Exceedingly Simple Principal Pivot Transforms

Jan de Leeuw

Version 001, March 23, 2016

Contents

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome. The directory deleeuwpdx.net/pubfolders/pivot has a pdf version, the complete Rmd file with all code chunks, the bib file, and the R source code.

1 Single Pivots

Suppose A is an $n \times m$ matrix with $a_{kk} \neq 0$. The k^{th} *principal single pivot* or *PSP* transforms A to the $n \times m$ matrix $\mathbf{piv}_k(A)$ with elements

$$[\mathbf{piv}_k(A)]_{ij} = \begin{cases} \frac{1}{a_{kk}} & \text{if } i = k \text{ and } j = k, \\ -\frac{a_{kj}}{a_{kk}} & \text{if } i = k \text{ and } j \neq k, \\ \frac{a_{ik}}{a_{kk}} & \text{if } i \neq k \text{ and } j = k, \\ a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}} & \text{if } i \neq k \text{ and } j \neq k. \end{cases} \tag{1}$$

$\mathbf{piv}_k(A)$ is undefined if $a_{kk} = 0$. Note that the matrix A is not necessarily square, symmetric, or positive semi-definite.

If A is the matrix

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  1.00  1.00  1.00  1.00  1.00
## [2,]  1.00  2.00  2.00  2.00  2.00
## [3,]  1.00  2.00  3.00  3.00  3.00
## [4,]  1.00  2.00  3.00  4.00  4.00
## [5,]  1.00  2.00  3.00  4.00  5.00
```

then pivoting on the second diagonal element, using our R function `pivotOneRC()`, produces $\mathbf{piv}_2(A)$

```
## $a
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  0.5  0.5    0    0    0
## [2,] -0.5  0.5   -1   -1   -1
## [3,]  0.0  1.0    1    1    1
## [4,]  0.0  1.0    1    2    2
## [5,]  0.0  1.0    1    2    3
##
## $refuse
## [1] 0
```

Note that the pivot operation is an involution, i.e. it is its own inverse, or $\mathbf{piv}_k(\mathbf{piv}_k(A)) = A$.

```
mprint(pivotOneRC(pivotOneRC(a,2)$a, 2)$a)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  1.00  1.00  1.00  1.00  1.00
## [2,]  1.00  2.00  2.00  2.00  2.00
## [3,]  1.00  2.00  3.00  3.00  3.00
## [4,]  1.00  2.00  3.00  4.00  4.00
## [5,]  1.00  2.00  3.00  4.00  5.00
```

PSP's are sometimes called *sweeps*, and the `SWP()` operator for symmetric matrices was introduced in statistical computing by Beaton (1964), section 3.2. See Goodnight (1979), Stewart (1998) , p. 330–333, and Lange (2010), p. 95–99, for discussions of the sweep operator in the context of numerical linear algebra. The version of the sweep operator that became popular in statistics, however, is slightly different from the one proposed by Beaton and from the one we use here. It is

$$[\mathbf{swp}_k(A)]_{ij}=\begin{cases}-\frac{1}{a_{kk}} & \text{if } i=k \text{ and } j=k,\\ \frac{a_{kj}}{a_{kk}} & \text{if } i=k \text{ and } j\neq k,\\ \frac{a_{ik}}{a_{kk}} & \text{if } i\neq k \text{ and } j=k,\\ a_{ij}-\frac{a_{ik}a_{kj}}{a_{kk}} & \text{if } i\neq k \text{ and } j\neq k.\end{cases} \tag{2}$$

This version preserves symmetry, which is convenient because symmetry can be used to minimize storage. But the symmetric sweep of \$(2) is not an involution. The inverse operation of this sweep is the *reverse sweep*, which is

$$[\mathbf{rswp}_k(A)]_{ij}=\begin{cases}-\frac{1}{a_{kk}} & \text{if } i=k \text{ and } j=k,\\ -\frac{a_{kj}}{a_{kk}} & \text{if } i=k \text{ and } j\neq k,\\ -\frac{a_{ik}}{a_{kk}} & \text{if } i\neq k \text{ and } j=k,\\ a_{ij}-\frac{a_{ik}a_{kj}}{a_{kk}} & \text{if } i\neq k \text{ and } j\neq k.\end{cases} \tag{3}$$

In the funcion `pivotOneRC()` we have a type parameter. For type equal to 1 we do a symmetric sweep, for type equal to 2 a reverse symmetric sweep. Thus

```
mprint (pivotOneRC(a, 2, type = 1)$a)
```

```
##      [,1]  [,2]  [,3]  [,4]  [,5]
## [1,]  0.50  0.50  0.00  0.00  0.00
## [2,]  0.50 -0.50  1.00  1.00  1.00
## [3,]  0.00  1.00  1.00  1.00  1.00
## [4,]  0.00  1.00  1.00  2.00  2.00
## [5,]  0.00  1.00  1.00  2.00  3.00
```

```
mprint (pivotOneRC(pivotOneRC(a, 2, type = 1)$a, 2, type = 2)$a)
```

```
##      [,1]  [,2]  [,3]  [,4]  [,5]
## [1,]  1.00  1.00  1.00  1.00  1.00
## [2,]  1.00  2.00  2.00  2.00  2.00
## [3,]  1.00  2.00  3.00  3.00  3.00
## [4,]  1.00  2.00  3.00  4.00  4.00
## [5,]  1.00  2.00  3.00  4.00  5.00
```

Type equal to 3 is the PSP we started out with, and type equal to 4 is its transpose, the involution

$$[\mathbf{qiv}_k(A)]_{ij} = \begin{cases} \frac{1}{a_{kk}} & \text{if } i = k \text{ and } j = k, \\ \frac{a_{kj}}{a_{kk}} & \text{if } i = k \text{ and } j \neq k, \\ -\frac{a_{ik}}{a_{kk}} & \text{if } i \neq k \text{ and } j = k, \\ a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}} & \text{if } i \neq k \text{ and } j \neq k. \end{cases} \tag{4}$$

The default in `pivotOneRC()` is to set type equal to 3, and we will use this default throughout the rest of the paper.

2 Sequences of pivots

Our R function `pivotRC()` performs a sequence of pivots on a matrix. If both PSP's are defined then $\mathbf{piv}_k(\mathbf{piv}_\ell(A)) = \mathbf{piv}_\ell(\mathbf{piv}_k(A))$, and thus the results are independent of the order in which we pivot. We obtain numerical stability by always pivoting on the largest diagonal element in the remainder of the sequence of pivots. This also ensures the function gives a sensible result if the matrices involved are singular.

Consider the matrix we get from our previous one by setting element $(1, 1)$ to zero. The resulting matrix is still non-singular.

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,] 0.00 1.00 1.00 1.00 1.00
## [2,] 1.00 2.00 2.00 2.00 2.00
## [3,] 1.00 2.00 3.00 3.00 3.00
## [4,] 1.00 2.00 3.00 4.00 4.00
## [5,] 1.00 2.00 3.00 4.00 5.00
```

If we pivot on the first four elements we get

```
##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] -2.00  1.00  0.00  0.00  0.00
## [2,]  1.00  1.00 -1.00  0.00  0.00
## [3,]  0.00 -1.00  2.00 -1.00 -0.00
## [4,] -0.00  0.00 -1.00  1.00 -1.00
## [5,]  0.00  0.00  0.00  1.00  1.00
```

and pivots are done in the order 4, 2, 1, 3.

Another example is the rank 2 matrix

```
mprint(a<-tcrossprod(matrix(c(1,1,1,1,1,-1,-1,1),4,2))/2)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  1.00  0.00  0.00  1.00
## [2,]  0.00  1.00  1.00  0.00
## [3,]  0.00  1.00  1.00  0.00
## [4,]  1.00  0.00  0.00  1.00
```

A complete pivot uses the order 1, 2, 3, 4 and finds

```
mprint(h$a)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  1.00 -0.00  0.00 -1.00
## [2,] -0.00  1.00 -1.00 -0.00
## [3,]  0.00  1.00  0.00  0.00
## [4,]  1.00  0.00  0.00  0.00
```

Note that if the function `pivotOneRC()` is asked to pivot on a zero diagonal element, it refuses and just returns its argument. Thus if zero diagonal elements are encountered, it is just as if the corresponding indices are not in the sequence, and we actually pivot on a shorter sequence. `pivotRC()` returns an indicator of the pivots that are skipped, which for our last example is 0, 0, 1, 1.

Also note that order-invariance for `pivotRC()` is no longer true if pivots are skipped. Consider

```
##      [,1] [,2] [,3] [,4]
## [1,]  0.00  0.00  0.00  1.00
## [2,]  0.00  0.00  0.00  1.00
## [3,]  0.00  0.00  0.00  1.00
## [4,]  1.00  1.00  1.00  1.00
```

Pivoting over all indices, but in a different order, gives different results.

```
pivotRC(a, 1:4)
```

```
## $a
##      [,1] [,2] [,3] [,4]
## [1,]   -1   -1   -1    1
## [2,]    1    0    0    0
## [3,]    1    0    0    0
## [4,]    1    0    0    0
##
## $pivot
## [1] 4 1 2 3
##
## $skip
## [1] 0 0 1 1
```

```
pivotRC(a, 4:1)
```

```
## $a
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    1    0
## [2,]    0    0    1    0
## [3,]   -1   -1   -1    1
## [4,]    0    0    1    0
##
## $pivot
## [1] 4 3 2 1
##
## $skip
## [1] 0 0 1 1
```

In the first case we compute $\mathbf{piv}_{\{1,4\}}(A)$, in the second case $\mathbf{piv}_{\{3,4\}}(A)$, and these are different.

Suppose

$$A = \begin{bmatrix} \alpha & a' \\ b & C \end{bmatrix}, \tag{5}$$

with $\alpha \neq 0$. Then

$$\mathbf{piv}_1(A) = \begin{bmatrix} \frac{1}{\alpha} & \frac{-a'}{\alpha} \\ \frac{b}{\alpha} & C - \frac{ba'}{\alpha} \end{bmatrix}.$$

The Schur complement (Ouellette (1981), Zhang (2005)) of $\frac{1}{\alpha}$ in $\mathbf{piv}_1(A)$ is C. Since rank is additive over the Schur complement, we have

$$\mathbf{rank}(\mathbf{piv}_1(A)) = 1 + \mathbf{rank}(C),$$

and thus $\mathbf{piv}_1(A)$ is non-singular whenever A is.

For a symmetric A in (5) we have that A is positive definite if and only if $\alpha > 0$ and $C - \frac{ba'}{\alpha}$ is positive definite (Bekker (1988)). Thus if A is positive definite, all subsequent pivot elements are positive. If A is unit triangular, either upper or lower, then $C - \frac{ba'}{\alpha}$ is unit triangular as well, and thus all subsequent pivot elements are equal to one.

Of course the choice of pivot $k = 1$ in (5) was for notational convenience only, the results are true for any k.

3 The partial inverse

Suppose K is a subset of $N := \{1, 2, \cdots, n\}$ and $\underline{K}$ is the complimentary subset. In our examples we usually take, for convenience, $K = \{1, 2, \cdots, k\}$. The *principal pivot transform* or *PPT* of A on K, which generalizes the simple pivot, is

$$\mathbf{piv}_K(A) = \begin{bmatrix} A_{KK}^{-1} & -A_{KK}^{-1}A_{K\underline{K}} \\ A_{\underline{K}K}A_{KK}^{-1} & A_{\underline{K}\underline{K}} - A_{\underline{K}K}A_{KK}^{-1}A_{K\underline{K}} \end{bmatrix}.$$

The PPT is defined only if A_{KK} is non-singular. A *complete* PPT is defined as $\mathbf{piv}_K()$ for $K = N$.

The PPT of a square matrix A is called the *partial inverse* by Wermuth, Wiedenbeck, and Cox (2006) and Wiedenbeck and Wermuth (2010). The name makes sense because if

$$\begin{bmatrix} A_{KK} & A_{\underline{K}} \\ A_{\underline{K}K} & A_{\underline{K}\underline{K}} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} u \\ v \end{bmatrix},$$

then

$$\begin{bmatrix} A_{KK}^{-1} & -A_{KK}^{-1}A_{K\underline{K}} \\ A_{\underline{K}K}A_{KK}^{-1} & A_{\underline{K}\underline{K}} - A_{\underline{K}K}A_{KK}^{-1}A_{K\underline{K}} \end{bmatrix} \begin{bmatrix} u \\ y \end{bmatrix} = \begin{bmatrix} x \\ v \end{bmatrix},$$

And conversely. Of course all this assumes that A_{KK} is non-singular.

The following results are true for arbitrary index sets K and L and their complement $\underline{K}$ and $\underline{L}$. For a proof, see Wermuth, Wiedenbeck, and Cox (2006), lemma 2.1. They assume that all principal submatrices are invertible, which is true for positive definite matrices and unit triangular matrices. In that case all PPT's are defined.

- $A = \mathbf{piv}_K(\mathbf{piv}_K(A))$
- $\mathbf{piv}_K(A) = (\mathbf{piv}_{\underline{K}}(A))^{-1}$
- $\mathbf{piv}_{\underline{K}}(\mathbf{piv}_K(A)) = A^{-1}$
- $\mathbf{piv}_L(\mathbf{piv}_K(A)) = \mathbf{piv}_K(\mathbf{piv}_L(A))$
- $\mathbf{piv}_K(A) = \mathbf{piv}_{\underline{K}}(A^{-1})$
- $(\mathbf{piv}_K(A))^{-1} = \mathbf{piv}_K(A^{-1})$

Wermuth, Wiedenbeck, and Cox (2006) summarize the main properties of partial inversion in their theorem 2.2. Suppose K, L, and M partition the n indices of a square matrix, and the PPT's are defined Then

- Commutativity: $\mathbf{piv}_K(\mathbf{piv}_L(A)) = \mathbf{piv}_K(\mathbf{piv}_L(A)) = \mathbf{piv}_H(A)$.
- Exchangeability: $[\mathbf{piv}_K(A)]_{K\cup L,K\cup L} = \mathbf{piv}_K(A_{K\cup L,K\cup L})$.
- Symmetric Difference: $\mathbf{piv}_{K\cup L}(A)\mathbf{piv}_{L\cup M}(A) = \mathbf{piv}_{K\cup M}(A)$.

If A is not square, then some of these results still make sense. For both a *broad* $n \times m$ matrix A, with $n < m$, and a *tall* A, with $n > m$, we can pivot on the diagonal elements and the defining equations for the principal pivot transform still apply.

Note that a PPT can be computed by a sequence of PSP's, provided all PSP's on the way are defined. But consider the example

```
##      [,1] [,2] [,3] [,4]
## [1,] 0.00 1.00 1.00 0.00
## [2,] 1.00 0.00 0.00 1.00
## [3,] 1.00 0.00 1.00 0.00
## [4,] 0.00 1.00 0.00 1.00
```

Now $\mathbf{piv}_1(A)$ and $\mathbf{piv}_2(A)$ are both not defined, and thus our function `pivotRC(a, 1:2)` does nothing, while

$$\mathbf{piv}_{\{1,2\}}(A) = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & -1 \\ 1 & 0 & -1 & 1 \end{bmatrix}.$$

4 Inverse and determinant

For a square matrix

$$A = \begin{bmatrix} \alpha & a' \\ b & C \end{bmatrix},$$

with $\alpha \neq 0$, by Schur's determinant formula,

$$\mathbf{det}(A) = \alpha\ \mathbf{det}(C - \frac{ba'}{\alpha}),$$

and thus $\mathbf{det}(A)$ is the product of the pivots in a complete PPT.

If A is square and non-singular, then carrying out a complete PPT produces the inverse of the matrix, or $\mathbf{piv}_N(A) = A^{-1}$. This assumes, of course, that a complete PPT can actually be carried out and we do not run into a zero pivot.

If A is singular, say of rank r, then we can permute rows and columns so that we have the leading principal submatrix of order r is non-singular. This easily follows from the pivoted LU decomposition, see Stewart (1998), theorem 2.13. If

$$A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$$

with B an $r \times r$ matrix such that $\mathbf{rank}(B) = \mathbf{rank}(A) = r$, and a pivot of A over the first r elements is possible, then

$$\mathbf{piv}_{\{1,2,\cdots,r\}}(A) = \begin{bmatrix} B^{-1} & -B^{-1}C \\ DB^{-1} & 0 \end{bmatrix}.$$

This implies that $A\left(\mathbf{piv}_{\{1,2,\cdots,r\}}(A)\right)A = A$, i.e. $\mathbf{piv}_{\{1,2,\cdots,r\}}(A)$ satisfies the first Penrose condition, i.e. is a (1)-inverse of A.

5 Least Squares

Statisticians like the **piv**() and **swp**() operators, because of their connection with linear least squares regression, and with stepwise regression in particular (Jennrich (1977)). In addition there are connections with covariance selection (Dempster (1972)) and with graphical models (Wermuth, Wiedenbeck, and Cox (2006)).

For the least squares problem of minimizing $(y - Xb)'(y - Xb)$ the normal equations are $X'Xb = X'y$. Suppose $X = (X_1 \mid X_2)$, with m_1 and m_2 columns, and consider the matrix

$$A = \begin{bmatrix} X_1'X_1 & X_1'X_2 & X_1'y \\ X_2'X_1 & X_2'X_2 & X_2'y \\ y'X_1 & y'X_2 & y'y \end{bmatrix}$$

Pivoting on the m_1 elements in $X_1'X_1$, assuming it is invertible, gives

$$\mathbf{piv}_{\{1,2,\cdots,m\}}(A) = \begin{bmatrix} (X_1'X_1)^{-1} & -(X_1'X_1)^{-1}X_1'X_2 & -(X_1'X_1)^{-1}X_1'y \\ X_2'X_1(X_1'X_1)^{-1} & X_2'(I - X_1(X_1'X_1)^{-1}X_1')X_2 & X_2'(I - X_1(X_1'X_1)^{-1}X_1')y \\ y'X_1(X_1'X_1)^{-1} & y'(I - X_1(X_1'X_1)^{-1}X_1')X_2 & y'(I - X_1(X_1'X_1)^{-1}X_1')y \end{bmatrix}.$$

All submatrices in the PPT give statistics which are relevant in the context of the least squares problem. We have the regression coefficients, their dispersion matrix, the residual sum of squares, and the partial covariances of X_2 given X_1. Moreover an additional PSP on one of the m_2 elements from X_2 will add a variable to the regression, and repeating a PSP from the first m_1 will remove that variable from the regression.

6 Code

The basic PSP routine `pivotOneC()` is written in C. If a pivot is zero, it is skipped. Performing a number of pivots to get a PPT is done in `pivotC()`, also in C. The `.C()` interface is used in R to write wrappers `pivotOneRC()` and `pivotRC()` for the C routines. Note that the `ggm` package from CRAN (Marchetti, Drton, and Sadeghi (2015)) has a `swp()` function that uses `solve()` from base R to compute the partial inverse. Pivoting on principal submatrices to compute the partial inverse will generally be faster than building up the partial inverse from pivoting on a sequence of diagonal elements like we do.

In previous work ((**deleeuw_E_15?**)) we implemented sweeping by writing a simple R wrapper for the fortran routine in AS 178 (Clarke (1982)). We improve this by switching to principal pivoting, to C, and to incorporating reordering the remaining pivots for numerical stability, similar to the suggestions of Ridout and Cobby (1989).

6.1 R Code

```r
dyn.load ("pivot.so")

pivotOneRC <- function (a, k, type = 3, eps = 1e-10) {
  n <- nrow (a)
  m <- ncol (a)
  h <-
    .C(
      "pivotOneC",
      a = as.double (a),
      k = as.integer(k),
      n = as.integer (n),
      m = as.integer (m),
      type = as.integer (type),
      refuse = as.integer (0),
      eos = as.double (eps)
    )
  return (list(a = matrix(h$a, n, m), refuse = h$refuse))
}

pivotRC <- function (a, ind, type = 3, eps = 1e-10) {
  n <- nrow (a)
  m <- ncol (a)
  p <- length (ind)
  done <- rep (0, p)
  h <-
    .C(
      "pivotC",
      a = as.double (a),
      ind = as.integer (ind),
      jnd = as.integer (rep (0, p)),
      done = as.integer (rep (0, p)),
      n = as.integer (n),
      m = as.integer (m),
      p = as.integer (p),
      type = as.integer (type),
      skip = as.integer (rep (0, p)),
      eps = as.double (eps)
    )
  return (list(a = matrix(h$a, n, m), pivot = h$jnd, skip = h$skip))
}
```

6.2 C Code

```c
void pivotOneC (double *, int *, int *, int *, int *, int *, double *eps);
void pivotC (double *, int *, int *, int *, int *, int *, int *, int *, int *, double *e
void mprint (double *, int *, int *);

void pivotOneC (double *a, int *kk, int *nn, int *mm, int *type, int *refuse, double *ep
    int i, j, ik, kj, ij, kp, s00, s01, s10, n = *nn, m = *mm, k = *kk, tp = *type;
    double pv, ee = *eps;
    *refuse = 0;
    if (tp == 1) {
        s00 = -1;
        s01 = 1;
        s10 = 1;
    }
    if (tp == 2) {
        s00 = -1;
        s01 = -1;
        s10 = -1;
    }
   if (tp == 3) {
        s00 = 1;
        s01 = -1;
        s10 = 1;
    }
   if (tp == 4) {
        s00 = 1;
        s01 = 1;
        s10 = -1;
    }
    kp = MINDEX(k, k, n);
    pv = a[kp];
    if (fabs (pv) < ee) {
        *refuse = 1;
        return;
    }
    for (i = 1; i <= n; i++) {
        if (i == k) continue;
        ik = MINDEX (i, k , n);
        for (j = 1; j <= m; j++) {
            if (j == k) continue;
            kj = MINDEX (k, j, n);
            ij = MINDEX (i, j, n);
            a[ij] = a[ij] - a[ik] * a[kj] / pv;
        }
    }
    for (i = 1; i <= n; i++) {
        if (i == k) continue;
        ik = MINDEX (i, k, n);
        a[ik] = s10 * a[ik] / pv;
    }
    for (j = 1; j <= m; j++) {
        if (j == k) continue;
        kj = MINDEX (k, j, n);
        a[kj] = s01 * a[kj] / pv;
    }
    a[kp] = s00 / pv;
}


void pivotC (double *a, int *ind, int *jnd, int *done, int *nn, int *mm, int *pp, int *t
    int ipiv, kpiv, lpiv, refuse;
    int n = *nn, m = *mm, p = *pp;
    double fmax, fpiv;
    for (int l = 0; l < p; l++) {
        jnd[l] = 0;
        done[l] = 0;
        skip[l] = 0;
    }
    for (int l = 1; l <= p; l++) {
        fmax = -1.0;
        for (int k = 1; k <= p; k++) {
            kpiv = ind[k - 1];
            if (done[k - 1] == 1) continue;
            fpiv = fabs (a[MINDEX (kpiv, kpiv, n)]);
            if (fpiv > fmax) {
                ipiv = k;
                lpiv = kpiv;
                fmax = fpiv;
            }
        }
        done[ipiv - 1] = 1;
        jnd [l - 1] = lpiv;
        pivotOneC(a, &lpiv, &n, &m, type, &refuse, eps);
        skip[l - 1] = refuse;
    }
}
```

```
void mprint (double * a, int *nn, int *mm) {
    int i, j, ij, n = *nn, m = *mm;
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= m; j++) {
            ij = MINDEX (i, j, n);
            printf (" %5.3f ", a[ij]);
        }
    printf ("\n");
    }
}
```

7 NEWS

References

Beaton, A. E. 1964. "The Use of Special Matrix Operators in Statistical Calculus." RB 64-51. Princeton, NJ: Educational Testing Service.

Bekker, P. A. 1988. "The Positive Semidefiniteness of Partitioned Matrices." *Linear Algebra and Its Applications* 111: 261–78.

Clarke, M. R. B. 1982. "Algorithm AS 178: The Gauss-Jordan Sweep Operator with Detection of Collinearity." *Journal of the Royal Statistical Society. Series C (Applied Statistics),* 31: 166–68.

Dempster, A. P. 1972. "Covariance Selection." *Biometrics* 28: 157–75.

Goodnight, J. H. 1979. "A Tutorial on the SWEEP Operator." *American Statistician* 33: 149–58.

Jennrich, R. I. 1977. "Stepwise Regression." In *Statistical Methods for Digital Computers*, edited by Enslein K., A. Ralston, and H. S. Wilf. Wiley.

Lange, K. 2010. *Numerical Analysis for Statisticians.* Second Edition. Springer.

Marchetti, G. M., M. Drton, and K. Sadeghi. 2015. *ggm: Functions for Graphical Markov Models.* https://CRAN.R-project.org/package=ggm.

Ouellette, D. V. 1981. "Schur Complements and Statistics." *Linear Algebra and Its Applications* 36: 187–295.

Ridout, M. S., and J. M. Cobby. 1989. "Remark AS R78: A Remark on Algorithm AS 178: The Gauss-Jordan Sweep Operator with Detection of Collinearity." *Journal of the Royal Statistical Society. Series C (Applied Statistics),* 38: 420–22.

Stewart, G. W. 1998. *Matrix Algorithms. Volume i: Basic Decompositions.* SIAM.

Wermuth, N., M. Wiedenbeck, and D. R. Cox. 2006. "Partial Inversion for Linear Systems and Partial Closures of Independence Graphs." *BIT Numerical Mathematics* 46: 883–901.

Wiedenbeck, M., and N. Wermuth. 2010. "Changing Parameters by Partial Mappings." *Statistica Sinica* 20: 823–36.

Zhang, F., ed. 2005. *The Schur Complement and its Applications.* Vol. 4. Numerical Methods and Algorithms. Springer.