BLMM notes

Table of Contents

1 Linear Mixed Models

This section aims to document what Linear Mixed Models are, do and what we could aim to implement in a distributed setting.

1.1 Formulation

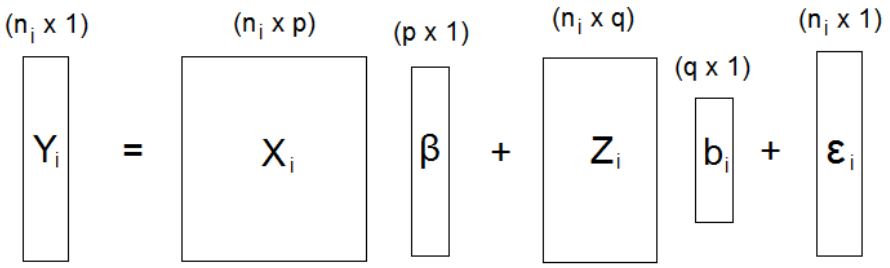
A Linear Mixed Model, for a specific site/subject/group/timeseries/cluster/etc, cluster i, is assumed to take the form:

$$Y_i = X_i\beta + Z_i b_i + \epsilon_i$$

Where:

- X_i is an $(n_i \times p)$ design matrix of covariates.
- β is a $(1 \times p)$ vector of Fixed Effects.
- Z_i is an $(n_i \times q)$ matrix of covariates.
- b_i is a $(q \times 1)$ vector of Random Effects.
- n_i is the number of observations for the i^{th} cluster.
- $N = \Sigma_{i=1}^{S} n_i$ is the total number of observations.
- S is the total number of subjects/sites/clusters.

The below image demonstrates what this would look like:



For all subjects, the model can be combined into one matrix equation as follows:

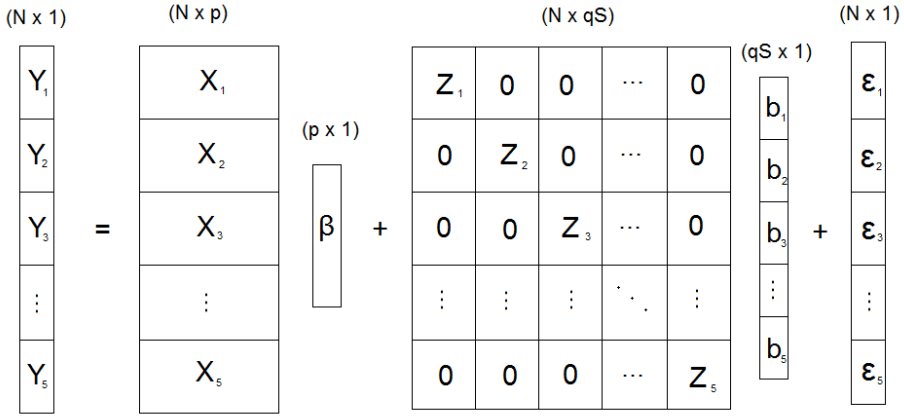
$$Y = X\beta + Zb + \epsilon$$

Where:

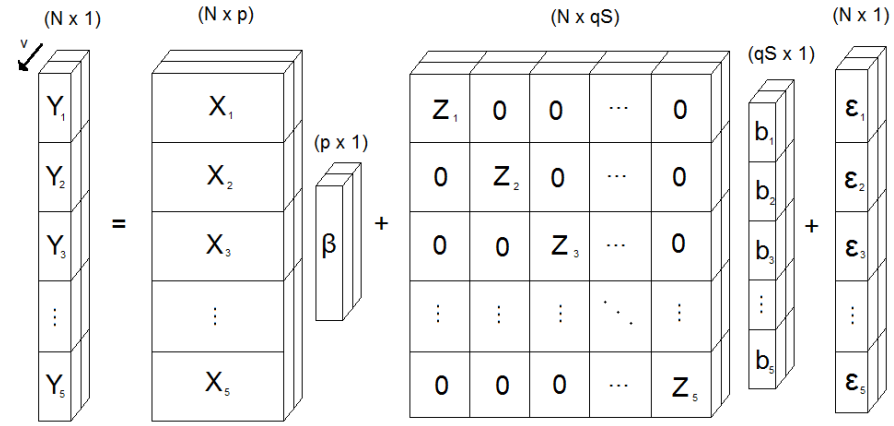
- X is an $(N \times p)$ design matrix of covariates, obtained by stacking all S X_i matrices on top of one another.

- β is a $(1 \times p)$ vector of Fixed Effects, the same as in the previous formulation.
- Z is an $(N \times Sq)$ matrix of covariates, obtained by creating a block diagonal matrix using all Z_i matrices.
- b is a $(Sq \times 1)$ vector of Random Effects, obtained by stacking all S b_i matrices on top of one another.

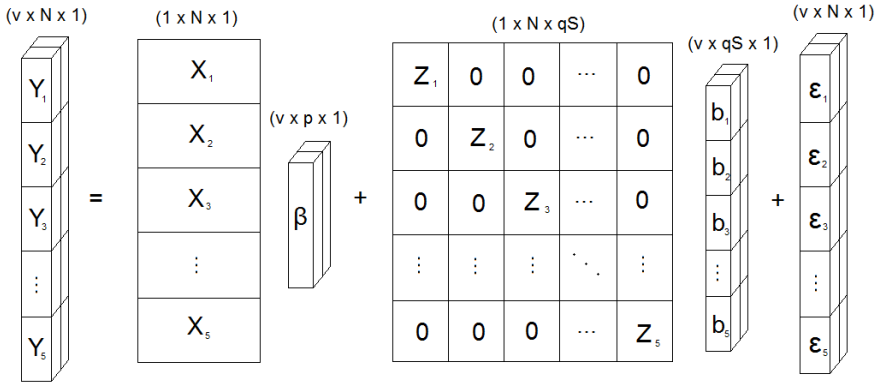
The below picture demonstrates what this would look like:



In NeuroImaging, we have v of these models, where v is the number of voxels in an image. Typically in NeuroImaging, S is number of subjects if dealing with longitudinal time series data or the number of groups if dealing with a group level design. The setup would now look like the below with each matrix being 3 dimensional with first dimension v:



Usually the covariate matrices, Z and X, are assumed to be constant across voxels as so;



The error term for cluster i observation j is often assumed to be $\epsilon_{ij} \sim N(0, \sigma^2)$. Within clusters ϵ_{ij} are usually assumed to be i.i.d. with covariance matrix for ϵ_i equal to $\Sigma_i = \sigma^2 I_{n_i}$. However, Σ_i can take any symmetric form so long as it only depends on i through it's dimension $(n_i \times n_i)$.

The random effects vector for cluster i, b_i, are usually assumed $N(\mathbf{0}_q, \boldsymbol{D})$ and i.i.d with respect to the other clusters (individual $b_{i,j}$ may be dependent on one another for fixed i, i.e. within a cluster, however).

All ϵ_i and b_i are also assumed independent of one another. The above model leads to the following distributional and log-likelihood results.

Distributions of model variables

Variable	Distribution	Notes
ϵ_i	$N(\mathbf{0}_{n_i}, \boldsymbol{\Sigma_i})$	Σ_i is often assumed to equal $\sigma^2 I_{n_i}$
b_i	$N(\mathbf{0}_q, \boldsymbol{D})$	D only depends on i through it's dimension $(n_i \times n_i)$
Y_i	$N(X_i\beta, \boldsymbol{Z_i D Z_i' + \Sigma_i})$	This is known as the marginal model. Common notation is $\boldsymbol{V_i = Z_i D Z_i' + \Sigma_i}$
$Y_i \vert b_i$	$N(X_i\beta + Z_i b_i, \boldsymbol{\Sigma_i})$	-

Log likelihood functions Define α Let $\theta = (\beta', \alpha')'$ be the vector of all parameters in the marginal model for Y_i. The marginal likelihood function

1.2 Examples

Random intercept model

Random intercept and Random slope model

Splines

Multi-factor

1.3 Assumptions made by lmer

1.4 Algorithms

Fixed Effects and covariance estimates In practice the Fixed effect, β, parameters and covariance estimates, α, are of primary interest. All algorithms for estimating $\theta = (\beta', \alpha')'$ have general form:

$$t_{s+1} = t_s + \lambda_s \delta_s$$

Where s is iteration number, λ_s is step size and t_s estimates θ in this setting. In all algorithms, $\delta_s = H_s^{-1} g_s$ where;

$$g_s = \frac{\delta l}{\delta t}\Big|_{t=t_s}$$

The only difference between algorithms is the matrix H_s, which is preferably positive definite. The table overleaf defines H_s for each of the "maximum likelihood" (ML)-based algorithms considered. The sections following describe the ML-based algorithms of this form in greater detail. The final algorithm described is the PLS algorithm used by "lmer" in R.

Algorithm	H_s	Notation
Newton-Raphson (NR)	Negative Hessian matrix	$H_s = -\left[\frac{\delta^2 l}{\delta\theta_{s,i}\theta_{s,j}}\right]$
Fisher Scoring (FS)	information matrix	$H_s = \mathbb{E}\left(-\left[\frac{\delta^2 l}{\delta\theta_{s,i}\theta_{s,j}}\right]\right)$
Simplified Fisher Scoring (SFS)	Information matrix block for D, standard estimators for β and σ^2	$H_s^{(D)} = \mathbb{E}\left(-\left[\frac{\delta^2 l}{\delta\theta_{s,i}\theta_{s,j}}\right]\right)_{(3,3)}$ $\beta_s = \hat{\beta}_{GLS}(D_s)$ $\beta_s = \hat{\sigma^2}(D_s, \beta_s)$
Empirical Fisher Scoring (EFS)	Approximated Information matrix	$H_s = \Sigma_{i=1}^{S}(\frac{\delta l_i}{\delta\theta})(\frac{\delta l_i}{\delta\theta})'$
Variance-profile Fisher Scoring (VPFS)	Information matrix for variance profile likelihood	$H_s = \mathbb{E}\left(-\left[\frac{\delta^2 l_{vp}}{\delta\bar{\theta}_{s,i}\bar{\theta}_{s,j}}\right]\right)$ (Where $\bar{\theta} = [\beta', \text{vech}(D)']'$)
Full-profile Fisher Scoring (FPFS)	Information matrix for full profile likelihood	$H_s = \mathbb{E}\left(-\left[\frac{\delta^2 l_{fp}}{\delta\bar{\theta}_{s,i}\bar{\theta}_{s,j}}\right]\right)$ (Where $\bar{\theta} = \text{vech}(D)$)
Expectation Maximization (EM)	Individual updates to $\hat{\beta}$, $\boldsymbol{D}$ and σ^2	$\hat{\beta}_s$ obtained via GLS $\Delta_{\sigma_s^2} = \frac{2\sigma_s^2}{N_T}\frac{\delta l}{\delta\sigma_s^2}$ $\Delta_{\boldsymbol{D}_s} = \frac{2}{N_T}\boldsymbol{D}_s\frac{\delta l}{\delta\boldsymbol{D}_s}\boldsymbol{D}_s$
Fixed Point (FP)	Dimension Reduced individual updates to $\hat{\beta}$, $\boldsymbol{D}$ and σ^2	Dimension Reduced equivalent of Expectation Maximization
Penalized Least Squares (PLS)	Iteration through solving reorganised PLS objective function	Solves components of: $\begin{bmatrix}\Lambda_s' Z'y \\ X'y\end{bmatrix} = \begin{bmatrix}\Lambda_s' Z'Z\Lambda_s & \Lambda_s' Z'X \\ X'Z\Lambda_s & X'X\end{bmatrix}\begin{bmatrix}\hat{b}_s \\ \hat{\beta}_s\end{bmatrix}$ (where $\Lambda_s\Lambda_s' = \frac{1}{\sigma_s^2}D_s$)

Two Restricted Maximum Likelihood (ReML) based algorithms are described in the below table:

Algorithm	H_s	Notation
Fisher Scoring (FS)	Information matrix	$H_s = \mathbb{E}\big(- \big[\frac{\delta^2 l_R}{\delta\theta_{s,i}\theta_{s,j}}\big]\big)$
Expectation Maximization (EM)	Individual updates to $\hat{\beta}$, $\boldsymbol{D}$ and σ^2	$\hat{\beta}_s$ obtained via GLS $\Delta_{\sigma_s^2} = \frac{2\sigma_s^2}{N_T - m}\frac{\delta l}{\delta \sigma_s^2}$ $\Delta_{\boldsymbol{D}_s} = \frac{2}{N_T - m}\boldsymbol{D}_s\frac{\delta l}{\delta \boldsymbol{D}_s}\boldsymbol{D}_s$

Newton-Raphson Algorithm

The following box contains the algorithm for NR estimation of $\theta = (\beta', \alpha')'$.

Input: β_0 (Initial Beta estimate)
σ_0^2 (Initial fixed effects variance estimate)
vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β_s (Final Beta estimate)
σ_s^2 (Final fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$
while *not converged* **do**
  for *all* i **do**
    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$
    Assign $e_{s,i} = y_i - X_i \beta_s$
    Assign $R_{s,i} = Z_i' V_{s,i}^{-1} Z_i$
  end
  Assign $\left[\frac{\delta l}{\delta \beta}\right]_s = \sigma_s^{-2} \sum X_i' V_{s,i}^{-1} e_{s,i}$
  Assign $\left[\frac{\delta l}{\delta \sigma^2}\right]_s = -\frac{1}{2} N_T \sigma_s^{-2} + \frac{1}{2} \sigma_s^{-4} \sum e_{s,i}' V_{s,i}^{-1} e_{s,i}$
  Assign $\left[\frac{\delta l}{\delta D}\right]_s = -\frac{1}{2} \sum [R_{s,i} - \sigma_s^{-2} Z_i' V_{s,i}^{-1} e_{s,i} e_{s,i}' V_{s,i}^{-1} Z_i]$
  Assign $\left[\frac{\delta l}{\delta \theta}\right]_s = \left(\left[\frac{\delta l}{\delta \beta}\right]_s', \left[\frac{\delta l}{\delta \sigma^2}\right]_s', \mathrm{vech}\left(\left[\frac{\delta l}{\delta D}\right]_s\right)'\right)'$
  Assign $H_{s,11} = \sigma_s^{-2} \sum X_i' V_{s,i}^{-1} X_i$
  Assign $H_{s,12} = \sigma_s^{-4} \sum X_i' V_{s,i}^{-1} e_{s,i}$
  Assign $H_{s,13} = \sigma_s^{-2} \sum e_{s,i}' V_{s,i}^{-1} Z_i \otimes X_i' V_{s,i}^{-1} Z_i$
  Assign $H_{s,22} = \sigma_s^{-6} \sum e_{s,i}' V_{s,i}^{-1} e_{s,i} - \frac{1}{2} N_T \sigma_s^{-4}$
  Assign $H_{s,23} = \frac{1}{2} \sigma_s^{-4} \sum e_{s,i}' V_{s,i}^{-1} Z_i \otimes e_{s,i}' V_{s,i}^{-1} Z_i$
  Assign $H_{s,33} = \frac{1}{2} \sum \{\sigma_s^{-2} (Z_i' V_{s,i}^{-1} e_{s,i} e_{s,i}' V_i^{-1} Z_i \otimes R_i +$
    $R_i \otimes Z_i' V_{s,i}^{-1} e_{s,i} e_{s,i}' V_i^{-1} Z_i) - R_i \otimes R_i\}$
  Assign $H_s = \begin{bmatrix} H_{s,11} & H_{s,12} & H_{s,13} \\ H_{s,12}' & H_{s,22} & H_{s,23} \\ H_{s,13}' & H_{s,23}' & H_{s,33} \end{bmatrix}$
  Assign $l_s = -\frac{1}{2} \left\{ N_T \ln(\sigma_s^2) + \sum \left[\ln|V_{s,i}| + \sigma_s^{-2} e_{s,i}' V_{s,i}^{-1} e_{s,i} \right] \right\}$
  if $l_s > l_{s-1}$ **then**
    Assign $\lambda = 1$
  else
    Assign $\lambda = \lambda/2$
  end
  Assign $[\beta_{s+1}', \sigma_{s+1}^{2\prime}, \mathrm{vec}(D_{s+1})']' = [\beta_s', \sigma_s^{2\prime}, \mathrm{vec}(D_s)']' + \lambda H_s^{-1} \left[\frac{\delta l}{\delta \theta}\right]_s$
  Assign s = s+1
end

There are several pros and cons to NR for mixed models. For example;

- Pro: The version of the algorithm proposed above is the dimension reduced NR algorithm, meaning a majority of the matrices involved in the above computation are of size $(p \times p)$ and $(q \times q)$.
- Pro: If the algorithm starts near the maximum then it is faster than FS.
- Pro: A convenient form of the NR algorithm exists involving QR decompositions (see 2.16).
- Con: NR does not guarantee a positive definite D matrix.
- Con: NR often doesn't converge if it starts far from the maximum.
- Con: NR is not robust to outliers as the observed Hessian is heavily influenced by the residuals.
- Con: The use of the observed Hessian massively complicates the computation compared to the expected Hessian.

Fisher Scoring Algorithm

There are many variations of the Fisher Scoring algorithm. The below gives the derivation of the Fisher information matrix and the following box contains the algorithm for FS estimation of $\theta = (\beta', \alpha')'$ in it's most well known formulation.

Derivation of $\mathcal{I}(\theta)$

To derive the Fisher Information matrix we first use the following lemma:

Lemma 1: The Fisher Information matrix, defined as $\mathcal{I}(\theta) = \mathbb{E}_\theta\{[l'(\theta|y)]^2\}$, is also given by $\mathcal{I}(\theta) = \mathrm{Var}_\theta[l'(\theta|y)]$.

Proof:

First, note that, by the definition of expectation:

$$\begin{aligned}\mathbb{E}_\theta\{l'(\theta|y)\} &= \int l'(\theta|y) f_y(y|\theta) dy \\ &= \int \frac{\delta}{\delta\theta}\big[\log(f_y(y|\theta))\big] f_y(y|\theta) dy \\ &= \int \frac{f'_y(y|\theta)}{f_y(y|\theta)} f_y(y|\theta) dy\end{aligned}$$

$$= \int f'_y(y|\theta)dy = 0$$

Therefore, by the definition of variance:

$$\begin{aligned}\mathrm{Var}_\theta[l'(\theta|y)] &= \mathbb{E}_\theta\{[l'(\theta|y) - \mathbb{E}_\theta\{l'(\theta|y)\}]^2\} \\ &= \mathbb{E}_\theta\{[l'(\theta|y) - 0]^2\} \\ &= \mathbb{E}_\theta\{[l'(\theta|y)]^2\} \\ &= \mathcal{I}(\theta)\end{aligned}$$

Lemma 2: The Fisher information matrix of the variance parameters of a linear mixed effects model, $\mathcal{I}(\sigma^2, \mathrm{vech}(D))$, is given by:

$$\mathcal{I}(\sigma^2, \mathrm{vech}(D)) = \begin{bmatrix} \frac{1}{2}n\sigma^{-4} & \frac{1}{2}\sigma^{-2}\mathrm{vec}'(R)\mathcal{D}^{+\prime} \\ \frac{1}{2}\sigma^{-2}\mathcal{D}^{+}\mathrm{vec}(R) & \frac{1}{2}\mathcal{D}^{+}(R \otimes R)\mathcal{D}^{+\prime} \end{bmatrix}$$

Where $R = Z'(I + ZDZ')^{-1}Z = ((Z'Z)^{-1} + D)^{-1}$ and $\mathcal{D}^+$ is the inverse of an appropriately sized duplication matrix.

Proof:

The log-likelihood of the mixed effects model is given below:

$$l(\theta|y) = -\frac{1}{2}\left\{n\ln(\sigma^2) + \sigma^{-2}(y - X\beta)'(I + ZDZ')^{-1}(y - X\beta) + \ln|I + ZDZ'|\right\}$$

The derivative of $l(\theta|y)$ with respect to σ^2 is given by:

$$\begin{aligned}\frac{\delta l(\theta|y)}{\delta\sigma^2} &= -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}(Y - X\beta)'(I + ZDZ')^{-1}(Y - X\beta) \\ &= -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^2}u'u\end{aligned}$$

Where $u = \sigma^{-1}(I + ZDZ)^{-\frac{1}{2}}(Y - X\beta) \sim N(0, I)$. Using the standard results for variance of a quadratic form (reference with proof commented out), we obtain:

$$\mathrm{Var}_{(\sigma^2, D)}\left(\frac{\delta l(\theta|y)}{\delta\sigma^2}\right) = \frac{1}{4\sigma^4}\mathrm{Var}_{(\sigma^2, D)}(u'u) = \frac{n}{2\sigma^4}$$

This gives the top left element of $\mathcal{I}(\sigma^2, \mathrm{vech}(D))$.

Now, consider the derivative of $l(\theta|y)$ with respect to D which is given by:

$$\frac{\delta l(\theta|y)}{\delta D} = \frac{1}{2\sigma^2} Z'(I+ZDZ')^{-1}(y-X\beta)(y-X\beta)'(I+ZDZ')^{-1}Z - \frac{1}{2}Z'(I+ZDZ')^{-1}Z$$
$$= \frac{1}{2}(Tu)(Tu)' - \frac{1}{2}Z'(I+ZDZ')^{-1}Z$$

Where $T = Z'(I+ZDZ')^{-\frac{1}{2}}$.

It follows that:

$$\frac{\delta l(\theta|y)}{\delta \text{vech}(D)} = \text{vech}\left(\frac{\delta l(\theta|y)}{\delta D}\right) = \text{vech}\left(\frac{1}{2}(Tu)(Tu)' - \frac{1}{2}Z'(I+ZDZ')^{-1}Z\right)$$

And, therefore:

$$\text{cov}\left(\frac{\delta l(\theta|y)}{\delta \text{vech}(D)}\right) = \text{cov}\left(\text{vech}\left[\frac{1}{2}(Tu)(Tu)' - \frac{1}{2}Z'(I+ZDZ')^{-1}Z\right]\right)$$
$$= \text{cov}\left(\text{vech}\left[\frac{1}{2}(Tu)(Tu)'\right]\right)$$
(As the second term was a fixed constant/non-random)
$$= \frac{1}{4}\text{cov}\left(\text{vech}\left[(Tu)(Tu)'\right]\right)$$
$$= \frac{1}{4}\text{cov}\left(\mathcal{D}^{+}\text{vec}\left[(Tu)(Tu)'\right]\right)$$
(As $\text{vech}(A) = \mathcal{D}^{+}\text{vec}(A)$, where $\mathcal{D}^{+}$ is the generalized inverse of the duplication matrix with correct dimension)
$$= \frac{1}{4}\mathcal{D}^{+}\text{cov}\left(\text{vec}\left[(Tu)(Tu)'\right]\right)\mathcal{D}^{+'}$$
$$= \frac{1}{4}\mathcal{D}^{+}\text{cov}\left((Tu)\otimes(Tu)\right)\mathcal{D}^{+'}$$
(As $\text{vec}((Tu)(Tu)') = \text{vec}(< Tu, Tu >) = (Tu)\otimes(Tu)$ where $< \cdot, \cdot >$ denotes the vector outer product, see wikipedia)
$$= \frac{1}{4}\mathcal{D}^{+}\text{cov}\left((T\otimes T)(u\otimes u)\right)\mathcal{D}^{+'}$$

(By the mixed-product property of the kronecker product)

$$= \frac{1}{4}\mathcal{D}^{+}(T \otimes T)\mathrm{cov}(u \otimes u)(T \otimes T)'\mathcal{D}^{+'}$$

$$= \frac{1}{4}\mathcal{D}^{+}(T \otimes T)2N_n(T' \otimes T')\mathcal{D}^{+'}$$

(As $\mathrm{cov}(u \otimes u) = 2N_n$, i.e. 2 times the commutation matrix)

$$= \frac{1}{2}\mathcal{D}^{+}(T \otimes T)(T' \otimes T')N_n\mathcal{D}^{+'}$$

(As the commutation matrix satisfies $(A \otimes A)N_n = N_n(A \otimes A)$ for all A, see Magnus 1986)

$$= \frac{1}{2}\mathcal{D}^{+}(T \otimes T)(T' \otimes T')\mathcal{D}^{+'}$$

(As the commutation matrix is symmetric and satisfies $\mathcal{D}^{+} = \mathcal{D}^{+}N_n$ for all duplication matrices $\mathcal{D}$, see Magnus 1986)

$$= \frac{1}{2}\mathcal{D}^{+}(TT' \otimes TT')\mathcal{D}^{+'}$$

(By the mixed product property of the kronecker product)

$$= \frac{1}{2}\mathcal{D}^{+}(R \otimes R)\mathcal{D}^{+'}$$

(As $TT' = Z'(I+ZDZ')^{\frac{1}{2}}(Z'(I+ZDZ')^{\frac{1}{2}})' = Z'(I+ZDZ')Z = R$)

This gives bottom right element of $\mathcal{I}(\sigma^2, \mathrm{vech}(D))$.

Finally, consider the remaining entries of the information matrix, specified by:

$$\mathrm{cov}\left(\frac{\delta l(\theta|y)}{\delta\sigma^2}, \frac{\delta l(\theta|y)}{\delta \mathrm{vech}(D)}\right)$$

$$= \mathrm{cov}\left(-\frac{n}{2\sigma^2} + \frac{1}{2\sigma^2}u'u, \mathrm{vech}\left(\frac{1}{2}(Tu)(Tu)' - \frac{1}{2}Z'(I+ZDZ')^{-1}Z\right)\right)$$

$$= \mathrm{cov}\left(\frac{1}{2\sigma^2}u'u, \mathrm{vech}\left(\frac{1}{2}(Tu)(Tu)'\right)\right)$$

$$= \mathrm{cov}\left(\frac{1}{2\sigma^2}u'u, \frac{1}{2}\mathcal{D}^{+}\mathrm{vec}\left((Tu)(Tu)'\right)\right)$$

(As $\mathrm{vech}(A) = \mathcal{D}^{+}\mathrm{vec}(A)$, where $\mathcal{D}^{+}$ is the generalized inverse of the duplication matrix with correct dimension)

$$= \mathrm{cov}\left(\frac{1}{2\sigma^2}u'u, \frac{1}{2}\mathcal{D}^{+}(Tu) \otimes (Tu)\right)$$

(As $\text{vec}((Tu)(Tu)') = \text{vec}(< Tu, Tu >) = (Tu) \otimes (Tu)$
where $< \cdot, \cdot >$ denotes the vector outer product, see wikipedia)

$$= \frac{1}{4\sigma^2}\text{cov}\bigg(u'u, \mathcal{D}^+(Tu) \otimes (Tu)\bigg)$$

(By moving the constants out, note the σ^{-2} only existed in one term)

$$= \frac{1}{4\sigma^2}\text{cov}\bigg(\text{vec}'(I_n)(u \otimes u), \mathcal{D}^+(Tu) \otimes (Tu)\bigg)$$

(As $b'b = (b \otimes b)'\text{vec}(I_n) = \text{vec}'(I_n)(b \otimes b)$ for all b,
see Demidenko 2013)

$$= \frac{1}{4\sigma^2}\text{cov}\bigg(\text{vec}'(I_n)(u \otimes u), \mathcal{D}^+(T \otimes T)(u \otimes u)\bigg)$$

(By the mixed product property of the kronecker product)

$$= \frac{1}{4\sigma^2}\text{vec}'(I_n)\text{cov}(u \otimes u)(T \otimes T)'\mathcal{D}^{+'}$$

(As $\text{cov}(Ax, Bx) = A\text{cov}(x, x)B' = A\text{cov}(x)B'$ for all x, A and B)

$$= \frac{1}{2\sigma^2}\text{vec}'(I_n)N_n(T \otimes T)'\mathcal{D}^{+'}$$

(As $\text{cov}(u \otimes u) = 2N_n$, i.e. 2 times the commutation matrix)

$$= \frac{1}{2\sigma^2}\text{vec}'(I_n)(T \otimes T)'N_n\mathcal{D}^{+'}$$

(As the commutation matrix satisfies $(A \otimes A)N_n = N_n(A \otimes A)$
for all A, and is symmetric, see Magnus 1986)

$$= \frac{1}{2\sigma^2}\text{vec}'(I_n)(T \otimes T)'\mathcal{D}^{+'}$$

(As the commutation matrix is symmetric and satisfies $\mathcal{D}^+ = \mathcal{D}^+N_n$
for all duplication matrices $\mathcal{D}$, see Magnus 1986)

$$= \frac{1}{2\sigma^2}\text{vec}'(TT')\mathcal{D}^{+'}$$

(As $\text{vec}'(I_n)(B \otimes B)' = ((B \otimes B)\text{vec}(I_n))' = (\text{vec}(BB'))' = \text{vec}'(BB')$
for all matrices, B)

$$= \frac{1}{2\sigma^2}\text{vec}'(R)\mathcal{D}^{+'}$$

This gives the top right element of $\mathcal{I}(\sigma^2, \text{vech}(D))$ and, by symmetry, it's transpose must also be the bottom left element.

Lemma 3: The information about β is orthogonal to the information about σ^2 and the information about vech(D). i.e.

$$\text{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, \frac{\delta l(\theta|y)}{\delta\sigma^2}\right) = 0$$

And:

$$\text{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, \frac{\delta l(\theta|y)}{\delta\text{vech}(D)}\right) = 0$$

Proof:

First, note that the derivative of $l(\theta|y)$ with respect to β is:

$$\frac{\delta l(\theta|y)}{\delta\beta} = \frac{1}{\sigma^2}X'(I + ZDZ')^{-1}(y - X\beta) = \sigma^{-1}X'V^{-\frac{1}{2}}u$$

Where u is as before and $V = I + ZDZ'$. Note now that:

$$\text{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, \frac{\delta l(\theta|y)}{\delta\sigma^2}\right) = \mathbb{E}\left[\frac{\delta l(\theta|y)}{\delta\beta}\frac{\delta l(\theta|y)}{\delta\sigma^2}\right] - \mathbb{E}\left[\frac{\delta l(\theta|y)}{\delta\beta}\right]\mathbb{E}\left[\frac{\delta l(\theta|y)}{\delta\sigma^2}\right]$$

(By the definition of covariance matrix)

$$= \mathbb{E}\left[\frac{\delta l(\theta|y)}{\delta\beta}\frac{\delta l(\theta|y)}{\delta\sigma^2}\right]$$

$$(\text{As } \mathbb{E}\left[\tfrac{\delta l(\theta|y)}{\delta\beta}\right] = \mathbb{E}[\sigma^{-1}XV^{-\frac{1}{2}}u] = \sigma^{-1}XV^{-\frac{1}{2}}\mathbb{E}[u] = 0, \text{as } U \sim N(0, I)\ )$$

$$= \mathbb{E}\left[\frac{1}{2\sigma^3}u'uX'V^{-\frac{1}{2}}u\right]$$

$$= \mathbb{E}\left[\frac{1}{2\sigma^3}\left(\sum_{i=1}^{q}u_i^2\right)X'V^{-\frac{1}{2}}u\right]$$

$$= \mathbb{E}\left[\frac{1}{2\sigma^3}X'V^{-\frac{1}{2}}u\left(\sum_{i=1}^{q}u_i^2\right)\right]$$

$$= \frac{1}{2\sigma^3}X'V^{-\frac{1}{2}}\mathbb{E}\left[u\left(\sum_{i=1}^{q}u_i^2\right)\right]$$

$$= 0$$

Where the last line follows as each element of the expression inside the expectation, say the i^{th} element, is a sum of u_i^3 and $u_iu_j^2$ for all $j \neq i$. However, as $u_i \sim N(0, 1)$ it follows that $\mathbb{E}(u_i^3)$ is the third moment of the normal distribution

and is therefore zero. And by independence, $\mathbb{E}(u_i u_j^2) = \mathbb{E}(u_i)\mathbb{E}(u_j^2) = 0$.

For the second part of the lemma consider the following, for some fixed vector c:

$$\operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, c'\frac{\delta l(\theta|y)}{\delta D}c\right) = \mathbb{E}\left[\frac{\delta l(\theta|y)}{\delta\beta}c'\frac{\delta l(\theta|y)}{\delta D}c\right] - \mathbb{E}\left[\frac{\delta l(\theta|y)}{\delta\beta}\right]\mathbb{E}\left[c'\frac{\delta l(\theta|y)}{\delta D}c\right]$$

(By the definition of covariance matrix)

$$= \mathbb{E}\left[\frac{\delta l(\theta|y)}{\delta\beta}c'\frac{\delta l(\theta|y)}{\delta D}c\right]$$

(As $\mathbb{E}\left[\frac{\delta l(\theta|y)}{\delta\beta}\right] = \mathbb{E}[\sigma^{-1}XV^{-\frac{1}{2}}u] = \sigma^{-1}XV^{-\frac{1}{2}}\mathbb{E}[u] = 0$, as $U \sim N(0, I)$)

(By the definition of covariance matrix)

$$= \mathbb{E}\left[\sigma^{-1}XV^{-\frac{1}{2}}uc'\left(\frac{1}{2}(Tu)(Tu)' - \frac{1}{2}Z'V^{-1}Z\right)c\right]$$

$$= \mathbb{E}\left[\sigma^{-1}XV^{-\frac{1}{2}}uc'\left(\frac{1}{2}(Tu)(Tu)'\right)c\right] - \mathbb{E}\left[\sigma^{-1}XV^{-\frac{1}{2}}uc'\left(\frac{1}{2}Z'V^{-1}Z\right)c\right]$$

$$= \mathbb{E}\left[\sigma^{-1}XV^{-\frac{1}{2}}uc'\left(\frac{1}{2}(Tu)(Tu)' - \frac{1}{2}Z'V^{-1}Z\right)c\right]$$

$$= \mathbb{E}\left[\sigma^{-1}XV^{-\frac{1}{2}}uc'\left(\frac{1}{2}(Tu)(Tu)'\right)c\right] - \sigma^{-1}XV^{-\frac{1}{2}}\mathbb{E}[u]c'\left(\frac{1}{2}Z'V^{-1}Zc\right)$$

$$= \mathbb{E}\left[\sigma^{-1}XV^{-\frac{1}{2}}uc'\left(\frac{1}{2}(Tu)(Tu)'\right)c\right]$$

(As $\mathbb{E}(u) = 0$)

$$= \frac{1}{2}\sigma^{-1}XV^{-\frac{1}{2}}\mathbb{E}\left[uc'Tuu'T'c\right]$$

$$= \frac{1}{2}\sigma^{-1}XV^{-\frac{1}{2}}\mathbb{E}\left[u\sum_{j=1}^{q}\sum_{k=1}^{q}\left(c_k T_{k,j}u_j\right)^2\right]$$

$$= \frac{1}{2}\sigma^{-1}XV^{-\frac{1}{2}}\sum_{j=1}^{q}\sum_{k=1}^{q}\begin{bmatrix} c_k^2T_{k,j}^2\mathbb{E}[u_j^2u_1] \\ c_k^2T_{k,j}^2\mathbb{E}[u_j^2u_2] \\ \vdots \\ c_k^2T_{k,j}^2\mathbb{E}[u_j^2u_q] \end{bmatrix}$$

$$= 0$$

(As $\mathbb{E}[u_j^2u_i] = \mathbb{E}[u_j^2]\mathbb{E}[u_i] = 0$ for $i \neq j$ and $\mathbb{E}[u_j^3] = 0$ as the third moment of a normal distribution is 0.)

Now, note that choosing $c = e_i$, the i^{th} standard basis vector, i.e. $c = (0, 0, ..., 1, ...0)$ where the 1 is in the i^{th} entry of c, gives the following:

$$\text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, c' \frac{\delta l(\theta|y)}{\delta D} c\right) = \text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{i,i}\right) = 0$$

In other words, the covariance between the derivative with respect to β and the derivative with respect to any element on the diagonal of D is zero. Similarly, choosing $c = e_i + e_j$ where $i \neq j$ gives:

$$\text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, c' \frac{\delta l(\theta|y)}{\delta D} c\right)$$

$$= \text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, \left(\left[\frac{\delta l(\theta|y)}{\delta D}\right]_{i,i} + \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{j,i} + \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{i,j} + \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{j,j}\right)\right)$$

$$= \text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, \left(\left[\frac{\delta l(\theta|y)}{\delta D}\right]_{i,i} + \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{j,j} + 2\left[\frac{\delta l(\theta|y)}{\delta D}\right]_{i,j}\right)\right)$$

(As $\frac{\delta l(\theta|y)}{\delta D}$ is symmetric, as D is symmetric.)

$$= \text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{i,i}\right) + \text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{j,j}\right) + 2\text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{i,j}\right)$$

(As $\text{cov}(A, B + C) = \text{cov}(A, B) + \text{cov}(A, C)$ for all A, B and C)

$$= 2\text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{i,j}\right)$$

$$\left(\text{As, by the previous, } \text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, \left[\frac{\delta l(\theta|y)}{\delta D}\right]_{i,i}\right) = 0 \text{ for all } i\right)$$

$$= 0$$

$$\left(\text{As, by the previous, } \text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, c' \frac{\delta l(\theta|y)}{\delta D} c\right) = 0 \text{ for all } c\right)$$

But this implies that the covariance between the derivative with respect to β and the derivative with respect to the $(i, j)^{th}$ element is zero for all (i, j). As $\text{vech}(D)$ is comprised of elements of D it therefore follows that this implies the covariance between the derivative with respect to β and the derivative with respect to every element of $\text{vech}(D)$ is zero. In other words:

$$\text{cov}\left(\frac{\delta l(\theta|y)}{\delta \beta}, \frac{\delta l(\theta|y)}{\delta \text{vech}(D)}\right) = 0$$

Lemma 4: The information of β is given by:

$$\operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}\right) = \sigma^{-2}X'VX$$

Proof:

$$\operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}\right) = \operatorname{cov}(\sigma^{-1}X'V^{-\frac{1}{2}}u)$$

$$= \sigma^{-2}X'V^{-\frac{1}{2}}\operatorname{cov}(u)V^{-\frac{1}{2}'}X'$$

$$= \sigma^{-2}X'V^{-\frac{1}{2}}\operatorname{cov}(u)V^{-\frac{1}{2}}X'$$

(As V is symmetric)

$$= \sigma^{-2}X'V^{-\frac{1}{2}}IV^{-\frac{1}{2}}X'$$

(As $u \sim N(0, I)$)

$$= \sigma^{-2}X'VX$$

Lemma 5: The total information matrix of $\theta = (\beta, \sigma^2, \operatorname{vech}(D))$, $\mathcal{I}(\theta)$, is given by:

$$\mathcal{I}(\theta) = \begin{bmatrix} \sigma^2X'VX & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}n\sigma^{-4} & \frac{1}{2}\sigma^{-2}\operatorname{vec}'(R)\mathcal{D}^{+'} \\ \mathbf{0} & \frac{1}{2}\sigma^{-2}\mathcal{D}^{+}\operatorname{vec}(R) & \frac{1}{2}\mathcal{D}^{+}(R\otimes R)\mathcal{D}^{+'} \end{bmatrix}$$

Proof:

By Lemma 1;

$$\mathcal{I}(\theta) = \begin{bmatrix} \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\sigma^2}, \frac{\delta l(\theta|y)}{\delta\beta}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}, \frac{\delta l(\theta|y)}{\delta\beta}\right) \\ \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, \frac{\delta l(\theta|y)}{\delta\sigma^2}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\sigma^2}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}, \frac{\delta l(\theta|y)}{\delta\sigma^2}\right) \\ \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, \frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\sigma^2}, \frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}\right) \end{bmatrix}$$

$$= \begin{bmatrix} \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, \frac{\delta l(\theta|y)}{\delta\sigma^2}\right)' & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, \frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}\right)' \\ \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, \frac{\delta l(\theta|y)}{\delta\sigma^2}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\sigma^2}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\sigma^2}, \frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}\right)' \\ \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\beta}, \frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\sigma^2}, \frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}\right) & \operatorname{cov}\left(\frac{\delta l(\theta|y)}{\delta\operatorname{vech}(D)}\right) \end{bmatrix}$$

(As $\text{cov}(A, B) = \text{cov}(B, A)'$ for all A and B)

$$= \begin{bmatrix} \sigma^2 X'VX & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}n\sigma^{-4} & \frac{1}{2}\sigma^{-2}\text{vec}'(R)\mathcal{D}^{+'} \\ \mathbf{0} & \frac{1}{2}\sigma^{-2}\mathcal{D}^{+}\text{vec}(R) & \frac{1}{2}\mathcal{D}^{+}(R \otimes R)\mathcal{D}^{+'} \end{bmatrix}$$

(By lemmas 2-4)

Input: β_0 (Initial Beta estimate)
σ_0^2 (Initial fixed effects variance estimate)
vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β_s (Final Beta estimate)
σ_s^2 (Final fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$
while *not converged* **do**
  for *all* i **do**
    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$
    Assign $e_{s,i} = y_i - X_i \beta_s$
    Assign $R_{s,i} = Z_i' V_{s,i}^{-1} Z_i$
  end
  Assign $\left[\frac{\delta l}{\delta \beta}\right]_s = \sigma_s^{-2} \sum X_i' V_{s,i}^{-1} e_{s,i}$
  Assign $\left[\frac{\delta l}{\delta \sigma^2}\right]_s = -\frac{1}{2} N_T \sigma_s^{-2} + \frac{1}{2}\sigma_s^{-4} \sum e_{s,i}' V_{s,i}^{-1} e_{s,i}$
  Assign $\left[\frac{\delta l}{\delta D}\right]_s = -\frac{1}{2} \sum [R_{s,i} - \sigma_s^{-2} Z_i' V_{s,i}^{-1} e_{s,i} e_{s,i}' V_{s,i}^{-1} Z_i]$
  Assign $\left[\frac{\delta l}{\delta \theta}\right]_s = \left(\left[\frac{\delta l}{\delta \beta}\right]_s', \left[\frac{\delta l}{\delta \sigma^2}\right]_s', \mathrm{vech}\left(\left[\frac{\delta l}{\delta D}\right]_s\right)'\right)'$
  Assign $\mathcal{I}_{s,11} = \sigma_s^{-2} \sum X_i' V_{s,i}^{-1} X_i$
  Assign $\mathcal{I}_{s,22} = \frac{1}{2} N_T \sigma_s^{-4}$
  Assign $\mathcal{I}_{s,33} = \frac{1}{2} \sum R_{s,i} \otimes R_{s,i}$
  Assign $\mathcal{I}_{s,23} = \frac{1}{2}\sigma_s^{-2} \sum \mathrm{vec}(R_i)'$
  Assign $\mathcal{I}_s = \begin{bmatrix} \mathcal{I}_{s,11} & 0 & 0 \\ 0 & \mathcal{I}_{s,22} & \mathcal{I}_{s,23} \\ 0 & \mathcal{I}_{s,23}' & \mathcal{I}_{s,33} \end{bmatrix}$
  Assign $l_s = -\frac{1}{2}\left\{ N_T \ln(\sigma_s^2) + \sum \left[\ln|V_{s,i}| + \sigma_s^{-2} e_{s,i}' V_{s,i}^{-1} e_{s,i}\right] \right\}$
  if $l_s > l_{s-1}$ **then**
    Assign $\lambda = 1$
  else
    Assign $\lambda = \lambda/2$
  end
  Assign $[\beta_{s+1}', {\sigma_{s+1}^2}', \mathrm{vec}(D_{s+1})']' = [\beta_s', \sigma_s^{2\prime}, \mathrm{vec}(D_s)']' + \lambda \mathcal{I}_s^{-1} \left[\frac{\delta l}{\delta \theta}\right]_s$
  Assign s = s+1
end

Some pros and cons of this algorithm are discussed below;

- Con: When both converge, FS is slower than NR.
- Pro: The expected Hessian is always positive definite, which means FS fails to converge much more infrequently than NR.
- Pro: The expected information matrix often gives a much better estimate of the asymptotic covariance than the empirical information matrix. The reason for this is that it less heavily influenced by outlier residuals.
- Pro: The code for this algorithm is much cleaner, and requires less computation than NR.
- Pro: The expected information matrix is also the same for both ML and REML, as it is based on asymptotic theory.
- Pro: FS is more robust to starting point that NR.

Simplified Fisher Scoring Algorithm

The simplified Fisher Scoring algorithm is based on the $(3,3)^{rd}$ block of the expected information matrix and uses this to update estimates for D based on the same logic as the standard FS algorithm.

However, for the estimate of β SFS uses the closed form generalized least squares (GSL) estimator, which would be correct given D were known. Similarly, σ^2 is estimated using a closed form estimator, $\hat{\sigma}^2$, which would be correct given D and β were known. Both estimators are given below:

$$\hat{\beta}_{GLS} = \left[\sum_{i=1}^{S} X_i' V_i^{-1} X_i\right]^{-1} \left[\sum_{i=1}^{S} X_i' V_i^{-1} Y_i\right]$$

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^{S} e_i' V_i^{-1} e_i$$

The main pro of this is that it is much shorter and quicker than the others.

However, there is little information on how it performs in terms of convergence. The algorithm is given by:

Input: β_0 (Initial Beta estimate)
σ_0^2 (Initial fixed effects variance estimate)
vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β_s (Final Beta estimate)
σ_s^2 (Final fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$
while *not converged* **do**
  for *all* i **do**
    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$
    Assign $e_{s,i} = y_i - X_i \beta_s$
    Assign $R_{s,i} = Z_i' V_{s,i}^{-1} Z_i$
  end
  Assign $\left[\frac{\delta l}{\delta D}\right]_s = \sum [R_{s,i} - \sigma_s^{-2} Z_i' V_{s,i}^{-1} e_{s,i} e_{s,i}' V_{s,i}^{-1} Z_i]$
  Assign $l_s = -\frac{1}{2}\left\{ N_T \ln(\sigma_s^2) + \sum \left[\ln|V_{s,i}| + \sigma_s^{-2} e_{s,i}' V_{s,i}^{-1} e_{s,i} \right] \right\}$
  if $l_s > l_{s-1}$ **then**
    Assign $\lambda = 1$
  else
    Assign $\lambda = \lambda/2$
  end
  Assign $\mathrm{vec}(D_{s+1}) = \mathrm{vec}(D_s) + \lambda (\sum R_{s,i} \otimes R_{s,i})^{-1} \left[\frac{\delta l}{\delta \theta}\right]_s$
  Assign $\beta_{s+1} = \left[\sum X_i' V_{s,i}^{-1} X_i \right]^{-1} \left[\sum X_i' V_{s,i}^{-1} Y_i \right]$
  Assign $\sigma_{s+1}^2 = \frac{1}{N_T} \sum e_{s,i}' V_{s,i}^{-1} e_{s,i}$
  Assign s = s+1
end

Empirical Fisher Scoring Algorithm

The empirical Fisher Scoring algorithm relies on the below approximation of

the Fisher Information matrix;

$$\mathcal{I} = \mathbb{E}\left[\left(\frac{\delta l}{\delta \theta}\right)\left(\frac{\delta l}{\delta \theta}\right)\right] \simeq \sum_{i=1}^{S}\left(\frac{\delta l_i}{\delta \theta}\right)\left(\frac{\delta l_i}{\delta \theta}\right)'$$

The algorithm for EFS is given below:

Input: β_0 (Initial Beta estimate)
σ_0^2 (Initial fixed effects variance estimate)
vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β_s (Final Beta estimate)
σ_s^2 (Final fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$
while *not converged* **do**
  for *all i* **do**
    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$
    Assign $e_{s,i} = y_i - X_i \beta_s$
    Assign $R_{s,i} = Z_i' V_{s,i}^{-1} Z_i$
    Assign $\left[\frac{\delta l_i}{\delta \beta}\right]_s = \sigma_s^{-2} X_i' V_{s,i}^{-1} e_{s,i}$
    Assign $\left[\frac{\delta l_i}{\delta \sigma^2}\right]_s = -\frac{1}{2} N_T \sigma_s^{-2} + \frac{1}{2}\sigma_s^{-4} e_{s,i}' V_{s,i}^{-1} e_{s,i}$
    Assign $\left[\frac{\delta l_i}{\delta D}\right]_s = -\frac{1}{2}[R_{s,i} - \sigma_s^{-2} Z_i' V_{s,i}^{-1} e_{s,i} e_{s,i}' V_{s,i}^{-1} Z_i]$
    Assign $\left[\frac{\delta l_i}{\delta \theta}\right]_s = \left(\left[\frac{\delta l_i}{\delta \beta}\right]_s', \left[\frac{\delta l_i}{\delta \sigma^2}\right]_s', \mathrm{vech}\left(\left[\frac{\delta l_i}{\delta D}\right]_s\right)'\right)'$
  end
  Assign $\mathcal{I}_s = \sum \left[\frac{\delta l_i}{\delta \theta}\right]_s \left[\frac{\delta l_i}{\delta \theta}\right]_s'$
  Assign $l_s = -\frac{1}{2}\left\{ N_T \ln(\sigma_s^2) + \sum \left[\ln|V_{s,i}| + \sigma_s^{-2} e_{s,i}' V_{s,i}^{-1} e_{s,i}\right] \right\}$
  if $l_s > l_{s-1}$ **then**
    Assign $\lambda = 1$
  else
    Assign $\lambda = \lambda/2$
  end
  Assign $[\beta_{s+1}', {\sigma_{s+1}^2}', \mathrm{vec}(D_{s+1})']' = [\beta_s', \sigma_s^{2\prime}, \mathrm{vec}(D_s)']' + \lambda \mathcal{I}_s^{-1} \sum \left[\frac{\delta l_i}{\delta \theta}\right]_s$
  Assign s = s+1
end

The pros and cons of EFS are:

– Pro: No matrix of second derivatives is needed! This makes this extremely quick and amenable to distributed programming.

– Pro: For large S (many clusters) this approximation is known to work well!

– Con: For small S (few clusters) this method may give unreliable results due to the approximation.

Variance-Profile Fisher Scoring Algorithm

The variance profile Fisher Scoring algorithm uses the variance profile log-likelihood and, as such, only maximizes with respect to D and β. The algorithm for VPFS is given overleaf.

The pros and cons of VPFS are:

– Pro: VFPS and FPFS are claimed to be some of the most economical and reliable FS algorithms in Demidenko (2004).

– Pro: Much of the computation is done before or after iteration, making the code faster.

– Con: Sometimes step sizes can fall victim to plateu effects with this likehood form, meaning a step size of size λ might not be enough to see an increase/decrease in likelihood.

Input: β_0 (Initial Beta estimate)
vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β_s (Final Beta estimate)
σ^2 (Fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$
for *all* i **do**
  Assign $T_i = X_i' Z_i$
  Assign $p_i = Z_i' y_i$
  Assign $M_i = Z_i' Z_i$
end
Assign $t = \sum X_i' y_i$
Assign $N = \sum X_i' X_i$
while *not converged* **do**
  for *all* i **do**
    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$
    Assign $e_{s,i} = y_i - X_i \beta_s$
    Assign $R_{s,i} = Z_i' V_{s,i}^{-1} Z_i$
    Assign $r_{s,i} = Z_i' e_{s,i}$
    Assign $S_{s,i} = e_{s,i}' e_{s,i}$
  end
  Assign $S_s = \sum \left[S_{s,i} - r_{s,i}' (D_s^{-1} + M_i)^{-1} r_{s,i} \right]$
  Assign $H_s = \frac{1}{2} \left[\sum R_{s,i} \otimes R_{s,i} - \frac{1}{N_T} \mathrm{vec}(\sum R_{s,i}) \mathrm{vec}(\sum R_{s,i})' \right]$
  Assign $l_{p,s} = -\frac{1}{2} \Big\{ N_T \ln\big(\sum [S_{s,i} - r_{s,i}' (D_s^{-1} + M_i)^{-1} r_{s,i})] \big) + \sum \ln |I + D_s M_i| \Big\}$
  Assign $\left[\frac{\delta l_p}{\delta D} \right]_s = \frac{1}{2} \sum \left[\frac{N_T}{S_s} (I + M_i D_s)^{-1} r_{s,i} r_{s,i}' (I + M_i D_s)^{-1} - M_i (I + M_i D_s)^{-1} \right]$
  if $l_{p,s} > l_{p,s-1}$ **then**
    Assign $\lambda = 1$
  else
    Assign $\lambda = \lambda / 2$
  end
  Assign $\mathrm{vec}(D_{s+1}) = \mathrm{vec}(D_s) + \lambda H_s^{-1} \left[\frac{\delta l_p}{\delta D} \right]_s$
  Assign $\beta_{s+1} = [N - \sum T_i (D_s^{-1} + M_i)^{-1} T_i']^{-1} [t - \sum T_i (D_s^{-1} + M_i)^{-1} p_i]$
  Assign s = s+1
end
Assign $\sigma^2 = \frac{1}{N_T} \sum (Y_i - X_i \beta_s)' (I + Z_i D_s Z_i')^{-1} (Y_i - X_i \beta_s)$

Full-Profile Fisher Scoring Algorithm

The Full Profile Fisher scoring algorithm maximizes the profile likehood for $D_- = D^{-1}$. The algorithm is given overleaf.

The pros and cons of FPFS are:

- Pro: D_- is optimized instead of D. This greatly reduces risk of numerical error as much less matrix inversion is carried out during optimization.

- Pro: This algorithm is faster than other FS algorithms as some of the computation can be done beforehand and the parameters in D are the only parameters maximized upon.

- Con: This algorithm is much less readable.

Input: vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β (Beta estimate)
σ^2 (Fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$

Assign $D_{-0} = D_0^{-1}$

for *all i* **do**

  Assign $T_i = X_i' Z_i$

  Assign $p_i = Z_i' y_i$

  Assign $M_i = Z_i' Z_i$

end

Assign $t = \sum X_i' y_i$

Assign $N = \sum X_i' X_i$

Assign $s_y = \sum y_i' y_i$

while *not converged* **do**

  for *all i* **do**

    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$

    Assign $R_{s,i} = Z_i' V_{s,i}^{-1} Z_i$

    Assign $F_{s,i} = D_{-s} + M_i$

  end

  Assign $G_s = N - \sum T_i F_{s,i}^{-1} T_i'$

  Assign $g_s = t - \sum T_i F_{s,i}^{-1} p_i'$

  Assign $S_s = s_y - \sum p_i' F_{s,i}^{-1} p_i' - g_s' G_s^{-1} g_s$

  Assign $Q_s = \sum F_{s,i}^{-1} (T_i' G_s^{-1} g_s p_i' + p_i g_s' G_s^{-1} T_i - T_i' G_s^{-1} g_s g_s' G_s^{-1} T_i) F_{s,i}^{-1}$

  Assign $H_s = \frac{1}{2} \left[\sum R_{s,i} \otimes R_{s,i} - \frac{1}{N_T} \text{vec}(\sum R_{s,i}) \text{vec}(\sum R_{s,i})' \right]$

  Assign $l_{p,s} = -\frac{1}{2} \left\{ N_T \ln(S_s) \ + \sum \ln|F_{s,i}| - N \ln|D_{-s}| \right\}$

  Assign $\left[\frac{\delta l_p}{\delta D_-} \right]_s = \frac{1}{2} \left\{ \frac{N_T}{S_s} \left(\sum F_{s,i}^{-1} p_i p_i' F_{s,i}^{-1} - Q_s \right) - \sum F_{s,i}^{-1} + N D_{-s}^{-1} \right\}$

  if $l_{p,s} > l_{p,s-1}$ **then**

    Assign $\lambda = 1$

  else

    Assign $\lambda = \lambda / 2$

  end

  Assign $\text{vec}(D_{-s+1}) = \text{vec}(D_{-s}) + \lambda (D_{-s} \otimes D_{-s}) H_s (D_{-s} \otimes D_{-s}) \left[\frac{\delta l_p}{\delta D_-} \right]_s$

  Assign s = s+1

end

Assign $\beta = [N - \sum T_i (D_{-s} + M_i)^{-1} T_i']^{-1} [t - \sum T_i (D_{-s} + M_i)^{-1} p_i]$

Assign $\sigma^2 = \frac{1}{N_T} \sum (Y_i - X_i \beta)' (I + Z_i D_{-s}^{-1} Z_i')^{-1} (Y_i - X_i \beta)$

Assign $\text{vec}(D_s) = \text{vec}(D_{-s}^{-1})$

Expectation Maximisation Algorithm

Expectation maximisation was one of the first algorithms used for linear mixed models. EM is not typically expressed in terms of H_s and the usual form of the algorithm is the form given below. The pros and cons of EM are;

- Con: EM is slower than FS.
- Con: EM may be extremely slow when the matrix D is close to zero.
- Pro: EM maximizes the log-likelihood from each iteration to each iteration (each step is guarenteed better than the last, hence consistent step sizes).
- Pro: The D_s matrix is positive definite if the initial D_0 is positive definite whereas in NR and FS non-negative D_s matrices are not guaranteed (Extra care must be taken in those cases to ensure the matrices are non-negative definite).
- Pro: EM is simple to implement.

Input: β_0 (Initial Beta estimate)
σ_0^2 (Initial fixed effects variance estimate)
vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β_s (Final Beta estimate)
σ_s^2 (Final fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$
while *not converged* **do**
  for *all* i **do**
    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$
    Assign $e_{s,i} = y_i - X_i \beta_s$
  end
  Assign $\sigma_{s+1}^2 = \sigma_s^2 - 1 + \frac{1}{\sigma_s^2 N_T} \sum e_{s,i}' V_{s,i}^{-1} e_{s,i}$
  Assign $D_{s+1} = D_s - \frac{1}{N} \sum [D_s Z_i' V_{s,i}^{-1} Z_i D_s - \sigma_s^{-2} D_s Z_i' V_{s,i}^{-1} e_{s,i} e_{s,i}' V_{s,i}^{-1} Z_i D_s]$
  Assign $\beta_{s+1} = \left[X_i' V_{s,i}^{-1} X_i\right]^{-1} X_i' V_{s,i}^{-1} y_i$
  Assign s = s+1
end

Fixed Point Algorithm

The fixed point algorithm for mixed models can be viewed as a dimension-reduced equivalent of expectation maximisation. The following box describes tha algorithm. The pros and cons of FP are:

Input: vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β_s (Final Beta estimate)
σ^2 (Fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$
for *all* i **do**
  Assign $T_i = X_i' Z_i$
  Assign $p_i = Z_i' y_i$
  Assign $M_i = Z_i' Z_i$
end
Assign $t = \sum X_i' y_i$
Assign $N = \sum X_i' X_i$
while *not converged* **do**
  for *all* i **do**
    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$
    Assign $R_{s,i} = Z_i' V_{s,i}^{-1} Z_i$
    Assign $F_{s,i} = D_s^{-1} + M_i$
    Assign $e_{s,i} = y_i - X_i \beta_s$
  end
  Assign $D_{s+1} = \frac{1}{N}\left[\frac{N_T}{\sum[||e||^2 - p_i' F_{s,i}^{-1} p_i]}\right] \sum F_{s,i}^{-1} p_i p_i' F_{s,i}^{-1} + \sum F_{s,i}^{-1}$
  Assign $\beta_{s+1} = [N - \sum T_i F_{s,i}^{-1} T_i']^{-1} [t - \sum T_i F_{s,i}^{-1} p_i]$
  Assign s = s+1
end
Assign $\sigma^2 = \frac{1}{N_T} \sum (Y_i - X_i \beta_s)' (I + Z_i D_s Z_i')^{-1} (Y_i - X_i \beta_s)$

PLS Algorithm ("lmer")

The penalised least squares ('PLS') algorithm is one of the most commonly used algorithms for estimating linear mixed models. It is employed by 'R's `lmer` package and assumes the following model (which is equivalent to the formulation given in section 1.1, obtained from the spherical transform of the random effects

given in section 2.22):

$$Y = X\beta + Z\Lambda_\theta u + \epsilon$$

(where $u \sim N(0, I)$ and θ is a vector of all non-zero elements of Λ.)

Given this model, PLS focuses on minimizing the following expression:

$$\underset{\beta,\sigma^2,\theta}{\arg\min} ||Y - X\beta - Z\Lambda_\theta u||_2^2 + ||u||_2^2$$

I.e. it minimizes the residual errors, whilst also penalising large values of u as they are less likely. Often, performing a weighted PLS mixed model analysis is also desirable. This involves the following modification to obtain the below objective function, denoted r^2:

$$\underset{\beta,\sigma^2,\theta}{\arg\min}\, r^2(\beta, \theta, u) = \underset{\beta,\sigma^2,\theta}{\arg\min} ||W^{\frac{1}{2}}(Y - X\beta - Z\Lambda_\theta u)||_2^2 + ||u||_2^2$$

Where W is a pre-specified (diagonal) weight matrix, such that $Y \sim N(0, \sigma^2 W^{-1})$. In practice, whether or not a weight matrix is specified is of little consequence to the computation.

The above objective function can be expressed equivalently as:

$$\underset{\beta,\sigma^2,\theta}{\arg\min} \left\| \begin{bmatrix} W^{\frac{1}{2}}Y \\ 0 \end{bmatrix} - \begin{bmatrix} W^{\frac{1}{2}}Z\Lambda_\theta & W^{\frac{1}{2}}X \\ I_q & 0 \end{bmatrix} \begin{bmatrix} u \\ \beta \end{bmatrix} \right\|_2^2$$

By considering the normal equations ($\hat{\beta} = \arg\min ||Y - X\beta|| \implies X'Y = X'X\hat{\beta}$) and the above formulation of the PLS objective function we arrive at the following expression for the MLE's (given θ) of this model:

$$\begin{bmatrix} \Lambda_\theta^T Z^T W Y \\ X^T W Y \end{bmatrix} = \begin{bmatrix} \Lambda_\theta^T Z^T W Z \Lambda_\theta + I & \Lambda_\theta^T Z^T W X \\ X^T W Z \Lambda_\theta & X^T W X \end{bmatrix} \begin{bmatrix} \hat{u}_\theta \\ \hat{\beta}_\theta \end{bmatrix}$$

From here, a cholesky decomposition is applied to the first matrix on the right hand side, in order to obtain:

$$\begin{bmatrix} \Lambda_\theta^T Z^T W Z \Lambda_\theta + I & \Lambda_\theta^T Z^T W X \\ X^T W Z \Lambda_\theta & X^T W X \end{bmatrix} = \begin{bmatrix} L_\theta & 0 \\ R_{ZX}^T & R_X^T \end{bmatrix} \begin{bmatrix} L_\theta^T & R_{ZX} \\ 0 & R_X \end{bmatrix}$$

In practice, the computation of L_θ can be sped up by using a fill-reducing permutation (see the matlab documentation for sparse matrix reordering), due to the sparsity of Z, like so:

$$L_\theta L_\theta^T = P(\Lambda_\theta^T Z^T W Z \Lambda_\theta + I)P^T$$

All of these steps are designed to improve computation speed and efficiency. Below, some brief derivations are provided in order to explain the steps of the

algorithm.

Derivations

Statement 1:

$$L_\theta c_u = P\Lambda_\theta^T Z^T WY$$

Where $c_u := L_\theta^T P\hat{u}_\theta + R_{ZX}\hat{\beta}_\theta$.

Proof:

From the previous section we have that:

$$\begin{bmatrix} \Lambda_\theta^T Z^T WY \\ X^T WY \end{bmatrix} = \begin{bmatrix} \Lambda_\theta^T Z^T WZ\Lambda_\theta + I & \Lambda_\theta^T Z^T WX \\ X^T WZ\Lambda_\theta & X^T WX \end{bmatrix} \begin{bmatrix} \hat{u}_\theta \\ \hat{\beta}_\theta \end{bmatrix} = \begin{bmatrix} P^T L_\theta & 0 \\ R_{ZX}^T & R_X^T \end{bmatrix} \begin{bmatrix} L_\theta^T P & R_{ZX} \\ 0 & R_X \end{bmatrix} \begin{bmatrix} \hat{u}_\theta \\ \hat{\beta}_\theta \end{bmatrix}$$

By multiplying out the right hand side we obtain:

$$\begin{bmatrix} \Lambda_\theta^T Z^T WY \\ X^T WY \end{bmatrix} = \begin{bmatrix} P^T L_\theta L_\theta^T P & P^T L_\theta R_{ZX} \\ R_{ZX}^T L_\theta^T P & R_{ZX}^T R_{ZX} + R_X^T R_X \end{bmatrix} \begin{bmatrix} \hat{u}_\theta \\ \hat{\beta}_\theta \end{bmatrix}$$

By comparing the top elements of both sides we obtain:

$$\begin{aligned} \Lambda_\theta^T Z^T WY &= P^T L_\theta L_\theta^T P\hat{u}_\theta + P^T L_\theta R_{ZX}\hat{\beta} \\ \implies \Lambda_\theta^T Z^T WY &= P^T L_\theta (L_\theta^T P\hat{u}_\theta + R_{ZX}\hat{\beta}) \\ \implies \Lambda_\theta^T Z^T WY &= P^T L_\theta c_u \\ \implies P\Lambda_\theta^T Z^T WY &= L_\theta c_u \end{aligned}$$

Statement 2:

$$L_\theta R_{ZX} = P\Lambda_\theta^T Z^T WX$$

Proof:

To prove statement 2, start from the cholesky decomposition, given as:

$$\begin{bmatrix} \Lambda_\theta^T Z^T WZ\Lambda_\theta + I & \Lambda_\theta^T Z^T WX \\ X^T WZ\Lambda_\theta & X^T WX \end{bmatrix} = \begin{bmatrix} P^T L_\theta & 0 \\ R_{ZX}^T & R_X^T \end{bmatrix} \begin{bmatrix} L_\theta^T P & R_{ZX} \\ 0 & R_X \end{bmatrix} = \begin{bmatrix} P^T L_\theta L_\theta^T P & P^T L_\theta R_{ZX} \\ R_{ZX}^T L_\theta^T P & R_{ZX}^T R_{ZX} + R_X^T R_X \end{bmatrix}$$

By comparing the upper right hand elements on both sides, we obtain:

$$\begin{aligned} \Lambda_\theta^T Z^T WX &= P^T L_\theta R_{ZX} \\ \implies P\Lambda_\theta^T Z^T WX &= L_\theta R_{ZX} \end{aligned}$$

Statement 3:

$$R_X^T R_X = X^T W X - R_{ZX}^T R_{ZX}$$

Proof:

To prove statement 3, start again from the cholesky decomposition, given as:

$$\begin{bmatrix} \Lambda_\theta^T Z^T W Z \Lambda_\theta + I & \Lambda_\theta^T Z^T W X \\ X^T W Z \Lambda_\theta & X^T W X \end{bmatrix} = \begin{bmatrix} P^T L_\theta L_\theta^T P & P^T L_\theta R_{ZX} \\ R_{ZX}^T L_\theta^T P & R_{ZX}^T R_{ZX} + R_X^T R_X \end{bmatrix}$$

By equating the bottom right elements we get:

$$\begin{aligned} & R_{ZX}^T R_{ZX} + R_X^T R_X = X^T W X \\ \implies & R_X^T R_X = X^T W X - R_{ZX}^T R_{ZX} \end{aligned}$$

Statement 4:

$$(R_X^T R_X)\hat{\beta}_\theta = X^T W Y - R_{ZX}^T c_u$$

Where $c_u := L_\theta^T P \hat{u}_\theta + R_{ZX}\hat{\beta}_\theta$.

Proof:

To see this, start from the below equation given in the proof of statement 1:

$$\begin{bmatrix} \Lambda_\theta^T Z^T W Y \\ X^T W Y \end{bmatrix} = \begin{bmatrix} P^T L_\theta L_\theta^T P & P^T L_\theta R_{ZX} \\ R_{ZX}^T L_\theta^T P & R_{ZX}^T R_{ZX} + R_X^T R_X \end{bmatrix} \begin{bmatrix} \hat{u}_\theta \\ \hat{\beta}_\theta \end{bmatrix}$$

Equating the bottom elements we obtain:

$$\begin{aligned} & X^T W Y = R_{ZX}^T L_\theta^T P \hat{u}_\theta + (R_{ZX}^T R_{ZX} + R_X^T R_X)\hat{\beta}_\theta \\ \implies & X^T W Y = R_{ZX}^T L_\theta^T P \hat{u}_\theta + R_{ZX}^T R_{ZX}\hat{\beta}_\theta + R_X^T R_X \hat{\beta}_\theta \\ \implies & X^T W Y = R_{ZX}^T (L_\theta^T P \hat{u}_\theta + R_{ZX}\hat{\beta}_\theta) + R_X^T R_X \hat{\beta}_\theta \\ \implies & X^T W Y = R_{ZX}^T c_u + (R_X^T R_X)\hat{\beta}_\theta \\ \implies & X^T W Y - R_{ZX}^T c_u = (R_X^T R_X)\hat{\beta}_\theta \end{aligned}$$

Statement 5:

$$L_\theta^T P \hat{u}_\theta = c_u - R_{ZX}\hat{\beta}_\theta$$

Where $c_u := L_\theta^T P\hat{u}_\theta + R_{ZX}\hat{\beta}_\theta$.

Proof:

This follows directly from the definition of c_u.

Statement 6:

$$r^2(\theta, \beta, u) = r^2(\theta) + ||L_\theta^T(u - \hat{u}_\theta) + R_{ZX}(\beta - \hat{\beta}_\theta)||_2^2 + ||R_X(\beta - \hat{\beta}_\theta)||_2^2$$

Where $r^2(\theta) := r^2(\theta, \hat{\beta}_\theta, \hat{u}_\theta)$.

Proof:

For simplicity of notation, denote the following:

$$M_Y := \begin{bmatrix} W^{\frac{1}{2}}Y \\ 0 \end{bmatrix}, M_X = \begin{bmatrix} W^{\frac{1}{2}}Z\Lambda_\theta & W^{\frac{1}{2}}X \\ I_q & 0 \end{bmatrix},$$

$$P = \begin{bmatrix} u \\ \beta \end{bmatrix}, \hat{P} = \begin{bmatrix} \hat{u}_\theta \\ \hat{\beta}_\theta \end{bmatrix}, M_L = \begin{bmatrix} L_\theta^T & R_{ZX} \\ 0 & R_X \end{bmatrix}$$

By the definition of $r^2(\theta, \beta, u)$;

$$r^2(\theta, \beta, u) = \|M_Y - M_X P\|_2^2$$

By the fact that $\hat{P}$ satisfies the normal equations for $M_Y - M_X P$, it is also true, by construction, that:

$$M_X^T M_X \hat{P} = M_X^T M_Y$$

Also, note that by the definition of M_L via the cholesky decomposition:

$$M_L' M_L = M_X' M_X$$

Now, note that:

$$r^2(\theta, \beta, u) = (M_Y - M_X P)^T (M_Y - M_X P)$$

$$= M_Y^T M_Y - M_Y^T M_X P - P^T M_X^T M_Y + P^T M_X^T M_X P$$

(By expanding)

$$= M_Y^T M_Y - \hat{P}^T M_X^T M_X P - P^T M_X^T M_X \hat{P} + P^T M_X^T M_X P$$

$$\text{(By } M_X^T M_X \hat{P} = M_X^T M_Y\text{)}$$

$$= M_Y^T M_Y - \hat{P}^T M_X^T M_X P - P^T M_X^T M_X \hat{P} + P^T M_X^T M_X P + \hat{P}^T M_X^T M_X \hat{P} - \hat{P}^T M_X^T M_X \hat{P}$$
$$\text{(By adding and subtracting } \hat{P}^T M_X^T M_X \hat{P}\text{)}$$

$$= M_Y^T M_Y + (P - \hat{P})^T M_X^T M_X (P - \hat{P}) - \hat{P}^T M_X^T M_X \hat{P}$$
$$= M_Y^T M_Y + (P - \hat{P})^T M_L^T M_L (P - \hat{P}) - \hat{P}^T M_X^T M_X \hat{P}$$
$$\text{(As } M_X^T M_X = M_L^T M_L\text{)}$$

$$= M_Y^T M_Y + \|M_L(P - \hat{P})\|_2^2 - \hat{P}^T M_X^T M_X \hat{P}$$
$$= \|M_L(P - \hat{P})\|_2^2 + M_Y^T M_Y - \hat{P}^T M_X^T M_X \hat{P} - \hat{P}^T M_X^T M_X \hat{P} + \hat{P}^T M_X^T M_X \hat{P}$$
$$\text{(By adding and subtracting } \hat{P}^T M_X^T M_X \hat{P}\text{)}$$

$$= \|M_L(P - \hat{P})\|_2^2 + M_Y^T M_Y - M_Y^T M_X \hat{P} - \hat{P}^T M_X^T M_Y + \hat{P}^T M_X^T M_X \hat{P}$$
$$\text{(By } M_X^T M_X \hat{P} = M_X^T M_Y\text{)}$$

$$= \|M_L(P - \hat{P})\|_2^2 + (M_Y - M_X \hat{p})^T (M_Y - M_X \hat{p})$$
$$= \|M_L(P - \hat{P})\|_2^2 + r^2(\theta, \hat{\beta}_\theta, \hat{u}_\theta)$$
$$\text{(By the definition of } r^2 \text{ and the definition of } \hat{p}\text{)}$$

$$= \|M_L(P - \hat{P})\|_2^2 + r^2(\theta)$$
$$\text{(By the definition of } r^2(\theta)\text{)}$$

$$= \left\| \begin{bmatrix} L_\theta^T P & R_{ZX} \\ 0 & R_X \end{bmatrix} \begin{bmatrix} (u - \hat{u}_\theta) \\ (\beta - \hat{\beta}_\theta) \end{bmatrix} \right\|_2^2 + r^2(\theta)$$
$$\text{(By the definition of } M_L\text{, } P \text{ and } \hat{P}\text{)}$$

$$= \|L_\theta^T (u - \hat{u}_\theta) + R_{XZ}(\beta - \hat{\beta}_\theta)\|_2^2 + \|R_X(\beta - \hat{\beta}_\theta)\|_2^2 + r^2(\theta)$$
$$\text{(By the definition of } M_L\text{, } P \text{ and } \hat{P}\text{)}$$

Statement 7:

The profiled log-likelihood $\mathcal{L}$ of θ given y is given by:

$$\mathcal{L}(\theta|y) = -\frac{1}{2}\log\left(\frac{|L_\theta|^2}{|W|}\right) - \frac{n}{2}\left[1 + \log\left(\frac{2\pi r^2(\theta)}{n}\right)\right]$$

In addition the θ-profile MLE for σ^2 is given by:

$$\hat{\sigma}_\theta^2 = \frac{r^2(\theta)}{n}$$

Proof:

As $Y|U \sim N(X\beta + Z\Lambda_\theta u, \sigma^2 W^{-1})$ and $U \sim N(0, \sigma^2 I)$, it follows by the multivariate normal PDF that the PDFs are $Y|U$ and U are given by:

$$f_{Y|U}(y|u) = \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left[\frac{-(y - X\beta - Z\Lambda_\theta)^T W (y - X\beta - Z\Lambda_\theta)}{2\sigma^2}\right]$$

$$= \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left[\frac{-\|W^{\frac{1}{2}}(y - X\beta - Z\Lambda_\theta)\|_2^2}{2\sigma^2}\right]$$

$$f_U(u) = \frac{1}{(2\pi\sigma^2)^{\frac{q}{2}}} \exp\left[\frac{-u^T u}{2\sigma^2}\right]$$

$$= \frac{1}{(2\pi\sigma^2)^{\frac{q}{2}}} \exp\left[\frac{-\|u\|_2^2}{2\sigma^2}\right]$$

(Where q is the length of u). By the definition of conditional density it is true that:

$$f_{Y,U}(y,u) = f_{Y|U}(y|u) f_U(u)$$

$$= \left(\frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left[\frac{-\|W^{\frac{1}{2}}(y - X\beta - Z\Lambda_\theta)\|_2^2}{2\sigma^2}\right]\right)\left(\frac{1}{(2\pi\sigma^2)^{\frac{q}{2}}} \exp\left[\frac{-\|u\|_2^2}{2\sigma^2}\right]\right)$$

$$= \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n+q}{2}}} \exp\left[\frac{-\|W^{\frac{1}{2}}(y - X\beta - Z\Lambda_\theta)\|_2^2 - \|u\|_2^2}{2\sigma^2}\right]$$

$$= \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n+q}{2}}} \exp\left[\frac{-r^2(\theta,\beta,u)}{2\sigma^2}\right]$$

By the definition of marginal distribution:

$$f_Y(y) = \int f_{Y,U}(y,u)du$$

$$= \int \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n+q}{2}}} \exp\left[\frac{-r^2(\theta,\beta,u)}{2\sigma^2}\right]du$$

$$= \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n+q}{2}}} \int \exp\left[\frac{-r^2(\theta,\beta,u)}{2\sigma^2}\right]du$$

Therefore, by statement 6:

$$f_Y(y) = \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n+q}{2}}} \int \exp\left[\frac{-r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2 - \|L_\theta^T(u - \hat{u}_\theta) + R_{ZX}(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2}\right]du$$

$$= \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n+q}{2}}} \exp\left[\frac{-r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2}\right] \int \exp\left[\frac{-\|L_\theta^T(u - \hat{u}_\theta) + R_{ZX}(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2}\right]du$$

A change of variables, $v = L_\theta^T(u - \hat{u}_\theta) + R_{ZX}(\beta - \hat{\beta}_\theta)$, gives:

$$f_Y(y) = \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n+q}{2}}} \exp\left[\frac{-r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2}\right] \int \exp\left[\frac{-\|v\|_2^2}{2\sigma^2}\right] |L_\theta|^{-1} dv$$

$$= \frac{|W|^{\frac{1}{2}}|L_\theta|^{-1}}{(2\pi\sigma^2)^{\frac{n+q}{2}}}\exp\left[\frac{-r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2}\right]\int \exp\left[\frac{-\|v\|_2^2}{2\sigma^2}\right]dv$$

$$= \frac{|W|^{\frac{1}{2}}|L_\theta|^{-1}}{(2\pi\sigma^2)^{\frac{n+q}{2}}}\exp\left[\frac{-r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2}\right](2\pi\sigma^2)^{\frac{q}{2}}$$

$$= \frac{|W|^{\frac{1}{2}}|L_\theta|^{-1}}{(2\pi\sigma^2)^{\frac{n}{2}}}\exp\left[\frac{-r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2}\right]$$

Therefore $\mathcal{L}(\theta, \beta, \sigma^2)$ is given by:

$$\mathcal{L}(\theta, \beta, \sigma^2|y) := \log(f_Y(y))$$

$$= \log\left(\frac{|W|^{\frac{1}{2}}|L_\theta|^{-1}}{(2\pi\sigma^2)^{\frac{n}{2}}}\exp\left[\frac{-r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2}\right]\right)$$

$$= -\frac{1}{2}\log\left(\frac{|L_\theta|^2}{|W|}\right) - \frac{n}{2}\log(2\pi\sigma^2) - \frac{r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2}$$

This can be profiled with respect to β easily as it only enters through the final term which is zero if $\beta = \hat{\beta}_\theta$. This gives:

$$\mathcal{L}(\theta, \sigma^2|y) = -\frac{1}{2}\log\left(\frac{|L_\theta|^2}{|W|}\right) - \frac{n}{2}\log(2\pi\sigma^2) - \frac{r^2(\theta)}{2\sigma^2}$$

To profile out σ^2 we can differentiate this the above with respect to σ^2 to obtain the following:

$$0 = \frac{n}{\hat{\sigma}_\theta^2} - \frac{r^2(\theta)}{\hat{\sigma}_\theta^4}$$

This gives the profiled MLE for σ^2 as:

$$\hat{\sigma}_\theta^2 = \frac{r^2(\theta)}{n}$$

Substituting this back into $\mathcal{L}(\theta, \sigma^2|y)$ gives:

$$\mathcal{L}(\theta|y) = -\frac{1}{2}\log\left(\frac{|L_\theta|^2}{|W|}\right) - \frac{n}{2}\left[1 + \log\left(\frac{2\pi r^2(\theta)}{n}\right)\right]$$

Statement 8:

The profiled restricted log-likelihood $\mathcal{L}_R$ of θ given y is given by:

$$\mathcal{L}_R(\theta|y) = -\frac{1}{2}\log\left(\frac{|L_\theta|^2|R_X|^2}{|W|}\right) - \frac{(n-p)}{2}\left[1 + \log\left(\frac{2\pi r^2(\theta)}{n-p}\right)\right]$$

In addition the θ-profile REML estimate for σ^2 is given by:

$$\hat{\sigma}_\theta^2 = \frac{r^2(\theta)}{n-p}$$

Proof:

By the definition of REML, $\mathcal{L}_R$ is given by:

$$\mathcal{L}_R(\theta, \sigma^2) = \log\Big(\int \exp(\mathcal{L}(\theta, \beta, \sigma^2)) d\beta \Big)$$

$$= \log\Big(\int f_Y(y) d\beta \Big)$$

By the previous:

$$\int f_Y(y) d\beta = \int \frac{|W|^{\frac{1}{2}} |L_\theta|^{-1}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\Big[\frac{-r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2} \Big] d\beta$$

$$= \frac{|W|^{\frac{1}{2}} |L_\theta|^{-1}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\Big[\frac{-r^2(\theta)}{2\sigma^2} \Big] \int \exp\Big[\frac{-\|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2} \Big] d\beta$$

Using a change of variables, $v = R_X(\beta - \hat{\beta}_\theta)$, we obtain:

$$\int f_Y(y) d\beta = \int \frac{|W|^{\frac{1}{2}} |L_\theta|^{-1}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\Big[\frac{-r^2(\theta) - \|R_X(\beta - \hat{\beta}_\theta)\|_2^2}{2\sigma^2} \Big] d\beta$$

$$= \frac{|W|^{\frac{1}{2}} |L_\theta|^{-1}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\Big[\frac{-r^2(\theta)}{2\sigma^2} \Big] \int \exp\Big[\frac{-\|v\|_2^2}{2\sigma^2} \Big] |R_X|^{-1} dv$$

$$= \frac{|W|^{\frac{1}{2}} |L_\theta|^{-1} |R_X|^{-1}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\Big[\frac{-r^2(\theta)}{2\sigma^2} \Big] \int \exp\Big[\frac{-\|v\|_2^2}{2\sigma^2} \Big] dv$$

$$= \frac{|W|^{\frac{1}{2}} |L_\theta|^{-1} |R_X|^{-1}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\Big[\frac{-r^2(\theta)}{2\sigma^2} \Big] (2\pi\sigma^2)^{\frac{q}{2}}$$

$$= \frac{|W|^{\frac{1}{2}} |L_\theta|^{-1} |R_X|^{-1}}{(2\pi\sigma^2)^{\frac{n-q}{2}}} \exp\Big[\frac{-r^2(\theta)}{2\sigma^2} \Big]$$

This gives:

$$\mathcal{L}_R(\theta, \sigma^2) = \log\Big(\int \exp(\mathcal{L}(\theta, \beta, \sigma^2)) d\beta \Big)$$

$$= -\frac{1}{2} \log\Big(\frac{|L_\theta|^2 |R_X|^2}{|W|} \Big) - \frac{(n-p)}{2} \log(2\pi\sigma^2) - \frac{r^2(\theta)}{2\sigma^2}$$

Differentiating this with respect to σ^2 and setting the result equal to zero yields the profiled REML estimate for σ^2, $\hat{\sigma}^2_\theta$, given as:

$$\hat{\sigma}^2_\theta = \frac{r^2(\theta)}{n-p}$$

Substituting this estimator back into $\mathcal{L}_R(\theta, \sigma^2)$ to obtain the θ-profiled REML criterion, $\mathcal{L}_R(\theta) = \mathcal{L}_R(\theta, \hat{\sigma}^2_\theta)$, yields:

$$\mathcal{L}_R(\theta) = -\frac{1}{2}\log\left(\frac{|L_\theta|^2|R_X|^2}{|W|}\right) - \frac{(n-p)}{2}\left[1 + \log\left(\frac{2\pi r^2(\theta)}{n-p}\right)\right]$$

Algorithm

The ML version of PLS is given below:

Input: σ_0^2 (Initial fixed effects variance estimate)
vech(D_0) (Initial random effects variance estimates)
X (random effects covariates matrix)
Z (fixed effects covariates matrix)
Y (response matrix)
W (weights matrix)
N_T (total number of observations)

Output: β (Final Beta estimate)
σ^2 (Final fixed effects variance estimate)
vech(D) (Final random effects variance estimates)

Assign $\Lambda_0 = \text{CholeskyDecomp}(\frac{D_0}{\sigma_0^2})$

Assign $\theta_0 = \text{VecNonZero}(\Lambda_0)$

Function $(\theta, \mathcal{L}(\theta), \beta) =$ **PLS**$(\theta)\{$

  Assign $\Lambda = \text{mapping}(\theta)$

  Assign $P = \text{SparsePerm}(Z\Lambda)$

  Assign $c_u = \text{solve}(L, P\Lambda^T Z^T WY)$

  Assign $R_{ZX} = \text{solve}(L, P\Lambda^T Z^T WX)$

  Assign $R_X^T R_X = X^T WX - R_{ZX}^T R_{ZX}$

  Assign $\beta = \text{solve}(R_X^T R_X, X^T WY - R_{ZX}^T c_u)$

  Assign $u = \text{solve}(L^T P, c_u - R_{ZX}\beta)$

  Assign $b = \Lambda u$

  Assign $\hat{y} = X\beta + Z^T b$

  Assign $\sqrt{WRSS} = W^{\frac{1}{2}}(y - \hat{y})$

  Assign $\text{Pen}WSS = \Sigma(\sqrt{WRSS})^2 + \Sigma u^2$

  Assign $\log(|L|^2) = 2\log(|L|)$

  Assign $\mathcal{L}(\theta) = -\frac{1}{2}(\log(|L|^2) + N_T(1 + \log(2\pi * \text{Pen}WSS) - \log(N_T)))$

$\}$

Assign $(\theta, \mathcal{L}(\theta), \beta) =$NonLinOptimise(**PLS**$(\theta)$, initial$= \theta_0$)

Assign $\Lambda = \text{mapping}(\theta)$

Assign $\sigma^2 = \frac{r^2(\Lambda)}{N_T}$

Assign $D = \sigma^2 \Lambda^T \Lambda$

In the above, line 6 is a consequence of statement 1, line 7 is a consequence of statement 2, line 8 is a consequence of statement 3, line 9 is a consequence of statement 4, line 10 is a consequence of statement 5 and line 16 is the likelihood given by statement 7.

Fisher Scoring (ReML)

The ReML version of FS is given below:

Input: β_0 (Initial Beta estimate)
σ_0^2 (Initial fixed effects variance estimate)
vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β_s (Final Beta estimate)
σ_s^2 (Final fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$
while *not converged* **do**
  for *all* i **do**
    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$
    Assign $e_{s,i} = y_i - X_i \beta_s$
    Assign $R_{s,i} = Z_i' V_{s,i}^{-1} Z_i$
  end
  Assign $\left[\frac{\delta l_R}{\delta \beta}\right]_s = \sigma_s^{-2} \sum X_i' V_{s,i}^{-1} e_{s,i}$
  Assign $\left[\frac{\delta l_R}{\delta \sigma^2}\right]_s = -\frac{1}{2}(N_T - m)\sigma_s^{-2} + \frac{1}{2}\sigma_s^{-4} \sum e_{s,i}' V_{s,i}^{-1} e_{s,i}$
  Assign $\left[\frac{\delta l_R}{\delta D}\right]_s = -\frac{1}{2} \sum [R_{s,i} - \sigma_s^{-2} Z_i' V_{s,i}^{-1} e_{s,i} e_{s,i}' V_{s,i}^{-1} Z_i$
    $-Z_i' V_{s,i}^{-1} X_i (\sum X_j' V_{s,j}^{-1} X_j)^{-1} X_i' V_{s,i}^{-1} Z_i)]$
  Assign $\left[\frac{\delta l_R}{\delta \theta}\right]_s = \left(\left[\frac{\delta l_R}{\delta \beta}\right]_s', \left[\frac{\delta l_R}{\delta \sigma^2}\right]_s', \mathrm{vech}\left(\left[\frac{\delta l_R}{\delta D}\right]_s\right)'\right)'$
  Assign $\mathcal{I}_{s,11} = \sigma_s^{-2} \sum X_i' V_{s,i}^{-1} X_i$
  Assign $\mathcal{I}_{s,22} = \frac{1}{2} N_T \sigma_s^{-4}$
  Assign $\mathcal{I}_{s,33} = \frac{1}{2} \sum R_{s,i} \otimes R_{s,i}$
  Assign $\mathcal{I}_{s,23} = \frac{1}{2}\sigma_s^{-2} \sum \mathrm{vec}(R_i)'$
  Assign $\mathcal{I}_s = \begin{bmatrix} \mathcal{I}_{s,11} & 0 & 0 \\ 0 & \mathcal{I}_{s,22} & \mathcal{I}_{s,23} \\ 0 & \mathcal{I}_{s,23}' & \mathcal{I}_{s,33} \end{bmatrix}$
  Assign $l_{Rs} = -\frac{1}{2}\Big\{(N_T - m)\ln(\sigma_s^2) + \ln\big|\sum X_i' V_{s,i}^{-1} X + i\big| + \sum \big[\ln|V_{s,i}|$
    $+\sigma_s^{-2} e_{s,i}' V_{s,i}^{-1} e_{s,i}\big]\Big\}$
  if $l_{Rs} > l_{R(s-1)}$ **then**
    Assign $\lambda = 1$
  else
    Assign $\lambda = \lambda/2$
  end
  Assign $[\beta_{s+1}', {\sigma_{s+1}^2}', \mathrm{vec}(D_{s+1})']' = [\beta_s', \sigma_s^{2\prime}, \mathrm{vec}(D_s)']' + \lambda \mathcal{I}_s^{-1} \left[\frac{\delta l_R}{\delta \theta}\right]_s$
  Assign s = s+1
end

Expectation Maximisation (ReML)

The expectation maximisation algorithm for ReML is given below:

Input: β_0 (Initial Beta estimate)
σ_0^2 (Initial fixed effects variance estimate)
vech(D_0) (Initial random effects variance estimates)
X_i (random effects covariates matrix, for all i)
Z_i (fixed effects covariates matrix, for all i)
Y_i (response matrix, for all i)
N_T (total number of observations)

Output: β_s (Final Beta estimate)
σ_s^2 (Final fixed effects variance estimate)
vech(D_s) (Final random effects variance estimates)

Assign s = 0, $l_0 = 0$
while *not converged* **do**
  for *all i* **do**
    Assign $V_{s,i} = Z_i D_s Z_i' + I_{n_i}$
    Assign $e_{s,i} = y_i - X_i \beta_s$
  end
  Assign $\sigma_{s+1}^2 = \sigma_s^2 - 1 + \frac{1}{\sigma_s^2 (N_T - m)} \sum e_{s,i}' V_{s,i}^{-1} e_{s,i}$
  Assign $D_{s+1} = \sum D_s [D_s^{-1} - Z_i V_{s,i}^{-1} Z_i] D_s +$
    $\sigma_s^{-2} \sum D_s Z_i' V_{s,i}^{-1} [e_{s,i} e_{s,i}' - X_i (\sum X_j' V_{s,j}^{-1} X_j)^{-1} X_i'] V_{s,i}^{-1} Z_i D_s$
  Assign $\beta_{s+1} = \left[X_i' V_{s,i}^{-1} X_i\right]^{-1} X_i' V_{s,i}^{-1} y_i$
  Assign s = s+1
end

PLS (ReML)

Algorithm

The ReML version of PLS is given below:

Input: σ_0^2 (Initial fixed effects variance estimate)
vech(D_0) (Initial random effects variance estimates)
X (random effects covariates matrix)
Z (fixed effects covariates matrix)
Y (response matrix)
W (weights matrix)
N_T (total number of observations)

Output: β (Final Beta estimate)
σ^2 (Final fixed effects variance estimate)
vech(D) (Final random effects variance estimates)

Assign $\Lambda_0 = \text{CholeskyDecomp}(\frac{D_0}{\sigma_0^2})$

Assign $\theta_0 = \text{VecNonZero}(\Lambda_0)$

Function $(\theta, \mathcal{L}_R(\theta), \beta) =$ **PLS**$(\theta)\{$

  Assign Λ=mapping(θ)

  Assign $P = \text{SparsePerm}(Z\Lambda)$

  Assign $c_u = \text{solve}(L, P\Lambda^T Z^T WY)$

  Assign $R_{ZX} = \text{solve}(L, P\Lambda^T Z^T WX)$

  Assign $R_X^T R_X = X^T WX - R_{ZX}^T R_{ZX}$

  Assign $\beta = \text{solve}(R_X^T R_X, X^T WY - R_{ZX}^T c_u)$

  Assign $u = \text{solve}(L^T P, c_u - R_{ZX}\beta)$

  Assign $b = \Lambda u$

  Assign $\hat{y} = X\beta + Z^T b$

  Assign $\sqrt{WRSS} = W^{\frac{1}{2}}(y - \hat{y})$

  Assign $\text{Pen}WSS = \Sigma(\sqrt{WRSS})^2 + \Sigma u^2$

  Assign $\log(|L|^2|R_X|^2) = 2\log(|L|) + 2\log(|R_X|)$

  Assign $\mathcal{L}_R(\theta) = -\frac{1}{2}(\log(|L|^2|R_X|^2) + (N_T - p)(1 + \log(2\pi * \text{Pen}WSS)$
  $-\log(N_T - p)))$

}

Assign $(\theta, \mathcal{L}_R(\theta), \beta)$=NonLinOptimise(**PLS**($\theta$), initial= θ_0)

Assign Λ=mapping(θ)

Assign $\sigma^2 = \frac{r^2(\Lambda)}{N_T - p}$

Assign $D = \sigma^2 \Lambda^T \Lambda$

In the above, line 6 is a consequence of statement 1, line 7 is a consequence of statement 2, line 8 is a consequence of statement 3, line 9 is a consequence of statement 4, line 10 is a consequence of statement 5 and line 16 is the likelihood given by statement 8. All statements can be found in the section of this document for the ML version of this algorithm.

Starting points

Recommended techniques for ensuring non negative definite D

A problem often encountered by the Newton-Raphson and Fisher scoring algorithms (But not expectation maximisation or Fixed point, so long as the starting estimate is positive definite) algorithms is that likelihood maximization for D is performed over the set:

$$C_1 = \{D | I + Z'DZ \text{ is non-negative definite}\}$$

Whereas the problem we are concerned with is actually the (further) constrained maximisation over:

$$C_2 = \{D | D \text{ is non-negative definite}\}$$

In other words, the maximisation algorithm may estimate D as a matrix which is not non-negative definite, which is a problem as D is a covariance matrix and must therefore be non-negative definite. To overcome this issue, 3 strategies are given:

Strategy 1: Allow D_s to be negative definite, project it into C_2 after maximization

The most simple strategy for ensuring the final estimator of D is non-negative definite is to let each iteration D_s be negative definite and then project the final result into C_2 to ensure D is positive definite. This can be done quickly by using the following method:

- Step 1: Maximise D_s via the previous algorithms mentioned to obtain an estimator $\hat{D}$.
- Step 2: Find the eigenvalue decomposition of $\hat{D}$, i.e. $\hat{D} = P\Lambda P'$ where P is a matrix of eigenvectors and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots \lambda_k)$ is a diagonal matrix of eigenvalues.
- Step 3: Define $\Lambda_+ = \text{diag}(\max(\lambda_1, 0), \max(\lambda_2, 0), \ldots \max(\lambda_k, 0))$.
- Step 4: Output $\hat{D}_+ = P\Lambda_+ P'$; this is the projection of $\hat{D}$ into C_2.

The method has a major benefit in that it does not affect maximisation time. If this matrix is now singular, then it is likely that the model is overspecified and some random effects may need to be removed.

Strategy 2: Force D_s to be non-negative definite in all iterations

Strategy 3: Reparameterise D_s using the Cholesky decomposition

Random Effects

Diagnostic tools correlation as well

Inference Statistics

1.5 Criterion for MLE exsitence

p74 Demidenko

1.6 Criterion for positive definiteness of D

Sometimes ML/ReML maximization will fail because the model has been incorrectly specified due to the random effects truly having covariance matrix zero. A simple check to see whether D is actually positive can save a lot of time and be performed relatively quickly. This check is provided by the following lemma:

Lemma: Let $\hat{\beta}_{OLS} = (X'X)^{-1}X'Y$ be the standard OLS estimator. Let $\hat{e_i} = y - \hat{\beta}_{OLS}$ be the OLS residual vector and $\hat{\sigma}^2_{OLS} = \sum \frac{||\hat{e}_i||^2}{N_T}$ denote the OLS variance. If the $(qbyq)$ matrix:

$$\sum Z_i' \hat{e}_i \hat{e}_i' Z_i - \hat{\sigma}^2_{OLS} \sum Z_i' Z_i$$

Is non-zero, non-negative efinite then the MLE of D is a non-zero matrix.

2 Computational Efficiency tricks for distributed computing

This chapter is a summary/documentation of all the interesting and useful tricks that I have read about that may be useful to consider when making the distributed algorithms for Mixed Models.

2.1 Distributed Calculation for Linear Models

Define X and Y as such:

$$X = \begin{bmatrix} X_{:,1} \\ X_{:,2} \\ \vdots \\ X_{:,n_K} \end{bmatrix}, Y = \begin{bmatrix} Y_{:,1} \\ Y_{:,2} \\ \vdots \\ Y_{:,n_K} \end{bmatrix}$$

The regression model is as follows:

$$Y = X\beta + \epsilon$$

For a standard regression problem we wish to compute:

$$\hat{\beta} = (X'X)^{-1}X'Y$$

$X'X$ and $X'Y$ can be computed as such:

$$X'X = [X'_{:,1}, X'_{:,2}, \ldots, X'_{:,n_K}] \begin{bmatrix} X_{:,1} \\ X_{:,2} \\ \vdots \\ X_{:,n_K} \end{bmatrix} = \sum_{k=1}^{n_K} X'_{:,k} X_{:,k}$$

$$X'Y = [X'_{:,1}, X'_{:,2}, \ldots, X'_{:,n_K}] \begin{bmatrix} Y_{:,1} \\ Y_{:,2} \\ \vdots \\ Y_{:,n_K} \end{bmatrix} = \sum_{k=1}^{n_K} X'_{:,k} Y_{:,k}$$

Therefore, to compute $\hat{\beta}$ in a distributed manor, each site k computes $X'_{:,k}X_{:,k}$ and $X'_{:,k}Y_{:,k}$, and then sends it to the central node. The central node will then calculate:

$$\hat{\beta} = \big(\sum_{k=1}^{n_K} X'_{:,k} X_{:,k}\big)^{-1} \big(\sum_{k=1}^{n_K} X'_{:,k} Y_{:,k}\big)$$

In addition, the variance estimator can be calculated in the same pass by noting first that the residuals for this regression are given by:

$e = Y - X\hat{\beta}$

$$= (I - X(X'X)^{-1}X')Y \qquad \text{(As } \hat{\beta} = (X'X)^{-1}X'Y\text{)}$$

$$= MY \qquad \text{(Defining } M = I - X(X'X)^{-1}X'\text{)}$$

Where M is symmetric (which can be seen from definition) and idempotent (see below).

$$M'M = (I - X(X'X)^{-1}X')'(I - X(X'X)^{-1}X')$$

$$= (I - 2X(X'X)^{-1}X' + X(X'X)^{-1}X'X(X'X)^{-1}X')$$

$$= (I - X(X'X)^{-1}X')$$

$$= M$$

Therefore;

$$e'e = y'M'My$$

$$= y'My = y'y - y'X(X'X)^{-1}X'y$$

$$= y'y - y'X(X'X)^{-1}X'X(X'X)^{-1}y$$

$$= y'y - (X\hat{\beta})'(X\hat{\beta})$$

$$= \sum y_i'y_i - \hat{\beta}' \sum (X_i'X_i)\hat{\beta}$$

As a result of this, each site, k, can compute $y_i'y_i$ and $X_i'X_i$ and return the result to the central node in order for the central node to obtain $e_i'e_i$ by using the above result. From this the variance estimator, $\hat{s}^2$, can be calculated as follows:

$$\hat{s}^2 = \frac{e'e}{N_s - N_p}$$

2.2 Missingness masking in a NeuroImaging Linear Regression

To account for missing data in a linear model for NeuroImaging, define the matrix M_v, for a specific voxel v, as follows:

$$(M_v)_{j,j'} := \begin{cases} 1, & \text{if } j = j' \text{ and } (Y_{v,:})_j \text{ is not missing} \\ 0, & \text{otherwise} \end{cases}$$

Where the notation $(A)_{i,j}$ represents the element at index (i, j) of matrix A. Note the dimensions of M_v are $n_s \times n_s$, where n_s is the number of scans.

The below is the linear regression model used for modelling Y for a specific voxel, $Y_{v,:}$, against X accounting for any missing data in Y at voxel v:

$$M_v Y_{v,:} = M_v X \beta_v + \epsilon$$

Note that, as $M_v Y_{v,:} = Y_{v,:}$ this simplifies to:

$$Y_{v,:} = M_v X \beta_v + \epsilon$$

Using the usual OLS, this gives the estimator for β_v as the following:

$$\hat{\beta}_v = (X' M_v' M_v X)^{-1} X' M_v' Y_{v,:} = (X' M_v X)^{-1} X' Y_{v,:}$$

(as M_v is symmetric and idempotent and $M_v Y_{v,:} = Y_{v,:}$)

Therefore, the problem is now to compute $X' M_v X$ and $X' Y_{v,:}$, across all sites, for each voxel v.

The latter matrix, $X' Y_{v,:}$, is computed in the same manner as in the previous section. Each site, site k, individually calculates $X'_{:,k} Y_{v,k}$ and then the central node simply computes the sum over all sites.

For the former matrix, $X' M_v X$, each site, site k, at each voxel, voxel v, must calculate $X'_{:,k} M_{v,k} X_{:,k}$. Where $M_{v,k}$ is the analogue of M_v but only containing information about scans at one specific site, site k.

$$(M_{v,k})_{j,j'} := \begin{cases} 1, & \text{if } j = j' \text{ and } (Y_{v,k})_j \text{ is not missing} \\ 0, & \text{otherwise} \end{cases}$$

This can be calculated at each site, site k, via doing the following:

$$X'_{:,k} M_{v,k} X_{:,k} = \sum_{i=1}^{n_{S,k}} (X'_{:,k})_i (M_{v,k})_{i,i} (X_{:,k})_i$$

Where $(X_{:,k})_i$ is the i^{th} row of $X_{:,k}$.

Each site then sends, for each voxel v, the matrix $X'_{:,k} M_{v,k} X_{:,k}$ to the central node. The final matrix $X' M_v X$, for each voxel v, can then be calculated by the central node by doing the following:

$$X' M_v X = \sum_{k=1}^{n_K} X'_{:,k} M_{v,k} X_{:,k}$$

Implementation tricks:

There are several computational tricks which can be employed in this setup to make code both quicker and more memory efficient. These are listed below:

- In practice, $X'_{:,k}M_{v,k}X_{:,k}$, does not likely need be recorded for every voxel v as the number of distinct observed values of $X'_{:,k}M_{v,k}X_{:,k}$ is likely to be much less than the number of voxels (n_v). i.e.

$$|\{X'_{:,k}M_{v,k}X_{:,k}|v \in \{1, ...n_V\}\}| << n_V$$

  As a result we only need record and index each distinct value of $X'_{:,k}M_{v,k}X_{:,k}$ we observe and then, for each voxel, the index recording the the value of $X'_{:,k}M_{v,k}X_{:,k}$ observed at v. This is noticably more memory efficient.
- In practice, if very few values are missing at a given voxel, instead of calculating

Pseudocode:

This section uses the following abbreviations:

- **CEN:** Central node/processor.
- **ns[k]:** Number of scans at site k.

Input: Each site k has access to its own design matrix X_k and its own dependent variables Y_k.

Output: Spatially varying $\hat{\beta}$ and sum of $e'e$ maps.

```
CEN sets sumXX = zeromatrix(dimensions=[np,np])
CEN sets sumXY = zeromatrix(dimensions=[np,nv])
CEN sets sumYY = zeromatrix(dimensions=[np,nv])
for each site k do
    site k computes XXk = product(transpose(Xk),Xk)
    site k computes XYk = product(transpose(Xk),Yk)
    site k computes YYk = product(transpose(Yk),Yk)
    site k sends XXk, XYk, YYk to CEN
    CEN updates sumXX = XXk + sumXX
    CEN updates sumXY = XYk + sumXY
    CEN updates sumYY = YYk + sumYY
end
CEN calculates Beta = product(inverse(sumXX), sumXY)
CEN calculates SSR = sumYY - product(transpose(Beta), inverse(sumXX),
  Beta)
return Beta, SSR
```

2.3 Condition numbers

A condition number of a matrix A is a measure which takes the following form:

$\kappa(A) = ||A||\ ||A^{-1}||$

Where $||\cdot||$ is usually taken to be the spectral ∞-norm, the spectral 2-norm or the spectral 1-norm, each of which is of the form below:

$||A||_p = \sup_{x \neq 0} \frac{|Ax|_p}{|x|_p}$

Where $|\cdot|$ is the corresponding vector norm. The condition number with respect to $||\cdot||_p$ is often denoted $\kappa_p(\cdot)$. The condition number is often used as a rule of thumb for measuring how ill-conditioned a matrix is, with large condition numbers meaning a matrix is more ill-conditioned and small condition numbers meaning a matrix is less ill-conditioned.

A standard Heuristic for testing how well-conditioned a matrix A is is that if the condition number of A, $\kappa(A)$, is approximately 10^k, then approximately k digits of accuracy may be lost during computation, on top of what would already be expected due to general numeric inefficiency caused by the arithmetic methods themselves.

An intuitive way of seeing why this is the case is to consider the linear system $Ax = b$. In this case the condition number can be viewed as the maximum ratio of the relative error in x ($\frac{||A^{-1}e||}{||A^{-1}b||}$) to the relative error in b ($\frac{||e||}{||b||}$). This is proved below:

$$\begin{aligned}
\max_{e,b\neq 0}\left\{\frac{\frac{||A^{-1}e||}{||A^{-1}b||}}{\frac{||e||}{||b||}}\right\} &= \max_{e,b\neq 0}\left\{\frac{||A^{-1}e||}{||e||}\frac{||b||}{||A^{-1}b||}\right\} \\
&= \max_{e\neq 0}\left\{\frac{||A^{-1}e||}{||e||}\right\}\max_{b\neq 0}\left\{\frac{||b||}{||A^{-1}b||}\right\} \\
&= \max_{e\neq 0}\left\{\frac{||A^{-1}e||}{||e||}\right\}\max_{x\neq 0}\left\{\frac{||Ax||}{||x||}\right\} \\
&= ||A^{-1}||\ ||A|| \\
&= \kappa(A)
\end{aligned}$$

The condition number, perhaps unsurprisingly, is therefore a good metric in determining whether the matrix A needs any pre-conditioning.

In R, the condition number is typically estimated with the functions 'kappa' or 'rcond' (although 'rcond' actually gives the reciprocal condition number, i.e. $\frac{1}{\kappa(A)}$). As both these functions use the same underlying methods, the following section briefly details how R performs condition number estimation in the function 'kappa'.

kappa condition estimation The function 'kappa' in R uses 3 different methods to estimate the condition number of a matrix.

Calculation of the 2-norm condition number using the svd The first method is only used in the case that the user wants to calculate the condition number using

the 2-norm and makes use of the following lemma.

Lemma: The condition number based on the 2-norm of a matrix A is given by the largest singular value of A divided by the smallest singular value of A, where the singular values are defined as the diagonal elements of Σ in the SVD decomposition $A = U'$. The largest and smallest singular values of A are commonly denoted $\sigma_{max}(A)$ and $\sigma_{min}(A)$ respectively. i.e.

$\kappa_2(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}$

Proof: It is true that for any orthogonal matrices Z and K, $||ZAK||_2 = ||A||_2$ (this can be seen from the definition of $||\cdot||_2$ by considering the vector norms which are unaffected by unit length transforms). Therefore, it follows that $||A||_2 = ||U'\Sigma V||_2 = ||\Sigma||_2$ (as U and V are orthogonal).

Following this, using the Lagrange multiplier, it can be seen that $||A||_1 = ||\Sigma||_2$ is $\sigma_{max}(A)$. Similarly, $||A^{-1}||_2 = ||U'\Sigma^{-1}V||_2 = ||\Sigma^{-1}||_2$ and, therefore, $||A^{-1}||_2 = \frac{1}{\sigma_{min}(A)}$. The result then follows.

Using this lemma, the 2-norm condition number is calculated simply and exactly in R using the inbuilt SVD functions.

Estimation of the ∞-norm or 1-norm condition number using LINPACK

If the user requests to calculate either the ∞-norm or the 1-norm of a matrix A in R, the code will estimate the result by first calculating the QR decomposition of A and then, assuming that $\kappa_{1(\infty)}(A) \approx \kappa_{1(\infty)}(R)$, finding the condition number of R.

Any algorithm for calculating κ_1 can be modified to calculate κ_∞ very easily by noting the fact that:

Estimation of the ∞-norm or 1-norm condition number using LAPACK

2.4 Alternative formulations of $\hat{\beta}_{OLS}$

Linear regression problems take the form below:

$Y = X\beta + \epsilon$

Where Y is an $(n \times 1)$ vector of response variables, X is an $(n \times p)$ matrix of explanatory variables and ϵ is an $(n \times 1)$ vector of standard gaussian errors.

The MLE estimator for this model, $\hat{\beta}$ is given by:

$\hat{\beta} = (X'X)^{-1}X'Y$

However, direct computation of this is extremely ill-advised due to the high condition number of $X'X$ which can cause numerical instabilities when inversed.

The QR formulation A more numerically stable approach is to use the QR decomposition.

$X = QR$

Where, if X is $(n \times p)$ then Q is orthogonal with dimensions $(n \times n)$ and R is upper triangular with dimensions $(n \times p)$. A common variant of this, used for tall, thin matrices (i.e. where $n >> p$) is the *"thin QR factorization"* where Q is instead $(n \times p)$ with orthogonal columns and R is still upper triangular but instead $(p \times p)$. In both cases the columns of Q are also unit vectors (orthonormal).

For the rest of this document it is assumed "QR" refers to the thin QR decomposition.

The reason the QR decomposition gives a more numerically stable way of computing the OLS estimator for linear regression is as follows:

$$\begin{aligned}
\hat{\beta} = (X'X)^{-1}X'Y &\implies X'X\hat{\beta} = X'Y \\
&\implies (QR)'(QR)\hat{\beta} = (QR)'Y && \text{(By the QR decomposition)} \\
&\implies R'Q'QR\hat{\beta} = R'Q'Y \\
&\implies R'R\hat{\beta} = R'Q'Y && \text{(As } Q \text{ is orthonormal)} \\
&\implies R\hat{\beta} = Q'Y
\end{aligned}$$

Since R is an upper-triangular matrix, this last equation is easy to solve for $\hat{\beta}$ by backwards substitutions and the numerical instability caused by calculatiing $X'X$ and $X'Y$ is avoided.

If the QR decomposition of X can be computed, theoretically the OLS estimates are much more numerically stable than the more direct computation. This should be checked in practice, however, as the extent of the benefit may be language specific.

The SVD formulation The SVD decomposition of an $(n \times p)$ matrix X is given by:

$$X = UDV^T$$

Where U has orthonormal columns and the same dimension as X, D is a (pp) diagonal matrix of singular values (usually in descending order) and V is a $(p \times p)$ orthogonal matrix.

For the OLS operator, using the SVD decomposition of X we arrive at the following estimator;

$$\begin{aligned}
\hat{\beta} &= (X'X)^{-1}X'Y \\
&= (VDU^TUDV^T)^{-1}VDU^Ty && \text{(By the SVD decomposition)} \\
&= (VD^2V^T)^{-1}VDU^Ty && \text{(As } U \text{ is orthonormal)} \\
&= VD^{-2}V^TVDU^Ty && \text{(By the properties of the SVD decomposition)} \\
&= VD^{-1}U^Ty && \text{(As } V \text{ is orthonormal)}
\end{aligned}$$

As D is the matrix of singular values of X, it has the same condition number as X (which is the square root of the condition number of $X'X$). This means that the inversion in the final line of the above is much more numerically stable than the inversion of $X'X$ which would be performed if directly computing $\hat{\beta}$ from the normal equations. The final result uses a pseudo-inverse for X, $VD^{-1}U^T$. This is the same pseudo-inverse as is used for Matlab's 'pinv'.

2.5 TSQR for OLS

A distributed and privacy protected algorithm for calculating $\hat{\beta}$ is the *"Tall Skinny QR"* algorithm. It works as follows:

Consider X is partitioned into k chunks of rows, so that each chunk of X, X_i, is of size $(\frac{n}{k} \times p)$. Similarly, consider Y is partitioned into row-wise chunks of size $(\frac{n}{k} \times 1)$. i.e.

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{bmatrix}, Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_k \end{bmatrix}$$

Suppose X and Y are shared across sites where each site, site number i only has access to one chunk of X and one chunk of Y, X_i and Y_i respectively. At each site, we can compute the QR decomposition of X_i, i.e. we can compute $Q_i^{(1)}$ and $R_i^{(1)}$ where:

$$X_i = Q_i^{(1)}R_i^{(1)}$$

Where $Q_i^{(1)}$ is has orthonormal columns and dimensions $(\frac{n}{k} \times p)$ and $R_i^{(1)}$ is upper triangular with dimensions $(p \times p)$. The notation $^{(1)}$ here refers to the fact that this is the first stage of QR decompositions we perform.

Note that, it is true that:

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{bmatrix} = \begin{bmatrix} Q_1^{(1)} & 0 & 0 \dots & 0 \\ 0 & Q_2^{(1)} & 0 \dots & 0 \\ \vdots & \vdots & \vdots \ddots & \vdots \\ 0 & 0 & 0 \dots & Q_k^{(1)} \end{bmatrix} \begin{bmatrix} R_1^{(1)} \\ R_2^{(1)} \\ \vdots \\ R_k^{(1)} \end{bmatrix}$$

However, the above is not a QR decomposition as $R^{(1)} = \left[R_1^{(1)'}, R_2^{(1)'}, \dots R_k^{(1)'}\right]'$ is not upper triangular. However, assuming each site has now saved $Q_i^{(1)}$ and $R_i^{(1)}$ and sent only $R_i^{(1)}$ to a central processor, we can now do the following:

The central processor calculates the QR decomposition of $R^{(1)} = \left[R_1^{(1)'}, R_2^{(1)'}, \dots R_k^{(1)'}\right]'$. I.e. it finds $Q^{(2)}$ and $R^{(2)}$ such that:

$R^{(1)} = Q^{(2)} R^{(2)}$

Where $Q^{(2)}$ is $(kp \times p)$ and has orthonormal columns and $R^{(2)}$ is $(p \times p)$ and upper triangular. Note that $R^{(1)}$ has dimensions $(kp \times p)$ and, therefore, this is a relatively small operation.

Note that it is true that:

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_k \end{bmatrix} = \begin{bmatrix} Q_1^{(1)} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & Q_k^{(1)} \end{bmatrix} R^{(1)} = \begin{bmatrix} Q_1^{(1)} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & Q_k^{(1)} \end{bmatrix} Q^{(2)} R^{(2)}$$

By considering a row-wise partition of $Q^{(2)}$ into k chunks, $Q_i^{(2)}$, of dimension $(p \times p)$, i.e. a partition of Q such that:

$$Q^{(2)} = \begin{bmatrix} Q_1^{(2)} \\ Q_2^{(2)} \\ \vdots \\ Q_k^{(2)} \end{bmatrix}$$

We get the following:

$$X = \begin{bmatrix} Q_1^{(1)} & 0 & 0 \dots & 0 \\ 0 & Q_2^{(1)} & 0 \dots & 0 \\ \vdots & \vdots & \vdots \ddots & \vdots \\ 0 & 0 & 0 \dots & Q_k^{(1)} \end{bmatrix} \begin{bmatrix} Q_1^{(2)} \\ Q_2^{(2)} \\ \vdots \\ Q_k^{(2)} \end{bmatrix} R^{(2)} = \begin{bmatrix} Q_1^{(1)} Q_1^{(2)} \\ Q_2^{(1)} Q_2^{(2)} \\ \vdots \\ Q_k^{(1)} Q_k^{(2)} \end{bmatrix} R^{(2)} = QR$$

Where the last step comes from defining $Q = \frac{1}{k}\left[\left(Q_1^{(1)}Q_1^{(2)}\right)', \left(Q_2^{(1)}Q_2^{(2)}\right)', \ldots, \left(Q_k^{(1)}Q_k^{(2)}\right)'\right]'$ and $R = kR^{(2)}$. Note that as, for any $i \in \{1, \ldots, k\}$:

$$\begin{aligned}
(Q_i^{(1)}Q_i^{(2)})'(Q_i^{(1)}Q_i^{(2)}) &= Q_i^{(2)'}Q_i^{(1)'}Q_i^{(1)}Q_i^{(2)} \\
&= Q_i^{(2)'}I_pQ_i^{(2)} \text{ (as } Q_i^{(1)} \text{ has orthonormal columns)} \\
&= Q_i^{(2)'}Q_i^{(2)} \\
&= I_p \text{ (as } Q_i^{(2)} \text{ has orthonormal columns)}
\end{aligned}$$

Therefore, each of the k $(\frac{n}{k} \times p)$ submatrices of kQ has orthonormal columns and, as a result, Q has orthonormal columns. This means Q is an $(n \times p)$ matrix with orthonormal columns and R is a $p \times p$ upper triangular matrix.

Consequently, by the uniqueness of the QR decomposition, Q and R are the QR decomposition of X.

Currently, at this point in the algorithm, each site, site i, has their own $Q_i^{(1)}$ and the central node has only $Q^{(2)}$ and R. The central node now sends $Q_i^{(2)}$ to site i and site i computes the following:

First $Q_i = \frac{1}{k}Q_i^{(1)}Q_i^{(2)}$ is computed. This is the i^{th} $(\frac{n}{k} \times p)$ submatrix of Q.

Next, $Q_i'Y_i$ is computed where Y_i the i^{th} $(\frac{n}{k} \times 1)$ submatrix of Y. This quantity is sent back to the central processor.

The central processor now has $Q_i'Y_i$ for every site i and also has R. Using these matrices it can compute $Q'Y$ as:

$Q'Y = \sum_i Q_i'Y_i$

And then backsolve $R\hat{\beta} = Q'Y$ to obtain $\hat{\beta}$.

To calculate covariances and other results, the matrix $X'X$ can still be obtained as $X'X = R'Q'QR = R'R$.

As the central node only ever sees $R, R^{(1)}$ and $Q'Y$ it cannot derive any information about X or Y; this ensures privacy is protected.

2.6 Diagonalising with orthogonal transforms to reduce computation

Habibs paper

2.7 Spectral decomposition useful results

The spectral/eigen decomposition of square matrix A is given by:

$$A = U\Lambda U^T$$

Where U is an orthogonal matrix with U's eigenvectors as it's columns and Λ is a diagonal matrix of eigenvalues of U. The below are useful results to bear in mind.

$$A^m = U\Lambda^m U^T$$

For any $m \in Z^+$. Similarly:

$$A^{-1} = U\Lambda^{-1} U^T$$

Notably, the inversion of Λ^{-1} is much simpler an inversion than doing A^{-1}. By the same logic, if f is any real valued function with a power series expansion;

$$f'(A) = Uf'(\Lambda)U^T$$

Note, if A is singular however, the matrix of eigenvectors U may be ill conditioned and not recommended for the above operations. In such settings the Jordan canonical form may be more appropriate (see next section).

2.8 Jordan canonical form useful results

The Jordan-Canonical form of a matrix A is the matrix J where;

$A = X\text{diag}(J_1, J_2, \ldots, J_q)X^{-1}$

Where J_i is a Jordan block of size $(n_i \times n_i)$ and has the below form:

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 & 0 & \ldots & 0 \\ 0 & \lambda_i & 1 & 0 & \ldots & 0 \\ 0 & 0 & \lambda_i & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & \lambda_i \end{bmatrix}$$

Where λ_i is the i^{th} eigenvalue of A. The Jordan-Canonical form is useful for computing matrix functions, for example functions such as;

$f(A) = I + A$
$f(A) = (I - \frac{A}{2})^{-1}(I + \frac{A}{2})$
$f(A) = \sum_{n=0}^{\infty} \frac{A^n}{n}$

With corresponding scalar functions:

$f(z) = 1 + z$
$f(z) = (1 - \frac{z}{2})^{-1}(1 + \frac{z}{2})$
$f(z) = \sum_{n=0}^{\infty} \frac{z^n}{n}$

Matrix functions are defined, and can be computed, using the Jordan-Canonical form

as such:

$$f(A) = X\text{diag}(F_1, F_2, \ldots, F_q)X^{-1}$$

Where:

$$F_i = \begin{bmatrix} f(\lambda_i) & f^{(1)}(\lambda_i) & \frac{f^{(2)}(\lambda_i)}{2} & \ldots & \frac{f^{(n_i-1)}(\lambda_i)}{(n_i-1)!} \\ 0 & f(\lambda_i) & f^{(1)}(\lambda_i) & \ldots & \frac{f^{(n_i-2)}(\lambda_i)}{(n_i-2)!} \\ 0 & 0 & f(\lambda_i) & \ldots & \frac{f^{(n_i-3)}(\lambda_i)}{(n_i-3)!} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & f(\lambda_i) \end{bmatrix}$$

To write the Taylor series of $f(A)$ about z_0, let r be such that f is analytic for scalar values in the r-neighbourhood of z_0, i.e.:

$$f(z) = \sum_{k=0}^{\infty} \frac{f^{(k)}(z_0)}{k!}(z - z_0)^k \qquad |z - z_0| < r$$

If $|\lambda_i - z_0| < r$ for all i then it follows that:

$$f(A) = \sum_{k=0}^{\infty} \frac{f^{(k)}(z_0)}{k!}(A - Iz_0)^k$$

This result has been used to prove useful common lemmas such as the following:

$$\exp(A) = \sum_{k=0}^{\infty}(-1)^k \frac{A^k}{k!}$$

$$\log(I - A) = \sum_{k=0}^{\infty}(-1)^k \frac{A^k}{k}$$

$$\sin(A) = \sum_{k=0}^{\infty}(-1)^k \frac{A^{2k+1}}{(2k+1)!}$$

$$\cos(A) = \sum_{k=0}^{\infty}(-1)^k \frac{A^{2k}}{(2k)!}$$

2.9 Diagonal preconditioning

If inverting a square by square matrix, S, we can reduce the condition number of the inversion by defining the diagonal matrix $D = \text{diag}(\sqrt{1/S_{ii}})$.

The inversion S, S^{-1} is equal to $D(DSD)^{-1}D$, however (DSD) has a much lower

condition number than S and, therefore, this operation is much more numerically stable.

If $S = X'X$, then $D = \text{diag}(\sqrt{1/S_{ii}}) = \text{diag}(\sqrt{1/(x_i'x_i)})$, where x_i is the i^{th} column of X. In other words, in this setting, the diagonal preconditioning is equivalent to descaling the columns of X before calculation and then rescaling them after, as in section 2.11.

2.10 Demeaning and Rescaling

The results in these sections describe how to demean and rescale columns of a matrix. The same can easily be done for rows by considering the transpose. It is common to want to do column operations for OLS regression with $(p \times p)$ transforms as these are (often extremely) smaller than the alternative $(n \times n)$ transforms and can be undone after caclulation of OLS estimates such as $\hat{\beta}$ like so:

$$\begin{aligned}
\hat{\beta} &= (X'X)^{-1}X'Y \\
&= ((\tilde{X}DS)'(\tilde{X}DS))^{-1}(\tilde{X}DS)'Y \\
&= (S'D'\tilde{X}'\tilde{X}DS))^{-1}(S'D')\tilde{X}'Y \\
&= (DS)^{-1}(\tilde{X}'\tilde{X})^{-1}(S'D')^{-1}(S'D')\tilde{X}'Y \\
&= (DS)^{-1}(\tilde{X}'\tilde{X})^{-1}\tilde{X}'Y \\
&= (DS)^{-1}\tilde{\beta}
\end{aligned}$$

Where $\tilde{\beta}$ is the OLS estimator based on the demeaned rescaled data matrix $\tilde{X}$.

Demeaning and rescaling with RHS $(p \times p)$ transforms An $(n \times p)$ design matrix X can be rescaled using $(p \times p)$ transformations in the following way. Define the $(p \times p)$ matrix S_X by:

$$S_X = \text{diag}(1/\sqrt{x_i'x_i})$$

Where x_i is the i^{th} column of X. The matrix XS_X corresponds to the matrix X with each column divided by it's L2-norm. In other words, this matrix rescales X.

If X contains an intercept column in the span of it's columns, it can be demeaned using (pp) transformations in the following way. Define the $(p \times p)$ matrix D_X by:

$$D_X = I_p - M\bar{X}$$

Where M is the $(1 \times p)$ solution to $XM = 1_n$, $\bar{X}$ is the $(P \times 1)$ vector of the column means of X, i.e. $\bar{X} = (\text{mean}(x_1), \text{mean}(x_2), \ldots \text{mean}(x_p))$ and I_p is the $(p \times p)$ identity

matrix. It follows that XD_X is the same as the matrix X with each column demeaned. The logic for this is as follows:

$$\begin{aligned} XD_X &= X(I_p - M\bar{X}) \\ &= X - XM\bar{X} \\ &= X - 1_n\bar{X} \qquad (\text{As } XM = 1) \end{aligned}$$

Where the last line is the standard operation for demeaning. It is worth noting, however, that in general we may not wish to demean all columns as demeaning columns whose span includes the intercept may lead to an ill-conditioned matrix. For example, the below shows some examples of matrices pre- and post- demeaning which are rank deficient only post-demeaning;

$$\begin{bmatrix} 1, & x_{21}, & x_{31} \\ 1, & x_{22}, & x_{32} \\ 1, & x_{23}, & x_{33} \\ 1, & x_{24}, & x_{34} \\ 1, & x_{25}, & x_{35} \\ 1, & x_{26}, & x_{36} \end{bmatrix} \rightarrow \begin{bmatrix} 0, & x_{21} - \bar{x}_2, & x_{31} - \bar{x}_3 \\ 0, & x_{22} - \bar{x}_2, & x_{32} - \bar{x}_3 \\ 0, & x_{23} - \bar{x}_2, & x_{33} - \bar{x}_3 \\ 0, & x_{24} - \bar{x}_2, & x_{34} - \bar{x}_3 \\ 0, & x_{25} - \bar{x}_2, & x_{35} - \bar{x}_3 \\ 0, & x_{26} - \bar{x}_2, & x_{36} - \bar{x}_3 \end{bmatrix}$$

$$\begin{bmatrix} 1, & 0, & x_{21}, & x_{31} \\ 1, & 0, & x_{22}, & x_{32} \\ 1, & 0, & x_{23}, & x_{33} \\ 0, & -1, & x_{24}, & x_{34} \\ 0, & -1, & x_{25}, & x_{35} \\ 0, & -1, & x_{26}, & x_{36} \end{bmatrix} \rightarrow \begin{bmatrix} 0.5, & 0.5, & x_{21} - \bar{x}_2, & x_{31} - \bar{x}_3 \\ 0.5, & 0.5, & x_{22} - \bar{x}_2, & x_{32} - \bar{x}_3 \\ 0.5, & 0.5, & x_{23} - \bar{x}_2, & x_{33} - \bar{x}_3 \\ -0.5, & -0.5, & x_{24} - \bar{x}_2, & x_{34} - \bar{x}_3 \\ -0.5, & -0.5, & x_{25} - \bar{x}_2, & x_{35} - \bar{x}_3 \\ -0.5, & -0.5, & x_{26} - \bar{x}_2, & x_{36} - \bar{x}_3 \end{bmatrix}$$

To overcome this issue The following adapted rescaling matrix can be used:

$$D_X = I_p - M(\bar{X} - M')$$

The logic behind this is that:

$$\begin{aligned} XD_X &= X(I_p - M(\bar{X} - M')) \\ &= X - XM(\bar{X} - M') \\ &= X - 1_n\bar{X} - 1_nM' \qquad (\text{As } XM = 1) \end{aligned}$$

As M_i is only non-zero for the subset of columns whose span includes the intercept, these columns are not fully demeaned due to this additional term and therefore no ill-conditioning occurs.

In summary, the below matrix is X rescaled and demeaned:

$$XD_XS_{D_XX}$$

Demeaning with a LHS ($n \times n$) transform The following matrix is an $(n \times n)$ matrix for demeaning an $(n \times p)$ matrix X.

$$D_X = I_n - \frac{1}{n} 1_n 1_n'$$

The logic behind this is:

$$\begin{aligned} D_X X &= (I_n - \tfrac{1}{n} 1_n 1_n')X \\ &= X - \tfrac{1}{n} 1_n 1_n' X \\ &= X - 1_n \bar{X} \end{aligned}$$

Where $\bar{X}$ is the vector of column means of X. To account for intercept columns the following can also be used:

$$D_X = I_n - \frac{1}{n} 1_n 1_n'$$

2.11 Gauss transformations for zeroing

A Gauss transformation of a column vector $v = [v_1, v_2, \dots v_n]'$ is characterised by a vector of the following form:

$$\tau = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \tau_{k+1} \\ \vdots \\ \tau_n \end{bmatrix}$$

Where $\tau_i = \frac{v_i}{v_k}$. It follows from definition and simple matrix multiplication that by defining $M_k = I_n - \tau e_k'$ (where e_k is a column vector with 1 in it's k^{th} position and 0 everywhere else):

$$M_k v = \begin{bmatrix} 1 \dots & 0 & 0 \dots 0 \\ \vdots \ddots & \vdots & \vdots \ddots \vdots \\ 0 \dots & 1 & 0 \dots 0 \\ 0 \dots & -\tau_{k+1} & 1 \dots 0 \\ \vdots \ddots & \vdots & \vdots \ddots \vdots \\ 0 \dots & -\tau_n & 0 \dots 1 \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_k \\ v_{k+1} \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_1 \\ \vdots \\ v_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

It can be seen immediately from the above that, given an arbitrary square matrix A of size $(n \times n)$, a series of $n-1$ Gauss transformations can be generated such that the product $M_{n-1} M_{n-2} \dots M_1 A$ is upper triangular.

If none of the "τ" elements (pivots) of the Gauss transformations are 0, it is true

that $L^{-1} = M_{n-1}M_{n-2}\dots M_1$ can be inverted:

$$L = M_1^{-1}M_2^{-1}\dots M_{n-1}^{-1} = (I_n + \tau^{(1)}e_1')(I_n + \tau^{(2)}e_2')\dots(I_n + \tau^{(n-1)}e_{n-1}')$$

Furthermore, L is lower triangular. This leads to the matrix factorization $A = LU$ where L is lower triangular and U is upper triangular. This is known as the LU factorization. Note: The LU factorization is not always guaranteed to exist and may not exist if 0 valued pivots are encountered in the above.

2.12 Givens rotations for zeroing

A Given's rotation is useful for introducing zeros into selective elements of a matrix. A Given's matrix has the following form:

$$G(i,k,\theta) = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix}$$

Where the $(k,k)^{th}$ and $(i,i)^{th}$ entries of G equal $c = \cos(\theta)$, the $(k,i)^{th}$ entry is $-s = -\sin(\theta)$ and the $(i,k)^{th}$ entry is $s = \sin(\theta)$. Givens rotations are orthogonal by construction and premultiplication by $G(i,k,\theta)$ corresponds to a rotation of θ in the (i,k) plane. One benefit of Givens rotations is that the matrix $G(i,k,\theta)$ can be used to induce a zero in the k^{th} element of a vector in the following manor. Note that if:

$$y = G(i,k,\theta)'x$$

Then:

$$y_j = \begin{cases} cx_i - sx_k & j = i \\ sx_i + cx_k & j = k \\ x_j & j \neq i,k \end{cases}$$

Note that this means that y_k can be forced to be 0 by just setting;

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}} \qquad\qquad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}$$

Thus a Givens matrix can be used to induce a zero in a vector and/or matrix entry.

Computation

Storage-wise a Given's matrix is extremely efficient. This is as $G(i,k,\theta)$ can be characterised completely by i,k and θ and thus requires only these 3 values to be stored (or just 1 value, θ, if the locations i and k are predetermined in the context $G(i,k,\theta)$ is being used). Note also in practice it is more efficient to record c or s rather than θ, noting $c = \sqrt{1-s^2}$, meaning only one value need be recorded. It is also worth noting that, when storing a sequence of givens matrices $G_1G_2 \dots G_n$, it is actually simpler to store each Given's matrix separately instead of the entire product.

Similarly, computation doesn't ever require the entire matrix $G(i,k,\theta)$ to be constructed. Instead to update a matrix A by pre-multiplying by $G(i,k,\theta)'$, i.e. to do $A = G(i,k,\theta)'A$, only the i^{th} and k^{th} rows need changing, which can be done in the following manner:

$$A_{([i,j],:)} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}' A_{([i,j],:)}$$

A small computational point worth noting is that in practice c is not computed as $\frac{x_i}{\sqrt{x_i^2+x_k^2}}$ but rather using the 'hypot' method which computes c using:

$c = \frac{1}{\sqrt{1+(\frac{x_i}{x_k})^2}}\left(\frac{x_i}{x_k}\right)$

This avoids over/underflow if x_i and x_k differ in magnitude by large amounts.

The QR factorization with Given's Rotations

A common use of Given's rotations is for working out the QR factorization of a matrix X. For simplicity of notation, in the below, let $G_{i,j}$ represents Givens rotation which zeros the $(i,j)^{th}$ element in whatever matrix/product of matrices is to the right of it. The QR decomposition of X can be derived using Given's rotations by noting the following:

$$G_{1,n} \dots G_{1,2}G_{2,n} \dots G_{2,3}G_{3,n} \dots G_{3,4} \dots G_{n,n-1}X = R$$

Where R is an upper triangular matrix, by construction. Similarly, as all Given's rotations are orthogonal, $Q' = G_{1,n} \dots G_{1,2}G_{2,n} \dots G_{2,3}G_{3,n} \dots G_{3,4} \dots G_{n,n-1}$ is also orthogonal.

Thus we have:

$$Q'X = R \implies X = QR$$

Where Q is orthogonal and R is upper triangular. This is the QR decomposition.

2.13 Householder rotations for zeroing

Householder rotations are useful for inducing columns of zeros in a matrix (as oppose to givens rotations which are more appropriate for zeroing specific elements), and are often used for diagonalizing or (QR) factorising matrices. Householder rotations also possess the useful properties of being characterised by symmetric orthogonal matrices. A Householder rotation can be expressed as any matrix of the below form:

$P = I_n - \frac{2vv'}{v'v}$

Where v is referred to as the Householder vector. If a vector x is multipied by P then it is reflected in $\text{span}(v)^{\perp}$. If we wish x to become a multiple of $e_1 = [1, 0, \ldots, 0]'$, this can be achieved by choosing:

$v = x \pm |x|_2 e_1$

Sketch proof:

We want Px to belong to $\text{span}(x, e_1)$. Note, Px has the below form:

$Px = x - \frac{2v'x}{v'v}v$

Therefore, from the above, if Px is in $\text{span}(x, e_1)$, v must be in $\text{span}(x, e_1)$. Rewriting $v = x + \alpha e_1$ for some α and working through some algebra, we arrive at:

$Px = \left(\frac{\alpha^2 - |x|_2^2}{x'x + 2\alpha x_1 + \alpha^2}\right)x - 2\alpha\frac{v'x}{v'v}e_1$

Substituting $\alpha = \pm|x|_2$ turns the first term above into 0 and shows that Px is now a multiple of e_1 as required.

Computation:

Below are listed some computational facts about the Householder rotation that often are useful/important:

- To make Px a positive multiple of e_1 (which is often desirable), often the sign of $\alpha = \pm|x|_2$ is chosen such that $v_1 = x_1 - |x|_2$. This can sometimes lead to issues if x is close to a positive multiple of e_1, as v_1 becomes close to 0 and it becomes hard to determine the sign of α. Often, to get around this $v_1 = \frac{-(x_2^2 + \ldots + x_n^2)}{x_1 + |x|_2}$ is used instead.
- Often v is normalized such that $v_1 = 1$. This means only $n-1$ "essential" elements of the householder vector need to be stored.
- In practice the householder matrix is never actually calculated. Instead when applying a householder reflection to matrix A, it is easier to compute $PA = A - (\frac{2v}{v'v})(v'A)$ and/or $AP = A - (A'v)(\frac{2v}{v'v})'$, which only involves vector multiplications.
- The WY representation allows for efficient storage of a series of householder reflections (see below).

The WY representation

Suppose $Q = Q_1 Q_2 \ldots Q_r$ is a product of $(n \times n)$ householder matrices. It follows that Q can be written in the form:

$Q = I_n - WY'$

Where W and Y are $(n \times r)$ matrices. These matrices can be derived recursively using the following lemma:

Suppose $Q = I_n - WY'$ if $(n \times n)$ and we wish to apply a Householder rotation, P, to Q. Then it is true that:

$Q_+ = QP = I_n = W_+ Y'_+$

Where $W_+ = [W, z]$ and $Y_+ = [y, v]$ where $z = \frac{2Qv}{v'v}$. This leads to a natural recursive argument starting from an initial Householder matrix Q_1 and building $Q_2 Q_1$ using the above lemma, then $Q_3 Q_2 Q_1$ and so on.

The QR factorization with Householder Reflections

A classical use of the Householder reflection is the QR decomposition. A series of Householder reflections can be used to zero all elements below the diagonal of a matrix X respectively leading to the following result:

$Q_{n-1} Q_{n-2} \dots Q_1 X = R$

Where R is an upper triangular matrix. Defining $Q = Q_1^{-1} Q_2^{-1} \dots Q_{n-1}^{-1} = Q'_1 Q'_2 \dots Q'_{n-1}$ (as Householder reflections are orthogonal) we get:

$X = QR$

Where Q is an orthogonal matrix. This is the QR decomposition. Another use of Householder reflections is to update the QR decomposition (see section 2.14).

2.14 Orthogonal transform for linear modelling

Given the linear model:

$$Y = X\beta + \epsilon$$

If A is a full rank $(n \times n)$ orthonormal matrix, then the below model gives the same OLS estimator as the original model:

$$AY = AX\beta + A\epsilon$$

Proof:

$\hat{\beta}_A = ((AX)'(AX))^{-1}(AX)'AY$

$= (X'A'AX)^{-1}X'A'AY$

$= (X'X)^{-1}X'Y$

$= \hat{\beta}$

Where $\hat{\beta}$ is the OLS estimate from the original model and $\hat{\beta}_A$ is the OLS estimate from the transformed model.

This can easily be turned into a privacy protecting method for distributed linear models like so:

- Each site, site k, generates their own private orthogonal matrix A and multiplies their data X_k and Y_k by $\frac{A}{K}$ where K is the total number of sites. They then share $\tilde{X}_k = \frac{A}{K}X$ and $\tilde{Y}_k = \frac{A}{K}X$.
- The central processor now obtains $\tilde{Y} = [\tilde{Y}_1', \tilde{Y}_2', \ldots, \tilde{Y}_K']'$ and $\tilde{X} = [\tilde{X}_1', \tilde{X}_2', \ldots, \tilde{X}_K']'$ and regresses $\tilde{Y}$ onto $\tilde{X}$.

The parameters obtained from this regression will be the same as those which would have been obtained by regressing Y onto X as the above is equivalent to running the original linear model but multiplied by:

$$A = \frac{1}{K}\begin{bmatrix} A_1 & 0 & 0 & \ldots & 0 \\ 0 & A_2 & 0 & \ldots & 0 \\ 0 & 0 & A_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & A_K \end{bmatrix}$$

Which is orthogonal by construction.

2.15 Updating the QR factorization

This section details how to update/downdate the QR decomposition of a matrix when introducing new data. All the following subsections assume we have an arbitrary $(m \times n)$ matrix, A, (except the first, which assumes A is square/$(n \times n)$), and it's QR decomposition denoted $A = QR$.

This section only details the most commonly used algorithms. Other algorithms can be found in the commented out reference.

Rank 1 changes Suppose we now wish to compute the QR decomposition of $\tilde{A} = A + uv'$ where $u, v \in \mathbb{R}^n$. Note that;

$$\tilde{A} = A + uv' = QR + QQ^{-1}uv' = QR + QQ'uv' = Q(R + Q'uv') = Q(R + wv')$$

Where $w = Q'u$. Define Given's rotations $J_1, \ldots, J_{n-1}'$ such that each J_k is rotation in the planes k and $k+1$ and:

$$J_1' \ldots J_{n-1}' w = \pm ||w||_2 e_1$$

It follows that these same rotations applied to R give an upper Hessenberg matrix;

$$H = J_1' \ldots J_{n-1}' R$$

Where a matrix M is upper Hessenberg if and only if $i > j + 1$ implies $M_{i,j} = 0$ for all i and j (i.e. everything below the sub-diagonal is zero). Note this implies that:

$$H_1 = (J_1' \dots J_{n-1}')(R + wv') = H + ||w||_2 e_1 v'$$

Must also be upper Hessenberg. This means only $n - 1$ additional Givens rotations ($G_1, \dots, G_n$, zero-ing out the $n-1$ elements below the diagonal) are required to obtain an upper triangular matrix R_1 from H_1:

$$G_{n-1}' \dots G_1' H_1 = R_1$$

Note also that Q_1, defined below, is orthogonal:

$$Q_1 = QJ_{n-1} \dots J_1 G_1 \dots G_{n-1}$$

And that:

$$Q_1 R_1 = QJ_{n-1} \dots J_1 G_1 \dots G_{n-1} G_{n-1}' \dots G_1' H_1 = Q(R + wv') = A + uv' = \tilde{A}$$

Therefore, $Q_1 R_1$ provides an updated QR factorisation of $\tilde{A}$.

Appending or deleting a column Denote the columns of A as $A = [a_1, a_2, \dots a_p]$ ($a_i \in \mathbb{R}^n$) and let R have the following form:

$$R = \begin{bmatrix} R_{11} & v & R_{13} \\ 0 & r_{kk} & w' \\ 0 & 0 & R_{33} \end{bmatrix}$$

Where R_{11} is $((k-1) \times (k-1))$, v is $((k-1) \times 1)$, R_{13} is $((k-1) \times (p-k))$, r_{kk} is scalar, w is $((p-k) \times 1)$ and R_{33} is $((n-k) \times (p-k))$.

Suppose we wish to find the QR decomposition of $\tilde{A} = [a_1, \dots a_{k-1}, a_{k+1}, \dots a_p]$ (i.e. A with it's k^{th} column removed). Note that:

$$H = Q'\tilde{A} = \begin{bmatrix} R_{11} & R_{13} \\ 0 & w' \\ 0 & R_{33} \end{bmatrix}$$

H is upper Hessenberg and therefore a sequence of $(p-k)$ Given's rotations, $G_{p-1}, \dots, G_k$, which zero the sub-diagonal elements of H can be defined such that $R_1 = G_{p-1}' \dots G_k H$ is diagonal and $Q_1 = QG_k \dots G_{p-1}$ is orthogonal. It follows by the construction of Q_1 and R_1 that $Q_1 R_1 = \tilde{A}$ is, therefore, the QR decomposition of $\tilde{A}$.

Now consider $\tilde{A}$ instead is $[a_1, \dots, a_k, z, a_{k+1}, \dots, a_p]$ (i.e. $\tilde{A}$ is now A with a column added). Consider:

$$Q'\tilde{A} = [Q'a_1, \dots, Q'a_k, w, Q'a_{k+1}, \dots, Q'a_p]$$

Where $w = Q'z$. The above matrix is still diagonal in all but the k^{th} column, w. It

therefore follows that a series of Given's rotations can be applied to the sub-diagonal elements in w to restore triangular form. Once these Given's rotations have been derived they can be applied to obtain an updated QR decomposition in the same way as is done when removing a column.

Appending or deleting a row Suppose now that $\tilde{A} = \begin{bmatrix} w' \\ A \end{bmatrix}$, (i.e. we have added a row in the first position of A). It follows that:

$$\text{diag}(1, Q')\tilde{A} = \begin{bmatrix} w' \\ R \end{bmatrix} = H$$

H is upper Hessenberg and therefore rotations $G_1, ...G_n$ can be applied such that $G'_n, \ldots, G'_1 H = R_1$ is upper triangular. It follows that $\tilde{A} = Q_1 R_1$ where $Q_1 = \text{diag}(1, Q)G_1 ... G_n$.

If the row is added in a different position to the first a permutation matrix can be applied first. Suppose now that the row is added between rows k and $k+1$ of A (i.e. let $\tilde{A} = [a'_1, \ldots, a'_k, w, a'_{k+1}, \ldots, a'_p]'$). Define a permutation matrix, P, by:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ I_k & 0 & 0 \\ 0 & 0 & I_{p-k} \end{bmatrix}$$

Where I_s is the $(s \times s)$ identity matrix. It can now be seen that;

$$\text{diag}(1, Q')P\tilde{A} = \text{diag}(1, Q')P \begin{bmatrix} A_1 \\ w' \\ A_2 \end{bmatrix} = \begin{bmatrix} w' \\ R \end{bmatrix} = H$$

Where $A_1 = [a'_1, \ldots, a'_k]'$, $A_2 = [a'_{k+1}, \ldots, a'_p]'$ and H is upper Hessenberg. From here we can proceed as before to get the updated QR factorisation by using Given's rotations to remove the p sub-diagonal elements.

In order to delete a row, consider the QR factorization with of $\tilde{A}$ where $A = \begin{bmatrix} v' \\ \tilde{A}_1 \end{bmatrix}$, where v is $(p \times 1)$ (i.e. $\tilde{A}$ is A with the first row removed). Let q' be the first row of Q, where $A = QR$ is the QR factorisation of A. Define Given's rotations $G_1, \ldots, G_{n-1}$ such that $G'_1 \ldots G_{n-1} q = \alpha e_1$ where α is either 1 or -1 and $e_1 = [1, 0, \ldots, 0]'$ is $(n \times 1)$. It now follows that:

$$H = G'_1 \ldots G'_{n-1} R = \begin{bmatrix} v' \\ R_1 \end{bmatrix}$$

is upper Hessenberg, where R_1 is upper triangular. Similarly:

$$QG_{n-1} \ldots G_1 = \begin{bmatrix} \alpha & 0 \\ 0 & Q_1 \end{bmatrix}$$

Where Q_1 is orthogonal. Therefore;

$$A = \begin{bmatrix} z' \\ \tilde{A} \end{bmatrix} = (QG_{n-1} \ldots G_1)(G'_1 \ldots G'_{n-1} R) = \begin{bmatrix} \alpha & 0 \\ 0 & Q_1 \end{bmatrix} \begin{bmatrix} v' \\ R_1 \end{bmatrix}$$

It follows that the QR decomposition of $\tilde{A}$ is therefore $Q_1 R_1$. If instead of remov-

ing the first row of A, we wish to remove a different row, a permutation matrix can be applied in similar fashion to in the example when a row is added.

2.16 Updating the Cholesky factorization

2.17 COINSTAC gradient descent

2.18 Differential privacy

2.19 Quick NR with QR

The following is an Newton-Raphson based recursion for deriving the maximised log likelihood β estimates for a exponential model, which can be resolved using functions such as 'solve' in R.

$$R_k(\beta_{k+1} - \beta_k) = Q_k'(w_k^{-1/2} \circ z_k)$$

Where the log likehood has the form:

$$l(\beta; y) = y'X\beta - 1_n'b(X\beta)$$

For some scalar function b, and:

- $w_k = b''(X\beta_k)$
- $z_k = y - b'(X\beta_k)$
- Q_k and R_k are from the QR decomposition of $X_k = \text{diag}(w_k)^{-1/2}X$
- And $\circ$ is the Hadamard/elementwise product.

Sketch proof

It can be shown via direct algebra that:

$\nabla l(\beta; y) = X'(y - b'(X\beta))$
$\nabla^2 l(\beta; y) = X'\text{diag}(b''(X\beta)X$

From here the general form of Newton Raphson can be applied:

$x_{n+1} = x_n - (\nabla f(x_n))^{-1} f(x_n)$

Where f is the function to be updated and x is the variable it is updated with respect to (note here $f = \nabla l$ and $x = \beta$). This yields:

$\beta_{k+1} = \beta_k + [X'\text{diag}(w_k)X]^{-1} z_k = \beta_k + [X_k'X_k]^{-1} X_k \text{diag}(w_k)^{-1/2} z_k$

Finally, taking the QR decomposition of X_k and rearranging gives the result. A full written proof can be found in the link commented out here (Latex is being difficult with links).

2.20 Frisch-Waugh-Lovell Theorem

The Frisch-Waugh-Lovell theorem makes several statements about the following linear regression models:

$$Y = X_1\beta_1 + X_2\beta_2 + \epsilon$$

And:

$$M_{X_1}Y = M_{X_1}X_2\beta_2 + M_{X_1}\epsilon$$

OLS estimation of the first, 'all in one', model obtains coefficients $\hat{\beta} = [\hat{\beta}_1'\hat{\beta}_2']'$ by using both X_1 and X_2 at once like so:

$$\hat{\beta} = \begin{bmatrix}\hat{\beta}_1 \\ \hat{\beta}_2\end{bmatrix} = (X'X)^{-1}X'Y$$

OLS estimation of the second, 'two-stage' model, however, is equivalent to taking a different approach by first estimating a coefficient $\hat{\beta}_1^*$, based only on X_1, then orthogonalizing the data using this coefficient to regress again using only the orthogonalized X_2 to obtain a second coefficient $\hat{\beta}_{2,R1}^*$. This is done like so:

$\hat{\beta}_1^* = (X_1'X_1)^{-1}X_1'Y$

$Y_{R1} = M_{X_1}Y = Y - X_1\hat{\beta}_1^*$

$X_{2,R1} = M_{X_1}X_2 = X_2 - X_1(X_1'X_1)^{-1}X_1'X_2$

$\hat{\beta}_{2,R1}^* = (X_{2,R1}'X_{2,R1})^{-1}X_{2,R1}'Y_{R1}$

Statement 1:

It is true that:

$\hat{\beta}_{2,R1}^* = \hat{\beta}_2$

Proof of statement 1:

Note first that:

$$\hat{\beta} = \begin{bmatrix}\hat{\beta}_1 \\ \hat{\beta}_2\end{bmatrix}$$

$$= \left(\begin{bmatrix}X_1' \\ X_2'\end{bmatrix}[X_1X_2]\right)^{-1}\begin{bmatrix}X_1' \\ X_2'\end{bmatrix}Y$$

$$= \begin{bmatrix}X_1'X_1 & X_1'X_2 \\ X_2'X_1 & X_2'X_2\end{bmatrix}^{-1}\begin{bmatrix}X_1' \\ X_2'\end{bmatrix}Y$$

$$= \begin{bmatrix} (X_1'X_1 - X_1'R_2X_1)^{-1} & -(X_1'X_1 - X_1'R_2X_1)^{-1}X_1'X_2(X_2'X_2)^{-1} \\ -(X_2'X_2 - X_2'R_1X_2)^{-1}X_2'X_1(X_1'X_1)^{-1} & (X_2'X_2 - X_2'R_1X_2)^{-1} \end{bmatrix} \begin{bmatrix} X_1' \\ X_2' \end{bmatrix} Y$$

(By the block matrix inversion formula and defining $R_k = X_k(X_k'X_k)^{-1}X_k'$)

$$= \begin{bmatrix} (X_1'X_1 - X_1'R_2X_1)^{-1}X_1'Y - (X_1'X_1 - X_1'R_2X_1)^{-1}X_1'R_2Y \\ -(X_2'X_2 - X_2'R_1X_2)^{-1}X_2'R_1Y + (X_2'X_2 - X_2'R_1X_2)^{-1}X_2'Y \end{bmatrix}$$

Therefore, by equating the bottom row of the above, it follows that:

$$\hat{\beta}_2 = -(X_2'X_2 - X_2'R_1X_2)^{-1}X_2'R_1Y + (X_2'X_2 - X_2'R_1X_2)^{-1}X_2'Y$$

$$= (X_2'X_2 - X_2'R_1X_2)^{-1}(X_2' - X_2'R_1)Y$$

(By Factorizing)

$$= (X_2'(I - R_1)X_2)^{-1}X_2'(I - R_1)Y$$

$$= (X_2'H_1X_2)^{-1}X_2'H_1Y$$

(By defining $H_1 = I - R_1$)

$$= (X_2'H_1'H_1X_2)^{-1}X_2'H_1'H_1Y$$

(By noting that H_1 is symmetric and idempotent, see proof of this below)

$$= ((H_1X_2)'H_1X_2)^{-1}(H_1X_2)'H_1Y$$

$$= (X_{2,R1}'X_{2,R1})^{-1}X_{2,R1}'H_1Y$$

(By the definition of $X_{2,R1}$)

$$= (X_{2,R1}'X_{2,R1})^{-1}X_{2,R1}'Y_{R1}$$

(By the definition of Y_{R1})

$$= \hat{\beta}_{2,R1}^*$$

(By the definition of $\hat{\beta}_{2,R1}^*$)

Statement 2:

H_1 is symmetric and idempotent.

Proof of statement 2:

See (section 2.1 "Distributed Calculation for Linear Models").

Statement 3:

The residuals given by the 'all in one' approach ($Y - X\hat{\beta}$) are equivalent to those given in the second stage of the 'two-stage' approach ($Y_{R1} - X_{2,R1}\hat{\beta}^*_{2,R1}$).

Proof of statement 3:

$$
\begin{aligned}
Y_{R1} - X_{2,R1}\hat{\beta}^*_{2,R1} &= Y_{R1} - X_{2,R1}(X'_{2,R1}X_{2,R1})^{-1}X'_{2,R1}Y_{R1} \\
&\quad \text{(By the definition of } \hat{\beta}^*_{2,R1}\text{)} \\
&= Y_{R1} - X_{2,R1}(X'_{2,R1}X_{2,R1})^{-1}X'_{2,R1}(I - X_1(X'_1X_1)^{-1}X'_1)Y \\
&\quad \text{(By the definition of } Y_{R1}\text{)} \\
&= Y_{R1} - X_{2,R1}(X'_{2,R1}X_{2,R1})^{-1}X'_{2,R1}Y + \\
&\quad X_{2,R1}(X'_{2,R1}X_{2,R1})^{-1}X'_{2,R1}X_1(X'_1X_1)^{-1}X'_1Y
\end{aligned}
$$

However, by noting that $X_{2,R1}$ is, by construction, orthogonal to X_1, we obtain that $X'_{2,R1}X_1 = 0$ and, therefore, the last term in the above becomes zero, leaving:

$$
\begin{aligned}
& Y_{R1} - X_{2,R1}(X'_{2,R1}X_{2,R1})^{-1}X'_{2,R1}Y \\
=\ & Y - X_1(X'_1X_1)^{-1}X'_1Y - X_{2,R1}(X'_{2,R1}X_{2,R1})^{-1}X'_{2,R1}Y \\
=\ & Y - R_1Y - R_{2,R1}Y \\
& \text{(By the definition of } R_k\text{)}
\end{aligned}
$$

Note that R_1Y and $R_{2,R1}Y$ are, by the construction of $X_{2,R1}$, orthogonal projections of Y satisfying $R_1Y + R_{2,R1}Y = RY$ (where $R = X(X'X)^{-1}X'$).

Therefore:

$$
\begin{aligned}
Y_{R1} - X_{2,R1}\hat{\beta}^*_{2,R1} &= Y - R_1Y - R_{2,R1}Y \\
&= Y - RY \\
&\quad \text{(As } R_1Y + R_{2,R1}Y = RY\text{)} \\
&= Y - X(X'X)^{-1}X'Y \\
&\quad \text{(By the definition of } R\text{)} \\
&= Y - X\hat{\beta} \\
&\quad \text{(By the definition of } \hat{\beta}\text{)}
\end{aligned}
$$

Therefore, $Y_{R1} - X_{2,R1}\hat{\beta}^*_{2,R1} = Y - X\hat{\beta}$ as given in Statement 3. In other words, the residuals of the two models are equal.

2.21 Dimension Reduction formulae

The dimension reduction formulas are a useful way to reduce $(n_i \times n_i)$ inversions and determinants into $(q \times q)$ problems which are more numerically stable and time efficient. The below inversion dimension reduction formula is used a lot for inversion;

$$
\begin{aligned}
V_i^{-1} &= (I_{n_i} + Z_i D Z_i')^{-1} && \text{(which involves an } (n_i \times n_i) \text{ inversion)} \\
&= I_{n_i} - Z_i(I_q + DZ_i'Z_i)^{-1}DZ_i' && \text{(which involves a } (q \times q) \text{ inversion)} \\
&= I_{n_i} - Z_i D(I_q + Z_i'Z_i D)^{-1}Z_i' && \text{(which involves a } (q \times q) \text{ inversion)} \\
&= I_{n_i} - Z_i(D^{-1} + Z_i'Z_i)^{-1}Z_i' && \text{(which involves two } (q \times q) \text{ inversions,} \\
& && \quad \text{unless working with } D^{-1} \text{ already)}
\end{aligned}
$$

The below dimension reduction formula is often used for calculating determinants:

$$
\begin{aligned}
|V_i| &= |I_{n_i} + Z_i D Z_i'| && \text{(which involves an } (n_i \times n_i) \text{ determinant calculation)} \\
&= |I_q + DZ_i'Z_i| && \text{(which involves a } (q \times q) \text{ determinant calculation)}
\end{aligned}
$$

In addition, if D^{-1} is known, the log of the determinant can be expressed as so:

$$
\begin{aligned}
\ln|V_i| &= \ln(|D||D^{-1} + Z_i'Z_i|) \\
&= \ln|D^{-1} + Z_i'Z_i| - \ln|D^{-1}| && \text{(which involves } (q \times q) \text{ determinant calculations only)}
\end{aligned}
$$

Finally the below are also useful results from the dimension reduction formulae.

$$
\begin{aligned}
Z_i'V_i^{-1}Z_i &= (I_q + Z_i'Z_i D)^{-1}Z_i'Z_i \\
&= Z_i'Z_i(I_q + DZ_i'Z_i)^{-1} && \text{(which involves } (q \times q) \text{ inversion only)} \\
&= Z_i'(I_q + Z_i'DZ_i)^{-1}Z_i && \text{(Note: This line is only true if } (Z_i'Z_i)^{-1} \text{ exists)} \\
&= ((Z_i'Z_i)^{-1} + D)^{-1} && \text{(Note: This line is only true if } (Z_i'Z_i)^{-1} \text{ exists)}
\end{aligned}
$$

Fixed Effects Variance Estimators under Dimension Reduction The following useful variants of the dimension reduction formulae are given in Demidenko.

$$\sum y_i' V_i^{-1} y_i = \sum y_i' y_i - \sum (Z_i' y_i)' (D_- + Z_i' Z_i)^{-1} (Z_i' y_i)$$
$$\sum X_i' V_i^{-1} y_i = \sum X_i' y_i - \sum (X_i' Z_i)(D_- + Z_i' Z_i)^{-1} (Z_i' y_i)$$
$$\sum X_i' V_i^{-1} X_i = \sum X_i' X_i - \sum (X_i' Z_i)(D_- + Z_i' Z_i)^{-1} (X_i' Z_i)'$$

2.22 Spherical Transform of mixed models

https://cran.r-project.org/web/packages/lme4/vignettes/Theory.pdf https://github.com/lme4/lme4pureR/tree/ma
lme4: Mixed-effects modeling with R on

Also make sure to note down transform to get from lmer cov to prev mentioned

2.23 The Woodbury identity and other related updates

All formulas in this section are common variants of the Woodbury Identity.

Woodbury Identity The Woodbury Identity is given by:

$$(A + UBV)^{-1} = A^{-1} - A^{-1}U(B^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

Sherman-Morrison Update Let $B = [A, v]$, the Sherman-Morrison Update allows the inverse of B to be derived from the inverse of A as so:

$$(A + bc')^{-1} = A^{-1} - \frac{A^{-1}bc'A^{-1}}{1 + c'A^{-1}b}$$

Rank 1 update of inverse of inner product From Sherman-Morrison the below can also be derived. Denote $A = (X'X)^{-1}$ and $\tilde{X} = [X, v]$. Then:

$$(\tilde{X}'\tilde{X})^{-1} = \begin{bmatrix} A + \frac{AX'vv'XA'}{v'v - v'XAX'v} & \frac{-AX'v}{v'v - v'XAX'v} \\ \frac{-v'XA'}{v'v - v'XAX'v} & \frac{1}{v'v - v'XAX'v} \end{bmatrix}$$

Rank 1 update of pseudo-inverse Applying Sherman-Morrison to the pseudo-inverse also gives the following theorem.

Let $B = [A, v]$ be an $(n \times (p+1))$ matrix with A an $(n \times p)$ matrix and v a column vector of length p. Let $B^+ = B'(BB')^{-1}$ and $A^+ = A'(AA')^{-1}$, it follows that:

$$B^+ = \begin{bmatrix} A' \\ v' \end{bmatrix} (AA')^{-1}(I - vu) = \begin{bmatrix} A^+(I - vu) \\ u \end{bmatrix}$$

Where $u = \frac{v'(AA')^{-1}}{1 + v'(AA')^{-1}v}$ and the first equality follows by Sherman-Morrison. Similarly defining $A^+ = (A'A)^{-1}A'$ and $B^+ = (B'B)^{-1}B'$, we get:

$$B^+ = (A'A)^{-1}(I - v'u)[A', v']$$

Where $u = \frac{v(A'A)^{-1}}{1+v(A'A)^{-1}v'}$.

2.24 Block matrix lemmas

This subsection lists some common forms used for block matrices.

Block matrix inverse formula The block matrix inverse formula gives the inverse for a partitioned matrix:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} A_{11}^{-1} + A_{11}^{-1}A_{12}(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}A_{21}A_{11}^{-1} & -A_{11}^{-1}A_{12}(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \\ -(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}A_{21}A_{11}^{-1} & (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \end{bmatrix}$$

Alternatively;

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} (A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1} & -(A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1}A_{12}A_{22}^{-1} \\ -A_{22}^{-1}A_{21}(A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1} & A_{22}^{-1} + A_{22}^{-1}A_{21}(A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1}A_{12}A_{22}^{-1} \end{bmatrix}$$

A consequence of this is that the inverse of a block diagonal matrix has the following form:

$$\begin{bmatrix} A_1 & 0 & 0 & \dots & 0 \\ 0 & A_2 & 0 & \dots & 0 \\ 0 & 0 & A_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & A_n \end{bmatrix}^{-1} = \begin{bmatrix} A_1^{-1} & 0 & 0 & \dots & 0 \\ 0 & A_2^{-1} & 0 & \dots & 0 \\ 0 & 0 & A_3^{-1} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & A_n^{-1} \end{bmatrix}$$

Schur complement The Schur Complement of block A_{11} in the notation of the previous section is:

$$A_{22} - A_{21}A_{11}^{-1}A_{12}$$

Similarly, the Schur Complement of block A_{22} in the notation of the previous section is:

$$A_{11} = A_{12}A_{22}^{-1}A_{21}$$

Using the Schur Complement, you can rewrite the inverse of a block matrix as:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}^{-1} = \begin{bmatrix} I & 0 \\ -A_{22}^{-1}A_{21} & I \end{bmatrix} \begin{bmatrix} (A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1} & 0 \\ 0 & A_{22}^{-1} \end{bmatrix} \begin{bmatrix} I & -A_{12}A_{22}^{-1} \\ 0 & I \end{bmatrix}$$

If solving a linear system of the form:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Which has the following equation to solve for x_1 only:

$$(A_{11} - A_{12}A_{22}^{-1}A_{21})x_1 = b_1 - A_{12}A_{22}^{-1}b_2$$

This can then be inserted to solve for x_2 also. This can reduce computation time if x_2 is not of interest (e.g. if x_2 are numerous nuisance parameters for instance).

2.25 Neuman and other approximations

This section outlines a series of miscellaneous matrix approximations which are commonly employed.

The Neuman series

If A is a matrix with $|\lambda_i| < 1$ for all eigenvalues, $|\lambda_i|$:

$$(I - A)^{-1} = \sum_{n=0}^{\infty} A^n$$

Or equivalently:

$$(I + A)^{-1} = \sum_{n=0}^{\infty} (-1)^n A^n$$

This leads to the following two commonly employe approximations:

$$(I - A)^{-1} \cong I + A + A^2$$
$$(I + A)^{-1} \cong I - A + A^2$$

Other approximations

This approximation holds for A large and symmetric:

$$A - A(I + A)^{-1}A \cong I - A^{-1}$$

If σ^2 is small compared to Q and M then:

$$(Q + \sigma^2 M)^{-1} \cong Q^{-1} - \sigma^2 Q^{-1} M Q^{-1}$$

Proof:

$$\begin{aligned}
(Q + \sigma^2 M)^{-1} &= (QQ^{-1}Q + \sigma^2 MQ^{-1}Q)^{-1} \\
&= ((I + \sigma^2 MQ^{-1})Q)^{-1} \\
&= Q^{-1}(I + \sigma^2 MQ^{-1})^{-1} \\
&= Q^{-1}(I - \sigma^2 MQ^{-1} + (\sigma^2 MQ^{-1})^2 - \ldots) \quad \text{(By the Taylor Expansion)} \\
&\cong Q^{-1}(I - \sigma^2 MQ^{-1}) \\
&= Q^{-1} - \sigma^2 Q^{-1} M Q^{-1}
\end{aligned}$$

3 WIP Ideas

This section is a work in progress scratchpad documenting ideas I have had during writing this document.

3.1 Demidenko Distributed Algorithms for one factor BLMM

Idea: The algorithms in section 1.5 from the Demidenko book for fixed effect and covariance parameter estimation, all look extremely amenable to distributed computation using the same trick we used for BLM (section 2.1).

It may be the case that some of these algorithms can be adapted so that each processor/site calculates quantities similar to $X_i'X_i$ in BLM and then the iteration is run on the central processor quickly (hopefully on $(p \times p)$ matrices) to obtain the final estimator. A drawback of these algorithms however is that they only allow for one factor LMMs and may also lead to negative definite matrices for the random effects covariance D.

At the very least though I could try and make these distributed and then test their performance against one another much like Habib did in 'fast and poweful GWAS'. At least this way we would have results.

I suggest this is priority so we have something to show and then the other ideas are something to consider once we have these down.

3.2 Bates Distributed Algorithms for multiple factor BLMM

Idea: The algorithm in section 1.5 used by 'lmer' by Doug Bates doesn't suffer from the issues of only being able to calculate one factor models or negative definite matrices. If possible, it would be good to try and adapt this into a distributed algorithm if possible. From what I've seen, however, as two stage doesn't assume Z is block diagonal, this might not be as amenable to distributed computing... again it needs more investigation.

Ideally I would like to have results for this algorithm alongside those in section 3.1 so that we can make reference to 'lmer'.

Updates: Using the package 'cvxopt' for sparse cholesky decomposition and the package 'scipy' for the optimization, I have implemented the PLS algorithm for one voxel and checked the results against R. The 'Powell' algorithm proved to be the fastest, being consistently around 8 times faster than than the alternative 'Nelder-Mead'.

The code took approximately 0.1 seconds to run one analysis. As array broadcasting is not available for sparse matrices in 'cvxopt', for a NIFTI of dimensions $(91 \times 109 \times 91)$ this means it will take approximately $91 \times 109 \times 91 \times 0.1/60^2 \approx 25$ hours without further improvement (accounting for the fact that roughly two thirds of the volume lies outside the mask reduces this to ~ 8 hours). Ideas are given in section 3.7 and 3.8 which aim to improve the computation time for this algorithm. Dask may also provide significant improvements.

The results also agreed with that R to at least 3 decimal places. In addition, I have reorganised the code to require only $X'Y$, $Z'Y$, $Z'X$, $X'X$, $Y'Y$ and $Z'Z$, meaning no matrices which scale with number of subjects are required for this algorithm.

3.3 Rework Demidenko Algorithms for two factor BLMM

Idea: The algorithms in section 1.5 from the Demidenko book for fixed effect and covariance parameter estimation do not account for the possibility of a multi-factor analysis and assume a block diagonal structure for Z. It may be worth re-deriving these algorithms without this assumption to see if we can still use them when we have the processing power of a distributed system to compensate for the time efficiency lost by not making this assumption. Again this needs further investigation...

I suggest this is medium priority, as if we find the algorithms from Demidenko provide better efficiency, the argument for using them will be weakened by the fact that the Bates algorithm allows for a wider range of models.

3.4 Rework Demidenko Algorithms for non-negative definite D

Idea: Some of the algorithms in section 1.5 from the Demidenko book for fixed effect and covariance parameter can sometimes give negative definite estimates for D. The book suggests we use one of the 3 solutions outlined in section 1.5, one of which is to reparameterize D into the cholesky decomposition $D = \Lambda'$ and then maximise the log-likelihood for Λ. This seems similar to the logic underlying the method used in 'lmer' and seems as though it is worth doing... unfortunately the book suggests doing this but doesn't actually give any algorithms for it.

It may be worth re-deriving the algorithms in section 1.5 but for Λ instead of D to ensure D is non-negative definite. Again this needs further investigation...

For the moment though I suggest this is low priority..

3.5 Applying Frisch-Waugh-Lovell to regress out fixed effects

Idea: I am unsure if this is possible but one idea I did have for an additional algorithm estimating the fixed effects and covariance parameters was to do the following:

- Much like in Frisch-Waugh-Lovell, use a matrix $M_X = I - X(X'X)^{-1}X'$ to regress out the fixed effects from the linear mixed model. This would transform the LMM:

  $$Y = X\beta + Zb + \epsilon$$

  Into:

  $$M_x Y = M_x Z b + M_x \epsilon$$

- Denoting $\tilde{Z} = M_x Z$ and $\tilde{Y} = M_x Y$, we still have a mixed effects model, just with no fixed effects, which we can then maximize the log likelihood for to get an estimator for D.

- Use the GLS formula to obtain an estimator for β and the closed form estimator for σ^2 based on β and D given in the Demidenko book.

The benefit of this is that this would hopefully be more computationally simple due to the neglection of any X matrix in the calculations. Perhaps this could speed up computation? It is just a thought... it may actually lead to the same algorithm as given for the profile likelihood for D, without checking I cannot be sure though. It seems like something someone would have thought of before.

Note also that in step one $\tilde{Y}$ can be obtained in a distributed manor by running BLM to obtain $\tilde{Y} = Y - X\hat{\beta}_{OLS}$. The issue here is obtaining $M_x Z$... this might take some more thought.

3.6 Handling the inversion of V

Problem: The classic algorithms for estimating parameters of linear mixed models all make the same key assumption; that there is only one factor present. I.e. in 'R' language, the model has the form:

`y ~ ffx + (rfx|factor1)`

Where `ffx` and `rfx` are the fixed effects and random effects, respectively. This leads to the random effects matrix, Z, having a nice block diagonal form (see fig 1).

The classical algorithms then often need estimate the scaled variance of the response Y during the likelihood estimation and inverse it. This scaled variance is given by $V = I + Z\Sigma Z'$ where Σ is the block-diagonal variance of the random effects b. It is easy to see where this inversion comes into the mathematics by considering the log likelihood which is given by:

$$l(\theta) = -\tfrac{1}{2}\big\{N\ln(\sigma^2) + \ln|I + Z\Sigma Z^T| + \sigma^{-2}(y - X\beta)^T(I + Z\Sigma Z^T)^{-1}(y - X\beta)\big\}$$

Note that V here has dimension $(n \times n)$ and this is an extremely large matrix inversion. In the classical setting, with one factor, however, this is not an issue. The reason for this is that, as Z and Σ are block diagonal, $Z\Sigma Z^T$ is block diagonal (see fig 2).

As the inverse of a block diagonal matrix is just the same matrix with each block replaced by the inverse of said block, this now reduces to simply computing the inverse of each $(n_{fgi} \times n_{fgi})$ block (where n_{fgi} is the number of values for each factor grouping i, for example if the factor was subject the n_{fg1} would be the number of recordings for subject 1).

In short, in the estimation of the one-factor model, the computation of $(I + Z\Sigma Z^T)^{-1}$

is broken down considerably, by noting the block-diagonal structure of Z.

However, since the development of the algorithms which use this trick, the definition of the linear mixed model has expanded somewhat to include crossed-factor models, i.e. models of the form:

`y` $\sim$ `ffx + (rfx1|factor1) + (rfx2|factor2) + ... + (rfxn|factorn)`

An example of such a model might be, say for instance, if there was a multi-site study which required a random intercept for both site location and gender (in that example, site location would be `factor1` and gender would be `factor2` whilst `rfx1` and `rfx2` would both be intercept columns).

Fig. 1. Z^T for a one factor model with a random intercept and random covariate.

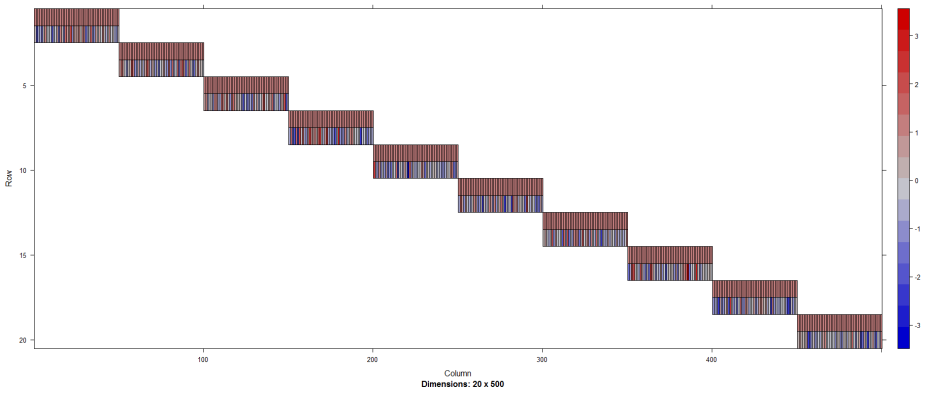
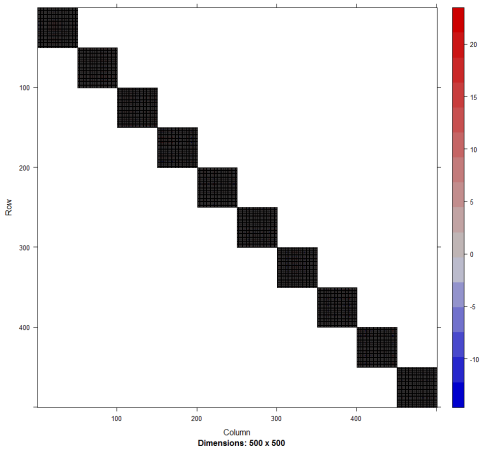


Fig. 2. $Z\Sigma Z^T$ for a one factor model with a random intercept and random covariate.



However, importantly, the inclusion of multiple factors now destroys the block diagonal structure of Z. For example, fig. 3 shows the same Z matrix as fig. 1 but with a second factor added.

The same computational trick cannot be applied to calculate $(I + Z\Sigma Z^T)^{-1}$ for the crossed factor model as was used in the one-factor model, as $Z\Sigma Z^T$ is no longer block diagonal (see fig. 4). This now requires an $(n \times n)$ matrix inversion, which is not feasible as n grows large.

Before discussing my proposed solution to this, I'd like to briefly point out that $(I + Z\Sigma Z^T)^{-1}$ almost always appears sandwiched between other matrices in computation. i.e. calculation almost always is of the form $A^T(I+Z\Sigma Z^T)^{-1}B$ where A and B are one of either X, Y or Z. This will be of use later.

Proposed Solution: Before going into details on my proposed solution, I would like to quickly introduce/recap the following notation:

- l_i - This is the number of levels for factor i. For example, if factor 1 was by group then l_1 would be number of subjects in group 1.

- q_i - This is the number of random effects coefficients for factor i. For example, if the first factor was a group and we had a random intercept and a reading of subjects weight over time, then this would be 2 random effects and therefore $q_1 = 2$.

- F - the total number of factors.

- q - This is the size of $Z'Z$ and Σ (both are $q \times q$). It is equal to $\Sigma_i q_i l_i$.

I will assume, for notational ease that the random effects are ordered in Z by $f_1, f_2, \ldots, f_F$ such that $q_1 l_1 \geq q_2 l_2 \geq \ldots \geq q_F l_F$. My proposed solution reduces the issues of an $(n \times n)$ matrix inversion for a multi-factor analysis to a series of matrix inversions of, at most, size $(a \times a)$ where $a = \max(q_2 l_2, q_1)$.

To understand my solution, first note that the dimension reduction formulae given in Verbeke (2004), (see section 2.21) tell us that:

$$(I + Z\Sigma Z^T)^{-1} = I - Z(I + \Sigma Z^T Z)^{-1}\Sigma Z^T$$

Fig. 3. Z^T for a two (crossed) factor model with a random intercept and random covariate for both factors.

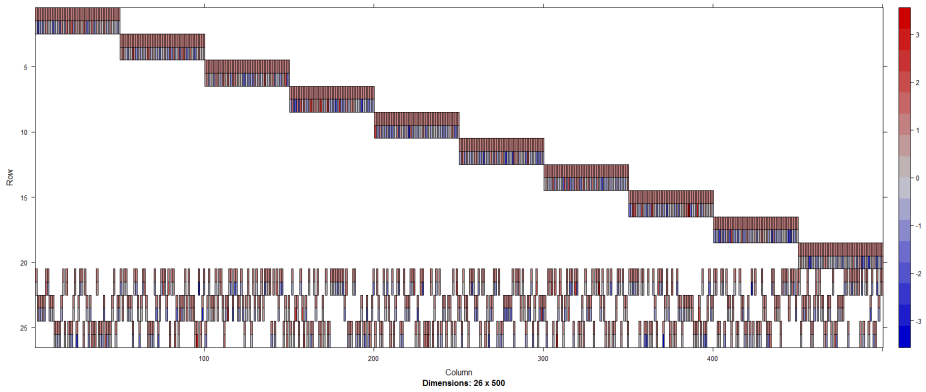
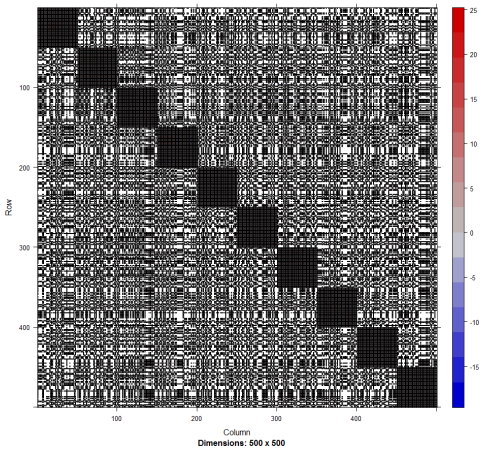


Fig. 4. $Z\Sigma Z^T$ for a two (crossed) factor model with a random intercept and random covariate for each factor.



This reduces the problem instantly to an inversion of the $(q \times q)$ matrix $(I + \Sigma Z^T Z)$ which is already much preferable to $(n \times n)$.

However, if say there are 2 factors each with 100 levels and 5 random effects to be estimated, this is now a (2000×2000) matrix inversion, which is still not ideal, especially when scaling to neuroimaging data. It is easy to think of such a situation where something like this may occur, for example, one factor may correspond to 'subject' in a study with 100 subjects and 5 random effects.

A key observation to reduce this problem further, is to note that, whilst the block-diagonal structure of $Z\Sigma Z^T$ was completely destroyed by adding crossing factors, it is not completely destroyed for $\Sigma Z^T Z$. Instead, certain blocks of $\Sigma Z^T Z$ retain a block-diagonal structure. To illustrate this, fig. 5 shows $\Sigma Z^T Z$ for the one-factor model from figs 1 and 3, whilst fig. 6 shows $\Sigma Z^T Z$ for the one-factor model from figs 2 and 4.

In general, for a 2 (crossed) factor model the matrix $\Sigma Z^T Z$ has the following form:

$$(\Sigma Z^T Z)_2 = \begin{bmatrix} A_{(q_1 l_1 \times q_1 l_1)} & B_{(q_1 l_1 \times q_2 l_2)} \\ B^T_{(q_1 l_1 \times q_2 l_2)} & C_{(q_2 l_2 \times q_2 l_2)} \end{bmatrix}$$

Where the subscripts indicate the dimension of the matrices and A and C are block-diagonal. For a general F (crossed) factor model, the matrix $\Sigma Z^T Z$ has the following recursive form:

$$(\Sigma Z^T Z)_F = \begin{bmatrix} (\Sigma Z^T Z)_{F-1} & M \\ M^T & D \end{bmatrix}$$

Where $(\Sigma Z^T Z)_{F-1}$ is the matrix for the model with factor F removed (dimensions are $M : (q_{F-1} l_{F-1} \times q_F l_F)$, $D : (q_F l_F \times q_F l_F)$ and $(\Sigma Z^T Z)_{F-1} : (\Sigma_{i=1}^{F-1} q_i l_i \times \Sigma_{i=1}^{F-1} q_i l_i)$).

For now just consider the 2-factor model. The block inverse matrix formula (see section 2.24) tells us this (the dimension subscript notation will be dropped from now on):

$$\begin{aligned} (\Sigma Z^T Z)_2^{-1} &= \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}^{-1} \\ &= \begin{bmatrix} A^{-1} + A^{-1}B(C - B^T A^{-1} B)^{-1} B^T A^{-1} & -A^{-1}B(C - B^T A^{-1} B)^{-1} \\ -(C - B^T A^{-1} B)^{-1} B^T A^{-1} & (C - B^T A^{-1} B)^{-1} \end{bmatrix} \end{aligned}$$

Fig. 5. $\Sigma Z^T Z$ for a one factor model with a random intercept and random covariate for each factor.

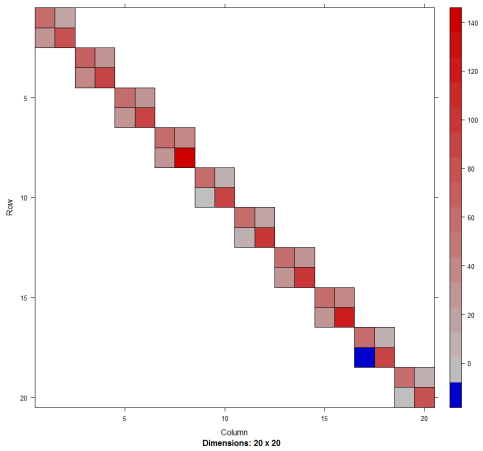
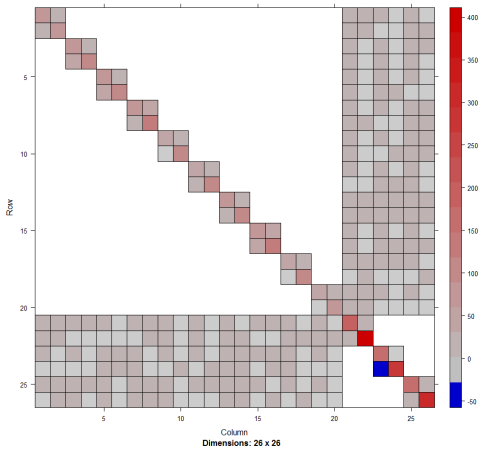


Fig. 6. $\Sigma Z^T Z$ for a two (crossed) factor model with a random intercept and random covariate for each factor.



This calculation only involves two inverses:

- A^{-1} - This inverse can be calculated extremely quickly an efficiently. The reason for this is because A is block diagonal, i.e. $A = \text{diag}(A_1, \dots A_{l_1})$ and thus it's inverse can be calculated as $A^{-1} = \text{diag}(A_1^{-1}, \dots A_{l_1}^{-1})$, thus involving only $(q_1 \times q_1)$ matrix inversions.

- $(C - B^T A^{-1} B)^{-1}$ - This inversion is only $(q_2 l_2 \times q_2 l_2)$, which may be substantially smaller than inverting a $(q \times q)$ matrix.

Therefore, the computation has been reduced to only at most $(a \times a)$ inversions where $a = \max(q_2 l_2, q_1)$, which may be much smaller and quicker than $(q \times q)$!

If we instead now wanted to invert a $(\Sigma Z^T Z)_F$ matrix for an F-crossed model, we can use a combination of the recursive definition of $(\Sigma Z^T Z)_F$ and the block matrix inverse lemma, i.e. the below:

$$(\Sigma Z^T Z)_F^{-1} =$$
$$\begin{bmatrix} (\Sigma Z^T Z)_{F-1}^{-1} + (\Sigma Z^T Z)_{F-1}^{-1} M (D - M^T (\Sigma Z^T Z)_{F-1}^{-1} M)^{-1} M^T (\Sigma Z^T Z)_{F-1}^{-1} & -(\Sigma Z^T Z)_{F-1}^{-1} M (D - M^T (\Sigma Z^T Z)_{F-1}^{-1} M)^{-1} \\ -(D - M^T (\Sigma Z^T Z)_{F-1}^{-1} M)^{-1} M^T (\Sigma Z^T Z)_{F-1}^{-1} & (D - M^T (\Sigma Z^T Z)_{F-1}^{-1} M)^{-1} \end{bmatrix}$$

This may seem daunting, but starting from $(\Sigma Z^T Z)_2^{-1}$ this can be calculated very quickly as, beyond the inversions needed for calculation of $(\Sigma Z^T Z)_2^{-1}$, the only inversion needed for each additional factor is the inverse of $(D - M^T (\Sigma Z^T Z)_{F-1}^{-1} M)^{-1}$, which, by the ordering assumption given previously in this section, is smaller than size $(q_2 l_2 \times q_2 l_2)$.

So, so far it's been shown that the inverse of $(I + Z \Sigma Z^T)$ can be calculated as $I - Z(I + \Sigma Z^T Z)^{-1} \Sigma Z^T$ and that this computation is much (much!) quicker and more efficient that inverting an $(n \times n)$ matrix.

There is one final thing to note, and that is that as often calculation involves operations of the form $A^T (I + Z \Sigma Z^T) B$ where A and B are one of either X, Y or Z. It can be seen that:

$$\begin{aligned} A^T (I + Z \Sigma Z^T)^{-1} B &= A^T (I - Z(I + \Sigma Z^T Z)^{-1} \Sigma Z^T) B \\ &= A^T B + A^T Z (I + \Sigma Z^T Z)^{-1} \Sigma Z^T B \end{aligned}$$

This is important to note as, in the distributed setting, we can calculate $A^T B$ for all $A, B \in \{X, Y, Z\}$ at each site separately and concatenate them in a similar way as to in BLM (see section 2.1), whilst still preserving the privacy of each site! The only moving part here is Σ! And the inversions are no longer an issue!

Updates:

I have now coded this idea in the Fisher Scoring python notebook. The computation was split into two components;

- Firstly, computation of the blockwise inverse of block-diagonal sparse matrix A was implemented as the function `blockInverse`.
- Secondly, a recursive function for doing the blockwise inverse of the form described above was implemented as the function `recursiveInverse`.

The `blockInverse` function was surprisingly unsuccessful. It gave exactly the same matrix as the `scipy` sparse inverse function but took approximately 4 times as long to do so (Example; scipy took approximately `0.009s`, whilst `blockInverse` took 0.039s in the toy data example in the notebook). In conclusion, I don't beleive this function gives any benefit over `scipy`.

The recursiveInverse function was notably more successful, however. To assess the function's performance, given a matrix A, I calculated it's estimated inverse, $\bar{A}$, using `numpy`, `scipy` and `recursiveInverse`. I then calculated the 2-norm of the difference between $A\bar{A}^{-1}$ and the identity matrix to measure accuracy and I recorded the time taken to invert A to measure computational efficiency. The results are in the table below:

Method	$\|\|A\bar{A}^{-1} - I\|\|_2$	Time (seconds)
`numpy`	2495.453	0.00033
`scipy`	4327.665	0.01395
`recursiveInverse`	722.506	0.01351

It is clear from this that in this example `recursiveInverse` was far superior in terms of accuracy (although no method was great...) and was no different in terms of time to `scipy`. However, both `scipy` and `recursiveInverse` were much (much) slower than `numpy`. This is something that may be observed for matrices of the size used in the toy example but which is unlikely to scale to larger sparse matrices. Whether or not to take this as a victory for `numpy` really depends on what size sparse matrices we intend to invert.

3.7 PLS speed improvement: Use of neighbouring voxel information

Problem: PLS currently does not scale very well to multiple voxels due to the inability to broadcast the sparse cholesky factor and the ignoring of neighbouring information between voxels. This section focuses on ideas aiming to speed up the computation of PLS for one voxel by using information from it's neighbouring voxels.

Idea 1: The idea behind this is relatively simple. Once we have estimated the model for one voxel, perhaps using the estimated θ vector from that voxel as the

initial estimate for it's neighbours might improve their convergence speed (as it is likely they may be starting closer to their true value if there is concordance between voxels).

To perform this idea it therefore makes sense to calculate θ for one voxel using the suggested initial estimate $\theta_0 = \text{vec}(I_n)$ (suggested in Demidenko 2013) and then use the resultant estimate $\hat{\theta}_{v_1}$ as the initial estimate θ_0 for the next voxel.

Update:

This approach has now been tried on the toy data example in "PLS_sandbox.ipynb". In the toy example, it provided very little improvement in terms of time efficiency. Computing the two voxels completely independently on average took around 0.38 seconds, whilst computing them using the suggestion in this idea consistently took around 0.003 seconds. Whilst there was a consistent improvement, it was negligible in comparison to the overall computation time.

A suggested cause for why this may have failed in this mannor is that while the initial estimate may have started closer to the true value of θ, the subsequent iterations may have quickly moved further from this value before eventually returning to the final estimate. This logic in part formed the motivation for idea 2.

Idea 2: The PLS objective function is given as:

$$\text{argmin}_{\beta,\sigma^2,\Sigma}||Y - X\beta - Z\Lambda u||_2^2 + ||u||_2^2$$

This formulation, for obvious reasons, doesn't include any information about neighbouring voxels. Suppose, we have already calculated β estimated for one voxel, β_n and wish to now estimate β for it's neighbour however.

As mentioned in the previous section, one issue encountered by PLS is that even if it starts relatively close to a reasonable estimate of β, it may stray relatively far from this estimate before returning to it. If we, however, assume that the true value of β lies somewhere in the vicinity of β_n, we can include this in the objective function like so:

$$\text{argmin}_{\beta,\sigma^2,\Sigma}||W^{-\frac{1}{2}}(Y - X\beta - Z\Lambda_\theta u)||_2^2 + ||u||_2^2 + ||\beta - \beta_n||_2^2$$

This acts like a "safety chord" pulling the iterations back towards β_n if they start to stray too far away. This is actually surprisingly easy to implement as the above has the below matrix formulation:

$$\text{argmin}_{\beta,\sigma^2,\Sigma}\left|\left|\begin{bmatrix} W^{-\frac{1}{2}}Y \\ \beta_n \\ 0 \end{bmatrix} - \begin{bmatrix} W^{-\frac{1}{2}}Z\Lambda_\theta & W^{-\frac{1}{2}}X \\ 0 & I_p \\ I_q & 0 \end{bmatrix} \begin{bmatrix} u \\ \beta \end{bmatrix}\right|\right|_2^2$$

Considering the normal equations for this, as in the derivation of the PLS algorithm, we obtain:

$$\begin{bmatrix} \Lambda_\theta^T Z^T W^{-\frac{1}{2}} & 0 & I_q \\ X^T W^{-\frac{1}{2}} & I_p & 0 \end{bmatrix} \begin{bmatrix} W^{-\frac{1}{2}}Y \\ \beta_n \\ 0 \end{bmatrix} = \begin{bmatrix} \Lambda_\theta^T Z^T W^{-\frac{1}{2}} & 0 & I_q \\ X^T W^{-\frac{1}{2}} & I_p & 0 \end{bmatrix} \begin{bmatrix} W^{-\frac{1}{2}}Z\Lambda_\theta & W^{-\frac{1}{2}}X \\ 0 & I_p \\ I_q & 0 \end{bmatrix} \begin{bmatrix} \hat{u}_\theta \\ \hat{\beta}_\theta \end{bmatrix}$$

Which implies:

$$\begin{bmatrix} \Lambda_\theta^T Z^T WY \\ X^T WY + \beta_n \end{bmatrix} = \begin{bmatrix} \Lambda_\theta^T Z^T WZ\Lambda_\theta + I_q & \Lambda_\theta^T Z^T WX \\ X^T WZ\Lambda_\theta & X^T WX + I_p \end{bmatrix} \begin{bmatrix} \hat{u}_\theta \\ \hat{\beta}_\theta \end{bmatrix}$$

Comparing this to the original PLS normal equations (below), we can see that this is exactly equivalent to performing PLS with $X^T WY$ replaced by $X^T WY + \beta_n$ and $X^T WX$ replaced by $X^T WX + I_p$. Given a pre-existing implementation of PLS which takes in $X^T WX$ and $X^T WY$ as input arguments, this is extremely easy to implement.

$$\begin{bmatrix} \Lambda_\theta^T Z^T WY \\ X^T WY \end{bmatrix} = \begin{bmatrix} \Lambda_\theta^T Z^T WZ\Lambda_\theta + I_q & \Lambda_\theta^T Z^T WX \\ X^T WZ\Lambda_\theta & X^T WX \end{bmatrix} \begin{bmatrix} \hat{u}_\theta \\ \hat{\beta}_\theta \end{bmatrix}$$

Update:

This approach has now been tried on the toy data example in "PLS_sandbox.ipynb". In the toy example, this performed extremely well. The original two voxels, when estimated independently took approximately 0.40 seconds to run, whereas when using this method they only took 0.32 seconds on average; a reduction of around a $\frac{1}{5}^{th}$ in terms of computation time.

More importantly however, in addition, when compared against the true values of β used to simulate the voxels, the "Neighbour-based" method appeared much more robust in terms of estimating the true β. The average squared difference between the true and estimated values of β for standard PLS over 100 runs in this simulation was approximately 40.28 when using PLS on each voxel independently but was approximately 2.32 when using the "Neighbour-based" method.

A few comments:

- Under further investigation, it could be seen that for a majority of cases the "Neighbour-based" method was only slightly outperforming the "Independent-voxel" method in terms of distance between the estimated and true β. However, when the "Independent-voxel" method failed, it failed wildy and unpredictably, often giving estimates differing from β by over 10.

- Cases in which the "Independent-voxel" method performed better than the "Neighbour-based" method in terms of distance between the estimated and true β were few and far between.

- This simulation may have been slightly biased in that the true β was drawn from the distributed $N(\beta_n, \frac{1}{2})$, in order to ensure there was concordance between the voxels. This may have been a favourable simulation as the derivation given previously can also be arrived at by treating β_n as an observation drawn from $N(\beta, \sigma^2)$, (see below).

- This method assumes concordance between voxels which may not be the case for rough areas of the map. However, given a value based on the estimate of smoothness between voxels, α, it may be possible to factor this into the objective function like so:

$$\text{argmin}_{\beta,\sigma^2,\Sigma}||W^{-\frac{1}{2}}(Y - X\beta - Z\Lambda_\theta u)||_2^2 + ||u||_2^2 + ||\alpha(\beta - \beta_n)||_2^2$$

  This would increase the effect of the penalty term on smooth areas of the image (penalizing vastly different estimates in smooth areas of the brain), and minimize the effect of the penalty term on rough areas (allowing for a wider search range over the parameter space in rough areas of the map). The result changes in the previous working just change I_p to αI_p and β_n to $\alpha\beta_n$ throughout. This would be equivalent to treating β_n as an observation from $N(\beta, \sigma^2\alpha^{-1})$. Perhaps a suggestion would be to choose $\alpha \approx \frac{\hat{\sigma}^2}{\hat{S}}$, where $\hat{S}$ is some reasonable measure of local smoothness.

Alternate derivation assuming $\beta_n \sim N(\beta, \sigma^2)$:

As in the proof of Statement 7 in the PLS section of chapter 2, $Y|U \sim N(X\beta + Z\Lambda_\theta u, \sigma^2 W^{-1})$ and $U \sim N(0, \sigma^2 I)$, and have PDFs given by:

$$f_{Y|U}(y|u) = \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n}{2}}}\exp\left[\frac{-\|W^{\frac{1}{2}}(y - X\beta - Z\Lambda_\theta)\|_2^2}{2\sigma^2}\right]$$

$$f_U(u) = \frac{1}{(2\pi\sigma^2)^{\frac{q}{2}}}\exp\left[\frac{-\|u\|_2^2}{2\sigma^2}\right]$$

If in addition we treat the precalculated β_n as an observation of B_n $N(\beta, \sigma^2)$, we have the following PDF for β_n:

$$f_{B_n}(\beta_n) = \frac{1}{(2\pi\sigma^2)^{\frac{p}{2}}} \exp\left[\frac{-\|\beta - \beta_n\|_2^2}{2\sigma^2}\right]$$

By the definition of conditional density it is true that:

$$\begin{aligned} f_{Y,B_n,U}(y, \beta_n, u) &= f_{Y,B_n|U}(y, \beta_n|u) f_U(u) \\ &= f_{Y|U}(y|u) f_{B_n|U}(\beta_n|u) f_U(u) \\ &= f_{Y|U}(y|u) f_{B_n}(\beta_n) f_U(u) \\ = \left(\frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left[\frac{-\|W^{\frac{1}{2}}(y - X\beta - Z\Lambda_\theta)\|_2^2}{2\sigma^2}\right]\right)&\left(\frac{1}{(2\pi\sigma^2)^{\frac{q}{2}}} \exp\left[\frac{-\|u\|_2^2}{2\sigma^2}\right]\right)\left(\frac{1}{(2\pi\sigma^2)^{\frac{p}{2}}} \exp\left[\frac{-\|\beta - \beta_n\|_2^2}{2\sigma^2}\right]\right) \\ &= \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n+q+p}{2}}} \exp\left[\frac{-\|W^{\frac{1}{2}}(y - X\beta - Z\Lambda_\theta)\|_2^2 - \|u\|_2^2 - \|\beta - \beta_n\|_2^2}{2\sigma^2}\right] \\ &= \frac{|W|^{\frac{1}{2}}}{(2\pi\sigma^2)^{\frac{n+q}{2}}} \exp\left[\frac{-\tilde{r}^2(\theta, \beta, u)}{2\sigma^2}\right] \end{aligned}$$

Where the equality between the 1^{st} and 2^{nd} lines follow as B_n and Y are independent (they have related population parameters but beyond that have no commonality), the equality between the 2^{nd} and 3^{rd} is as B_n does not depend on u and finally $\tilde{r}^2$ is the suggested modified objective function.

From here, $f_{Y,B_n}(y, \beta_n)$ can be obtained as a function of $\tilde{r}$ by integrating out u in the same manor as in the proof of statement 7. Following this, the log-likelihood, $\mathcal{L}(\theta, \beta, \sigma^2|y, \beta_n) := \log(f_{Y,B_n}(y, \beta_n))$, can be calculated and seen to be minimized only when $\tilde{r}^2$ is minimized. See the proof of statement 7 for further details.

3.8 PLS speed improvement: Diagonalized broadcasting

Problem: Matrix broadcasting is not available for sparse matrices in 'cvxopt'. This means that, for the PLS algorithm, currently the speed for doing the same computation for all voxels is the speed of the computation for one voxel multiplied by the number of voxels. This could be improved.

Idea 1: The packages 'sparse' and 'dask' claim to have distributed functions for sparse matrices. A majority of the computation may be able to be done with these packages. However, the sparse cholesky decomposition cannot be done in this manor and may require conversion to 'cvxopt' sparse arrays and looping.

Idea 2: The idea for this is relatively straightforward. Consider two separate linear mixed model problems of the form:

$$\underset{\beta_1,\sigma_1^2,\theta_1}{\arg\min}\, ||Y_1 - X_1\beta_1 - Z_1\Lambda_{1\theta_1}u_1||_2^2 + ||u_1||_2^2$$

And:

$$\underset{\beta_2,\sigma_2^2,\theta_2}{\arg\min}\, ||Y_2 - X_2\beta_2 - Z_2\Lambda_{2\theta_2}u_2||_2^2 + ||u_2||_2^2$$

This problem can be expressed alternatively as:

$$\underset{\beta_1,\sigma_1^2,\theta_1,\beta_2,\sigma_2^2,\theta_2}{\arg\min} \left|\left| \begin{bmatrix} Y_1 & 0 \\ 0 & Y_2 \end{bmatrix} - \begin{bmatrix} X_1 & 0 \\ 0 & X_2 \end{bmatrix} \begin{bmatrix} \beta_1 & 0 \\ 0 & \beta_2 \end{bmatrix} - \begin{bmatrix} Z_1 & 0 \\ 0 & Z_2 \end{bmatrix} \begin{bmatrix} \Lambda_{2\theta_2} & 0 \\ 0 & \Lambda_{2\theta_2} \end{bmatrix} \begin{bmatrix} u_1 & 0 \\ 0 & u_2 \end{bmatrix} \right|\right|_2^2 + \left|\left| \begin{bmatrix} u_1 & 0 \\ 0 & u_2 \end{bmatrix} \right|\right|_2^2$$

Thus solving the above, would be expected to give the same results. In fact, by noting that the sparse cholesky decomposition (spChol) satisfies:

$$\text{spChol}\left(\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \right) = \begin{bmatrix} \text{spChol}(A) & 0 \\ 0 & \text{spChol}(B) \end{bmatrix}$$

It can be noted that performing PLS on the above expression is doing a majority of exactly the same computations as when performing PLS on the two seperate mixed model formulations. However, it will most likely be doing them faster.

There is no reason to stack the models for only two voxels together in this way, it is easy to generalise the above to n voxels in the same manner. If this works, in fact, it may even reduce storage space as, for example, $Y'Y$ is now only one value for n voxels. Controlling the stopping criteria may be more difficult in this case however if using `scipy` optimize.

Obviously, this idea wouldn't be possible for all voxels at once (the Y vectors and X vectors alone may become massive very quickly... that said $X'X$ and $X'Y$ may not become so large). However, given this, even just 10-20 at a time may improve speed drastically.

3.9 Demidenko book error

$$\text{cov}\left[\text{vec}\left(\tfrac{\delta l}{\delta D}\right)\right] = \text{cov}\left[\mathcal{D}\text{vech}\left(\tfrac{\delta l}{\delta D}\right)\right] \qquad (\text{As } \text{vec}(M) = \mathcal{D}\text{vech}(M))$$

$$= \mathcal{D}\text{cov}\left[\text{vech}\left(\frac{\delta l}{\delta D}\right)\right]\mathcal{D}'$$

$$= \tfrac{1}{2}\mathcal{D}\mathcal{D}^{+}\Sigma(R_i \otimes R_i)\mathcal{D}^{+\prime}\mathcal{D}' \qquad \text{(By Section 3.3, Demidenko 2013)}$$

$$= \tfrac{1}{2}N_n\Sigma(R_i \otimes R_i)N_n' \qquad \text{(As } \mathcal{D}\mathcal{D}^{+} = N_n\text{, where } N_n = (I_{n^2} + K_n)/2\text{)}$$

$$= \tfrac{1}{2}N_n\Sigma(R_i \otimes R_i)N_n \qquad \text{(As } N_n \text{ is symmetric)}$$

$$= \tfrac{1}{2}N_n\Sigma(R_i \otimes R_i) \qquad \text{(As } N_n(A \otimes A)N_n = N_n(A \otimes A)\text{, c.f. Magnus 1986)}$$

Bibliography