4.4 Sequential Preprocessing of Linear Least-Squares Data

A. Purpose

Let a linear least-squares problem be denoted by

$$A\mathbf{x} \simeq \mathbf{b}$$

where A is a given $m \times n$ matrix with $m \geq n$, $\mathbf{b}$ is a given m-vector (or $m \times nb$ matrix) and it is required to find an n-vector (or $n \times nb$ matrix), $\mathbf{x}$, that is an approximate solution to this equation in the least-squares sense. The given data for this problem can be regarded as the composite matrix, $[A : \mathbf{b}]$.

The subroutine, SACCUM or DACCUM, can be used to preprocess the data matrix, $[A : \mathbf{b}]$, sequentially, *i.e.*, it can process the matrix by individual rows or by blocks of rows. This is useful in problems in which m is much larger than n and the product $m \times n$ is so large that it would be inconvenient or impossible to allocate storage arrays of total size $(m \times n) + (m \times nb)$ to hold the data matrix, $[A : \mathbf{b}]$. It is also useful if it is desired to have the processing begin before the value of m is known. This subroutine can function using total array storage space as small as $(n + 2) \times (n + nb)$ floating-point numbers.

The result of this preprocessing is an upper-triangular array of data defining a least-squares problem that is equivalent to the original problem in the sense that it has the same solution, the same covariance matrix, and the same condition number. The transformed problem will have at most $n + 1$ rows. The resulting triangular system can be solved directly if there is no concern about the problem being ill-conditioned. Otherwise, one can apply the subroutines of Chapters 4.2 or 4.3 to the transformed problem.

B. Usage

B.1 Program Prototype, Single Precision

INTEGER LDA, N, LDB, NB, IR1, NROWS, NCOUNT

REAL A(LDA, ≥N)**, B**(LDB, ≥NB) or **B**(LDB)

On the initial call to SACCUM for a new problem set LDA, N, LDB, NB, IR1= 1, and NROWS. On the initial call and all subsequent calls accumulating data for the same problem, set NROWS and store NROWS rows of new data into A(,) and B(,) beginning at row IR1.

CALL SACCUM(A, LDA, N, B, LDB, NB, IR1, NROWS, NCOUNT)

Following the call the contents of A(,) and B(,) will have been altered reflecting the processing of the new data. The value of IR1 may have been increased.

B.2 Argument Definitions

A(,) [inout] On each call to SACCUM the user assigns an integer value to NROWS and places NROWS rows of the data matrix, $[A : \mathbf{b}]$, into rows IR1 through IR1 + NROWS − 1 of A(,) and B(,).

On each return the contents of A(,) and B(,) and possibly the value of IR1 will have been modified. The output data in the first IR1 − 1 rows of A(,) and B(,) will constitute a least-squares problem having the same solution as the problem represented by the data accumulated so far. The elements to the left of the diagonal in rows 2 through IR1 − 1 of A(,) will be zero on return.

LDA [in] The leading dimensioning parameter for the array A(,). LDA must be at least as large as the largest value the expression IR1 + NROWS − 1 will ever have on entry to SACCUM during the processing of the current problem. Note that IR1 will never exceed N + 2. If μ is the largest value that will be assigned to NROWS, then it suffices to set LDA ≥ N + 1 + μ.

N [in] Number of columns in the data matrix, A.

B(,) [inout] See discussion above for A(,).

LDB [in] Leading dimensioning parameter for the array B(,). Must satisfy the same constraints as described above for LDA.

NB [in] Number of columns in the right-side data matrix, B. Set NB = 1 if the right-side is a single vector. If NB = 0 the array B(,) will not be referenced.

IR1 [inout] Must be set to the value 1 before the first call to SACCUM for a problem. On each return IR1 will be updated by SACCUM to the value min(IR1 + NROWS, N + 2). See discussion of A(,) above for the meaning and usage of IR1.

NROWS [in] The number of rows of new data being provided to SACCUM on the current entry. The user must set this value on the initial entry for a new problem. After that the user may leave it unchanged or change it on any entry.

NCOUNT [inout] Total number of rows of data provided so far for the current problem. This is simply the sum of the values of NROWS provided on all entries to SACCUM for the current problem. The subroutine initializes the counting when entered with IR1 = 1.

B.3 Changes for Double Precision

For double precision usage change the REAL statement to DOUBLE PRECISION and change the subroutine name from SACCUM to DACCUM.

C. Examples and Remarks

As an example of the use of SACCUM the program DRSACCUM computes a least-squares fit to a set of twelve data points by a seventh degree polynomial. The output is shown in ODSACCUM.

For simplicity we have chosen to process just one row at a time. Thus NROWS has the value 1 in each call to SACCUM. It would be more efficient to process more rows on each call.

To investigate the dependence of execution time on the setting of NROWS we solved a 200×5 problem with two right-side vectors using a number of different settings of NROWS. The transformed problem was then solved using DHFTI. For comparison we also solved the problem directly using DHFTI. We found the time for solution processing only one row at a time was about 1.7 times the time to solve it directly. The time to solve it processing 10 rows at a time was about 1.2 times the direct solution time. These ratios might differ substantially in different computing environments.

D. Functional Description

This subroutine uses orthogonal transformations to process the given data, producing an equivalent least-squares problem. This method of sequential accumulation is treated in detail in [1]. To avoid complications we discuss here only the usual case in which the total number of rows of data accumulated exceeds n. Then the transformed problem will be of the form

$$\begin{bmatrix} R \\ 0 \end{bmatrix} \mathbf{x} \simeq \begin{bmatrix} \mathbf{y} \\ \alpha \end{bmatrix}$$

where R is an $n \times n$ upper triangular matrix, $\mathbf{y}$ is an n-vector, and α is a scalar quantity. The matrix R will be in the array A(,) and $\mathbf{y}$ and α will be in B(,).

For enhanced efficiency the subroutine uses Givens orthogonal transformations when processing a small number of new rows of data and Householder orthogonal transformations when processing a larger set of new rows.

The transformed quantities are related to the data, $[A : \mathbf{b}]$, by the relations, $R^t R = A^t A$, $R^t \mathbf{y} = A^t \mathbf{b}$, and $\mathbf{y}^t \mathbf{y} + \alpha^2 = \mathbf{b}^t \mathbf{b}$.

The solution, $\mathbf{x}$, for the transformed problem is also the solution for the original least-squares problem, $A\mathbf{x} \simeq \mathbf{b}$. If the matrix A is of rank N and sufficiently well-conditioned, the solution can be computed by solving the triangular system, $R\mathbf{x} = \mathbf{y}$. Alternatively the transformed system can be analyzed and/or solved using subroutines from Chapters 4.2 or 4.3.

The residual vector for the transformed least-squares problem is $\begin{bmatrix} 0 \\ \alpha \end{bmatrix}$.

The norm of this residual vector is $|\alpha|$ and this is also equal to $\|\mathbf{b} - A\mathbf{x}\|$.

References

1. Charles L. Lawson and Richard J. Hanson, **Solving Least-Squares Problems**, Prentice-Hall, Englewood Cliffs, N. J. (1974) 340 pages.

E. Error Procedures and Restrictions

IR1 must be set to 1 on the first call to SACCUM and must not be altered subsequently by the user during the processing for one problem.

SACCUM will return immediately, with no error message, if NROWS ≤ 0.

Let k = IR1 + NROWS $-$ 1. This will be the index of the last row of A(,) and B(,) containing new data on entry to SACCUM. If $k >$ LDA or $k >$ LDB, the subroutine will issue an error message using the error processing routines of Chapter 19.2 with an error level of 0 and return, doing no computation.

F. Supporting Information

The source language is ANSI Fortran 77.

Entry	Required Files
DACCUM	DACCUM, DHTCC, DNRM2, DROTG, ERFIN, ERMSG, IERM1, IERV1
SACCUM	ERFIN, ERMSG, IERM1, IERV1, SACCUM, SHTCC, SNRM2, SROTG

These subroutines are adaptations to the JPL MATH77 library of the algorithms that were developed by C. L. Lawson and R. J. Hanson at JPL in 1972 and described in detail in [1].

DRSACCUM

```
c      program DRSACCUM
c>> 1996-06-18 DRSACCUM Krogh Special code for C conversion.
c>> 1996-05-28 DRSACCUM Krogh Added external state. & moved up formats
c>> 1995-09-15 DRSACCUM Krogh Remove '0' in format (again?)
c>> 1994-10-19 DRSACCUM Krogh  Changes to use M77CON
c>> 1994-08-09 DRSACCUM WVS Removed '0' in formats
c>> 1991-11-20 DRSACCUM CLL Edited for Fortran 90.
c>> 1987-12-09 DRSACCUM Lawson  Initial Code.
c      Demonstration driver for SACCUM.
c--S replaces "?": DR?ACCUM, ?ACCUM, ?HFTI, ?COPY, ?MPVAL
c      ------------------------------------------------------------------
      external SMPVAL
      real             SMPVAL
      integer NMAX, LDIM, LPMAX, NB, MDATA
      parameter(NMAX = 8, LDIM = NMAX + 2, NB = 1, MDATA = 12)
      parameter(LPMAX = NMAX + 2)
      real             X(MDATA),Y(MDATA),P(LPMAX)
      real             A(LDIM,NMAX),B(LDIM)
      real             RNORM(NB), WORK(NMAX)
      real             DOF, R, SIGFAC, TAU, U, YFIT
      integer I, IP(NMAX), IR1, IROW, J, KRANK, N, NCOUNT, NDEG, NROWS
      parameter(TAU = 1.0E-5)
c      ------------------------------------------------------------------
      data X / 2.0E0,  4.0E0,  6.0E0,  8.0E0,  10.0E0,  12.0E0,
     *       14.0E0, 16.0E0, 18.0E0, 20.0E0,  22.0E0,  24.0E0/
      data Y / 2.2E0,  4.0E0,  5.0E0,  4.6E0,  2.8E0,    2.7E0,
     *        3.8E0,  5.1E0,  6.1E0,  6.3E0,  5.0E0,    2.0E0/
      data P(1), P(2) / 13.0E0, 11.0E0 /
c
c      ------------------------------------------------------------------
      N = NMAX
      NDEG = N - 1
      IR1 = 1
      NROWS = 1
c
      do 20 IROW = 1,MDATA
        U = (X(IROW)-P(1)) / P(2)
        I = IR1
        A(I,1) = 1.
        do 10 J = 2, NDEG+1
            A(I,J) = A(I,J-1)*U
   10   continue
        B(I) = Y(IROW)
        call SACCUM(A, LDIM, N, B, LDIM, NB, IR1, NROWS, NCOUNT)
   20 continue
c
      print*,'DRSACCUM..  Demo driver for SACCUM.'
      print '(1x,a,i4,a,i4)','MDATA = ',MDATA,', NCOUNT = ',NCOUNT
      call SHFTI(A,LDIM,IR1-1,N,B,LDIM,NB,TAU,KRANK,RNORM,WORK,IP)
      print '(1x,a,I4)','KRANK = ',KRANK
c
c                       The following stmt does a type conversion.
      DOF = NCOUNT - N
      SIGFAC = RNORM(1) / sqrt(DOF)
      call SCOPY(N, B, 1, P(3), 1)
c++ Code for .C. is inactive
```

```
c%%    printf(
c%%       "\n NDEG =%2ld             RNORM =%8.4f             SIGFAC =%8.4f",
c%%        ndeg, Rnorm[1], sigfac );
c%%    printf(
c%%    "\n\n P(1),P(2) =           %15.5f%15.5f\n\n P(3),...,P(NDEG+3) =",
c%%    p[0], p[1]);
c%%    for (i = 2; i < (n + 2); i+=3){
c%%       for (j = i; j <= (i < n ? i+2 : n+1); j++)
c%%            printf("%15.5f", p[j] );
c%%            if (i < n-1) printf("\n                              ");}
c%%    printf( "\n" );
c++ Code for ~.C. is active
      print
     *'(/'' NDEG ='',I2,10X,''RNORM ='',F8.4,10X,''SIGFAC ='',F8.4//
     *'' P(1),P(2) ='',9X,2F15.5//'' P(3),...,P(NDEG+3) ='',3F15.5/
     *(21X,3F15.5))', NDEG,RNORM(1),SIGFAC,(P(I),I=1,N+2)
c++ End
      print '(1X/''   I     X          Y          YFIT   R=Y-YFIT''/1X)'
      do 30 I=1,MDATA
         YFIT=SMPVAL(P,NDEG,X(I))
         R=Y(I)-YFIT
         print '(1X,I2,F6.0,2F9.3,F10.3)', I,X(I),Y(I),YFIT,R
   30 continue
      stop
c
      end
```

ODSACCUM

```
DRSACCUM..  Demo driver for SACCUM.
MDATA =    12, NCOUNT =    12
KRANK =     8

NDEG = 7            RNORM =  0.4432            SIGFAC =  0.2216

P(1),P(2) =                13.00000        11.00000

P(3),...,P(NDEG+3) =        3.04312         5.42386        16.15689
                          -23.08771       -28.52199        36.65454
                           11.42051       -19.09685

  I     X          Y          YFIT   R=Y-YFIT

  1     2.     2.200     2.205    -0.005
  2     4.     4.000     3.959     0.041
  3     6.     5.000     5.147    -0.147
  4     8.     4.600     4.333     0.267
  5    10.     2.800     3.028    -0.228
  6    12.     2.700     2.699     0.001
  7    14.     3.800     3.651     0.149
  8    16.     5.100     5.156    -0.056
  9    18.     6.100     6.196    -0.096
 10    20.     6.300     6.187     0.113
 11    22.     5.000     5.048    -0.048
 12    24.     2.000     1.992     0.008
```