OLS.c

```
/*
*/
#include <stdio.h>
#include <R_ext/Lapack.h>
#include <R_ext/BLAS.h>

int main(){

        int i,info, ipiv[2];
        char trans = 't', notrans ='n';
        double alpha = 1.0, beta=0.0;
        int ncol=2;
        int nrow=5;
        int one=1;
        double XprimeX[4];
        double X[10] = {1,1,1,1,1,0.3,-0.2,0.4,-0.5,0.3};
        double Y[5] = {0.7,-0.5,0.9,-1.1,0.7};
        double XXinv[4] = {1,0,0,1};
        double XXinvX[10];
        double coef[2];

        printf("\n\nX = ");
        for(i=0;i<5;i++) printf("\n%f %f", X[i],X[i+5]);
        printf("\n\nY = ");
        for(i=0;i<5;i++) printf("\n%f", Y[i]);

        dgemm_(&trans,&notrans,&ncol,&ncol,&nrow,&alpha,X,&nrow,X,&nrow,&beta,
XprimeX,&ncol);
        printf("\n\nX'X = ");
        for(i=0;i<2;i++) printf("\n%f %f",XprimeX[i], XprimeX[i+2]);

        dgesv_(&ncol,&ncol,XprimeX,&ncol,ipiv,XXinv,&ncol,&info);
        printf("\n\n(X'X)-1 = ");
        for(i=0;i<2;i++) printf("\n%f %f",XXinv[i], XXinv[i+2]);


//XXinv is 2x2
//X' is 2x5
//X is 5x2

        dgemm_(&notrans,&trans,&ncol,&nrow,&ncol,&alpha,XXinv,&ncol,X,&nrow,&b
eta,XXinvX,&ncol);

//XXinvX is 2x5
//Y is 5x1

        dgemm_(&notrans,&notrans,&ncol,&one,&nrow,&alpha,XXinvX,&ncol,Y,&nrow,
&beta,coef,&nrow);
        printf("\n\nB0 = %f", coef[0]);
        printf("\nB1 = %f\n\n", coef[1]);

        return(0);
}
```

OLS.c (annotated)

```
/*

C:\docs_c_summer_course>gcc -I"c:/program files/R/R-2.9.0/include" -
L"C:/Program
 Files/R/R-2.9.0/bin" -Wall ols.c -o ols.exe -lRlapack -lRblas

C:\docs_c_summer_course>ols


X =
1.000000 0.300000
1.000000 -0.200000
1.000000 0.400000
1.000000 -0.500000
1.000000 0.300000

Y =
0.700000
-0.500000
0.900000
-1.100000
0.700000

X'X =
5.000000 0.300000
0.300000 0.630000

(X'X)-1 =
0.205882 -0.098039
-0.098039 1.633987

B0 = 0.003922
B1 = 2.267974


*/
#include <stdio.h>
/*  Here is the path statement to the LAPACK and BLAS header files:
/*  C:/Program Files/R/R-2.9.0/include/R_ext/ /*
#include <R_ext/Lapack.h>
#include <R_ext/BLAS.h>

int main(){

        int i,info, ipiv[2];
/* These are used in the function call for dgemm – it tells the function
whether or not the indicated matrices are to be transposed or not */
        char trans = 't', notrans ='n';
        double alpha = 1.0, beta=0.0;
        int ncol=2;
        int nrow=5;
        int one=1;
        double XprimeX[4];
```

```
        double X[10] = {1,1,1,1,1,0.3,-0.2,0.4,-0.5,0.3};
        double Y[5] = {0.7,-0.5,0.9,-1.1,0.7};
        double XXinv[4] = {1,0,0,1};
        double XXinvX[10];
        double coef[2];

        printf("\n\nX = ");
        for(i=0;i<5;i++) printf("\n%f %f", X[i],X[i+5]);
        printf("\n\nY = ");
        for(i=0;i<5;i++) printf("\n%f", Y[i]);
/* This BLAS function takes 3 matrices, A, B, and C, and two scalars, alpha
and beta, and computes: Y = alpha*(AB) + beta*C. Note that the result, the
matrix Y, is returned in C.  Here A=X_transpose, B=X, C=XprimeX (note that
XprimeX is empty), alpha=1, beta=0. The function then returns
Y = (1.0)*(X_transpose*X) + (0.0)*XprimeX = X_transposeX  */
        dgemm_(&trans,&notrans,&ncol,&ncol,&nrow,&alpha,X,&nrow,X,&nrow,&beta,
XprimeX,&ncol);
        printf("\n\nX'X = ");
        for(i=0;i<2;i++) printf("\n%f %f",XprimeX[i], XprimeX[i+2]);
/* This LAPACK function solves the linear system:  Y=A*BETA, where A is a
square n by n matrix of real numbers, BETA is a matrix of coefficients that
is n by m (m<=n), and Y is an n by m matrix.  The solution matrix, BETA, is
returned in Y.  Here Y=I, that is, Y is set equal to the n by n identity
matrix, the result is that BETA = A_inverse is returned in Y.  Below,
A=XprimeX, and XXinv=I when the function is called.  The function returns
XprimeX_inverse in XXinv */
        dgesv_(&ncol,&ncol,XprimeX,&ncol,ipiv,XXinv,&ncol,&info);
        printf("\n\n(X'X)-1 = ");
        for(i=0;i<2;i++) printf("\n%f %f",XXinv[i], XXinv[i+2]);


//XXinv is 2x2
//X' is 2x5
//X is 5x2
/* Same BLAS function as above.  Here A=(X'X)-1 (XXinv), B=X_transpose,
C=XXinvX (note that XXinvX is empty), alpha=1, beta=0. The function then
returns Y = (1.0)*[(X'X)-1]X') + (0.0)*XXinvX = [(X'X)-1]X') in XXinvX  */

        dgemm_(&notrans,&trans,&ncol,&nrow,&ncol,&alpha,XXinv,&ncol,X,&nrow,&b
eta,XXinvX,&ncol);

//XXinvX is 2x5
//Y is 5x1
/* Same BLAS function as above.  Here A=[(X'X)-1]X') (XXinvX), B=Y, C=coef
(note that coef is empty), alpha=1, beta=0. The function then returns
Y = (1.0)*( [(X'X)-1]X')Y) + (0.0)*coef = coef, the two Beta coefficients  */

        dgemm_(&notrans,&notrans,&ncol,&one,&nrow,&alpha,XXinvX,&ncol,Y,&nrow,
&beta,coef,&nrow);
        printf("\n\nB0 = %f", coef[0]);
        printf("\nB1 = %f\n\n", coef[1]);
        return(0);

}
```

Subroutine DGEMM from BLAS Library

```
OLS.C --dgemm_(&trans,&notrans,&ncol,&ncol,&nrow,&alpha,X,&nrow,X,&nrow,&beta,XprimeX,&ncol);

      SUBROUTINE DGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
*     .. Scalar Arguments ..
      DOUBLE PRECISION ALPHA,BETA
      INTEGER K,LDA,LDB,LDC,M,N
      CHARACTER TRANSA,TRANSB
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION A(LDA,*),B(LDB,*),C(LDC,*)
*     ..
*
*  Purpose
*  =======
*
*  DGEMM  performs one of the matrix-matrix operations
*
*     C := alpha*op( A )*op( B ) + beta*C,
*
*  where  op( X ) is one of
*
*     op( X ) = X   or   op( X ) = X',
*
*  alpha and beta are scalars, and A, B and C are matrices, with op( A )
*  an m by k matrix,  op( B )  a  k by n matrix and  C an m by n matrix.
*
*  Arguments
*  ==========
*
*  TRANSA - CHARACTER*1.
*           On entry, TRANSA specifies the form of op( A ) to be used in
*           the matrix multiplication as follows:
*
*              TRANSA = 'N' or 'n',  op( A ) = A.
*
*              TRANSA = 'T' or 't',  op( A ) = A'.
*
*              TRANSA = 'C' or 'c',  op( A ) = A'.
*
*           Unchanged on exit.
*
*  TRANSB - CHARACTER*1.
*           On entry, TRANSB specifies the form of op( B ) to be used in
*           the matrix multiplication as follows:
*
*              TRANSB = 'N' or 'n',  op( B ) = B.
*
*              TRANSB = 'T' or 't',  op( B ) = B'.
*
*              TRANSB = 'C' or 'c',  op( B ) = B'.
*
```

```
*           Unchanged on exit.
*
*  M      - INTEGER.
*           On entry,  M  specifies  the number  of rows  of the  matrix
*           op( A )  and of the  matrix  C.  M must be at least zero.
*           Unchanged on exit.
*
*  N      - INTEGER.
*           On entry,  N  specifies the number  of columns of the matrix
*           op( B ) and the number of columns of the matrix C. N must be
*           at least zero.
*           Unchanged on exit.
*
*  K      - INTEGER.
*           On entry,  K  specifies  the number of columns of the matrix
*           op( A ) and the number of rows of the matrix op( B ). K must
*           be at least  zero.
*           Unchanged on exit.
*
*  ALPHA  - DOUBLE PRECISION.
*           On entry, ALPHA specifies the scalar alpha.
*           Unchanged on exit.
*
*  A      - DOUBLE PRECISION array of DIMENSION ( LDA, ka ), where ka is
*           k  when  TRANSA = 'N' or 'n',  and is  m  otherwise.
*           Before entry with  TRANSA = 'N' or 'n',  the leading  m by k
*           part of the array  A  must contain the matrix  A,  otherwise
*           the leading  k by m  part of the array  A  must contain  the
*           matrix A.
*           Unchanged on exit.
*
*  LDA    - INTEGER.
*           On entry, LDA specifies the first dimension of A as declared
*           in the calling (sub) program. When  TRANSA = 'N' or 'n' then
*           LDA must be at least  max( 1, m ), otherwise  LDA must be at
*           least  max( 1, k ).
*           Unchanged on exit.
*
*  B      - DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where kb is
*           n  when  TRANSB = 'N' or 'n',  and is  k  otherwise.
*           Before entry with  TRANSB = 'N' or 'n',  the leading  k by n
*           part of the array  B  must contain the matrix  B,  otherwise
*           the leading  n by k  part of the array  B  must contain  the
*           matrix B.
*           Unchanged on exit.
*
*  LDB    - INTEGER.
*           On entry, LDB specifies the first dimension of B as declared
*           in the calling (sub) program. When  TRANSB = 'N' or 'n' then
*           LDB must be at least  max( 1, k ), otherwise  LDB must be at
*           least  max( 1, n ).
*           Unchanged on exit.
*
*  BETA   - DOUBLE PRECISION.
*           On entry,  BETA  specifies the scalar  beta.  When  BETA  is
```

```
*           supplied as zero then C need not be set on input.
*           Unchanged on exit.
*
*  C      - DOUBLE PRECISION array of DIMENSION ( LDC, n ).
*           Before entry, the leading  m by n  part of the array  C must
*           contain the matrix  C,  except when  beta  is zero, in which
*           case C need not be set on entry.
*           On exit, the array  C  is overwritten by the  m by n  matrix
*           ( alpha*op( A )*op( B ) + beta*C ).
*
*  LDC    - INTEGER.
*           On entry, LDC specifies the first dimension of C as declared
*           in  the  calling  (sub)  program.   LDC  must  be  at  least
*           max( 1, m ).
*           Unchanged on exit.
*
*
*  Level 3 Blas routine.
*
*  -- Written on 8-February-1989.
*     Jack Dongarra, Argonne National Laboratory.
*     Iain Duff, AERE Harwell.
*     Jeremy Du Croz, Numerical Algorithms Group Ltd.
*     Sven Hammarling, Numerical Algorithms Group Ltd.
*
```

Subroutine DGESV from LAPACK Library

CALLED IN OLS.C: **dgesv_(&ncol,&ncol,XprimeX,&ncol,ipiv,XXinv,&ncol,&info);**

```
      SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*
*  -- LAPACK driver routine (version 3.2) --
*  -- LAPACK is a software package provided by Univ. of Tennessee,    --
*  -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
*     November 2006
*
*     .. Scalar Arguments ..
      INTEGER            INFO, LDA, LDB, N, NRHS
*     ..
*     .. Array Arguments ..
      INTEGER            IPIV( * )
      DOUBLE PRECISION   A( LDA, * ), B( LDB, * )
*     ..
*
*  Purpose
*  =======
*
*  DGESV computes the solution to a real system of linear equations
*     A * X = B,
*  where A is an N-by-N matrix and X and B are N-by-NRHS matrices.
*
*  The LU decomposition with partial pivoting and row interchanges is
*  used to factor A as
*     A = P * L * U,
*  where P is a permutation matrix, L is unit lower triangular, and U is
*  upper triangular.  The factored form of A is then used to solve the
*  system of equations A * X = B.
*
*  Arguments
*  =========
*
*  N       (input) INTEGER
*          The number of linear equations, i.e., the order of the
*          matrix A.  N >= 0.
*
*  NRHS    (input) INTEGER
*          The number of right hand sides, i.e., the number of columns
*          of the matrix B.  NRHS >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the N-by-N coefficient matrix A.
*          On exit, the factors L and U from the factorization
*          A = P*L*U; the unit diagonal elements of L are not stored.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,N).
*
*  IPIV    (output) INTEGER array, dimension (N)
*          The pivot indices that define the permutation matrix P;
*          row i of the matrix was interchanged with row IPIV(i).
```

```
*
*  B       (input/output) DOUBLE PRECISION array, dimension (LDB,NRHS)
*          On entry, the N-by-NRHS matrix of right hand side matrix B.
*          On exit, if INFO = 0, the N-by-NRHS solution matrix X.
*
*  LDB     (input) INTEGER
*          The leading dimension of the array B.  LDB >= max(1,N).
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*          > 0:  if INFO = i, U(i,i) is exactly zero.  The factorization
*                has been completed, but the factor U is exactly
*                singular, so the solution could not be computed.
*
*  =====================================================================
*
```