NIASRA

NATIONAL INSTITUTE FOR APPLIED STATISTICS RESEARCH AUSTRALIA

National Institute for Applied Statistics Research Australia

University of Wollongong, Australia

Working Paper

01-23

Computational Implementation of Supernodal Variants for Cholesky Factorization and Calculation of the SIS

Luke Mazur and Robin Thompson

National Institute for Applied Statistics Research Australia, University of Wollongong, Wollongong NSW 2522, Australia T: +61 2 42215076. E: karink@uow.edu.au

Computational implementation of supernodal variants for Cholesky factorization and calculation of the SIS

Luke Mazur, Robin Thompson

September 4, 2023

1. The AI algorithm

This chapter briefly discusses the various quantities required in the calculation of the AI algorithm (Gilmour *et al.*, 1995).

1.1 Introducing the general linear mixed model

If the $n_{\boldsymbol{y}}$-vector of observations is denoted by $\boldsymbol{y}$, the linear mixed model can be written as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\tau} + \boldsymbol{Z}\boldsymbol{u} + \boldsymbol{e} \tag{1.1}$$

where $\boldsymbol{\tau}$ is the $n_{\boldsymbol{\tau}}$-vector of fixed effects, $\boldsymbol{X}$ is the $n_{\boldsymbol{y}} \times n_{\boldsymbol{\tau}}$ design matrix that associates observations with the appropriate combination of fixed effects, $\boldsymbol{u}$ is the $n_{\boldsymbol{u}}$-vector of random effects, $\boldsymbol{Z}$ is the $n_{\boldsymbol{y}} \times n_{\boldsymbol{u}}$ design matrix which associates observations with the appropriate combination of random effects, and $\boldsymbol{e}$ is the $n_{\boldsymbol{y}}$-vector of residuals. The column rank of $\boldsymbol{X}$ is denoted $d_{\boldsymbol{X}}$ and $d_{\boldsymbol{X}} \leq n_{\boldsymbol{\tau}}$. The model, in Equation (1.1), is called a linear mixed model or linear mixed-effects model. It is assumed

$$\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{e} \end{bmatrix} \sim N\left(\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}, \begin{bmatrix} \boldsymbol{G} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R} \end{bmatrix} \right). \tag{1.2}$$

The variance models given by the matrices $\boldsymbol{G}$ and $\boldsymbol{R}$ are called G-structures and R-structures respectively. In this report the only the uni-variate case is considered, however extensions to the multi-variate case are straightforward by replacing $\boldsymbol{y}$, $\boldsymbol{\tau}$, $\boldsymbol{u}$ and $\boldsymbol{e}$ by matrix equivalents, where each column corresponds to a response variable.

Typically $\boldsymbol{G}$ and $\boldsymbol{R}$ are functions of parameters that need to be estimated

$$\boldsymbol{G} = \boldsymbol{G}(\boldsymbol{\sigma}_{\boldsymbol{G}}) \tag{1.3}$$

and

$$\boldsymbol{R} = \boldsymbol{R}(\boldsymbol{\sigma}_{\boldsymbol{R}}). \tag{1.4}$$

The vectors $\boldsymbol{\sigma}_{\boldsymbol{G}}$ and $\boldsymbol{\sigma}_{\boldsymbol{R}}$ are parameter vectors associated with the random effects ($\boldsymbol{u}$) and residuals ($\boldsymbol{e}$) respectively.

Then a matrix $\boldsymbol{V}$ is defined such that

$$\boldsymbol{V} = \boldsymbol{R} + \boldsymbol{Z}\boldsymbol{G}\boldsymbol{Z}^\top,$$

so that

$$\boldsymbol{y} \sim N(\boldsymbol{X}\boldsymbol{\tau}, \boldsymbol{V}). \tag{1.5}$$

1.1.1 Variance structures for the residuals

In many datasets the vector of residuals represents the residuals from a single experiment and it is assumed that $\boldsymbol{R}$ is simply a scaled identity matrix, that is $\boldsymbol{R} = \sigma \boldsymbol{I}_{\boldsymbol{n_y}}$, where σ is the first and only element of the parameter vector $\boldsymbol{\sigma}_{\boldsymbol{R}}$ implying the the residuals were assumed independent and identically distributed.

In the datasets of interest in this report, that is MET datasets (Smith *et al.*, 2001), the vector $\boldsymbol{e}$ is a series of vectors indexed by a factor or factors. The sub-vectors relate to *sections* of the data, each with their own associated variance matrix. Thus in general it is written $\boldsymbol{e} = (\boldsymbol{e}_1^\top, \boldsymbol{e}_2^\top, \ldots, \boldsymbol{e}_s^\top)^\top$, so that $\boldsymbol{e}_j$ represents the vector of residuals of the j^{th} *section* of the data. The variance matrix for each section may differ, but it is assumed that the residuals from different sections are independent (if they are not then the dependent components can be coalesced into a single component and hence the independence structure can be maintained).

In matrix terms this gives

$$\boldsymbol{R} = \oplus_{j=1}^{s} \boldsymbol{R}_j,$$

where $\oplus$ is the direct sum operator. In MET datasets usually the indexing factor for a section is either `Trial` or `Environment`.

1.1.2 Variance structures for the random effects

The vector of random effects is often composed of ϕ sub-vectors $\boldsymbol{u} = (\boldsymbol{u}_1^\top, \boldsymbol{u}_2^\top, \ldots, \boldsymbol{u}_\phi^\top)^\top$ where the sub-vectors $\boldsymbol{u}_i$ are of length $n_{\boldsymbol{G_i}}$. These sub-vectors are assumed independent normally distributed with variance matrices $\boldsymbol{G}_i$. Thus, as for $\boldsymbol{R}$, this leads to

$$\boldsymbol{G} = \oplus_{i=1}^{\phi} \boldsymbol{G}_i.$$

There is a corresponding partition in $\boldsymbol{Z}$, namely $\boldsymbol{Z} = [\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots, \boldsymbol{Z}_\phi]$.

1.2 Prediction in the linear mixed model

The general aim is the prediction of the linear combination $\boldsymbol{\zeta}_{\boldsymbol{1}}^\top \boldsymbol{\tau} + \boldsymbol{\zeta}_{\boldsymbol{2}}^\top \boldsymbol{u}$ of fixed and random effects where $\boldsymbol{\zeta}_{\boldsymbol{1}}$ and $\boldsymbol{\zeta}_{\boldsymbol{2}}$ are known $n_{\boldsymbol{\tau}}$- and $n_{\boldsymbol{u}}$-vectors respectively, and further $\boldsymbol{\zeta}_{\boldsymbol{1}}^\top \boldsymbol{\tau}$ is estimable, i.e. $\boldsymbol{\zeta}_{\boldsymbol{1}}^\top$ lies in the row space of $\boldsymbol{X}$ (Searle & Khuri,

2017, p. 266). If $\boldsymbol{V}$ is known, implying $\boldsymbol{\sigma_G}$ and $\boldsymbol{\sigma_R}$ are known, the predictor which has the minimum mean square error (MSE) among the class of linear unbiased predictors is given by $\boldsymbol{\zeta_1^\top}\hat{\boldsymbol{\tau}} + \boldsymbol{\zeta_2^\top}\tilde{\boldsymbol{u}}$ where

$$\hat{\boldsymbol{\tau}} = (\boldsymbol{X}^\top \boldsymbol{V}^{-1} \boldsymbol{X})^- \boldsymbol{X}^\top \boldsymbol{V}^{-1} \boldsymbol{y}, \tag{1.6}$$

$$\tilde{\boldsymbol{u}} = \boldsymbol{G}\boldsymbol{Z}^\top \boldsymbol{P}\boldsymbol{y}, \tag{1.7}$$

$\boldsymbol{P} = \boldsymbol{V}^{-1} - \boldsymbol{V}^{-1}\boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{V}^{-1}\boldsymbol{X})^- \boldsymbol{X}^\top \boldsymbol{V}^{-1}$ and $(\boldsymbol{X}^\top \boldsymbol{V}^{-1}\boldsymbol{X})^-$ is a generalised inverse of $\boldsymbol{X}^\top \boldsymbol{V}^{-1}\boldsymbol{X}$ (Henderson, 1950).

These can be obtained by solving a system of equations known as the mixed model equations (MMEs), as proposed by Henderson (1950, 1973). They can be written in matrix-vector notation by:

$$\begin{bmatrix} \boldsymbol{X}^\top \boldsymbol{R}^{-1} \boldsymbol{X} & \boldsymbol{X}^\top \boldsymbol{R}^{-1} \boldsymbol{Z} \\ \boldsymbol{Z}^\top \boldsymbol{R}^{-1} \boldsymbol{X} & \boldsymbol{Z}^\top \boldsymbol{R}^{-1} \boldsymbol{Z} + \boldsymbol{G}^{-1} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{\tau}} \\ \tilde{\boldsymbol{u}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{X}^\top \boldsymbol{R}^{-1} \boldsymbol{y} \\ \boldsymbol{Z}^\top \boldsymbol{R}^{-1} \boldsymbol{y} \end{bmatrix}. \tag{1.8}$$

Letting $\boldsymbol{W} = [\boldsymbol{X}\ \boldsymbol{Z}], \tilde{\boldsymbol{\beta}}^\top = \left[\hat{\boldsymbol{\tau}}^\top\ \tilde{\boldsymbol{u}}^\top\right]$, $\boldsymbol{b} = \boldsymbol{W}^\top \boldsymbol{R}^{-1} \boldsymbol{y}$,

$$\boldsymbol{C} = \boldsymbol{W}^\top \boldsymbol{R}^{-1} \boldsymbol{W} + \boldsymbol{G}^*,$$

$$\boldsymbol{G}^* = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{G}^{-1} \end{bmatrix},$$

leads to a more abbreviated representation of the MMEs

$$\boldsymbol{C}\tilde{\boldsymbol{\beta}} = \boldsymbol{b}. \tag{1.9}$$

It can be shown that a specific solution to the MMEs is in fact equivalent to the estimates $\hat{\boldsymbol{\tau}}$ and $\tilde{\boldsymbol{u}}$ in Equations (1.6) and (1.7), known as empirical BLUEs (E-BLUEs) and empirical BLUPs (E-BLUPs) respectively.

1.3 The REML log-likelihood function

The REML log-likelihood (Patterson & Thompson, 1971) for Equation (1.1) can be written as

$$\ell_{\boldsymbol{R}} = -\frac{1}{2}\left\{2\log|\boldsymbol{F}| - \log|\boldsymbol{X}_{\boldsymbol{0}}^\top \boldsymbol{X}_{\boldsymbol{0}}| + \log|\boldsymbol{V}| + \log|\boldsymbol{X}_{\boldsymbol{0}}^\top \boldsymbol{V}^{-1} \boldsymbol{X}_{\boldsymbol{0}}| + \boldsymbol{y}^\top \boldsymbol{P}_{\boldsymbol{0}} \boldsymbol{y}\right\}, \tag{1.10}$$

where the choice of $\boldsymbol{F} = \boldsymbol{I}_{(n_{\boldsymbol{y}} - d_{\boldsymbol{X}})}$ and $\boldsymbol{X}_{\boldsymbol{0}}$ refers to the linearly independent columns of $\boldsymbol{X}$, see Mazur (2022) for more information on that.

1.4 Score equations and AI updating

Letting $\dot{\boldsymbol{R}}_{\boldsymbol{i}} = \partial \boldsymbol{R}/\partial \sigma_{\boldsymbol{R_i}}$ and $\boldsymbol{q}_{\boldsymbol{R_i}} = \dot{\boldsymbol{R}}_{\boldsymbol{i}} \boldsymbol{P_0 y}$, the REML score for a variance parameter $\sigma_{\boldsymbol{R_i}}$ associated with the residuals $\boldsymbol{e}$, following Gilmour *et al.* (1995), is given by

$$\boldsymbol{U}_{\boldsymbol{R}}(\sigma_{\boldsymbol{R_i}}) = -\frac{1}{2}\left\{\mathrm{tr}\left(\boldsymbol{R}^{-1}\dot{\boldsymbol{R}}_{\boldsymbol{i}}\right) - \mathrm{tr}\left(\boldsymbol{C}_{\boldsymbol{0}}^{-1}\boldsymbol{W}_{\boldsymbol{0}}^{\top}\boldsymbol{R}^{-1}\dot{\boldsymbol{R}}_{\boldsymbol{i}}\boldsymbol{R}^{-1}\boldsymbol{W}_{\boldsymbol{0}}\right) - \boldsymbol{y}^{\top}\boldsymbol{P_0}\boldsymbol{q}_{\boldsymbol{R_i}}\right\}, \tag{1.11}$$

where $\boldsymbol{C_0}$ refers to $\boldsymbol{C}$ with all instances of $\boldsymbol{X}$ replaced by $\boldsymbol{X_0}$. Letting $\dot{\boldsymbol{G}}_{\boldsymbol{i}} = \partial \boldsymbol{G}/\partial \sigma_{\boldsymbol{G_i}}$, $\boldsymbol{C}^{\boldsymbol{ZZ}}$ be the partition of $\boldsymbol{C}_{\boldsymbol{0}}^{-1}$ which corresponds to $\boldsymbol{u}$ and $\boldsymbol{q}_{\boldsymbol{G_i}} = \boldsymbol{Z}\dot{\boldsymbol{G}}_{\boldsymbol{i}}\boldsymbol{Z}^{\top}\boldsymbol{P_0 y}$, the REML score for a variance parameter $\sigma_{\boldsymbol{G_i}}$ is given by

$$\boldsymbol{U}_{\boldsymbol{R}}(\sigma_{\boldsymbol{G_i}}) = -\frac{1}{2}\left\{\mathrm{tr}\left(\boldsymbol{G}^{-1}\dot{\boldsymbol{G}}_{\boldsymbol{i}}\right) - \mathrm{tr}\left(\boldsymbol{G}^{-1}\dot{\boldsymbol{G}}_{\boldsymbol{i}}\boldsymbol{G}^{-1}\boldsymbol{C}^{\boldsymbol{ZZ}}\right) - \boldsymbol{y}^{\top}\boldsymbol{P_0}\boldsymbol{q}_{\boldsymbol{G_i}}\right\}. \tag{1.12}$$

The elements of the average information matrix are given by:

$$\boldsymbol{\mathcal{I}}_{\boldsymbol{A}}(\kappa_i, \kappa_j) = \frac{1}{2}\left(\dot{\boldsymbol{V}}_{\boldsymbol{i}}\boldsymbol{P_0 y}\right)^{\top}\boldsymbol{P_0}\left(\dot{\boldsymbol{V}}_{\boldsymbol{j}}\boldsymbol{P_0 y}\right). \tag{1.13}$$

2. Cholesky factorization and calculation of the sparse inverse subset by recursion

2.1 Defining the CF and the SIS

Typically $\boldsymbol{C}$ is a sparse matrix, meaning it has many zero elements, and the goal is to exploit this as much as possible to minimize runtime by avoiding redundant operations involving multiplication by zeroes. This report concentrates on the two most computationally intensive steps of the algorithm, that is the Cholesky factorization (CF) used in solving the MMEs and the calculation of the sparse inverse subset (SIS) which is the set of elements of the inverse of $\boldsymbol{C}$ necessary for the scores, i.e. the elements of $\boldsymbol{C}_{\mathbf{0}}^{-1}$ corresponding to non-zeroes in $\boldsymbol{C}_{\mathbf{0}}$. A CF typically refers to the calculation of a lower triangular matrix $\boldsymbol{L}$ ($\boldsymbol{L}$ is the convention in the literature as L stands for Lower) such that $\boldsymbol{L}\boldsymbol{L}^{\top} = \boldsymbol{C}$. In the algorithms present in this report, following Gilmour *et al.* (1995) and Meyer (1989), an operator `Factorize` is defined such that $\boldsymbol{L} = \texttt{Factorize}(\boldsymbol{A})$ denotes finding a lower triangular matrix $\boldsymbol{L}$ such that $\boldsymbol{L}^{\top}\boldsymbol{L} = \boldsymbol{A}$ for some SPD matrix $\boldsymbol{A}$. This could be considered an "inversely permuted" CF, and is simply referred to as a CF where the meaning is clear from context. This can be viewed as a form of extension of the $\sqrt{}$ operator to matrices, and if `Factorize` is applied to a 1×1 matrix then it behaves as the $\sqrt{}$ operator. For example, if $\boldsymbol{A}$ and $\boldsymbol{L}$ are 1×1 matrices then $\boldsymbol{L} = \texttt{Factorize}(\boldsymbol{A})$ is equivalent to $l_{11} = \sqrt{a_{11}}$ in this case.

CF of $\boldsymbol{C}$ can be used to transform a system of equations:

$$\boldsymbol{C}\tilde{\boldsymbol{\beta}} = \boldsymbol{b} \tag{2.1}$$

to a triangular system:

$$\boldsymbol{L}\tilde{\boldsymbol{\beta}} = (\boldsymbol{L}^{\top})^{-1}\boldsymbol{b} = \boldsymbol{h}. \tag{2.2}$$

The calculation of $\boldsymbol{h}$ is known as the back solution and this could occur after the CF is complete, but it is shown how it can be performed as part of the CF process.

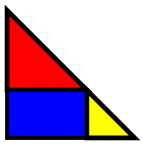
2.2 Recursion using multi-nodes

The recursion occurs by considering a multi-node at each level of the recursion. The lower triangle of a symmetric matrix $\boldsymbol{A}$ can be partitioned such that:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & * & * \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{m1} & \dots & \boldsymbol{A}_{mm} \end{bmatrix}.$$

Then it can be partitioned into m multi-nodes such that the i^{th} multi-node consists of all blocks of $\boldsymbol{A}$ in the i^{th} block row, i.e. $[\boldsymbol{A}_{i1}\ \boldsymbol{A}_{i2}\ \dots\ \boldsymbol{A}_{ii}]$. Each multi-node consists of two parts: a diagonal part (of which only the lower triangle is stored) and an off-diagonal part. The diagonal part of the i^{th} multi-node is simply the diagonal block $\boldsymbol{A}_{ii}$, while the off-diagonal part refers to all of the off-diagonal blocks, i.e. $\boldsymbol{A}_{i,1:i-1}$. The first multi-node only has a diagonal part i.e. the off-diagonal part of the first multi-node has no columns. A visual representation of the first two multi-nodes is shown in Figure 2.1.

Figure 2.1: A visual representation of the top two multi-nodes in terms of the elements that need to be stored. First multi-node (red) with diagonal part (yellow) and off-diagonal part (blue) of second multi-node.



In particular, the rows of $\boldsymbol{C}$ and $\boldsymbol{L}$ can be grouped into multi-nodes with m denoting the number of such multi-nodes. Then $\{\mathbb{r}_i\}$ denotes the set of rows in the i^{th} multi-node, numbering $|\mathbb{r}_i|$. Further $[i]$ is denoted as a multi-node comprised of the first i multi-nodes and $\{\mathbb{r}_{[i]}\} = \cup\{\mathbb{r}_j\}, j \leq i$. Then the number of rows associated with the first i multi-nodes is denoted by $|\mathbb{r}_{[i]}|$ (so $\{\mathbb{r}_{[m]}\}$ refers to all the rows and $|\mathbb{r}_{[m]}| = n$ is the total number of rows of $\boldsymbol{C}$). As a result, $\{\mathbb{r}_i\} = \{|\mathbb{r}_{[i-1]}| + 1 : |\mathbb{r}_{[i]}|\}$ Similarly $\{\mathbb{c}_i\}$ denotes the columns of the off-diagonal part of the i^{th} multi-node, numbering $|\mathbb{c}_i|$. The columns of the diagonal part correspond to $\{\mathbb{r}_i\}$.

There are two main advantages of multi-nodes. The first is that the off-diagonal part can be treated as a dense rectangular matrix. The second is that the diagonal part of a multi-node of $\boldsymbol{C}$ can be treated as a dense symmetric matrix while the diagonal part of a multi-node of $\boldsymbol{L}$ is treated as a dense triangular matrix.

The recursive process is initialized such that:

$$\boldsymbol{C}^{(m)} = \boldsymbol{C}, \tag{2.3}$$

$$\boldsymbol{b}^{(m)} = \boldsymbol{b}. \tag{2.4}$$

To denote the step of the recursion, k is used. Finally, in order to save memory in computer implementations, a separate object for $\boldsymbol{L}$ is not created. Instead, $\boldsymbol{C}$ is overwritten by its Cholesky factor, and then its SIS. This occurs using the *assignment operator* := as the recursion moves ahead.

Additionally, there are situations where $\boldsymbol{C}$ of size $n \times n$ is stored in contiguous memory but operations are only required on $\boldsymbol{C}_{:\{\mathbb{A}\}\{\mathbb{B}\}:}$ of size $|\mathbb{A}| \times |\mathbb{B}|$ which is not stored in contiguous memory, nor can it be treated as contiguous through the use of a fixed offset. Such a memory structure that is unsuitable for BLAS operations must be copied into a memory structure which is suitable, and this is denoted by the subscript of the partition being pre- and post-ceded by ":" symbols. Then for the sake of argument say it is desired to apply a function `Function` to $\boldsymbol{C}_{:\{\mathbb{A}\}\{\mathbb{B}\}:}$ but because it is not stored in contiguous memory they are instead applied to $\boldsymbol{A} = \boldsymbol{C}_{:\{\mathbb{A}\}\{\mathbb{B}\}:}$. This requires gathering $\boldsymbol{C}_{:\{\mathbb{A}\}\{\mathbb{B}\}:}$ into the shallow copy $\boldsymbol{A}$, performing the function (or some other operation) and then scattering $\boldsymbol{A}$ back into the relevant locations of $\boldsymbol{C}_{:\{\mathbb{A}\}\{\mathbb{B}\}:}$.

The function `Gather` gives a name to the targeted copying of rows of $\boldsymbol{C}$ contained in $\{\mathbb{A}\}$ and columns of $\boldsymbol{C}$ contained in $\{\mathbb{B}\}$ into $\boldsymbol{A}$. Letting $\{\mathbb{A}_i\}$ be a set containing only the i^{th} element of $\{\mathbb{A}\}$ such that $\{\mathbb{A}\} = \cup\{\mathbb{A}_i\}$ and similarly letting $\{\mathbb{B}_j\}$ be a set containing only the j^{th} element of $\{\mathbb{B}\}$. Then:

$$\boldsymbol{A}_{ij} = \boldsymbol{C}_{\{\mathbb{A}_i\}\{\mathbb{B}_j\}}.$$

Similarly the function `Scatter` gives a name to the copying of $\boldsymbol{A}$ back into $\boldsymbol{C}$ such that:

$$\boldsymbol{C}_{\{\mathbb{A}_i\}\{\mathbb{B}_j\}} := \boldsymbol{A}_{ij}.$$

The symbol :=: is used to refer to this procedure of gathering, assigning to and scattering back (if a term on the right hand side is not stored in a convenient memory structure this does not pose a problem, it is simply gathered and used as necessary without a change in notation) and so equivalently:

$$\boldsymbol{C}_{:\{\mathbb{A}\}\{\mathbb{B}\}:} :=: \mathit{Function}(\boldsymbol{C}_{:\{\mathbb{A}\}\{\mathbb{B}\}:}).$$

2.3 Sparsity and supernodes

To incorporate sparsity, sets $\{\mathbb{z}_i\}$ are defined such that $a \in \{\mathbb{z}_i\}$ if $a < i$ and $c_{ia} = 0$ after the CF is complete, i.e. the zeroes in the i^{th} row of the Cholesky factor, with i taking values from 1 to n. Similarly sets $\{\mathbb{c}_i\}$ are defined such that $a \in \{\mathbb{c}_i\}$ if $a < i$ and $c_{ia} \neq 0$, i.e. the non-zeroes in the i^{th} row of the Cholesky factor. Supernodes are multi-nodes with the constraint that all of the rows in the off-diagonal part of a supernode have the same non-zero structure. These supernodes are discovered in a matrix using a supernodal symbolic factorization which merges rows into sets $\{\mathbb{r}_k\}$, as well as the sets of non-zero columns for the supernodes denoted by $\{\mathbb{c}_k\}$ and the sets of zero columns for the supernodes denoted by $\{\mathbb{z}_k\}$. These are calculated using an elimination tree, see Liu

(1990). Then another set of merging occurs through the supernodal symbolic factorization such that if $\{\mathbb{z}_{\boldsymbol{d}}\}$ is sufficiently similar to $\{\mathbb{z}_{(\boldsymbol{d}+1)}\}$ then these two supernodes are merged together, even if this means explicitly storing some zero entries, as it was found to reduce runtime in the majority of cases (Ashcraft & Grimes, 1989).

Prior to this symbolic factorization, typically a permutation is used to minimize the number of structural non-zero entries in $\boldsymbol{L}$, by reducing the number of fill-in that occurs as zeroes are added to/subtracted from during the CF process: turning them into non-zeroes. After this permutation, a post-ordering occurs which permutes the children of each node in the elimination tree prior to its parents. This ordering can be computed using a recursive depth first search, where in each step it does a depth first search through all of the children of the current node, until it encounters leaf nodes (nodes with no children). Then it moves its way back up the stack of recursion, numbering the nodes in the order they were visited, i.e. with children being numbered before the parents. The procedure starts at the root, and the root is given its number once all of its children have been visited, i.e. at the end. This post-ordering has no impact on the number of non-zeroes in $\boldsymbol{L}$ but it means that children are typically placed consecutively, and these are the most likely to be able to be merged into supernodes as they are likely to have similar non-zero structures. In the following it is assumed that both of these orderings have been applied to $\boldsymbol{C}$ prior to the CF.

The code written by the author uses the supernodal amalgamation rules of CHOLMOD (Chen *et al.*, 2008) (an open source alternative discussed in Section 5.4), as it uses CHOLMOD for the entire symbolic factorization phase. The rules are that two supernodes are merged if any of the following conditions:

- The number of rows in the merged supernode is less than some parameter (by default 4)
- No new structural non-zeroes are formed by merging them
- The number of rows in the merged supernode is less than some parameter (by default 16) and the ratio of stored explicit zeroes to total structural non-zeroes in the merged supernode is less than some parameter (by default 0.8)
- The number of rows in the merged supernode is less than some parameter (by default 48) and the ratio of stored explicit zeroes to total structural non-zeroes in the merged supernode is less than some parameter (by default 0.1)
- The ratio of stored explicit zeroes to total structural non-zeroes in the merged supernode is less than some parameter (by default 0.05)

are met.

Then Equation (2.1) can be partitioned into:

$$\begin{bmatrix} \boldsymbol{C}^{(k)}_{\{\mathbb{T}^{(k)}\}\{\mathbb{T}^{(k)}\}} & \boldsymbol{C}^{(k)}_{\{\mathbb{T}^{(k)}\}\{\mathbb{B}^{(k)}\}} \\ \boldsymbol{C}^{(k)}_{\{\mathbb{B}^{(k)}\}\{\mathbb{T}^{(k)}\}} & \boldsymbol{C}^{(k)}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\beta}}_{\{\mathbb{T}^{(k)}\}} \\ \tilde{\boldsymbol{\beta}}_{\{\mathbb{B}^{(k)}\}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}^{(k)}_{\{\mathbb{T}^{(k)}\}} \\ \boldsymbol{b}^{(k)}_{\{\mathbb{B}^{(k)}\}} \end{bmatrix}, \tag{2.5}$$

with at the k^{th} step of the algorithm $\{\mathbb{T}^{(k)}\} = \{\mathbb{r}_{[\boldsymbol{k-1}]}\}$, $\{\mathbb{B}^{(k)}\} = \{\mathbb{r}_{\boldsymbol{k}}\}$. Then defining the following sets $\{\mathbb{Z}^{(k)}\} = \{\mathbb{z}_{\{\mathbb{r}_{\boldsymbol{k}}\}}\}$, $\{\mathbb{S}^{(k)}\} = \{\mathbb{c}_{\boldsymbol{k}}\} = \{\mathbb{T}^{(k)}\} - \{\mathbb{Z}^{(k)}\}$, the relevant elements at the k^{th} step can be accessed using $\{\mathbb{S}^{(k)}\}$ and $\{\mathbb{B}^{(k)}\}$. In the case where the matrix is entirely dense then there is only one supernode with $\{\mathbb{r}_{\boldsymbol{1}}\} = \{1 : n\}$, and this is equivalent to the multi-nodal case with one multi-node.

A direct consequence of supernodal methods is that $\boldsymbol{C}_{\{\mathbb{T}^{(k)}\}\{\mathbb{T}^{(k)}\}}$ is no longer stored, as many of the elements within it would be zero. Rather, $\boldsymbol{C}_{[\mathbb{F}^{(k-1)}]}$ is stored such that at the k^{th} step of the algorithm $[\mathbb{F}^{(k-1)}] = \cup\{a, b\}$ for $a \in \{\mathbb{r}_{\boldsymbol{i}}\}, b \in \{\mathbb{c}_{\boldsymbol{i}}\}$ for $i < k$, i.e. it is the union of all the elements present in the first $k-1$ supernodes. Similarly letting $[\mathbb{F}^{(k)}] = \cup\{a, b\}$ for $a \in \{\mathbb{r}_{\boldsymbol{i}}\}, b \in \{\mathbb{c}_{\boldsymbol{i}}\}$ for $i <= k$, i.e. it is the union of all the elements present in the first k supernodes.

3. Function notation

3.1 Cholesky factorization of the coefficient matrix of the MMEs

First $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ is factorized then $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}$ and $\boldsymbol{b}_{\{\mathbb{B}^{(k)}\}}$ are back-solved for and finally $\boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:}$ and $\boldsymbol{b}_{:\{\mathbb{S}^{(k)}\}:}$ are updated using the back-solutions, where the Schur complement is a specific case of a back-update.

$$
\begin{aligned}
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \texttt{Factorize}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}),\\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}} &:= ((\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}})^{-1})^{\top}\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}},\\
\boldsymbol{b}_{\{\mathbb{B}^{(k)}\}} &:= ((\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}})^{-1})^{\top}\boldsymbol{b}_{\{\mathbb{B}^{(k)}\}},\\
\boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:} &:=: \boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:} -\\
&\quad (\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}})^{\top}\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}},\\
\boldsymbol{b}_{:\{\mathbb{S}^{(k)}\}:} &:=: \boldsymbol{b}_{:\{\mathbb{S}^{(k)}\}:} - (\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}})^{\top}\boldsymbol{b}_{\{\mathbb{B}^{(k)}\}},
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \texttt{Factorize}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}), && (3.1)\\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}} &:= \texttt{Back-Solve}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}, \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}), && (3.2)\\
\boldsymbol{b}_{\{\mathbb{B}^{(k)}\}} &:= \texttt{Back-Solve}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}, \boldsymbol{b}_{\{\mathbb{B}^{(k)}\}}),\\
\\
\boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:} &:=: \texttt{SchurComplement}(\boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:}, \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}), && (3.3)\\
\boldsymbol{b}_{:\{\mathbb{S}^{(k)}\}:} &:=: \texttt{Back-UpdateSCP}(\boldsymbol{b}_{:\{\mathbb{S}^{(k)}\}:}, \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}, \boldsymbol{b}_{\{\mathbb{B}^{(k)}\}}). && (3.4)
\end{aligned}
$$

3.2 Calculation of the sparse inverse subset of the coefficient matrix of the MMEs

The first Assignment forward-solves for $\boldsymbol{A}$, while the second multiplies $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}$ by a symmetric matrix. After this $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ is inverted then multiplied by its transpose and finally $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ is forward-updated.

$$
\begin{aligned}
\boldsymbol{A} &:= (\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}})^{-1}\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}},\\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}} &:= -\boldsymbol{A}\boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:},\\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= (\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}})^{-1},\\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}})^{\top},\\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} - \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}\boldsymbol{A}^{\top},
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{A} &:= \texttt{Forward-Solve}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}, \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}), && (3.5)\\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}} &:= \texttt{SymmetricMultiply}(-\boldsymbol{A}, \boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:}), && (3.6)\\
&\quad ,\\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \texttt{InvertCrossMultiply}(\boldsymbol{I}, \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}), && (3.7)\\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \texttt{Forward-UpdateInv}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}, \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}, \boldsymbol{A}). && (3.8)
\end{aligned}
$$

There is an alternate formulation worth considering (after re-ordering Assignments (3.5-3.8) and initializing $\boldsymbol{S} := \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$, $\boldsymbol{A} := \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}$ and $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} := \boldsymbol{I}$) which first multiplies $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}$ by a symmetric matrix then $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ is forward-updated. After this $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}$ is forward solved for and finally $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ is pre- and post-multiplied by the inverse of $\boldsymbol{S}$:

$$
\begin{aligned}
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}} &:= -\boldsymbol{A}\boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:}, \\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} - \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}\boldsymbol{A}^\top, \\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}} &:= \boldsymbol{S}^{-1}\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}, \\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \boldsymbol{S}^{-1}\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}(\boldsymbol{S}^{-1})^\top,
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}} &:= \texttt{SymmetricMultiply}(-\boldsymbol{A}, \boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:}), && (3.9) \\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \texttt{Forward-UpdateInv}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}, \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}, \boldsymbol{A}), && (3.10) \\
\boldsymbol{A} &:= \texttt{Forward-Solve}(\boldsymbol{S}, \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}), \\
\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}} &:= \texttt{InvertCrossMultiply}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}, \boldsymbol{S}). && (3.11)
\end{aligned}
$$

4. Singularity-handling supernodal AI algorithm

4.1 Literature review

While CF is typically thought to only work for SPD matrices, it can work for SPSD (Symmetric Positive Semi-Definite) matrices in two main ways. The first is known as a modified CF (Bunch, 1971; Schnabel & Eskow, 1990; Eskow & Schnabel, 1991) where small numbers are added to the diagonal elements to effectively make the matrix SPD. A block approach to this is discussed in Dayde & Camichel (1995). If this matrix was the coefficient matrix for a linear system then such a modification modifies the solution of the system and so will not be discussed further herein.

The second way is to simply avoid the rows that have diagonal entries below a certain threshold in some way. There are two methods to avoid singularities when solving a linear system where the coefficient matrix is SPSD, without affecting the solution. In the first method they are set to extremely large numbers with off-diagonals set to zero and corresponding elements of the solution set to zero, essentially "fixing" them so that they do not cause numerical instability. In fact numerically they will have no impact on the remainder of the operations. This method is denoted the "zeroing-out" method. The second method is to mark them and avoid them entirely, such that they take no part in any future computation, denoted the "avoiding" method. The first method is discussed in detail in Gupta *et al.* (1997); Gupta (2000).

Of the two methods the avoiding should be more efficient as redundant operations are avoided, and this is the choice method in the dense linear algebra space, and in some sparse single-nodal codes such as ASReml. This method has also been applied for sparse unsymmetric nondefinite matrices and sparse symmetric indefinite matrices, but these problems are different as the tiny pivots cannot immediately be flagged as singularities, as it is possible that a negative pivot further in the matrix will modify the tiny pivot back to a suitable size. As a result, dynamic permutation must occur of the matrix in order to delay this pivot, and this typically causes fill-in to occur as it represents a departure from

the original ordering scheme. As far as the author is aware there has not been an implementation that avoids these singularities in a supernodal algorithm for the factorization of an SPSD matrix, and this report presents such an algorithm that while it features dynamic permutation, it does not lead to an increase in fill-in, as rows within a supernode can be interchanged without introducing any fill-in.

Since the software ASReml uses a CF that can handle singularities for every iteration, a supernodal CF for an SPSD is demonstrated herein through an adjustment to the `Factorize` operator, in Subsection 4.1.1. However, typically in the AI algorithm the singularities are only present in $\boldsymbol{C}_{\boldsymbol{xx}}$ due to aliasing in the fixed effects and so such a CF could simply be applied to this partition of $\boldsymbol{C}$, which is relatively small for most models used for plant variety trials. The singularities could be detected, then a full rank parameterization could be selected and used for all remaining iterations since the singularities are typically the same for each iteration as they are independent of the variance parameters. The time for this CF would be a relatively minor portion of the time required for fitting most models, then the AI algorithm could proceed with an SPD $\boldsymbol{C}$ matrix. Nevertheless, other applications such as chemical physics or linear programming feature matrices that can have singularities throughout the coefficient matrix, and only require one CF to be performed in their respective algorithms. In these applications having a faster method to factorize an SPSD matrix could potentially be quite useful.

For a more comprehensive review of the Cholesky factorization of SPSD matrices see Mazur (2022).

4.1.1 Singularity-Handling Factorization Operator

The operator $\texttt{Factorize}^{+}$ is defined to refer to a `Factorize` operation that can handle a singular input matrix. While for example $\texttt{Factorize}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}})$ only modifies $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$, $\texttt{Factorize}^{+}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}})$ may modify either $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ to make it non-singular (as in the zeroing-out method), or modify $\{\mathbb{B}^{(k)}\}$ to create a $\{\mathbb{B}^{(\Bbbk)+}\}$ to ignore the linearly dependent rows of $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ (as in the avoiding method). This is achieved by permuting all of the linearly dependent rows in the supernode to the top of the supernode such that $\{\mathbb{B}^{(\Bbbk)+}\}$ remains a contiguous set, and keeping track of their number denoted as $\zeta_{\{\mathbb{B}^{(k)}\}}$. As a result $d_{\{\mathbb{B}^{(k)}\}} = |\mathbb{r}_{\boldsymbol{k}}| - \zeta_{\{\mathbb{B}^{(k)}\}}$ denotes the rank of $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$. In either case $\texttt{Factorize}^{+}$ denotes finding an $\boldsymbol{L}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} = \texttt{Factorize}^{+}(\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}})$ such that $(\boldsymbol{L}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}})^{\top}\boldsymbol{L}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} = \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}$, sans potential modification of the linearly dependent rows, and $\{\mathbb{B}^{(\Bbbk)+}\} = \{\mathbb{B}^{(k)}\}$ for the zeroing-out method (and thus $\zeta_{\{\mathbb{B}^{(k)}\}} = 0$ and $d_{\{\mathbb{B}^{(k)}\}} = |\mathbb{B}^{(\Bbbk)}|$). In the SPD case `Factorize` and $\texttt{Factorize}^{+}$ achieve the same result.

Further bookkeeping is required for the avoiding method, as all of the permutations are applied only within each supernode locally during the CF, and once the CF has been completed for all supernodes, they can be "synchronised" such that they all share the new matrix ordering, and the columns corresponding to linearly dependent rows can be avoided at the cost of some additional bookkeeping which leads to a potential reduction in the size of $\{\mathbb{S}^{(k)}\}$.

5. Computational Implementation

5.1 Introduction

This chapter shows how the functions (defined in Chapter 3) are then implemented in an efficient fashion, primarily through the use of highly optimized third party library routines. These routines are first introduced in Section 5.2, and then it is shown how with the use of some minor pre-processing steps (including pre- and post-multiplication by permutation matrices see Appendix A), the routines can be used to perform the computations necessary for the functions (Section 5.3). Existing open source implementations of parts of the AI algorithm based on some of these routines are also introduced in Section 5.4. Concluding remarks are presented in Section 5.5.

5.2 Basic Linear Algebra Subprograms (BLAS), Linear Algebra PACKage (LAPACK) and BLAS-like subroutines

The following is a brief alphabetically ordered list of Basic Linear Algebra Subprograms (BLAS), LAPACK and BLAS-like subroutines. Their purpose is to implement some specific mathematical operation on a dense matrix which is stored as a collection of some form of decimal number. The first letter of the routine determines the type of decimal number the subroutine operates on (except for `TWOSIDEDTRSM` because it is written in C++ and is templated, so it automatically works for any of the various data types), in this work all subroutines work with double precision numbers, so all subroutines are preceded with a "D" which is prepended when the subroutine is formally referred to. In the following this first letter is omitted. The names are typically somewhat informative regarding what the subroutine does but there is no fixed rule to determine it. The following has the name and level of the subroutine (level 3

is typically order n^3 FLOPs: matrix-matrix operations while level 2 is typically order n^2 matrix-vector operations and level 1 is typically order n vector scalar operations), followed by the purpose of the subroutine, followed by the algebra, followed by roughly how many FLOPs it requires (many of the calculations are of the form of the number of FLOPs per element multiplied by the number of elements, for the derivation of the non-trivial FLOP results see Mazur (2022)), and finally where it is used. The below are implemented in the Intel MKL and are documented in Intel (2019), with the exception of `TWOSIDEDTRSM` is documented in Poulson *et al.* (2013).

GEMM–3	"Computes a matrix-matrix product with general matrices. $\boldsymbol{C} := \alpha\times$ op$(\boldsymbol{A}) \times$ op$(\boldsymbol{B}) + \beta \times \boldsymbol{C}$, where op$(\boldsymbol{X})$ is one of op$(\boldsymbol{X}) = \boldsymbol{X}$, or op$(\boldsymbol{X}) = \boldsymbol{X}^\top$, α and β are scalars, $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are matrices: op$(\boldsymbol{A})$ is an m-by-k matrix, op$(\boldsymbol{B})$ is a k-by-n matrix, $\boldsymbol{C}$ is an m-by-n matrix", roughly $2mkn$ FLOPs. Used in back-updating and in updating the inverse.
GEMMT–3	"Computes a matrix-matrix product with general matrices but updates only the upper or lower triangular part of the result matrix. The operation is defined as $\boldsymbol{C} := \alpha\times$ op$(\boldsymbol{A}) \times$ op$(\boldsymbol{B}) + \beta \times \boldsymbol{C}$, where op$(\boldsymbol{X})$ is one of op$(\boldsymbol{X}) = \boldsymbol{X}$, or op$(\boldsymbol{X}) = \boldsymbol{X}^\top$, α and β are scalars, $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are matrices: op$(\boldsymbol{A})$ is an n-by-k matrix, op$(\boldsymbol{B})$ is a k-by-n matrix, $\boldsymbol{C}$ is an n-by-n upper or lower triangular matrix", roughly n^2k FLOPs. Used in forward-updating the inverse.
GEMV–2	"Computes a matrix-vector product using a general matrix." "The operation is defined as $\boldsymbol{y} := \alpha \times \boldsymbol{A} \times \boldsymbol{x} + \beta \times \boldsymbol{y}$ or $\boldsymbol{y} := \alpha \times \boldsymbol{A}^\top \times \boldsymbol{x} + \beta \times \boldsymbol{y}$ where α and β are scalars, $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors and $\boldsymbol{A}$ is an m-by-n matrix.", roughly $2mn$ FLOPs. Used in forward-updating.
GER–2	"Performs a rank-1 update of a general matrix. $\boldsymbol{A} := \alpha \times \boldsymbol{x} \times \boldsymbol{y}^\top + \boldsymbol{A}$ where α is a scalar, $\boldsymbol{x}$ is an m-element vector, $\boldsymbol{y}$ is an n-element vector, $\boldsymbol{A}$ is an m-by-n general matrix", roughly $2mn$ FLOPs. Used in factorization.
LAUUM–3	"Computes the product $[\boldsymbol{A} :=] \boldsymbol{U} \times \boldsymbol{U}^\top$ or $[\boldsymbol{A} :=] \boldsymbol{L}^\top \times \boldsymbol{L}$", roughly $n^3/3$ FLOPs where n is the order of the matrix. Used in invert and cross multiply.
POTRF–3	"Computes the Cholesky factorization of a symmetric (Hermitian) positive-definite matrix." Finds the

	$\boldsymbol{L}$ (order n) such that $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^\top$ then $\boldsymbol{A} := \boldsymbol{L}$, roughly $n^3/3$ FLOPs. Used in factorization.
PSTRF–3	"Computes the Cholesky factorization with complete pivoting of a real symmetric (complex Hermitian) positive semi-definite matrix." Finds the $\boldsymbol{L}$ (order n) such that $\boldsymbol{P}^\top \boldsymbol{A}\boldsymbol{P} = \boldsymbol{L}\boldsymbol{L}^\top$ "where $\boldsymbol{P}$ is a permutation matrix stored as vector piv" then $\boldsymbol{A} := \boldsymbol{L}$, roughly $n^3/3$ FLOPs. Used in factorization.
SCAL–1	"Computes the product of a vector by a scalar. $\boldsymbol{x} := a \times \boldsymbol{x}$ where a is a scalar, $\boldsymbol{x}$ is an n-element vector", roughly n FLOPs. Used in factorization.
SWAP–1	"Swaps a vector with another vector" $\boldsymbol{x}$,$\boldsymbol{y} := \boldsymbol{y}$,$\boldsymbol{x}$, roughly $2n$ FLOPs where n is the size of the vector. Used in factorization.
SYMM–3	"Computes a scalar-matrix-matrix product with one symmetric matrix and add the result to a scalar-matrix product . The operation is defined as $\boldsymbol{C} := \alpha \times \boldsymbol{A} \times \boldsymbol{B} + \beta \times \boldsymbol{C}$ or $\boldsymbol{C} := \alpha \times \boldsymbol{B} \times \boldsymbol{A} + \beta \times \boldsymbol{C}$, where where α and β are scalars, $\boldsymbol{A}$ is a symmetric matrix, $\boldsymbol{B}$ and $\boldsymbol{C}$ are m-by-n matrices", roughly m^2n FLOPs in the first case. Used in symmetric multiplication.
SYRK–3	"Performs a symmetric rank-k update." "The operation is defined as $\boldsymbol{C} := \alpha \times \boldsymbol{A} \times \boldsymbol{A}^\top + \beta \times \boldsymbol{C}$, or $\boldsymbol{C} := \alpha \times \boldsymbol{A}^\top \times \boldsymbol{A} + \beta \times \boldsymbol{C}$, where α and β are scalars, $\boldsymbol{C}$ is an n-by-n symmetric matrix, $\boldsymbol{A}$ is an n-by-k matrix in the first case and a k-by-n matrix in the second case", roughly n^2k FLOPs. Used in back-updating.
TRSM–3	"Solves a triangular matrix equation" (and thus essentially multiplies a matrix by the inverse of a triangular matrix in a cheaper way) "$\mathrm{op}(\boldsymbol{A}) \times \boldsymbol{X} = \alpha \times \boldsymbol{B}$, or $\boldsymbol{X} \times \mathrm{op}(\boldsymbol{A}) = \alpha \times \boldsymbol{B}$, where α is a scalar, $\boldsymbol{X}$ and $\boldsymbol{B}$ are m-by-n matrices, $\boldsymbol{A}$ is a unit, or non-unit, upper or lower triangular matrix, $\mathrm{op}(\boldsymbol{A})$ is one of $\mathrm{op}(\boldsymbol{A}) = \boldsymbol{A}$, or $\mathrm{op}(\boldsymbol{A}) = \boldsymbol{A}^\top$" "The matrix $\boldsymbol{B}$ is overwritten by the solution matrix $\boldsymbol{X}$", i.e. $\boldsymbol{X} := \boldsymbol{B}$, roughly m^2n FLOPs. Used in forward-solution and back-solution.
TRTRI–3	"Computes the inverse of a triangular matrix". $\boldsymbol{A} := \boldsymbol{A}^{-1}$ where $\boldsymbol{A}$ is a triangular matrix, roughly $n^3/3$ FLOPs where n is the order of the matrix. Used in invert and cross multiply.

TWOSIDEDTRSM–3 "Performs a two-sided triangular solves with multiple right-hand sides which preserves the symmetry of the input matrix, either $\boldsymbol{A} := \boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-\top}$ or $\boldsymbol{A} := \boldsymbol{U}^{-\top}\boldsymbol{A}\boldsymbol{U}^{-1}$", roughly n^3 FLOPs where n is the order of the matrix. Used in invert and cross multiply.

5.3 BLAS implementations

In this section function names from Section 3 can be shown on the left with their BLAS/LAPACK/BLAS-like implementations on the right.

5.3.1 Implementation of `Factorize`

5.3.1.1 LAPACK variant

This variant factorizes $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ using a LAPACK optimized routine built for this purpose which achieves the factorization presumably via some recursively blocked process, as used in the open source version. It is impossible to know exactly how the Intel MKL implementation operates because it is not open source. The subroutine `DPOTRF` computes the factorization $\boldsymbol{L}\boldsymbol{L}^{\top}$ rather than $\boldsymbol{L}^{\top}\boldsymbol{L}$ as the CF described previously requires. The matrix $\boldsymbol{T} \equiv \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ is denoted as the diagonal part of the k^{th} supernode, where in this case there is no copying as $\boldsymbol{T}$ is not actually formed, but it is purely for notational purposes and any changes that occur to $\boldsymbol{T}$ occur to $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$—in other words a deep copy. Reverse permutations were used to create a $\boldsymbol{F}$ such that $\boldsymbol{F} = \boldsymbol{P}_{\boldsymbol{Rev}}\boldsymbol{T}\boldsymbol{P}_{\boldsymbol{Rev}}$. Then $\{\mathbb{A}^{(k)}\} = \{|\mathbb{B}^{(k)}| - i + 1\}$, $\{\mathbb{C}^{(k)}\} = \{|\mathbb{B}^{(k)}| - j + 1\}$, $\{\mathbb{D}^{(k)}\} = \{j\}$ and $\{\mathbb{E}^{(k)}\} = \{i\}$ with j iterating from $|\mathbb{B}^{(k)}|$ to 1 and i iterating from 1 to j:

$$\boldsymbol{F}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} := \boldsymbol{T}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}} \quad \Big| \quad \boldsymbol{F}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} = \boldsymbol{T}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}}. \tag{5.1}$$

Then $\boldsymbol{F}$ is factorized:

$$\boldsymbol{F} := \texttt{Factorize}(\boldsymbol{F}) \quad \Big| \quad \texttt{DPOTRF}(\text{“}L\text{”}, |\mathbb{B}^{(k)}|, \boldsymbol{F}, |\mathbb{B}^{(k)}|, \texttt{INFO}), \tag{5.2}$$

where `INFO` holds an error value that provides information in case something went wrong. Finally $\boldsymbol{F}$ needs to be reversely permuted back into $\boldsymbol{T}$:

$$\boldsymbol{T}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} := \boldsymbol{F}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}} \quad \Big| \quad \boldsymbol{T}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} = \boldsymbol{F}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}}. \tag{5.3}$$

5.3.2 Implementation of $\mathtt{Factorize}^+$

5.3.2.1 LAPACK variant with singularity handling

This variant is an alternative LAPACK variant that uses the avoiding method to handle singularities, which works in the same way as the above except it uses DPSTRF instead of DPOTRF.

$$\boldsymbol{F}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} := \boldsymbol{T}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}} \Big| \qquad \boldsymbol{F}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} = \boldsymbol{T}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}}. \tag{5.4}$$

The matrix $\boldsymbol{T}$ could have been permuted in place without the use of $\boldsymbol{F}$ but such permutation requires some temporary space anyway, so $\boldsymbol{F}$ simplifies operations in this regard. Then $\boldsymbol{F}$ is factorized:

$$\boldsymbol{F} := \mathtt{Factorize}^+(\boldsymbol{F}) \quad \Big| \quad \mathtt{DPSTRF}(\text{“}L\text{”}, |\mathbb{B}^{(\Bbbk)}|, \boldsymbol{F}, |\mathbb{B}^{(\Bbbk)}|, \mathtt{PERM}, d_{\{\mathbb{B}^{(k)}\}}, 0.0001, \mathtt{WORK}, \mathtt{INFO}). \tag{5.5}$$

The subroutine DPSTRF factorizes the input matrix column by column starting from the left and after each column it permutes the column with the largest pivot value to be the next column factorized, as this is what the open source version does. It is hard to know exactly what goes on in the Intel MKL implementation as it is closed source, but this is the best guess of the author. This permutation is stored in PERM. Once none of the remaining pivots are greater than the tolerance (0.0001 was chosen for this) then the rest of the columns are classed as singular and then the rank of $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ is reported in $d_{\{\mathbb{B}^{(k)}\}}$. The vector WORK is just some extra memory (presumably to facilitate the permutations) and INFO holds an error value that provides information in case something went wrong. Finally $\boldsymbol{F}$ needs to be reversely permuted back into $\boldsymbol{T}$:

$$\boldsymbol{T}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} := \boldsymbol{F}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}} \qquad \Big| \qquad \boldsymbol{T}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} = \boldsymbol{F}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}}, \tag{5.6}$$

then $\zeta_{\{\mathbb{B}^{(k)}\}} = |\mathbb{B}^{(\Bbbk)}| - d_{\{\mathbb{B}^{(k)}\}}$ and finally $\{\mathbb{B}^{(\Bbbk)+}\} := \{\mathbb{B}^{(k)}\}_{1+\zeta_{\{\mathbb{B}^{(k)}\}}:|\mathbb{B}^{(\Bbbk)}|}$

5.3.2.2 Matrix variant with the zeroing-out method for singularity-handling

This variant factorizes $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ row by row with j iterating from $|\mathbb{B}^{(\Bbbk)}|$ to 1 and $\boldsymbol{T}$ is as above. Then defining $\{\mathbb{T}^*\} = \{1 : (j-1)\}$, $\{\mathbb{B}^*\} = \{j\}$,

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{T}_{:\{\mathbb{T}^*\}\{\mathbb{T}^*\}:} & (\boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}})^\top \\ \boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}} & t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} \end{bmatrix}. \tag{5.7}$$

If $(t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} < 0.0001)$ Then
$t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} := 100000000$
$\boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}} := 0$
Else
$t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} := t^{1/2}_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}}$
$\boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}} := t^{-1}_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} \boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}$
$\boldsymbol{T}_{:\{\mathbb{T}^*\}\{\mathbb{T}^*\}:} := \boldsymbol{T}_{:\{\mathbb{T}^*\}\{\mathbb{T}^*\}:} -$
$(\boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}})^\top \boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}$
End If

and finally $\{\mathbb{B}^{(\Bbbk)+}\} := \{\mathbb{B}^{(k)}\}$.

If $(t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} < 0.0001)$ Then
$t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} := 100000000$ (5.8)
$\boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}} := 0$ (5.9)
Else
$t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} := \mathtt{SQRT}(t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}})$ (5.10)
$\mathtt{DSCAL}(j-1, (t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}})^{-1}, \boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}, 1)$ (5.11)
$\mathtt{DSYRK}(\text{“}L\text{”}, \text{“}T\text{”}, j-1, 1, -1.0, \boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}},$
$1, 1.0, \boldsymbol{T}_{\{1\},\{1\}}, |\mathbb{B}^{(\Bbbk)}|)$ (5.12)
End If

5.3.2.3 Matrix variant with the avoiding method for singularity-handling

This variant factorizes $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ row by row with j iterating from $|\mathbb{B}^{(\Bbbk)}|$ to 1, and finishing prematurely if $j = \zeta_{\{\mathbb{B}^{(k)}\}}$ as this means all the remaining rows are linearly dependent. The matrix $\boldsymbol{T}$ is as above and denoting $\boldsymbol{S} \equiv \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}(\{\mathbb{S}^{(k)}\}\cup\{\mathbb{B}^{(k)}\})} \equiv [\,\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{S}^{(k)}\}}\ \boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}\,] \equiv [\,\boldsymbol{R}\ \boldsymbol{T}\,]$ (i.e. $\boldsymbol{S}$ refers to the entire supernode while $\boldsymbol{R}$ refers to the off-diagonal part) and $\boldsymbol{f} \equiv \boldsymbol{b}_{\{\mathbb{B}^{(k)}\}}$. Both $\boldsymbol{S}$ and $\boldsymbol{f}$ are dense. Then defining $\{\mathbb{T}^*\} = \{(1 + \zeta_{\{\mathbb{B}^{(k)}\}}) : (j-1)\}$, $\{\mathbb{B}^*\} = \{j\}$. Finally denoting a temporary a and $\boldsymbol{A}$ that are meaningless and serve only to temporarily store values due to swapping (the equivalent of `WORK` in `DPSTRF`)

If $(t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} < 0.0001)$ Then

$\zeta_{\{\mathbb{B}^{(k)}\}} := \zeta_{\{\mathbb{B}^{(k)}\}} + 1$

$\boldsymbol{A} := \boldsymbol{T}_{1:|\mathbb{B}^{(\Bbbk)}|,\zeta_{\{\mathbb{B}^{(k)}\}}}$

$\boldsymbol{T}_{1:|\mathbb{B}^{(\Bbbk)}|,\zeta_{\{\mathbb{B}^{(k)}\}}} := \boldsymbol{T}_{1:|\mathbb{B}^{(\Bbbk)}|,j}$

$\boldsymbol{T}_{1:|\mathbb{B}^{(\Bbbk)}|,j} := \boldsymbol{A}$

$\boldsymbol{A} := \boldsymbol{S}_{j,1:(|\mathbb{B}^{(\Bbbk)}|+|\mathbb{S}^{(k)}|)}$

$\boldsymbol{S}_{j,1:(|\mathbb{B}^{(\Bbbk)}|+|\mathbb{S}^{(k)}|)} := \boldsymbol{S}_{\zeta_{\{\mathbb{B}^{(k)}\}},1:(|\mathbb{B}^{(\Bbbk)}|+|\mathbb{S}^{(k)}|)}$

$\boldsymbol{S}_{\zeta_{\{\mathbb{B}^{(k)}\}},1:(|\mathbb{B}^{(\Bbbk)}|+|\mathbb{S}^{(k)}|)} := \boldsymbol{A}$

$a := \boldsymbol{f}_{\zeta_{\{\mathbb{B}^{(k)}\}}}$

$\boldsymbol{f}_{\zeta_{\{\mathbb{B}^{(k)}\}}} := \boldsymbol{f}_j$

$\boldsymbol{f}_j := a$

Else

$t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} := t^{1/2}_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}}$

$\boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}} := t^{-1}_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} \boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}$

$\boldsymbol{T}_{:\{\mathbb{T}^*\}\{\mathbb{T}^*\}:} := \boldsymbol{T}_{:\{\mathbb{T}^*\}\{\mathbb{T}^*\}:} -$
$(\boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}})^\top \boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}$

End If

If $(t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} < 0.0001)$ Then

$$\zeta_{\{\mathbb{B}^{(k)}\}} := \zeta_{\{\mathbb{B}^{(k)}\}} + 1 \tag{5.13}$$

$$\texttt{DSWAP}(|\mathbb{B}^{(\Bbbk)}|, \boldsymbol{T}_{1:|\mathbb{B}^{(\Bbbk)}|,\zeta_{\{\mathbb{B}^{(k)}\}}}, 1, \boldsymbol{T}_{1:|\mathbb{B}^{(\Bbbk)}|,j}, 1) \tag{5.14}$$

$$\texttt{DSWAP}(|\mathbb{S}^{(k)}|, \boldsymbol{S}_{j,1:(|\mathbb{B}^{(\Bbbk)}|+|\mathbb{S}^{(k)}|)}, 1, \boldsymbol{S}_{\zeta_{\{\mathbb{B}^{(k)}\}},1:(|\mathbb{B}^{(\Bbbk)}|+|\mathbb{S}^{(k)}|)}, 1) \tag{5.15}$$

$$\texttt{SWAPDOUBLE}(\boldsymbol{f}_{\zeta_{\{\mathbb{B}^{(k)}\}}}, \boldsymbol{f}_j) \tag{5.16}$$

Else

$$t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} := \texttt{SQRT}(t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}}) \tag{5.17}$$

$$\texttt{DSCAL}(j-1, (t_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}})^{-1}, \boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}, 1) \tag{5.18}$$

$$\texttt{DSYRK}(\text{“}L\text{”}, \text{“}T\text{”}, j-1-\zeta_{\{\mathbb{B}^{(k)}\}}, 1, -1.0, \boldsymbol{T}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}, 1, 1.0, \boldsymbol{T}_{\{1+\zeta_{\{\mathbb{B}^{(k)}\}}\},\{1+\zeta_{\{\mathbb{B}^{(k)}\}}\}}, |\mathbb{B}^{(\Bbbk)}|) \tag{5.19}$$

End If

Where SWAPDOUBLE is a subroutine written by the author to swap two double precision numbers rather than using any BLAS for that task. Finally $\{\mathbb{B}^{(\Bbbk)+}\} := \{\mathbb{B}^{(k)}\}_{1+\zeta_{\{\mathbb{B}^{(k)}\}}:|\mathbb{B}^{(\Bbbk)}|}$.

5.3.3 Implementation of Back-Solve

There is only one variant of this which uses the dense BLAS triangular solve to do the solve. Then newly defining $\boldsymbol{T}^+ \equiv \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}$, $\boldsymbol{R}^+ \equiv \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{S}^{(k)}\}}$ to be the off-diagonal part of the supernode :

$$\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{S}^{(k)}\}} := (\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}})^{-1})^\top \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{S}^{(k)}\}} \quad \texttt{DTRSM}(\text{“}L\text{”}, \text{“}L\text{”}, \text{“}T\text{”}, \text{“}N\text{”}, |\mathbb{B}^{(\Bbbk)+}|, |\mathbb{S}^{(k)}|, 1.0, \boldsymbol{T}^+_{\{1\},\{1\}}, |\mathbb{B}^{(\Bbbk)}|, \boldsymbol{R}^+_{\{1\},\{1\}}, |\mathbb{B}^{(\Bbbk)}|) \tag{5.20}$$

The implementation of Forward-Solve occurs in essentially the same way except that the triangular matrix is not transposed.

5.3.4 Implementation of **Factorize** and **Back-Solve** together with the zeroing-out method for singularity-handling

This vector-vector variant achieves the Factorize and Back-Solve steps together row by row with j iterating from $|\mathbb{B}^{(\Bbbk)}|$ to 1. The matrices $\boldsymbol{S}$ and $\boldsymbol{f}$ are defined as above. Then defining $\lambda = (|\mathbb{S}^{(k)}|+j)$, $\iota = (|\mathbb{S}^{(k)}|+1)$, $\{\mathbb{T}^*\} = \{1 : (j-1)\}$, $\{\mathbb{B}^*\} = \{j\}$, $\{\mathbb{L}^*\} = \{1 : (\lambda - 1)\}$, $\{\mathbb{R}^*\} = \{\lambda\}$, $\{\mathbb{D}^*\} = \{\iota : (\lambda - 1)\}$.

Then the following partitioning can occur:

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{S}_{:\{\mathbb{T}^*\}\{\mathbb{L}^*\}:} & (\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}})^\top \\ \boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}} & s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} \end{bmatrix}, \tag{5.21}$$

$$\boldsymbol{f} = \begin{bmatrix} \boldsymbol{f}_{\{\mathbb{T}^*\}} \\ f_{\{\mathbb{B}^*\}} \end{bmatrix}. \tag{5.22}$$

If $(s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} < 0.0001)$ Then
- $s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} := 100000000$
- $f := 0$
- $\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}} := 0$

Else
- $s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} := (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{1/2}$
- $f_{\{\mathbb{B}^*\}} := (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{-1} f_{\{\mathbb{B}^*\}}$
- $\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}} := (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{-1} \boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}}$
- $\boldsymbol{f}_{\{\mathbb{T}^*\}} := \boldsymbol{f}_{\{\mathbb{T}^*\}} -$
- $(\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{D}^*\}})^\top f_{\{\mathbb{B}^*\}}$
- $\boldsymbol{S}_{:\{\mathbb{T}^*\}\{\mathbb{L}^*\}:} := \boldsymbol{S}_{:\{\mathbb{T}^*\}\{\mathbb{L}^*\}:} -$
- $(\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{D}^*\}})^\top \boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}}$

End If

If $(s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} < 0.0001)$ Then
- $s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} := 100000000$ (5.23)
- $f := 0$ (5.24)
- $\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}} := 0$ (5.25)

Else
- $s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} := \mathtt{SQRT}(s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})$ (5.26)
- $\mathtt{DSCAL}(1, (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{-1}, f_{\{\mathbb{B}^*\}}, 1)$ (5.27)
- $\mathtt{DSCAL}(\lambda - 1, (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{-1}, \boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}})$ (5.28)
- $\mathtt{DGER}(j-1, 1, -1.0, (\boldsymbol{s}_{\{\mathbb{B}^*\}\{\iota\}})^\top, 1,$
- $f_{\{\mathbb{B}^*\}}, 1, 1, \boldsymbol{f}_{\{\mathbb{T}^*\}}, |\mathbb{B}^{(\Bbbk)}|)$ (5.29)
- $\mathtt{DGER}(j-1, \lambda-1, -1.0, (\boldsymbol{s}_{\{\mathbb{B}^*\}\{\iota\}})^\top, 1,$
- $\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}}, 1, \boldsymbol{S}_{\{1\}\{1\}}, |\mathbb{B}^{(\Bbbk)}|)$ (5.30)

End If

Finally $\{\mathbb{B}^{(\Bbbk)+}\} := \{\mathbb{B}^{(k)}\}$.

5.3.5 Implementation of **Factorize** and **Back-Solve** together with the avoiding method for singularity-handling

This vector-vector variant achieves the Factorize and Back-Solve steps together row by row with j iterating from $|\mathbb{B}^{(\Bbbk)}|$ to 1, and finishing prematurely if $j = \zeta_{\{\mathbb{B}^{(k)}\}}$ as this means all the remaining rows are linearly dependent. $\boldsymbol{S}$ and

$\boldsymbol{f}$ are defined as above. Then defining $\lambda = (|\mathbb{S}^{(k)}| + j)$, $\iota = (|\mathbb{S}^{(k)}| + 1 + \zeta_{\{\mathbb{B}^{(k)}\}})$, $\{\mathbb{T}^*\} = \{(1 + \zeta_{\{\mathbb{B}^{(k)}\}}) : (j-1)\}$, $\{\mathbb{B}^*\} = \{j\}$, $\{\mathbb{L}^*\} = \{1 : (\lambda - 1)\}$, $\{\mathbb{R}^*\} = \{\lambda\}$, $\{\mathbb{D}^*\} = \{\iota : (\lambda - 1)\}$, while also defining a and $\boldsymbol{A}$ as meaningless temporaries.

If $(s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} < 0.0001)$ Then

$\zeta_{\{\mathbb{B}^{(k)}\}} := \zeta_{\{\mathbb{B}^{(k)}\}} + 1$

$s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} := 100000000$

$f := 0$

$\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}} := 0$

Else

$s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} := (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{1/2}$

$f_{\{\mathbb{B}^*\}} := (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{-1} f_{\{\mathbb{B}^*\}}$

$\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}} := (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{-1} \boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}}$

$\boldsymbol{f}_{\{\mathbb{T}^*\}} := \boldsymbol{f}_{\{\mathbb{T}^*\}} -$
$(\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{D}^*\}})^\top f_{\{\mathbb{B}^*\}}$

$\boldsymbol{S}_{:\{\mathbb{T}^*\}\{\mathbb{L}^*\}:} := \boldsymbol{S}_{:\{\mathbb{T}^*\}\{\mathbb{L}^*\}:} -$
$(\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{D}^*\}})^\top \boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}}$

End If

Finally $\{\mathbb{B}^{(\Bbbk)+}\} := \{\mathbb{B}^{(k)}\}_{1+\zeta_{\{\mathbb{B}^{(k)}\}}:|\mathbb{B}^{(\Bbbk)}|}$

If $(s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} < 0.0001)$ Then

$$\zeta_{\{\mathbb{B}^{(k)}\}} := \zeta_{\{\mathbb{B}^{(k)}\}} + 1 \tag{5.31}$$

$$s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} := 100000000 \tag{5.32}$$

$$f := 0 \tag{5.33}$$

$$\boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}} := 0 \tag{5.34}$$

Else

$$s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}} := \mathtt{SQRT}(s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}}) \tag{5.35}$$

$$\mathtt{DSCAL}(1, (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{-1}, f_{\{\mathbb{B}^*\}}, 1) \tag{5.36}$$

$$\mathtt{DSCAL}(\lambda - 1, (s_{\{\mathbb{B}^*\}\{\mathbb{R}^*\}})^{-1}, \boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}}) \tag{5.37}$$

$$\mathtt{DGER}(j - 1 - \zeta_{\{\mathbb{B}^{(k)}\}}, 1, -1.0, (\boldsymbol{s}_{\{\mathbb{B}^*\}\{\iota\}})^\top, 1, f_{\{\mathbb{B}^*\}}, 1, 1, \boldsymbol{f}_{\{\mathbb{T}^*\}}, |\mathbb{B}^{(\Bbbk)}|) \tag{5.38}$$

$$\mathtt{DGER}(j - 1 - \zeta_{\{\mathbb{B}^{(k)}\}}, \lambda - 1, -1.0, (\boldsymbol{s}_{\{\mathbb{B}^*\}\{\iota\}})^\top, 1, \boldsymbol{s}_{\{\mathbb{B}^*\}\{\mathbb{L}^*\}}, 1, \boldsymbol{S}_{\{1+\zeta_{\{\mathbb{B}^{(k)}\}}\}\{1\}}, |\mathbb{B}^{(\Bbbk)}|) \tag{5.39}$$

End If

5.3.6 Implementation of `SchurComplement`

The following establishes a right-looking scheme and as a result rather than updating supernodes $\boldsymbol{C}_{\{\mathbb{T}^{(k)}\}\{\mathbb{T}^{(k)}\}}$ all at once the updating process is split across the various supernodes. Then denoting $\{\mathbb{T}^*\} = (\{\mathbb{c}_{\boldsymbol{k}}\} \cap \{\mathbb{r}_{\boldsymbol{i}}\})$, $\{\mathbb{L}^*\} = (\{\mathbb{c}_{\boldsymbol{k}}\} \cap \{\mathbb{c}_{\boldsymbol{i}}\})$ with i iterating in any order such that $\{\mathbb{T}^*\} \neq \emptyset$ and letting $\boldsymbol{R}^+ \equiv \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{T}^*\}}$ and $\boldsymbol{Q}^+ \equiv \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{L}^*\}}$, and a single copy $\boldsymbol{A}^+ := (\boldsymbol{R}^+)^\top$ made to avoid having to transpose $\boldsymbol{R}^+$ multiple times.

$\boldsymbol{C}_{:\{\mathbb{T}^*\}\{\mathbb{T}^*\}:} =: \boldsymbol{C}_{:\{\mathbb{T}^*\}\{\mathbb{T}^*\}:} -$
$(\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{T}^*\}})^\top \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{T}^*\}}$

$\boldsymbol{C}_{:\{\mathbb{T}^*\}\{\mathbb{L}^*\}:} =: \boldsymbol{C}_{:\{\mathbb{T}^*\}\{\mathbb{L}^*\}:} -$
$(\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{T}^*\}})^\top * \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{L}^*\}}$

$$\mathtt{DSYRK}(\text{“}L\text{”}, \text{“}T\text{”}, |\mathbb{T}^*|, |\mathbb{B}^{(\Bbbk)+}|, -1.0, \boldsymbol{R}^+_{\{1\}\{1\}}, |\mathbb{B}^{(\Bbbk)}|, 1.0, \boldsymbol{C}_{:\{\mathbb{T}^*\}\{\mathbb{T}^*\}:}, |\mathbb{T}^*|,) \tag{5.40}$$

$$\mathtt{DGEMM}(\text{“}N\text{”}, \text{“}N\text{”}, |\mathbb{T}^*|, |\mathbb{L}^*|, |\mathbb{B}^{(\Bbbk)+}|, -1.0, \boldsymbol{A}^+_{\{1\}\{1\}}, (|\mathbb{B}^{(\Bbbk)}| + |\mathbb{S}^{(k)}|), \boldsymbol{Q}^+_{\{1:|\mathbb{B}^{(\Bbbk)+}|\}\{1:|\mathbb{L}^*|\}}, |\mathbb{B}^{(\Bbbk)+}|, 1.0, \boldsymbol{C}_{:\{\mathbb{T}^*\}\{\mathbb{L}^*\}:}, |\mathbb{T}^*|). \tag{5.41}$$

Additionally the overhead in performing the gathering and scattering is reused where possible in the performing of `Back-UpdateSCP`. In the case of updates to r the overhead of gathering and scattering into copies can be avoided since it is entirely dense.

5.3.7 Implementation of `Back-UpdateSCP`

Similarly updating the right hand side is split across the various supernodes, letting $\{\mathbb{T}^*\} = (\{\mathbb{c}_{\boldsymbol{k}}\} \cap \{\mathbb{r}_{\boldsymbol{i}}\})$,with i iterating in any order such that $\{\mathbb{T}^*\} \neq \emptyset$. The matrix $\boldsymbol{A}^+$ and vector $\boldsymbol{f}^+$ are defined as above.

$$\begin{array}{l|l} \boldsymbol{b}_{\{\mathbb{T}^*\}} :=: \boldsymbol{b}_{\{\mathbb{T}^*\}} - & \texttt{DGEMM}(\text{“}N\text{”}, \text{“}N\text{”}, |\mathbb{T}^*|, 1, |\mathbb{B}^{(\Bbbk)+}|, -1.0, \boldsymbol{A}^+_{\{1\}\{1\}}, \\ (\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{L}^*\}})^\top \boldsymbol{b}_{\{\mathbb{B}^{(\Bbbk)+}\}} & |\mathbb{B}^{(\Bbbk)}| + |\mathbb{S}^{(k)}|, \boldsymbol{f}^+_{\{1:|\mathbb{B}^{(\Bbbk)+}|\}}, |\mathbb{B}^{(\Bbbk)+}|, 1.0, \boldsymbol{b}_{\{\mathbb{T}^*\}}, |\mathbb{T}^*|). \end{array} \tag{5.42}$$

5.3.8 Implementation of `SymmetricMultiply`

This demonstration of `SymmetricMultiply` uses $\boldsymbol{A} = (\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}})^{-1}\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{S}^{(k)}\}}$ assuming that it has already been calculated in a previous `Forward-Solve` step. Then letting $\boldsymbol{T}^+ \equiv \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}$, which needs to be gathered as it is not stored in contiguous memory and also $\boldsymbol{R}^+$ is as before.

$$\begin{array}{l|l} \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{S}^{(k)}\}} := -\boldsymbol{A}\boldsymbol{C}_{:\{\mathbb{S}^{(k)}\}\{\mathbb{S}^{(k)}\}:} & \texttt{DSYMM}(\text{“}R\text{”}, \text{“}L\text{”}, |\mathbb{B}^{(\Bbbk)+}|, |\mathbb{S}^{(k)}|, -1.0, \boldsymbol{T}_{\{1\}\{1\}}, |\mathbb{S}^{(k)}|, \\ & \boldsymbol{A}, |\mathbb{B}^{(\Bbbk)+}|, 0, \boldsymbol{R}^+_{\{1\}\{1\}}, |\mathbb{B}^{(\Bbbk)}|). \end{array} \tag{5.43}$$

5.3.9 Implementation of `InvertCrossMultiply`

There are multiple implementations of this step, the `DTRTRI` and `DLAUUM` method which exploits symmetry and Intel MKL BLAS, a method based on `twosidedtrsm` from the Elemental (Poulson *et al.*, 2013) library which exploits symmetry in a more general way and the standard two calls of `TRSM` (the way CHOLMOD-Extra does it) which does not exploit symmetry.

5.3.9.1 Implementation using `DTRTRI` and `DLAUUM`

First $\boldsymbol{C}_{\{\mathbb{B}^{(k)}\}\{\mathbb{B}^{(k)}\}}$ is inverted using `DTRTRI` then the final result is calculated using `DLAUUM`, however because `DLAUUM` computes $\boldsymbol{L}^\top\boldsymbol{L}$ rather than $\boldsymbol{L}\boldsymbol{L}^\top$ then reverse permutations must be used. With $\boldsymbol{T}^+ \equiv \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}$ and $\boldsymbol{F}^+ = \boldsymbol{P}_{\boldsymbol{Rev}}\boldsymbol{T}^+\boldsymbol{P}_{\boldsymbol{Rev}}$. Then denoting $\{\mathbb{A}^{(k)}\} = \{|\mathbb{B}^{(\Bbbk)+}|-i+1\}$, $\{\mathbb{C}^{(k)}\} = \{|\mathbb{B}^{(\Bbbk)+}|-j+1\}$,

$\{\mathbb{D}^{(k)}\} = \{j\}$ and $\{\mathbb{E}^{(k)}\} = \{i\}$ with j iterating from $|\mathbb{B}^{(\Bbbk)+}|$ to 1 and i iterating from 1 to j:

$$\boldsymbol{F}^{+}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} := \boldsymbol{T}^{+}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}} \quad \Big| \quad \boldsymbol{F}^{+}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} = \boldsymbol{T}^{+}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}}. \tag{5.44}$$

Then $\boldsymbol{F}^{+}$ is inverted and cross multiplied in place:

$$\begin{aligned} \boldsymbol{F}^{+} &:= (\boldsymbol{F}^{+})^{-1} \quad \Big| \quad \texttt{DTRTRI}(\text{“}L\text{”}, \text{“}N\text{”}, |\mathbb{B}^{(\Bbbk)+}|, \boldsymbol{F}^{+}, |\mathbb{B}^{(\Bbbk)+}|, \texttt{INFO}), && (5.45) \\ \boldsymbol{F}^{+} &:= \boldsymbol{F}^{+}(\boldsymbol{F}^{+})^{\top} \quad \Big| \quad \texttt{DLAUUM}(\text{“}L\text{”}, |\mathbb{B}^{(\Bbbk)+}|, \boldsymbol{F}^{+}, |\mathbb{B}^{(\Bbbk)+}|, \texttt{INFO}). && (5.46) \end{aligned}$$

where `INFO` holds an error value that provides information in case something went wrong. Finally $\boldsymbol{F}^{+}$ needs to be reversely permuted back into $\boldsymbol{T}^{+}$:

$$\boldsymbol{T}^{+}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} := \boldsymbol{F}^{+}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}} \quad \Big| \quad \boldsymbol{T}^{+}_{\{\mathbb{A}^{(k)}\}\{\mathbb{C}^{(k)}\}} = \boldsymbol{F}^{+}_{\{\mathbb{D}^{(k)}\}\{\mathbb{E}^{(k)}\}}. \tag{5.47}$$

5.3.9.2 Implementation based on `twosidedtrsm`

This implementation operates differently, and it requires a temporary $\boldsymbol{S}^{+} := \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} = \boldsymbol{L}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}$, set prior to the beginning of the SIS process for the k^{th} supernode, because `twosidedtrsm` cannot multiply a matrix by its own inverse, instead it cross-multiplies a matrix by the inverse of another matrix. Then $\boldsymbol{T}^{+} \equiv \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}$ is pre and post-multiplied by the inverse of $\boldsymbol{S}^{+}$ via the solution of triangular systems. An interface was written by the author called `forttwosidedtrsm` that was used since Elemental is not written in FORTRAN (the implementation of this is in Appendix C).

$$\begin{aligned} \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} := (\boldsymbol{S}^{+})^{-1}\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}((\boldsymbol{S}^{+})^{-1})^{\top} \quad \Big| \quad & \texttt{forttwosidedtrsm}(|\mathbb{B}^{(\Bbbk)+}|, |\mathbb{B}^{(\Bbbk)+}|, |\mathbb{B}^{(\Bbbk)}|, \\ & \boldsymbol{S}^{+}_{:,\{1:|\mathbb{B}^{(\Bbbk)+|}\}}, |\mathbb{B}^{(\Bbbk)}|, \boldsymbol{T}^{+}_{:,\{1:|\mathbb{B}^{(\Bbbk)+|}\}}, \zeta_{\{\mathbb{B}^{(k)}\}}). \end{aligned} \tag{5.48}$$

5.3.9.3 Implementation using two `TRSM` calls

This implementation is very similar to the implementation using `twosidedtrsm` except it does this using two `TRSM` calls, which fails to exploit symmetry.

$\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} := (\boldsymbol{S}^{+})^{-1}\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}$	DTRSM("L", "L", "N", "N", $\lvert\mathbb{B}^{(\Bbbk)+}\rvert, \lvert\mathbb{B}^{(\Bbbk)+}\rvert, 1.0,$ $\boldsymbol{S}^{+}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert, \boldsymbol{T}^{+}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert$). (5.49)
$\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} := \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}((\boldsymbol{S}^{+})^{-1})^{\top}$	DTRSM("R", "L", "T", "N", $\lvert\mathbb{B}^{(\Bbbk)+}\rvert, \lvert\mathbb{B}^{(\Bbbk)+}\rvert, 1.0,$ $\boldsymbol{S}^{+}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert, \boldsymbol{T}^{+}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert$). (5.50)

In theory the two TRSM calls could be executed in reverse order without consequence, but for reasons unknown to the author calling them in this order was faster by a non-trivial amount in preliminary tests.

5.3.10 Implementation of Forward-UpdateInv

5.3.10.1 Implementation to exploit symmetry DGEMMT

This version is very similar to Back-UpdateSCP, except that since $\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}$ is symmetric, only the lower triangle of it needs to be updated. There is an Intel MKL subroutine DGEMMT that exploits this symmetry in the result matrix however, so this is used instead. Afterwards a subroutine called COMPLETESQUARE written by the author is used to copy the lower triangle of the diagonal part into the top triangle, which may be necessary for the invert and cross multiply step. This is significantly cheaper than computing both triangles, to the point of being completely dominated by the calculation of the lower triangle. Symmetry is exploited using DGEMMT and COMPLETESQUARE. Using $\boldsymbol{R}^{+}$ and $\boldsymbol{T}^{+}$ as before, with $\boldsymbol{A} = (\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}})^{-1}\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{S}^{(k)}\}}$:

$\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} := \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} -$ $\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{S}^{(k)}\}}\boldsymbol{A}^{\top}$	DGEMMT("L", "N", "T", $\lvert\mathbb{B}^{(\Bbbk)+}\rvert, \lvert\mathbb{S}^{(k)}\rvert, -1.0,$ $\boldsymbol{R}^{+}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert, \boldsymbol{A}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert, 1.0, \boldsymbol{T}^{+}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert$), (5.51) COMPLETESQUARE($\boldsymbol{T}^{+}_{\{1:\lvert\mathbb{B}^{(\Bbbk)+}\rvert\}\{1:\lvert\mathbb{B}^{(\Bbbk)+}\rvert\}}$). (5.52)

5.3.10.2 Implementation using DGEMM

This is the same as above, except that symmetry of $\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}}$ is not exploited:

$\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} := \boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{B}^{(\Bbbk)+}\}} -$ $\boldsymbol{C}_{\{\mathbb{B}^{(\Bbbk)+}\}\{\mathbb{S}^{(k)}\}}\boldsymbol{A}^{\top}$	DGEMM("N", "T", $\lvert\mathbb{B}^{(\Bbbk)+}\rvert, \lvert\mathbb{B}^{(\Bbbk)+}\rvert, \lvert\mathbb{S}^{(k)}\rvert, -1.0,$ $\boldsymbol{R}^{+}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert, \boldsymbol{A}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert, 1.0, \boldsymbol{T}^{+}_{\{1\}\{1\}}, \lvert\mathbb{B}^{(\Bbbk)}\rvert$). (5.53)

5.3.11 Leading dimension

Many of the BLAS and LAPACK routines have arguments associated with the dimensions of the matrices, as well as leading dimensions. When the matrix

arguments are entire matrices, the latter are effectively redundant, however when the matrices to be operated on are partitions of larger dense matrices then these can be used to avoid unnecessary copying. Matrices are stored as vectors in a computer, with additional information regarding the number of rows and columns in order to treat this vector as a matrix. FORTRAN stores matrices by columns, which means that all of the elements in the same column are stored contiguously. As a result of this, partitioning a matrix by column is very straightforward for the processor, the relevant columns can simply be taken and put together in the partition since it knows how many elements there are per column. In particular, taking the first however many columns of a matrix requires no copying, one simply takes the elements associated with these columns from the vector of memory. However, partitioning by row is not as straight forward, if one wanted to take a number of rows of a matrix these would have to be found and then copied into a new contiguous memory vector, regardless where these rows are in the original matrix. The leading dimension argument is a way that the user can supply a matrix with say b rows but only take a partition of the first $a \leq b$ rows. In this case the processor takes the first a elements in the first column, then skips the next $b - a$ elements and moves onto the next column and so on.

5.4 Existing open source alternatives

5.4.1 CHOLMOD and CHOLMOD-Extra

It is easiest to call the functions from C (ISO, 2011) as CHOLMOD (Chen *et al.*, 2008) and CHOLMOD-Extra (Luttinen, 2018) are written in C.

5.4.1.1 Symbolic factorization and reverse permutation

CHOLMOD computes a $\boldsymbol{L}\boldsymbol{L}^\top$ rather than a $\boldsymbol{L}^\top\boldsymbol{L}$ so reverse permutations need to be applied, so letting $\texttt{CHOLMODPERM} = n-1, n-2, \ldots 0$ (as CHOLMOD is zero-indexed):

$$\boldsymbol{L} := \texttt{cholmod\_l\_analyze\_p2}(\texttt{TRUE}, \boldsymbol{C}, \texttt{CHOLMODPERM}, \texttt{NULL}, 0, \&\texttt{CHOLMODCOMMON}); \tag{5.54}$$

Where `TRUE` refers to a CF as opposed to an LDLT factorization, `NULL` is an option to not subset the matrix and `CHOLMODCOMMON` (passed by reference hence the &) stores auxiliary information at different steps in the process. This creates the symbolic factorization and stores it in $\boldsymbol{L}$.

5.4.1.2 Cholesky factorization

$$\texttt{cholmod\_l\_factorize}(\boldsymbol{C}, \boldsymbol{L}, \&\texttt{CHOLMODCOMMON}); \tag{5.55}$$

CHOLMOD uses a forward left-looking algorithm for the factorization.

5.4.1.3 Forward-solving and back-solving

CHOLMOD does not do any solving in factorization, so both of these steps must be applied, but this is all wrapped up in the following function. Curiously $\boldsymbol{L}$ does not overwrite $\boldsymbol{C}$ but $\boldsymbol{C}$ can simply be deleted at this point.

$$\tilde{\boldsymbol{\beta}} := \texttt{cholmod\_l\_solve}(\texttt{CHOLMOD\_A}, \boldsymbol{L}, \boldsymbol{b}, \&\texttt{CHOLMODCOMMON}); \tag{5.56}$$

`CHOLMOD_A` simply tells CHOLMOD to use $\boldsymbol{C}$ as the coefficient matrix rather than $\boldsymbol{C}\boldsymbol{C}^\top$ which is another option in the case of rectangular $\boldsymbol{C}$. Similarly $\tilde{\boldsymbol{\beta}}$ does not overwrite $\boldsymbol{b}$ but $\boldsymbol{b}$ can be deleted at this point.

5.4.1.4 Calculation of the sparse inverse subset

$$\boldsymbol{A} := \texttt{cholmod\_l\_spinv}(\boldsymbol{L}, \&\texttt{CHOLMODCOMMON}); \tag{5.57}$$

where in this case $\boldsymbol{A}$ is used to store the SIS of $\boldsymbol{C}$. The software CHOLMOD-Extra is used to compute the SIS as CHOLMOD in its base form does not have this functionality. CHOLMOD-Extra uses a back left-looking algorithm to compute the SIS.

5.4.1.5 Simplicial mode

Within CHOLMOD (and by extension CHOLMOD-Extra) there is the option to run in either supernodal mode, or simplicial mode (which is single-nodal mode). This is denoted as `CHOLMOD-sn` and `CHOLMOD-Extra-sn` respectively.

5.4.1.6 Key BLAS libraries

The factorization occurs using LAPACK variant (Subsubsection 5.3.1.1), `Back-Solve` (Subsubsection 5.3.3) and `SchurComplement` (Subsubsection 5.3.6), while the calculation of the SIS occurs using `Forward-Solve` (Subsubsection 5.3.3), `SymmetricMultiply` (Subsubsection 5.3.8), two `TRSM` calls (Subsubsection 5.3.9.3) and `DGEMM` (Subsubsection 5.3.10.2).

5.4.2 CHOMPACK

It is easiest to call the various functions from Python (Van Rossum & Others, 2007) as CHOMPACK (Vandenberghe & Andersen, 2015) is written in Python.

5.4.2.1 Symbolic factorization and reverse permutation

CHOMPACK computes a $\boldsymbol{L}\boldsymbol{L}^\top$ rather than a $\boldsymbol{L}^\top\boldsymbol{L}$ so reverse permutations need to be applied, so letting $\texttt{CHOLMODPERM} = n-1, n-2, \ldots 0$ (as CHOMPACK is zero-indexed):

$$\texttt{symb} := \texttt{symbolic}(\boldsymbol{C}, \texttt{p = CHOMPACKPERM}, \texttt{merge\_function = fmerge}) \tag{5.58}$$

Where `merge_function = fmerge` is an option to add duplicate elements if they appear, which doesn't matter since they are not present. This creates the symbolic factorization and stores it in `symb`.

5.4.2.2 Cholesky factorization

$$\texttt{cholesky}(\boldsymbol{C}) \tag{5.59}$$

CHOMPACK uses a forward multifrontal algorithm for the factorization.

5.4.2.3 Forward-solving and back-solving

CHOMPACK does not do any solving in factorization, so both of these steps must be applied, but this is all wrapped up in the following function.

$$\texttt{trsm}(\boldsymbol{C}, \boldsymbol{b}) \tag{5.60}$$

This `trsm` is not to be confused with the level 3 BLAS `DTRSM`.

5.4.2.4 Calculation of the sparse inverse subset

$$\texttt{projected\_inverse}(\boldsymbol{C}) \tag{5.61}$$

CHOMPACK uses a back multifrontal algorithm for the calculation of the SIS.

5.4.2.5 Key BLAS Libraries

CHOMPACK uses an LDLT algorithm for the factorization, which means its factorization has a forward solve and a backward solve, but its SIS calculation doesn't have a backward solve component. Its factorization uses the LAPACK variant (Subsubsection 5.3.1.1), `Back-Solve` (Subsubsection 5.3.3), `Forward-Solve` (Subsubsection 5.3.3) and a scheme similar to `SchurComplement` (Subsubsection 5.3.6) except that it does update $\boldsymbol{C}_{\{\mathbb{T}^{(k)}\}\{\mathbb{T}^{(k)}\}}$ all at once (as mentioned) using the level 3 BLAS `DSYRK` as it is multifrontal. For the SIS it uses, `SymmetricMultiply` (Subsubsection 5.3.8), `DTRTRI` and `DSYRK` (taking the place of `DLAUUM` in Subsubsection 5.3.9.1) and `DGEMM` (Subsubsection 5.3.10.2).

5.4.3 MUMPS

It is easiest to call the various functions from FORTRAN (Reid, 2008) as MUMPS (Amestoy *et al.*, 2000) is written in mainly FORTRAN.

5.4.3.1 Symbolic factorization and reverse permutation

MUMPS computes a $\boldsymbol{L}\boldsymbol{L}^\top$ rather than a $\boldsymbol{L}^\top\boldsymbol{L}$ so reverse permutations need to be applied, so letting $\texttt{MUMPSPERM} = n, n-1, \ldots 1$ (as MUMPS is one-indexed):

$$\texttt{mumps\_par\%JOB} = 1 \tag{5.62}$$
$$\texttt{DMUMPS(mumps\_par)} \tag{5.63}$$

Where `mumps_par` is a common object that stores everything related to the matrix in question, i.e. $\boldsymbol{C}$ and $\boldsymbol{b}$ are parts of it and `DMUMPS` is the subroutine that is called for each step, with `mumps_par%JOB` being used to select which operation is to be performed, in this case it is the symbolic factorization due to the value of 1. A value of 2 refers to CF and a value of 3 refers to one of the types of solves.

5.4.3.2 Cholesky factorization

$$\texttt{mumps\_par\%JOB} = 2 \tag{5.64}$$
$$\texttt{DMUMPS(mumps\_par)} \tag{5.65}$$

MUMPS uses a forward multifrontal algorithm for the factorization.

5.4.3.3 Forward-solving and back-solving

MUMPS has the capability to do forward solving in factorization but this feature is used for a specific application, and is more likely to be less efficient in the general case, so for that reason it is turned off. Therefore both of these steps must be applied, but this is all wrapped up in the following function.

$$\texttt{mumps\_par\%JOB} = 3 \tag{5.66}$$
$$\texttt{DMUMPS(mumps\_par)} \tag{5.67}$$

5.4.3.4 Calculation of the sparse inverse subset

$$\texttt{mumps\_par\%ICNTL(30)} = 1 \tag{5.68}$$
$$\texttt{mumps\_par\%JOB} = 3 \tag{5.69}$$
$$\texttt{DMUMPS(mumps\_par)} \tag{5.70}$$

MUMPS does not have a true SIS interface, instead it has a selected inversion interface where if it is provided with various indices of elements it will find those elements of $\boldsymbol{C}^{-1}$. The relevant elements are placed into `mumps_par` and the option `mumps_par%ICNTL(30)` must be set to 1 to enable this option. MUMPS finds these elements via its forward-solve and back-solve routines, and so `mumps_par%JOB` $= 3$. MUMPS does this by solving for a number of columns of the identity matrix, and exploiting sparsity such that it only performs the necessary calculations to find the selected elements of the inverse. This number of columns is controlled by a blocking parameter `mumps_par%ICNTL(27)` and for the remainder of this thesis the value of 4096 is chosen. This value was chosen using some initial testing, and as the MUMPS user guide offers little guidance in this matter this was what the author decided was sensible.

5.4.3.5 Key BLAS Libraries

MUMPS uses the LAPACK variant (Subsubsection 5.3.1.1), `Back-Solve` (Subsubsection 5.3.3), `Forward-Solve` (Subsubsection 5.3.3) and a scheme similar to `SchurComplement` (Subsubsection 5.3.6) except that it does update $\boldsymbol{C}_{\{\mathbb{T}^{(k)}\}\{\mathbb{T}^{(k)}\}}$ all at once (as mentioned) using the level 3 BLAS `DSYRK` as it is multifrontal. For the calculation of the SIS it uses only `Back-Solve` (Subsubsection 5.3.3), `Forward-Solve` (Subsubsection 5.3.3) and `DGEMM` (Subsubsection 5.3.10.2).

5.5 Conclusions

This chapter introduced the Basic Linear Algebra Subprograms, as well as LAPACK and BLAS-Like routines, which are highly optimized dense linear algebra libraries that can be used to perform various standard matrix operations very efficiently in Section 5.2. The way that these routines could be used to perform the various functions required for the AI algorithm was demonstrated in Section 5.3. Existing open source codes that could perform the most computationally intensive steps of the AI algorithm were also introduced, in Section 5.4.

Bibliography

Amestoy, Patrick R, Duff, Iain S, L'Excellent, Jean-Yves & Koster, Jacko (2000). MUMPS: a general purpose distributed memory sparse solver. In *International Workshop on Applied Parallel Computing*. Springer, Boston, MA, 121-130.

Ashcraft, Cleve & Grimes, Roger (1989). The influence of relaxed supernode partitions on the multifrontal method. *ACM Transactions on Mathematical Software (TOMS)* **15**(4), 291–309.

Bunch, James R (1971). Analysis of the diagonal pivoting method. *SIAM Journal on Numerical Analysis* **8**(4), 656–680.

Chen, Yanqing, Davis, Timothy A, Hager, William W & Rajamanickam, Sivasankaran (2008). Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)* **35**(3), 22.

Dayde, Michel J & Camichel, Rue (1995). A Block Version of the Eskow-Schnabel Modified Cholesky Factorization.
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.9977{&}rep=rep1{&}type=pdf

Eskow, Elizabeth & Schnabel, Robert B (1991). Algorithm 695: Software for a new modified Cholesky factorization. *ACM Transactions on Mathematical Software (TOMS)* **17**(3), 306–312.

Gilmour, A R (1998). ASREML, Technical Details. Technical report, NSW DPI.

Gilmour, A R, Gogel, B J, Cullis, B R, Welham, S J & Thompson, R (2015). ASReml User Guide Release 4.1 Functional Specification.
URL www.vsni.co.uk

Gilmour, Arthur R, Thompson, Robin & Cullis, Brian R (1995). Average information REML: An efficient algorithm for variance parameter estimation in linear mixed models. *Biometrics* **51**(4), 1440–1450.

Gupta, Anshul (2000). WSMP: Watson sparse matrix package (Part-I: direct solution of symmetric sparse systems). *IBM TJ Watson Research Center, Yorktown Heights, NY, Tech. Rep. RC* **21886**.

Gupta, Anshul, Joshi, Mahesh & Kumar, Vipin (1997). *WSSMP: Watson symmetric sparse matrix package*. IBM TJ Watson Research Center.

Henderson, C R (1950). Estimation of genetic parameters (abstract). *Annals of Mathematical Statistics* **21**, 309–310.

Henderson, C R (1973). Sire Evaluation and Genetic Trends. *Journal of Animal Science* **1973**(Symposium), 10–41.
URL https://doi.org/10.1093/ansci/1973.Symposium.10

Intel (2019). Developer Reference for Intel Math Kernel Library - Fortran.
URL https://software.intel.com/en-us/mkl-developer-reference-fortran

ISO (2011). IEC 9899: 2011 Information technology—Programming languages—C. *International Organization for Standardization, Geneva, Switzerland* **27**, 59.

Liu, Joseph W H (1990). The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications* **11**(1), 134–172.

Luttinen, Jaakko (2018). cholmod-extra Documentation.
URL https://buildmedia.readthedocs.org/media/pdf/cholmod-extra/stable/cholmod-extra.pdf

Mazur, Luke (2022). *Computational Methods for the Fitting of Factor Analytic Linear Mixed Models with Applications to Plant Variety Trials*. Ph.D. thesis, University of Wollongong.

Meyer, Karin (1989). Restricted maximum likelihood to estimate variance components for animal models with several random effects using a derivative-free algorithm. *Genetics Selection Evolution* **21**(3), 317.

Patterson, H D & Thompson, R (1971). Recovery of interblock information when block sizes are unequal. *Biometrika* **31**, 545–554.

Poulson, Jack, Geijn, Robert A V A N D E & Bennighof, Jeffrey (2012). (Parallel) Algorithms for Two-sided Triangular Solves and Matrix Multiplication.
URL https://www.cs.utexas.edu/users/flame/pubs/ElementalGenEig.pdf

Poulson, Jack, Marker, Bryan, de Geijn, Robert A, Hammond, Jeff R & Romero, Nichols A (2013). Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software (TOMS)* **39**(2), 1–24.

Reid, John (2008). The new features of Fortran 2008. In *ACM SIGPLAN Fortran Forum*. ACM New York, NY, USA, **27**(2), 8-21.

Schnabel, Robert B & Eskow, Elizabeth (1990). A new modified Cholesky factorization. *SIAM Journal on Scientific and Statistical Computing* **11**(6), 1136–1158.

Searle, Shayle Roy & Khuri, Andre I (2017). *Matrix Algebra Useful for Statistics*. John Wiley & Sons.

Smith, Alison B, Cullis, Brian R & Thompson, Robin (2001). Analyzing variety by environment data using multiplicative mixed models and adjustments for spatial field trend. *Biometrics* **57**(4), 1138–1147.

Van Rossum, Guido & Others (2007). Python Programming Language. In *USENIX annual technical conference*. **41**, 36.

Vandenberghe, Lieven & Andersen, Martin S (2015). Chordal Graphs and Semidefinite Optimization. *Foundations and Trends® in Optimization* **1**(4), 241–433.
URL http://www.nowpublishers.com/article/Details/OPT-006

A. Permutation matrices

A brief notation introduction to the concept of a permutation matrix $\boldsymbol{P}^*$ (not to be confused with $\boldsymbol{P}$ which is not a permutation matrix). A permutation matrix is simply the identity matrix with some of the rows or columns rearranged. In other words, it is a square matrix with exactly one 1 in each row and in each column with the rest of the elements being 0. Pre-multiplying by a permutation matrix permutes the rows of a matrix for example $\boldsymbol{P}^*\boldsymbol{A}$ will have the rows in a permuted order as compared to $\boldsymbol{A}$ while $\boldsymbol{A}\boldsymbol{P}^*$ will have the columns rearranged (except in the special case where $\boldsymbol{P}^* = \boldsymbol{I}$).An important property is that $\boldsymbol{P}^*(\boldsymbol{P}^*)^\top = \boldsymbol{P}^*\boldsymbol{I}(\boldsymbol{P}^*)^\top = \boldsymbol{I}$. For the remainder of this report the superscript is dropped from $\boldsymbol{P}^*$ where it is clear from context that $\boldsymbol{P}$ is a permutation matrix.

A special case of a permutation matrix is one that is anti-diagonal, i.e. it has 1s along the diagonal from the bottom left to the top right and 0s elsewhere. Let this be denoted as $\boldsymbol{P}_{\boldsymbol{Rev}}$. A symmetric permutation using $\boldsymbol{P}_{\boldsymbol{Rev}}$ produces a matrix that is reversely permuted, i.e. the first diagonal element becomes the last, second diagonal element becomes second last etc. Also $\boldsymbol{P}_{\boldsymbol{Rev}} = \boldsymbol{P}_{\boldsymbol{Rev}}^{-1} = \boldsymbol{P}_{\boldsymbol{Rev}}^\top$. These are relevant because ASReml (See Gilmour *et al.* (2015) for a user guide and Gilmour (1998) for technical details) implements absorption with k iterating from n to 2 while conventional sparse and dense packages, including the BLAS, expect the factorizations to occur with k iterating from 1 to n (or to m in a supernodal setting).

B. Calculation of FLOPs for BLAS routines

In the following a supernode k is considered such that it has rows represented by the set $\{\mathbb{B}^{(k)}\}$ meaning that it has $|\mathbb{B}^{(k)}|$ rows. It has columns in the off-diagonal part represented by the set $\{\mathbb{S}^{(k)}\}$ and thus has $|\mathbb{S}^{(k)}|$ such columns. The FLOPs performed by various subroutines are presented, in alphabetical order.

LAUUM–3 is used to multiply two matrices, the first upper triangular of order $|\mathbb{B}^{(k)}|$ and the second the transpose of the former, forming the symmetric result matrix row by row. The number of multiplications and additions for the j^{th} element in the i^{th} row of the resultant matrix are summed:

$$
\begin{aligned}
&\sum_{i=1}^{|\mathbb{B}^{(k)}|}\sum_{j=1}^{i}(\min(|\mathbb{B}^{(k)}|-i+1,|\mathbb{B}^{(k)}|-j+1)+(\min(|\mathbb{B}^{(k)}|-i+1,|\mathbb{B}^{(k)}|-j+1)-1))\\
&=\sum_{i=1}^{|\mathbb{B}^{(k)}|}\sum_{j=1}^{i}((|\mathbb{B}^{(k)}|-i+1)+(|\mathbb{B}^{(k)}|-i+1-1))\\
&=\sum_{i=1}^{|\mathbb{B}^{(k)}|}\sum_{j=1}^{i}(2|\mathbb{B}^{(k)}|-2i+1)\\
&=\sum_{i=1}^{|\mathbb{B}^{(k)}|}(2|\mathbb{B}^{(k)}|i-2i^2+i)\\
&=2|\mathbb{B}^{(k)}|(|\mathbb{B}^{(k)}|(|\mathbb{B}^{(k)}|+1)/2)-2((2|\mathbb{B}^{(k)}|+1)|\mathbb{B}^{(k)}|(|\mathbb{B}^{(k)}|+1)/6)+|\mathbb{B}^{(k)}|(|\mathbb{B}^{(k)}|+1)/2\\
&=|\mathbb{B}^{(k)}|^2(|\mathbb{B}^{(k)}|+1)-(2|\mathbb{B}^{(k)}|+1)|\mathbb{B}^{(k)}|(|\mathbb{B}^{(k)}|+1)/3+|\mathbb{B}^{(k)}|(|\mathbb{B}^{(k)}|+1)/2\\
&\approx|\mathbb{B}^{(k)}|^3-2|\mathbb{B}^{(k)}|^3/3\\
&=|\mathbb{B}^{(k)}|^3/3.
\end{aligned}
$$

POTRF–3 Dense Cholesky factorization of the diagonal block of size $|\mathbb{B}^{(k)}|$. The number of multiplications and additions (Equation **??**), divisions (Equation **??**) and square roots (Equation **??**) for the i^{th} row respectively are

summed below:

$$\begin{aligned}
&\sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|} (i(i-1)/2 + i(i-1)/2 + (i-1) + 1) \\
&= (|\mathbb{B}^{(\Bbbk)}| - 1)|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}| + 1)/6 + (|\mathbb{B}^{(\Bbbk)}| - 1)|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}| + 1)/6 + (|\mathbb{B}^{(\Bbbk)}| - 1)|\mathbb{B}^{(\Bbbk)}|/2 + |\mathbb{B}^{(\Bbbk)}| \\
&= (|\mathbb{B}^{(\Bbbk)}| - 1)|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}| + 1)/3 + (|\mathbb{B}^{(\Bbbk)}| + 1)|\mathbb{B}^{(\Bbbk)}|/2 \\
&\approx |\mathbb{B}^{(\Bbbk)}|^3/3.
\end{aligned}$$

PSTRF–3 As above, except that in the singular case for the diagonal block the location of the singularities are not yet known, they are discovered in this phase. If one were to assume that they are discovered last (worst possible case) then the final ζ steps are not applied. The number of multiplications, additions, divisions and square roots in the i^{th} row not applied as a result of this are summed below:

$$\begin{aligned}
&\sum_{i=1}^{\zeta_{\{\mathbb{B}^{(k)}\}}} (i(i-1)/2 + i(i-1)/2 + i - 1 + 1) \\
&= (\zeta_{\{\mathbb{B}^{(k)}\}} - 1)\zeta_{\{\mathbb{B}^{(k)}\}}(\zeta_{\{\mathbb{B}^{(k)}\}} + 1)/6 + (\zeta_{\{\mathbb{B}^{(k)}\}} - 1)\zeta_{\{\mathbb{B}^{(k)}\}}(\zeta_{\{\mathbb{B}^{(k)}\}} + 1)/6 + (\zeta_{\{\mathbb{B}^{(k)}\}} - 1)\zeta_{\{\mathbb{B}^{(k)}\}}/2 + \zeta_{\{\mathbb{B}^{(k)}\}} \\
&= (\zeta_{\{\mathbb{B}^{(k)}\}} - 1)\zeta_{\{\mathbb{B}^{(k)}\}}(\zeta_{\{\mathbb{B}^{(k)}\}} + 1)/3 + \zeta_{\{\mathbb{B}^{(k)}\}}(\zeta_{\{\mathbb{B}^{(k)}\}} + 1)/2 \\
&\approx \zeta^3_{\{\mathbb{B}^{(k)}\}}/3.
\end{aligned}$$

So the final FLOPs for PSTRF in the singular case are $\approx (|\mathbb{B}^{(\Bbbk)}|^3 - \zeta^3_{\{\mathbb{B}^{(k)}\}})/3$.

TRSM–3 The number of multiplications and subtractions (Equation **??**) and divisions (Equation **??**) for the i^{th} row to solve for the corresponding elements of the right hand side are summed below:

$$\begin{aligned}
&\sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|} ((i-1) + (i-1) + 1) \\
&= (|\mathbb{B}^{(\Bbbk)}| - 1)|\mathbb{B}^{(\Bbbk)}|/2 + (|\mathbb{B}^{(\Bbbk)}| - 1)|\mathbb{B}^{(\Bbbk)}|/2 + |\mathbb{B}^{(\Bbbk)}| \\
&= (|\mathbb{B}^{(\Bbbk)}| - 1)|\mathbb{B}^{(\Bbbk)}| + |\mathbb{B}^{(\Bbbk)}| \\
&= |\mathbb{B}^{(\Bbbk)}|^2.
\end{aligned}$$

TRTRI–3 This can be considered as using the triangular part of the supernode (post CF) of order $|\mathbb{B}^{(\Bbbk)}|$ to solve for the identity matrix order $|\mathbb{B}^{(\Bbbk)}|$ taking advantage of the fact that the inverse of a lower triangular matrix is lower triangular. The number of multiplications, subtractions and divisions (Equation **??**) in the i^{th} row for the j^{th} right hand side are

summed below:

$$
\begin{aligned}
&\sum_{j=1}^{|\mathbb{B}^{(\Bbbk)}|}\sum_{i=j}^{|\mathbb{B}^{(\Bbbk)}|}((i-1)+(i-1)+1)\\
&=\sum_{j=1}^{|\mathbb{B}^{(\Bbbk)}|}\sum_{i=j}^{|\mathbb{B}^{(\Bbbk)}|}(2i-1)\\
&=\sum_{j=1}^{|\mathbb{B}^{(\Bbbk)}|}(\sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|}(2i-1)-\sum_{i=1}^{j-1}(2i-1))\\
&=\sum_{j=1}^{|\mathbb{B}^{(\Bbbk)}|}(2(|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}|+1)/2)-|\mathbb{B}^{(\Bbbk)}|-(2(j(j-1)/2)-j+1))\\
&=\sum_{j=1}^{|\mathbb{B}^{(\Bbbk)}|}(|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}|+1)-|\mathbb{B}^{(\Bbbk)}|-2j(j-1)+j-1)\\
&=\sum_{j=1}^{|\mathbb{B}^{(\Bbbk)}|}(|\mathbb{B}^{(\Bbbk)}|^2-2j^2+3j-1)\\
&=|\mathbb{B}^{(\Bbbk)}|^3-2((2|\mathbb{B}^{(\Bbbk)}|+1)|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}|+1)/6)+3(|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}|+1)/2)-|\mathbb{B}^{(\Bbbk)}|\\
&=|\mathbb{B}^{(\Bbbk)}|^3-(2|\mathbb{B}^{(\Bbbk)}|+1)|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}|+1)/3+3|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}|+1)/2-|\mathbb{B}^{(\Bbbk)}|\\
&\approx|\mathbb{B}^{(\Bbbk)}|^3-2|\mathbb{B}^{(\Bbbk)}|^3/3\\
&=|\mathbb{B}^{(\Bbbk)}|^3/3.
\end{aligned}
$$

TWOSIDED TRSM–3

Poulson *et al.* (2012) claim that their variant 4 is the best one, so the calculations of FLOPs herein pertain to that variant. There is a recursive blocked algorithm at work such that the subscripts with a $\{\mathbb{C}^{(k)}\}$ refer to the current block, which splits the matrix into the top and bottom parts. Therefore those with a $\{\mathbb{T}^*\}$ refer to those prior to the current block, i.e. parts that have already been computed, while those with a $\{\mathbb{B}^*\}$ refer to those after the current block. It is helpful to realise that $\boldsymbol{L}_{\{\mathbb{C}^{(k)}\}\{\mathbb{C}^{(k)}\}}$ and $\boldsymbol{C}_{\{\mathbb{C}^{(k)}\}\{\mathbb{C}^{(k)}\}}$ are $|\mathbb{C}|$ by $|\mathbb{C}|$ matrices where $|\mathbb{C}|$ is the blocksize (and thus there are $|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|$ blocks, assuming that $|\mathbb{C}|$ evenly divides $|\mathbb{B}^{(\Bbbk)}|$, for the sake of brevity), and so any $\{\mathbb{C}^{(k)}\}$ subscript refers to a dimension of size $|\mathbb{C}|$, and any variable with a $\{\mathbb{B}^*\}$ subscript has dimension $|\mathbb{C}|(|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|-i)=|\mathbb{B}^{(\Bbbk)}|-i|\mathbb{C}|$ where i refers to the current block. Similarly anything with a $\{\mathbb{T}^*\}$ subscript has a dimension of $(i-1)|\mathbb{C}|$. There are two key components which are shaded that are considered. The first of these:

$$\boldsymbol{C}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}:=\boldsymbol{C}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}-\boldsymbol{L}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}}\boldsymbol{C}_{\{\mathbb{C}^{(k)}\}\{\mathbb{T}^*\}},$$

which becomes after ignoring subtraction (since this is lower order):

$$\boldsymbol{C}_{\{\mathbb{B}^*\}\{\mathbb{T}^*\}}:=\boldsymbol{L}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}}\boldsymbol{C}_{\{\mathbb{C}^{(k)}\}\{\mathbb{T}^*\}}.$$

The matrices $\boldsymbol{L}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}}$ and $\boldsymbol{C}_{\{\mathbb{C}^{(k)}\}\{\mathbb{T}^*\}}$ are standard rectangular matrices of sizes $(|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|) \times |\mathbb{C}|$ and $|\mathbb{C}| \times (i-1)|\mathbb{C}|$ respectively. So multiplying them together leads to the following number of multiplications and additions summed over blocks:

$$
\begin{aligned}
&\sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|} ((|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|) \times |\mathbb{C}| \times (i-1)|\mathbb{C}| + (|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|) \times |\mathbb{C}| \times (i-1)|\mathbb{C}|) \\
&= \sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|} (2|\mathbb{C}|^2(|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|)(i-1)) \\
&= \sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|} (2|\mathbb{C}|^2\ |\mathbb{B}^{(\Bbbk)}|(i-1) - 2|\mathbb{C}|^3 i(i-1)) \\
&= \sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|} (2|\mathbb{C}|^2|\mathbb{B}^{(\Bbbk)}|(i-1) - 4|\mathbb{C}|^3 i(i-1)/2) \\
&= 2|\mathbb{B}^{(\Bbbk)}|^2(|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|(|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}| - 1)/2) - 4|\mathbb{C}|^3((|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}| - 1)|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|(|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}| + 1)/6) \\
&\approx |\mathbb{B}^{(\Bbbk)}|^3 - 2|\mathbb{B}^{(\Bbbk)}|^3/3 \\
&= |\mathbb{B}^{(\Bbbk)}|^3/3.
\end{aligned}
$$

It is now worth considering the second shaded area:

$$\boldsymbol{C}_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} := \boldsymbol{C}_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} - (\boldsymbol{L}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}} \boldsymbol{C}^{\top}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}} + \boldsymbol{C}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}} \boldsymbol{L}^{\top}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}}),$$

which is achieved using HER2K. The author assumes that since the first product in brackets is equal to the transpose of the second product that the second product comes for a lower order cost than the first product, and thus can be ignored. Similarly subtraction at the front of the equation will be ignored. This leaves the following:

$$\boldsymbol{C}_{\{\mathbb{B}^*\}\{\mathbb{B}^*\}} := (\boldsymbol{L}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}} \boldsymbol{C}^{\top}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}}).$$

The matrices $\boldsymbol{L}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}}$ and $\boldsymbol{C}^{\top}_{\{\mathbb{B}^*\}\{\mathbb{C}^{(k)}\}}$ are standard rectangular matrices of sizes $(|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|) \times |\mathbb{C}|$ and $|\mathbb{C}| \times (|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|)$ respectively. So multiplying them together leads to the following number of multiplica-

tions and additions summed over blocks:

$$
\begin{aligned}
&\sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|} ((|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|) \times |\mathbb{C}| \times (|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|) + (|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|) \times |\mathbb{C}| \times (|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|)) \\
&= \sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|} (2|\mathbb{C}|(|\mathbb{B}^{(\Bbbk)}| - i|\mathbb{C}|)^2) \\
&= \sum_{i=1}^{|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|} (2|\mathbb{C}||\mathbb{B}^{(\Bbbk)}|^2 - 4|\mathbb{C}|^2|\mathbb{B}^{(\Bbbk)}|i + 2|\mathbb{C}|^3 i^2) \\
&= 2|\mathbb{C}||\mathbb{B}^{(\Bbbk)}|^2(|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|) - 4|\mathbb{C}|^2|\mathbb{B}^{(\Bbbk)}|(|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|(|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}| + 1)/2) \\
&+ 2|\mathbb{C}|^3(|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}|(|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}| + 1)(2|\mathbb{B}^{(\Bbbk)}|/|\mathbb{C}| + 1)) \\
&\approx 2|\mathbb{B}^{(\Bbbk)}|^3 - 2|\mathbb{B}^{(\Bbbk)}|^3 + 4|\mathbb{B}^{(\Bbbk)}|^3/6 \\
&= 2|\mathbb{B}^{(\Bbbk)}|^3/3.
\end{aligned}
$$

Adding these two together (ignoring the non-shaded parts as those are lower order of magnitude) leads to $2|\mathbb{B}^{(\Bbbk)}|^3/3 + |\mathbb{B}^{(\Bbbk)}|^3/3 = |\mathbb{B}^{(\Bbbk)}|^3$ which is in agreement with the aforementioned paper.

C. C wrapper for implementation of `forttwosidedtrsm`

The following is the implementation of the C wrapper to enable the use of `twosidedtrsm` via FORTRAN. First an environment is created as the Elemental library requires an Elemental environment to be created in order to function. Then options are specified, in this case a lower triangular input matrix that is not unit diagonal. Then Elemental matrices are created for both the Cholesky factor and for the output matrix, as this type is required to work with the functions of the Elemental library. Finally, the function `twosidedtrsm` is called to perform the invert cross-multiply.

```
void forttwosidedtrsm(long* numrows, long* numcols, long* ldl, double** l,
long* ldz, double** z, long* nbr)
{
int tempArgc = 0;
char** tempArgv = new char* [0];

El::Environment* env = new El::Environment( tempArgc, tempArgv );

char uplo = 'L';
El::UpperOrLowerNS::UpperOrLower ElUplo =
El::UpperOrLowerNS::CharToUpperOrLower(uplo);
char diag = 'N';
El::UnitOrNonUnitNS::UnitOrNonUnit ElDiag =
El::UnitOrNonUnitNS::CharToUnitOrNonUnit(diag);

long tempNBR = *nbr;
double* tempL = *l;
double* adjustedL = &tempL[tempNBR];
double* tempZ = *z;
double* adjustedZ = &tempZ[tempNBR];
```

```
El::Matrix<double> fortLMatrix( (El::Int ) (*numrows), (El::Int ) (*numcols),
adjustedL, (El::Int ) (*ldl));
El::Matrix<double> fortZMatrix( (El::Int ) (*numrows), (El::Int ) (*numcols),
adjustedZ, (El::Int ) (*ldz));


// All the matrices are square here so numrows = numcols
// Further all their dimensions must be the same so this one parameter is all that is nee

El::TwoSidedTrsm( ElUplo, ElDiag, fortZMatrix, fortLMatrix);

}
```