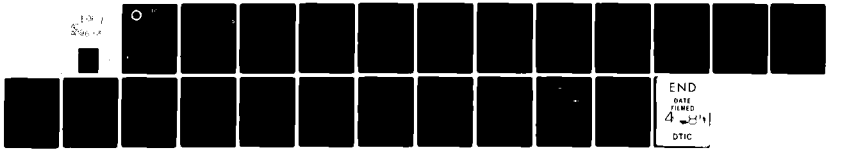
AD-A096 131 STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB F/G 12/1
MODULES FOR USE WITH MINOS/AUGMENTED IN SOLVING SEQUENCES OF MA--ETC(U)
NOV 80 P V PRECKEL DAAG29-79-C-0110
UNCLASSIFIED SOL-80-15 NL



END
DATE FILMED
4-81
DTIC

Systems
Optimization
Laboratory

LEVEL II

AD A096131

DTIC FILE COPY

Department of Operations Research
Stanford University
Stanford, CA 94305

81 3 3 044

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

Modules for use with MINOS/AUGMENTED in Solving Sequences of Mathematical Programming

by

Paul V. Preckel

TECHNICAL REPORT SOL 80-15

November 1980

Research and reproduction of this report were supported by the Department of Energy Contract DE-AC03-76SF00326, PA# DE-AT03-76ER72018; Office of Naval Research Contract N00014-75-C-0267; National Science Foundation Grants MCS-7926009, ECS-8012974 (formerly ENG77-06761), and SOC 78-16811; Army Research Office Contract DAA29-79-C-0110.

Modules for Use with MINOS/AUGMENTED in Solving Sequences of Mathematical Programs

Paul V. Preckel

Department of Operations Research
Stanford University
Stanford, California 94305

ABSTRACT

The modules documented herein are designed for use with the nonlinear programming system MINOS/AUGMENTED when implementing algorithms requiring the solution of sequences of similar mathematical programs. Large-scale constrained mathematical programming requires the use of special data structures which make in-core problem modification difficult. These modules allow in-core modification of nonlinear programs without knowledge of the specialized data structures used in MINOS/AUGMENTED.

The modifications addressed by these modules are: modification of elements in the constraint matrix, linear objective or right hand side; modification of bounds on individual variables; modification of the nonlinear functions; and appending columns to the constraint matrix. The user specifies the flow and nature of problem modifications via a Fortran subroutine. A simple example of the use of each of the modules is presented.

This manual supplements Reports SOL 77-9 and SOL 80-14, the MINOS User's Guide and the MINOS/AUGMENTED User's Manual.

CONTENTS

1. OVERVIEW

It has become apparent that it would be useful to have a nonlinear programming code suitable for solving a sequence of mathematical programs of the following form; at the k-th step

$$\begin{aligned} \text{minimize} \quad & F^k(x) + (c^k)^T x \\ \text{subject to} \quad & f^k(x) + A^k x = b^k \\ & l^k \leq x \leq u^k \end{aligned} \tag{P^k}$$

where l^k, u^k, c^k are n vectors, b^k is an m vector, and A^k is an m by n matrix, and where F^k and f^k are continuous and differentiable over the feasible region. Dantzig-Wolfe Decomposition [1] for large–scale linear programming and the Manne-Chao-Wilson algorithm for solving for competitive equilibria [2] are two algorithms that require solving a sequence of similar mathematical programs.

The author has created a set of four subroutines to assist the implementation of algorithms involving the solution of sequences of mathematical programs which are substantially the same. Their use avoids the need to learn the data structures involved in expressing the sparse problem, and they provide a framework for the efficient implementation of such algorithms.

These subroutines are available on the MINOS Distribution Tape. The tape is documented in Report SOL 80-100.

2. ON UNDERSTANDING THIS SYSTEM

Modification of the problem P^k to express P^{k+1} can be achieved by a combination of:

1. modifying the bounds l^k and u^k;
2. adding constraints;
3. adding columns to A^k;
4. modifying elements of A^k, c^k and b^k; and
5. modifying the nonlinear functions F^k and f^k.

Operation 2 is implemented as a combination of 1 and 4 as follows. Declare a free row and input elements which will not be rejected as too small in the original matrix A^1. (See note regarding AIJTOL in §4.3.) Note that a free row is a constraint whose corresponding logical variable has a *very* large negative lower bound, and a *very* large positive upper bound. Row type "N" in the MPS file corresponds to a free row. (See SOL 77-9.) After the k-th problem is solved, and it is desired to add the row, the matrix elements are individually modified to

create the proper coefficients and right-hand-side. The final step which completes the addition of this constraint to the problem is the modification to the bounds of the logical variable associated with the row to be added.

The other transformations are straight forward. The reader is directed to the subroutine specifications found later in this report.

Rule to remember—one can not enlarge or change the "non-zero pattern" in a column once it has been added to the problem.

3. ADDITIONAL PROBLEM SPECIFICATIONS

The problem specifications, or SPECS file, is fully documented in SOL 77-9, and this information is supplemented in SOL 80-14. A list of the additional keywords useful for solving sequences of mathematical programs follows with a short explanation of each.

Note: The syntax for the SPECS file is discussed in SOL 77-9.

`CYCLE LIMIT` k (default $k = 1$)

This parameter indicates that the maximum number of problems to be solved is k.

`CYCLE PRINT` j (default $j = 1$)

This parameter controls printing. At most, the last j solutions will be output.

After returning from a subroutine call to `MATMOD` which returns `FINISH=.FALSE.`, the system checks to see if the maximum number of cycles minus the current number of calls to `MATMOD` is less than j. If so, then the next solution is printed. After a call to `MATMOD` which returns `FINISH=.TRUE.`, if the solution from the previous cycle was not printed, it is printed now.

`CYCLE TOLERANCE` t (default $t = 0.0$)

The value t is passed to `MATMOD` as the parameter `CNVTOL` for use in a user specified convergence test. This gives one the facility to vary the convergence criterion for the cycling algorithm without recompiling the `MATMOD` routine. (See §4.1.)

`PHANTOM COLUMNS` k (default $k =$ `PHANTOM ELEMENTS`$/10$)

This parameter specifies the expected number of columns to be added beyond those in the original MPS input deck.

PHANTOM ELEMENTS k (default $k = 10 \times$ **PHANTOM COLUMNS**)

This parameter specifies the total number of nonzero elements to be allocated to the "phantom columns" after input of the original matrix. It should include space for possible linear objective entries.

Note: If both the **PHANTOM COLUMNS** and the **PHANTOM ELEMENTS** parameters are not specified, both will be set to 0. Also, during input of the MPS file, it may be desirable to accept "non-zero" elements of A^1 which are arbitrarily small in the absolute sense, or even zero. Normally these are not stored. However, one can suppress the rejection of small elements by manipulating the parameter **AIJ TOLERANCE** (see SOL 77-9). Set the value of this parameter to zero to accept arbitrarily small non-zero values as "non-zeros". Set the value of this parameter to a negative number to accept any matrix element as a "non-zero" regardless of its absolute size. (Remember, one can not change the "non-zero pattern" of a column once it has become part of the current problem.)

4. SUBROUTINE DOCUMENTATION

The following six subroutines provide a framework for problem modification. **MATMOD** is provided by the user, and **MINOS** (the subroutine) may be modified by the user if necessary. The other subroutines are to be called where needed within **MATMOD**. These subroutines are available on the MINOS Distribution Tape. (See SOL 80-100.)

4.1 SUBROUTINE MATMOD

This subroutine is provided by the user to direct the flow and nature of problem modifications. The constraint matrix, bounds, linear objective, right-hand-side and/or the nonlinear functions of the problem can be modified within **MATMOD**.

Specification:

```
      SUBROUTINE MATMOD( NCYCLE,CNVTOL,FINISH,
     1   M,N,NROWS,NCOLS,NE,NP1,
     2   A,BL,BU,HA,KA,ID1,ID2,
     3   PI,XL,XS,Z,MAXZ,LFREE )
      IMPLICIT    REAL*8(C-G,O-Z)
      REAL        A(NE),BL(N),BU(N)
      REAL*8      PI(NROWS),XL(NROWS),XS(NCOLS),Z(MAXZ)
      INTEGER*2   HA(NE)
      INTEGER     KA(NP1),ID1(N),ID2(N)
```

```
LOGICAL      FINISH
```

Declarations of variable type REAL*8 should not be used when single precision is adequate; e.g. on Burroughs and CDC machines. Similarly, INTEGER*2 should be replaced by INTEGER when not available.

Parameters:

CNVTOL (Input) This constant is input from the SPECS file (see page 3, CYCLE TOLERANCE) to be used within MATMOD to determine "convergence" of the user's algorithm.

FINISH (Output) This logical variable is used to signal convergence of the user's algorithm. Set FINISH = .TRUE. to break out of the problem solution loop and output the final solution according to the SOLUTION keyword in the SPECS file. After FINISH is set to .TRUE. no more mathematical programs are solved. FINISH is initialized to be .FALSE. in MATMOD.

NCYCLE (Input) This variable is equal to (the number of mathematical programs solved)+1. For example, NCYCLE is 1 on the first entry to MATMOD before any mathematical programs have been solved.

NCOLS (Input) The number of columns (excluding logicals and the right-hand-side) in the problem. Note that PHANTOM COLUMNS are counted in NCOLS.

NROWS (Input) The number of rows in the problem, including the linear objective, if it exists.

XL(*) (Input) The solution values of the logical (slack or artificial) variables.

XS(*) (Input) The solution values of the structural variables.

PI(*) (Input) The dual solution to the mathematical program.

Others (Input/Output) Parameters for the subroutines called by MATMOD. Note that the constraint and bound data lies in this storage. It is a good idea not to try to manipulate this storage except via the subroutines documented herein.

Note: XL(*), XS(*) and PI(*) may be used as work vectors within MATMOD.

For some applications it may be important to stop solution of the sequence of mathematical programs if the previous program was not solved as intended. This can be accomplished by testing the common variable IERR in common block LPCOM. If the value of this variable is not zero, then some error condition (e.g., infeasibility, unboundedness, too many iterations) has arisen within the MINOS/AUGMENTED system. In general, it is a good idea to declare common block LPCOM in MATMOD (see example) and insert the following statement as the first line of executable code:

```
IF (IERR.NE.0) STOP
```

4.2 SUBROUTINE MINOS

This subroutine, provided on the MINOS distribution tape, is fully documented within the source code. Standard MINOS/AUGMENTED contains subroutine MINOS similar to the one documented in Report SOL 77-31. One important enhancement is that it allows for a "flying start", wherein the basis, superbasis, Lagrange multiplier estimates and reduced Hessian approximation are retained from the *previous* problem, P^k, to provide an advanced start for P^{k+1}.

If all problem modifications during the sequence modify only the nonlinear functions based on the values of the *nonlinear variables*, then standard MINOS/AUGMENTED may be used as is, and subroutine MATMOD is unnecessary. (See §7 for details.)

If problem modifications include modifying the linear constraints or bounds on individual variables, and/or if the nonlinear functions can not be modified solely on the basis of the optimal values of the *nonlinear variables* for the previous problems, then a special version of subroutine MINOS should be used. Files 13–15 of the tape contain this special version of subroutine MINOS along with the skeleton of MATMOD and the modules MKCOL, MODBND and MODELM.

Normally neither of these two versions of subroutine MINOS will be changed. For a variety of reasons, the user may wish to suppress the flying start feature. This can be accomplished by replacing NCYCLE by 1 in the call to subroutine DRIVER.

4.3 SUBROUTINE MKCOL

This subroutine is provided to facilitate adding columns to the constraint matrix within MATMOD. Given the vector ACOL, below, store the elements of this vector which are larger than AIJTOL, in the absolute sense, in A and HA. Components smaller than AIJTOL are not stored. Effectively, add the vector corresponding to ACOL, as a new column, to the constraint matrix. The bounds corresponding to the new column are set to the default values for structural variables. (See SOL

77-9.) Note that it is necessary to provide for PHANTOM COLUMNS if this subroutine is used. If the phantom column space runs out, MKCOL prints a message to that effect and flags an error to stop solution of the next mathematical program. Similarly, if the new column has no components larger than AIJTOL, then a message is printed and an error is flagged.

Specification:

```
SUBROUTINE MKCOL( A,ACOL,BL,BU,HA,KA,LENCOL,N,NE,NP1 )
IMPLICIT   REAL*8(C-G,O-Z)
INTEGER*2  HA(NE)
INTEGER    KA(NP1)
REAL       A(NE),ACOL(LENCOL),BL(N),BU(N)
```

Parameters:

ACOL(*) (Input) The new column to be appended to A^k.

LENCOL (Input) This parameter indicates the length of the column to be appended to the matrix. Note that LENCOL must satisfy LENCOL $\leq$ NROWS.

Others (Input/Output) Data to be modified via this subroutine.

For some applications it may be useful to read in and store some zeros in the original matrix, A^1. To accomplish this, one method is to set the parameter "AIJ TOLERANCE" in the SPECS file to a negative number. However, this is the same tolerance (AIJTOL) used to filter "zero" elements in new columns by MKCOL. Hence, to avoid storing all elements of the new column regardless of size, the common variable AIJTOL should be reset within MATMOD, before the first call to MKCOL. AIJTOL is located in common block MPSCOM. This is illustrated in the example of MATMOD.

4.4 SUBROUTINE MODBND

This subroutine, provided for the user to call within MATMOD, modifies an upper or lower bound for a single variable. If the index of this variable is zero, or too large (bigger than N, the number of structurals + 1 + the number of logicals), then an error message is printed and a flag is set to halt execution of the present sequence of mathematical programs.

Note that both structurals and logicals have upper and lower bounds. For each type of inequality, the logicals have the following bound structure:

$$
\begin{array}{ll}
\leq & l_i = 0 \text{ and } u_i = \texttt{PLINFY}; \\
= & l_i = 0 \text{ and } u_i = 0; \\
\geq & l_i = \text{-}\texttt{PLINFY} \text{ and } u_i = 0; \\
\text{free} & l_i = \text{-}\texttt{PLINFY} \text{ and } u_i = \texttt{PLINFY}.
\end{array}
$$

PLINFY is a very large number (usually 1.0E+20) and is used to specify infinite bounds. It is located in the common block EPSCOM. (All common blocks are listed in the source code of subroutine MINOS for reference.)

Specification:

```
SUBROUTINE MODBND( IMOD,BVAL,BL,BU,N )
IMPLICIT   REAL*8(C-G,O-Z)
INTEGER    IMOD
REAL       BL(N),BU(N),BVAL
```

Parameters:

IMOD (Input) This parameter is the number of the variable whose bound is to be modified with a possible change in sign. If IMOD is negative then the lower bound will be changed; IMOD positive indicates the upper bound will be changed.

BVAL (Input) This parameter is the new value of the bound indicated by IMOD.

Others (Input/Output) Data to be modified via this subroutine.

4.5 SUBROUTINE MODELM

This subroutine, provided for the user to call within MATMOD, resets an *existing* element of either the constraint matrix, the linear portion of the objective function, or the right-hand-side, to a given value. The effect of this subroutine is equivalent to setting "A(I,J) = AVAL".

Note that the linear objective and right-hand-side are imbedded in A^k as a free row and as a column with upper and lower bounds both equal to minus one respectively.

Specification:

```
SUBROUTINE MODELM( A,I,J,AVAL,HA,KA,M,N,NE,NP1 )
IMPLICIT   REAL*8(C-G,O-Z)
INTEGER*2  HA(NE)
INTEGER    I,J,KA(NP1)
REAL       A(NE),AVAL
```

Parameters:

I	(Input) The row number of the element to be modified.
J	(Input) The column number of the element to be modified.
AVAL	(Input) The new value for the element in row I and column J.
Others	(Input/Output) Data to be modified via this subroutine.

Note: There must already exist in the matrix an element in row I and column J before this subroutine is called. If not, an error message is printed, and execution of the current sequence of mathematical programs is stopped.

4.6 SUBROUTINE NMSRCH

This subroutine can be called from MATMOD to determine the index of a variable with a given name. It is documented in SOL 77-31. Since the phantom column names are not stored in-core until after P^1 has been solved, a call to NMSRCH to find the index of a phantom column via its name will not succeed until that time. Determining the index of a phantom column is better accomplished by computing its value from pointers available in common. (See §5.)

5. ON STORAGE AND INDICES

Constraint rows and objective rows of the matrix are indexed in the order that they are read from the MPS input file. Rows with no elements specified will be retained and indexed. Variables corresponding to columns in the MPS deck are numbered in the order that they are read. A column with no elements is assigned a zero in row one. (A warning message will be printed in this case unless the variable is specified to be nonlinear. In any case, execution continues.) Phantom columns come after the original columns and are indexed accordingly. The next "column" in the A file is the right-hand-side which is treated as a fixed variable. Finally come the slack, or logical, variables. KRHS in the common block LPCOM is the column number of the right-hand-side, and IOBJ in the common block FXCOM is the row number of the linear portion of the objective function. (IOBJ is zero if there is no linear objective.)

Note that the logical variable corresponding to the objective row is stored as column number KRHS+IOBJ. Similarly, the i-th logical variable is stored in column KRHS+i.

6. AN EXAMPLE OF MATMOD

The following example illustrates the use of subroutines MKCOL, MODBND, MODELM and NMSRCH within a user's version of MATMOD.

1. Solve the linear program;

$$\text{minimize} \quad -5x_1 - 8x_2 - 5x_3 - 6x_4 - 7x_5$$

$$\text{subject to} \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & -1 & -3 & 5 \\ 2 & 2 & 3 & 0 & 0 \\ -3 & 0 & 4 & 5 & 6 \\ -9 & 3 & -3 & 0 & -1 \\ -4 & 0 & -2 & -1 & 5 \end{bmatrix} x \leq \begin{bmatrix} 4 \\ 6 \\ 4 \\ 6 \\ 9 \\ 4 \end{bmatrix},$$

$$x \geq 0$$

2. Add the constraint;

$$-5x_1 - 8x_2 - 5x_3 - 6x_4 - 7x_5 \geq -23$$

and re-solve using the previous solution as the starting point.

3. Add a column corresponding to a new variable, x_6. Let this variable have a coefficient of 1.0 in the first constraint, and a cost coefficient of –30.0. Let all other coefficients be zero for this variable. Re-solve, again starting from the previous solution.

Here is a listing of **MATMOD** corresponding to the above example. This is followed by an appropriate SPECS file and MPS file.

```
      SUBROUTINE MATMOD( NCYCLE,CNVTOL,FINISH,
     1   M,N,NROWS,NCOLS,NE,NP1,
     2   A,BL,BU,HA,KA,ID1,ID2,
     3   PI,XL,XS,Z,MAXZ,LFREE )
      IMPLICIT   REAL*8(C-G,O-Z)
      LOGICAL    FINISH
      DIMENSION  A(NE),BL(N),BU(N)
      INTEGER*2  HA(NE)
      INTEGER    KA(NP1),ID1(N),ID2(N)
      REAL*8     PI(NROWS),XL(NROWS),XS(NCOLS),Z(MAXZ)
C
C     EXAMPLE OF USER-WRITTEN SUBROUTINE MATMOD.
C
C     THE FOLLOWING 4 COMMON BLOCKS ARE USED WITHIN MINOS
C     THEY SHOULD NOT BE OVERWRITTEN, WITH THE POSSIBLE
C     EXCEPTION OF AIJTOL IN COMMON BLOCK MPSCOM.
C
      COMMON    /FXCOM / FX,FXNONL,SINF,WTOBJ,MINIMZ,NINF,IOBJ,NPROB
      COMMON    /IOCOMM/ IREAD,IPRINT
      COMMON    /LPCOM / KRNS,NS1,MAXR,IERR,IDEBUG,LPRINT
      COMMON    /MPSCOM/ AIJTOL,BSTRUC(2),MINMAX,MLST,MER,
     1                   NAME(2),MOBJ(2),MRNS(2),MRNG(2),MBND(2)
```

```
C
C     DECLARE LOCAL STORAGE.
C
      REAL AVAL(6),ACOL(10)
      DATA NAMEA,NAMEB/4HROWN,4HUM07/
C
C     TEST FOR ERROR CONDITION WITHIN MINOS.
C
      IF (IERR.NE.0) STOP
C
C     NOTE THAT ON THE FIRST ENTRY (BEFORE A PROBLEM IS SOLVED)
C     WHEN NCYCLE = 1, WE RETURN IMMEDIATELY.
C
C     ALSO NOTE THAT THE PROBLEM HAS BEEN READ BY THIS TIME, SO
C     WE CAN RESET AIJTOL TO A POSITIVE VALUE TO CONSERVE STORAGE
C     LATER WHEN ADDING COLUMNS.
C
      AIJTOL = .00001
      IF (NCYCLE .EQ. 1) RETURN
      IF (NCYCLE .EQ. 3) GO TO 150
C
C     ON THE SECOND CALL TO MATMOD WE ADD A CONSTRAINT BY MODIFYING
C     SOME CURRENTLY EXISTING ZERO MATRIX ELEMENTS, AND  BY MODIFYING
C     THE BOUND ON THE SLACK VARIABLE CORRESPONDING TO THAT ROW.
C
      AVAL(1) = -5.0
      AVAL(2) = -8.0
      AVAL(3) = -5.0
      AVAL(4) = -6.0
      AVAL(5) = -7.0
      AVAL(6) = -23.0
C
C     NOW LET NMSRCH FIND THE INDEX FOR THE ROW NAMED ROWNUM07.
C
C     SINCE THE ROW NAMES FOLLOW THE COLUMN NAMES AND RHS NAME,
C     WE MUST SUBTRACT ONE PLUS THE NUMBER OF VARIABLES (=KRHS) FROM
C     JFOUND TO GET THE ROW INDEX.
C
      ISTART = 1
      CALL NMSRCH(N,ID1,ID2,NAMEA,NAMEB,1,NOTFND,
     1   1,1,N,ISTART,JFOUND)
C
C     CHECK TO MAKE SURE WE FOUND THE NAME.
C
      IF (JFOUND.EQ.0) GO TO 125
      JFOUND = JFOUND - KRHS
      DO 100 J=1,5
         CALL MODELM( A,JFOUND,J,AVAL(J),HA,KA,M,N,NE,NP1 )
  100 CONTINUE
      CALL MODELM( A,JFOUND,KRHS,AVAL(6),HA,KA,M,N,NE,NP1 )
C
C     COMPUTE THE INDEX OF THE SLACK VARIABLE CORRESPONDING
C     TO ROW JFOUND.
C
      JFOUND = JFOUND + KRHS
      BVAL = 0.0
      CALL MODBND( JFOUND,BVAL,BL,BU,N )
      RETURN
C
C     NAME NOT FOUND.
C
  125 WRITE(IPRINT,1000) NAMEA,NAMEB
      IERRO = IERRO + 1
      RETURN
C
C     ON THE THIRD CALL TO MATMOD (NCYCLE = 3) A COLUMN IS ADDED TO
C     THE MATRIX USING THE LAST OF THE NEW SUBROUTINES, MKCOL.
C
  150 DO 200 I=2,M
         ACOL(I) = 0.0
  200 CONTINUE
      ACOL(1) = 1.0
```

```
C
C     USE IOBJ FROM COMMON BLOCK FXCOM TO FIND THE OBJECTIVE ROW.
C
      ACOL(IOBJ) = -30.0
      LENCOL = M
      CALL MKCOL( A,ACOL,BL,BU,HA,KA,LENCOL,N,NE,NP1 )
      RETURN
C
 1000 FORMAT (1X,27HXXXX VARIABLE OR ROW INDEX ,2A4,11H NOT FOUND.)
C     END OF MATMOD
      END
```

SPECS file:

```
BEGIN CLP
   OBJECTIVE     OBJECTIV
   ROWS               10
   COLUMNS            30
   ELEMENTS          250
   SOLUTION    IF OPTIMAL
   AIJTOL        -0.0001
   CYCLE PRINT         1
   CYCLE TOLERANCE 0.001
   CYCLE LIMIT         3
   PHANTOM COLUMNS    10
   PHANTOM ELEMENTS  100
END
```

MPS file:

```
NAME          CLP
ROWS
 L  ROWNUM01
 L  ROWNUM02
 L  ROWNUM03
 L  ROWNUM04
 L  ROWNUM05
 L  ROWNUM06
 N  ROWNUM07
 N  OBJECTIV
COLUMNS
    X1        ROWNUM01   1.0        ROWNUM02   2.0
    X1        ROWNUM03   2.0        ROWNUM04  -3.0
    X1        ROWNUM05  -9.0        ROWNUM06  -4.0
    X1        ROWNUM07   0.0        OBJECTIV  -5.0
    X2        ROWNUM01   1.0        ROWNUM02   2.0
    X2        ROWNUM03   2.0        ROWNUM05   3.0
    X2        ROWNUM07   0.0        OBJECTIV  -8.0
    X3        ROWNUM01   1.0        ROWNUM02  -1.0
    X3        ROWNUM03   3.0        ROWNUM04   4.0
    X3        ROWNUM05  -3.0        ROWNUM06  -2.0
    X3        ROWNUM07   0.0        OBJECTIV  -5.0
    X4        ROWNUM01   1.0        ROWNUM02  -3.0
    X4        ROWNUM04   5.0        ROWNUM06  -1.0
    X4        ROWNUM07   0.0        OBJECTIV  -6.0
    X5        ROWNUM01   1.0        ROWNUM02   5.0
    X5        ROWNUM04   6.0        ROWNUM05  -1.0
    X5        ROWNUM06   5.0        ROWNUM07   0.0
    X5        OBJECTIV  -7.0
RHS
    RHS00001  ROWNUM01   4.0
    RHS00001  ROWNUM02   6.0
    RHS00001  ROWNUM03   4.0
    RHS00001  ROWNUM04   6.0
    RHS00001  ROWNUM05   9.0
    RHS00001  ROWNUM06   4.0
    RHS00001  ROWNUM07   0.0
ENDATA
```

7. ON MODIFYING NONLINEAR PROGRAMS

To this point the examples and modules have been aimed at modifying linear problems. The generalization to nonlinear problems is straightforward.

To communicate with CALCFG or CALCON, declare a new common block (with a non-reserved name) and append it to the declarations in MATMOD, CALCFG and CALCON. If it is determined to be advantageous to modify the nonlinear functions, MATMOD can pass information to CALCFG and CALCON via the new common block.

In some cases, the nonlinear functions could be modified directly in CALCFG or CALCON, as follows. If the SPECS file uses both of the specifications:

```
CYCLE LIMIT I
CALL FUNCTIONS WHEN OPTIMAL
```

where I is some positive integer, then a special call to CALCFG and CALCON will occur with the parameter value NSTATE = 2, whenever a subproblem P^k has been solved to optimality. (Note that this special call will also occur after P^I.) Some information in an appropriate common block could then be altered. These modifications may be based on the values for the nonlinear variables (held in array X), but the dual multipliers and strictly linear variables are inaccessible to CALCFG and CALCON. If all problem modifications for the sequence $\{P^k\}$ can be performed in CALCFG and CALCON, then standard MINOS/AUGMENTED may be used as is. (The printing parameters documented in this paper will be ignored if standard MINOS/AUGMENTED is used.)

8. SUMMARY

The modules provided are designed for use with the nonlinear programming system MINOS/AUGMENTED to ease the difficulty of efficiently implementing algorithms requiring the solution of sequences of similar mathematical programs. Using this system, all problem modifications can be performed in-core; hence, overhead for disk input/output is kept to a minimum. The modules described here make learning the specific data structures used in MINOS/AUGMENTED unnecessary.

In the case of nonlinear programs, information can be saved from problem P^k. MINOS/AUGMENTED can begin to solve the mathematical program P^{k+1} using the reduced Hessian approximation, Lagrange multiplier estimates, basis and superbasis from P^k. This flying start feature will usually result in significant computational savings.

This package was created to facilitate research in the field of solving for economic equilibria via optimization methods. Restrictions on the types of modifications of the mathematical programs in the sequence have been kept to a minimum to provide greater flexibility to users. Hence, this system could be

useful for research in other areas such as integer programming and very large scale linear programming.

ACKNOWLEDGEMENTS

The extension of MINOS to the solution of sequences of mathematical programs was suggested by Alan S. Manne [3]. Subsequent discussions with Michael A. Saunders [6] yielded the implementation strategy that has been used. I would like to thank Alan Manne for his support of this project and for his encouragement. I would like to thank Michael Saunders for many helpful suggestions and for his encouragement. The author is solely responsible for any remaining errors in this work.

APPENDIX–ADDITIONAL COMMON INFORMATION

One of the common blocks in MINOS/AUGMENTED contains items relevant to solving sequences of problems. This common block contains the control parameters for cycling and should not be overwritten. The parameters held therein are defined below.

Declaration:

```
COMMON /CYCLCM/ CNVTOL,IERRO,IFREE,MAXCY,NEPHNT,NPHANT,NPRINT
```

Variables:

CNVTOL See parameter list for **MATMOD**.

IERRO Error flag for inconsistencies in **MATMOD**.

IFREE Marks the beginning of the currently unused phantom column space.

MAXCY The maximum number of mathematical programs to be solved. The value of this parameter is defined by the **CYCLE LIMIT**.

NEPHNT The number of nonzero elements allowed for in the phantom column space. The value of this parameter is defined by the **PHANTOM ELEMENTS** keyword.

NPHANT The number of **PHANTOM COLUMNS** specified in the SPECS file.

NPRINT The value of **NPRINT** is specified by **CYCLE PRINT** in the SPECS file. The solutions for problems $P^{\mathbf{MAXCY}-\mathbf{NPRINT}}$ through $P^{\mathbf{MAXCY}}$ will be printed.

REFERENCES

[1] Dantzig, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.

[2] Manne, A. S., Chao, H. P. and Wilson, R. (1980). Computation of Competitive Equilibria by a Sequence of Linear Programs. *Econometrica*, 48, pp. 1595–1615.

[3] Manne, A. S. (1979). Private communication.

[4] Murtagh, B. A. and Saunders, M. A. (1977). MINOS User's Guide, Report SOL 77-9, Department of Operations Research, Stanford University.

[5] Murtagh, B. A. and Saunders, M. A. (1980). MINOS/AUGMENTED User's Manual, Report SOL 80-14, Department of Operations Research, Stanford University.

[6] Saunders, M. A. (1979-80). Private communications.

[7] Saunders, M. A. (1977). MINOS System Manual, Report SOL 77-31, Department of Operations Research, Stanford University.

[8] Saunders, M. A. (1980). MINOS Distribution Documentation, Report SOL 80-100, Department of Operations Research, Stanford University.

INDEX

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) SOL-80-15 ✓ 2. GOVT ACCESSION NO. AD-A096	3. RECIPIENT'S CATALOG NUMBER 432
4. TITLE (and Subtitle) (6) Modules for use with MINOS/AUGMENTED in Solving Sequences of Mathematical Programming.	5. TYPE OF REPORT & PERIOD COVERED (9) Technical Report.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Paul V. Preckel	8. CONTRACT OR GRANT NUMBER(s) (15) DAAG29-79-C-0110 ✓ N00014-75-C-0267 ✓
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research - SOL ✓ Stanford University Stanford, CA 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-047-143
11. CONTROLLING OFFICE NAME AND ADDRESS (11) U.S. Army Research Office P.O. Box 1221 Research Triangle Park, NC 27709 Office of Naval Research Department of the Navy 800 N. Quincy Street Arlington, VA 22217	12. REPORT DATE Nov[redacted]80 13. NUMBER OF PAGES 17 (12) 23 ✓ 15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

NONLINEAR PROGRAMMING
MINOS
MINOS/AUGMENTED
LINEAR PROGRAMMING
OPERATIONS RESEARCH
SEQUENCES OF MATHEMATICAL PROGRAMS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SEE ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

408765 xlt

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

SOL 80-15: Paul V. Preckel, "Modules for use with MINOS/AUGMENTED in Solving Sequences of Mathematical Programming

ABSTRACT

The modules documented herein are designed for use with the nonlinear programming system MINOS/AUGMENTED when implementing algorithms requiring the solution of sequences of similar mathematical programs. Large-scale constrained mathematical programming requires the use of special data structures which make in-core problem modification difficult. These modules allow in-core modification of nonlinear programs without knowledge of the specialized data structures used in MINOS/AUGMENTED.

The modifications addressed by these modules are: modification of elements in the constraint matrix, linear objective or right hand side; modification of bounds on individual variables; modification of the nonlinear functions; and appending columns to the constraint matrix. The user specifies the flow and nature of problem modifications via a Fortran subroutine. A simple example of the use of each of the modules is presented.

This manual supplements Reports SOL 77-9 and SOL 80-14, the MINOS User's Guide and the MINOS/AUGMENTED User's Manual.