```
BBBBBBBBBBBB        AAAAAAAAA        SSSSSSSSSSS  RRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
BBBBBBBBBBBB        AAAAAAAAA        SSSSSSSSSSS  RRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
BBBBBBBBBBBB        AAAAAAAAA        SSSSSSSSSSS  RRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
BBB         BBB   AAA         AAA   SSS          RRR         RRR         TTT        LLL
BBB         BBB   AAA         AAA   SSS          RRR         RRR         TTT        LLL
BBB         BBB   AAA         AAA   SSS          RRR         RRR         TTT        LLL
BBB         BBB   AAA         AAA   SSS          RRR         RRR         TTT        LLL
BBB         BBB   AAA         AAA   SSS          RRR         RRR         TTT        LLL
BBB         BBB   AAA         AAA   SSS          RRR         RRR         TTT        LLL
BBBBBBBBBBBB      AAA         AAA     SSSSSSSSS  RRRRRRRRRRR             TTT        LLL
BBBBBBBBBBBB      AAA         AAA     SSSSSSSSS  RRRRRRRRRRR             TTT        LLL
BBBBBBBBBBBB      AAA         AAA     SSSSSSSSS  RRRRRRRRRRR             TTT        LLL
BBB         BBB   AAAAAAAAAAAAAAA           SSS  RRR   RRR               TTT        LLL
BBB         BBB   AAAAAAAAAAAAAAA           SSS  RRR   RRR               TTT        LLL
BBB         BBB   AAAAAAAAAAAAAAA           SSS  RRR   RRR               TTT        LLL
BBB         BBB   AAA         AAA           SSS  RRR      RRR            TTT        LLL
BBB         BBB   AAA         AAA           SSS  RRR   RRR               TTT        LLL
BBB         BBB   AAA         AAA           SSS  RRR      RRR            TTT        LLL
BBBBBBBBBBBB      AAA         AAA   SSSSSSSSSSS     RRR         RRR      TTT        LLLLLLLLLLLLLLL
BBBBBBBBBBBB      AAA         AAA   SSSSSSSSSSS     RRR         RRR      TTT        LLLLLLLLLLLLLLL
BBBBBBBBBBBB      AAA         AAA   SSSSSSSSSSS     RRR         RRR      TTT        LLLLLLLLLLLLLLL
```

**FILE**ID**BASMATINV

```
BBBBBBBB      AAAAAA      SSSSSSSS  MM      MM    AAAAAA    TTTTTTTTTT     IIIIII    NN      NN  VV      VV
BBBBBBBB      AAAAAA      SSSSSSSS  MM      MM    AAAAAA    TTTTTTTTTT     IIIIII    NN      NN  VV      VV
BB      BB   AA    AA   SS        MMMM  MMMM   AA    AA        TT           II      NN      NN  VV      VV
BB      BB   AA    AA   SS        MMMM  MMMM   AA    AA        TT           II      NN      NN  VV      VV
BB      BB   AA    AA   SS        MM  MM  MM   AA    AA        TT           II      NNNN    NN  VV      VV
BB      BB   AA    AA   SS        MM  MM  MM   AA    AA        TT           II      NNNN    NN  VV      VV
BBBBBBBB     AA    AA     SSSSSS  MM      MM   AA    AA        TT           II      NN  NN  NN  VV      VV
BBBBBBBB     AA    AA     SSSSSS  MM      MM   AA    AA        TT           II      NN  NN  NN  VV      VV
BB      BB   AAAAAAAAAA         SS MM      MM   AAAAAAAAAA    TT           II      NN    NNNN  VV      VV
BB      BB   AAAAAAAAAA         SS MM      MM   AAAAAAAAAA    TT           II      NN    NNNN  VV      VV
BB      BB   AA    AA           SS MM      MM   AA    AA      TT           II      NN      NN    VV  VV      ....
BB      BB   AA    AA           SS MM      MM   AA    AA      TT           II      NN      NN    VV  VV      ....
BBBBBBBB     AA    AA   SSSSSSSS  MM      MM   AA    AA      TT        IIIIII     NN      NN      VV        ....
BBBBBBBB     AA    AA   SSSSSSSS  MM      MM   AA    AA      TT        IIIIII     NN      NN      VV        ....
LL           IIIIII     SSSSSSSS
LL           IIIIII     SSSSSSSS
LL             II     SS
LL             II     SS
LL             II     SS
LL             II     SS
LL             II       SSSSSS
LL             II       SSSSSS
LL             II             SS
LL             II             SS
LL             II             SS
LL             II             SS
LLLLLLLLLL   IIIIII   SSSSSSSS
LLLLLLLLLL   IIIIII   SSSSSSSS
```

Table of contents

```
0000     1          .TITLE  BAS$MAT_INV             ; BASIC matrix invert
0000     2          .IDENT  /1-014/                 ; File: BASMATINV.MAR Edit: MDL1014
0000     3
0000     4 ;
0000     5 ;*****************************************************************************
0000     6 ;*                                                                           *
0000     7 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000     8 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     9 ;*  ALL RIGHTS RESERVED.                                                     *
0000    10 ;*                                                                           *
0000    11 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    12 ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    13 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    14 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    15 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    16 ;*  TRANSFERRED.                                                             *
0000    17 ;*                                                                           *
0000    18 ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    19 ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    20 ;*  CORPORATION.                                                             *
0000    21 ;*                                                                           *
0000    22 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    23 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    24 ;*                                                                           *
0000    25 ;*                                                                           *
0000    26 ;*****************************************************************************
0000    27 ;
0000    28
0000    29 ;++
0000    30 ; FACILITY: BASIC code support
0000    31 ;
0000    32 ; ABSTRACT:
0000    33 ;
0000    34 ;       This module finds the inverse of a source matrix and stores the inverse
0000    35 ;       in the destination matrix (which may be the same as the source matrix).
0000    36 ;       As a side result of the inversion, this routine also calculates the
0000    37 ;       determinant of the source matrix and stores that result in the OWN
0000    38 ;       storage in the BAS$DET module.
0000    39 ;
0000    40 ; ENVIRONMENT: User Mode, AST Reentrant
0000    41 ;
0000    42 ;--
0000    43 ; AUTHOR: R. Will, CREATION DATE: 28-Aug-79
0000    44 ;
0000    45 ; MODIFIED BY:
0000    46 ;++
0000    47 ; 1-001 - Original
0000    48 ; 1-002 - Fix scaling.  RW  6-Dec-79
0000    49 ; 1-003 - Change shared external references to G^ RNH 25-Sep-81
0000    50 ; 1-004 - Add support for byte, g & h floating.  PLL 29-Sep-81
0000    51 ; 1-005 - Substitute a macro for the calls to the array fetch and store
0000    52 ;         routines.  This should speed things up.  PLL 3-Nov-81
0000    53 ; 1-006 - Change a MOVL to MOV'array_dtype'.
0000    54 ;       - Change compile-time tests in FETCH and STORE to run-time tests.
0000    55 ;         PLL 19-Jan-82
0000    56 ; 1-007 - Don't list macro expansions.  PLL 16-Mar-82
0000    57 ; 1-008 - Fix g, h floating.  PLL 13-Apr-82
```

```
0000     58 ; 1-009 - Fix hfloat problem in subroutine.  PLL 15-Apr-82
0000     59 ; 1-010 - Remove FETCH and STORE macros; they are now found in macro
0000     60 ;         library MATRIXMAC.OLB.  LB 19-May-82
0000     61 ; 1-011 - Change local storage to stack storage.  PLL 9-Jul-1982
0000     62 ; 1-012 - Use G^ for ALL externals.  SBL 16-Nov-1982
0000     63 ; 1-013 - fix call to MTH$DINT_R4 - should put parameter in R0 prior
0000     64 ;         to calling.  MDL 28-Mar-1983
0000     65 ; 1-014 - scale before calling BAS$$STORE_DET_G&H- this is a coordinated
0000     66 ;         change with BAS$DET to get G&H_floating to work.  MDL 7-Mar-1984
0000     67 ;--
```

```
0000     69          .SBTTL  DECLARATIONS
0000     70 ;
0000     71 ; INCLUDE FILES:
0000     72 ;
0000     73
0000     74          $DSCDEF                         ; define descriptor offsets
0000     75          $SFDEF                          ; use to get scale
0000     76
0000     77 ;
0000     78 ; EXTERNAL DECLARATIONS:
0000     79 ;
0000     80
0000     81          .DSABL  GBL                     ; Prevent undeclared
0000     82                                          ;  symbols from being
0000     83                                          ;  automatically global.
0000     84          .EXTRN  BAS$K_ARGDONMAT         ; signalled if all 3 blocks
0000     85                                          ;  not present in array desc
0000     86          .EXTRN  BAS$K_DATTYPERR         ; signalled if dtype of array
0000     87                                          ; isn't word long float double
0000     88          .EXTRN  BAS$K_MATDIMERR         ; signalled if dimct on any
0000     89                                          ;  array isn't 2
0000     90          .EXTRN  BAS$K_CANINVMAT         ; signalled if largest remaining
0000     91                                          ;  element in array is 0, array
0000     92                                          ;  appears to be singular
0000     93          .EXTRN  BAS$K_ARRMUSSQU         ; signalled if input array
0000     94                                          ;  was not square
0000     95          .EXTRN  BAS$STO_FA_B_R8         ; array element store for byte
0000     96          .EXTRN  BAS$STO_FA_W_R8         ; array element store for word
0000     97          .EXTRN  BAS$STO_FA_L_R8         ; array element store for long
0000     98          .EXTRN  BAS$STO_FA_F_R8         ; array element store - float
0000     99          .EXTRN  BAS$STO_FA_D_R8         ; array element store - double
0000    100          .EXTRN  BAS$STO_FA_G_R8         ; array element store - gfloat
0000    101          .EXTRN  BAS$STO_FA_H_R8         ; array element store - hfloat
0000    102          .EXTRN  BAS$FET_FA_B_R8         ; array element fetch - byte
0000    103          .EXTRN  BAS$FET_FA_W_R8         ; array element fetch - word
0000    104          .EXTRN  BAS$FET_FA_L_R8         ; array element fetch - long
0000    105          .EXTRN  BAS$FET_FA_F_R8         ; array element fetch - float
0000    106          .EXTRN  BAS$FET_FA_D_R8         ; array element fetch - double
0000    107          .EXTRN  BAS$FET_FA_G_R8         ; array element fetch - gfloat
0000    108          .EXTRN  BAS$FET_FA_H_R8         ; array element fetch - hfloat
0000    109          .EXTRN  BAS$MAT_ASSIGN          ; copy input matrix to destination
0000    110                                          ;  matrix for in place inversion
0000    111                                          ;  will also redim dest mat
0000    112          .EXTRN  BAS$$SCALE_R1           ; scale for double procision
0000    113          .EXTRN  BAS$$STORE_DET          ; store dbl determinant
0000    114          .EXTRN  BAS$$STORE_DET_G        ; store gfloat determinant
0000    115          .EXTRN  BAS$$STORE_DET_H        ; store hfloat determinant
0000    116          .EXTRN  MTH$DINT_R4             ; routine to integerize double
0000    117          .EXTRN  BAS$$STOP               ; signal fatal errors
0000    118          .EXTRN  BAS$FETCH_BFA
0000    119          .EXTRN  BAS$STORE_BFA
0000    120
0000    121 ;
0000    122 ; MACROS:
0000    123 ;
0000    124
0000    125 ;       $BAS$MAT_INV    inverse algorithm,  see next page
```

```
                     0000   126 ;       FETCH           fetch an array element (found in macro library
                     0000   127 ;                       MATRIXMAC.OLB)
                     0000   128 ;       STORE           store an array element (found in macro library
                     0000   129 ;                       MATRIXMAC.OLB)
                     0000   130
                     0000   131 ;
                     0000   132 ; EQUATED SYMBOLS:
                     0000   133 ;
                     0000   134
                     0000   135 ;+
                     0000   136 ; stack offsets
                     0000   137 ;-
                     0000   138
00000000             0000   139         maximum_sub = 0                 ; upper bound of subs
00000004             0000   140         current_i = 4                   ; loop counter i
00000008             0000   141         current_j = 8                   ; loop counter j
0000000C             0000   142         current_k = 12                  ; look counter k
00000010             0000   143         pivi = 16                       ; row of max remaining elem
00000014             0000   144         pivj = 20                       ; col of max remaining elem
00000018             0000   145         pivot = 24                      ; abs of max remaining elem
                     0000   146                                         ;  (this may be hfloat)
00000028             0000   147         scale = 40                      ; scale (1-10**6) for double
00000030             0000   148         determinant = 48                ; compute determinant
                     0000   149                                         ;  (this may be hfloat)
00000040             0000   150         value_desc = 64                 ; output descriptor
00000040             0000   151         str_len = 64                    ; length field within desc
00000042             0000   152         dtype = 66                      ; data type field within desc
00000043             0000   153         class = 67                      ; class field within desc
00000048             0000   154         data = 72                       ; data (4 longwords worth)
00000044             0000   155         pointer = 68                    ; pointer field within desc
00000058             0000   156         r_sub_k = 88                    ; value of R(K)
00000058             0000   157         loc_sub_r_sub_i = 88            ; another name for same storage
00000058             0000   158         loc_sub_c_sub_i = 88            ; yet another name for same
0000005C             0000   159         r_sub_i = 92                    ; value of R(I)
00000060             0000   160         c_sub_i = 96                    ; value of C(I)
00000064             0000   161         a_sub_rkci = 100                ; output_matrix(R(K), C(I))
00000064             0000   162         save_element = 100              ; another name for same storage
00000064             0000   163         c_sub_0 = 100                   ; address for addressing c vector
00000068             0000   164         r_sub_0 = 104                   ; address for addressing r vector
0000006C             0000   165         tag_sub_0 = 108                 ; address for addressing tag vect
00000070             0000   166         loc_sub_0 = 112                 ; address for addressing loc vector
00000074             0000   167         c_sub_1 = 116                   ; first element of c vector
00000078             0000   168         r_sub_1 = 120                   ; first element of r vector
0000007C             0000   169         tag_sub_1 = 124                 ; first element of tag vector
00000080             0000   170         loc_sub_1 = 128                 ; first element of loc vector
                     0000   171
                     0000   172 ;+
                     0000   173 ; Stack offsets for subroutine
                     0000   174 ;-
                     0000   175
00000000             0000   176         temp_pivi = 0                   ; cell to store new pivot row
00000004             0000   177         temp_pivj = 4                   ; cell to store new pivot col
00000008             0000   178         temp_pivot = 8                  ; cell to calculate new pivot
00000018             0000   179         temp_pivot_abs = 24             ; absolute value of new pivot
00000028             0000   180         save_result = 40                ; temp area to store a result
                     0000   181
                     0000   182 ;+
```

```
                        0000   183 ; descriptor offsets
                        0000   184 ;-
                        0000   185
00000018                0000   186         dsc$l_l1_1 = 24                         ; desc offset if 1 sub
0000001C                0000   187         dsc$l_u1_1 = 28                         ; desc offset if 1 sub
0000001C                0000   188         dsc$l_l1_2 = 28                         ; desc offset if 2 sub
00000020                0000   189         dsc$l_u1_2 = 32                         ; desc offset if 2 sub
00000024                0000   190         dsc$l_l2_2 = 36                         ; desc offset if 2 sub
00000028                0000   191         dsc$l_u2_2 = 40                         ; desc offset if 2 sub
                        0000   192
                        0000   193 ;
                        0000   194 ; OWN STORAGE:
                        0000   195 ;
                        0000   196
                00000000  0000   197         .PSECT _BAS$DATA PIC, USR, CON, REL, LCL, NOSHR, NOEXE, -
                        0000   198                         RD, WRT
                        0000   199
00000010                0000   200         temp:   .BLKB   16      ; 16 bytes - large enough to hold h-floating
                        0010   201                                 ; datum
                        0010   202
                        0010   203 ;
                        0010   204 ; PSECT DECLARATIONS:
                        0010   205 ;
                        0010   206         .PSECT _BAS$CODE PIC, USR, CON, REL, LCL, SHR, -
                00000000  0000   207                         EXE, RD, NOWRT, LONG
                        0000   208
                        0000   209
```

```
0000    211 ;+
0000    212 ; This macro contains the algorithm for inverting array.  The inversion
0000    213 ; algorithm used is from the COLLECTED ALGORITHMS FROM CACM, algorithms
0000    214 ; 230 and 231.  It inverts a matrix in place using the Gauss_Jordan method with
0000    215 ; complete matrix pivoting.  It contain the logic for all the data
0000    216 ; types and scaling.  A macro is used to make it easy to maintain the parallel
0000    217 ; code for all the different data types.
0000    218 ;-
0000    219
0000    220         .MACRO  $BAS$MAT_INVERT array_dtype     ; inversion algorithm
0000    221
0000    222 FIND_1ST_PIVOT'array_dtype':
0000    223         MOV'array_dtype'        #1, determinant(SP)     ; initialize det
0000    224         .IF     IDN     array_dtype, D          ; if double
0000    225         MULD2   scale(SP), determinant(SP)      ; scale it, no integerize needed
0000    226         .ENDC
0000    227         CLR'array_dtype'        pivot(SP)       ; set pivot to 0
0000    228         MOVL    #1, R10                         ; lowerbound for outer loop
0000    229 1$:     MOVL    #1, R11                         ; lowerbound for inner loop
0000    230 2$:     MOVL    R9, R0                          ; pointer to output array
0000    231         MOVL    R10, R1                         ; current row
0000    232         MOVL    R11, R2                         ; current col
0000    233         FETCH   'array_dtype'                   ; fetch data from output array
0000    234         TST'array_dtype'        R0              ; see if need ABS
0000    235         BGEQ    3$                              ; positive, no ABS needed
0000    236         MNEG'array_dtype'       R0, R0          ; negative, negate
0000    237                                                 ;  could get performance improv
0000    238                                                 ;  for f and d if divided on
0000    239                                                 ;  data types and used BIC
0000    240 3$:     CMP'array_dtype'        R0, pivot(SP)   ; see if larger than pivot
0000    241         BLEQ    4$                              ; not large, go to next element
0000    242         MOV'array_dtype'        R0, pivot(SP)   ; new initial pivot
0000    243         MOVL    R10, pivi(SP)                   ; new pivot row
0000    244         MOVL    R11, pivj(SP)                   ; new pivot column
0000    245 4$:     INCL    R11                             ; next element
0000    246         CMPL    maximum_sub(SP),R11             ;\replaced AOBLEQ instruction
0000    247         BLEQ    7$                              ; due to Branch destination
0000    248         BRW     2$                              ;/out of range errors
0000    249 7$:     ACBL    maximum_sub(SP), #1, R10, 1$    ; next element (AOBLEQ will not
0000    250                                                 ;  work here)
0000    251
0000    252
0000    253
0000    254 ;+
0000    255 ; Now get the initial pivot without the absolute value.
0000    256 ;-
0000    257
0000    258         MOVL    R9, R0                          ; pointer to output array
0000    259         MOVL    pivi(SP), R1                    ; current row
0000    260         MOVL    pivj(SP), R2                    ; current col
0000    261         FETCH   'array_dtype'                   ; fetch data from output array
0000    262         MOV'array_dtype'        R0, pivot(SP)   ; initial pivot
0000    263
0000    264 ;+
0000    265 ; The initial pivot element and it's position are now stored on the
0000    266 ; stack.  perform the inversion algorithm.
0000    267 ;-
```

```
0000    268
0000    269          MOVL    #1, current_i(SP)                ; initialize I loop
0000    270 LOOP_I_'array_dtype':
0000    271
0000    272 ;+
0000    273 ; Make remaining element with greatest magnitude the next pivot element.
0000    274 ; Keep track in permutation vectors.
0000    275 ;-
0000    276
0000    277          INDEX   current_i(SP), #1, maximum_sub(SP), #4, #0, R0 ; ith element
0000    278          CMPL    pivi(SP), current_i(SP)                  ; any row swapping?
0000    279          BEQL    5$                                       ; no
0000    280          MNEG'array_dtype'       determinant(SP), determinant(SP) ; change sign of de
0000    281 5$:      INDEX   pivi(SP), #1, maximum_sub(SP), #4, #0, R1 ; pivith element
0000    282          MOVL    r_sub_0(SP)[R1], r_sub_i(SP)             ; temp <- R(PIVI), new R(I)
0000    283          MOVL    r_sub_0(SP)[R0], r_sub_0(SP)[R1]         ; R(PIVI) <- R(I)
0000    284          MOVL    r_sub_i(SP), r_sub_0(SP)[R0]             ; R(I) <- temp
0000    285          CMPL    pivj(SP), current_i(SP)                  ; any column swapping?
0000    286          BEQL    6$                                       ; no
0000    287          MNEG'array_dtype'       determinant(SP), determinant(SP) ; change sign of de
0000    288 6$:      INDEX   pivj(SP), #1, maximum_sub(SP), #4, #0, R2 ; pivjth element
0000    289          MOVL    c_sub_0(SP)[R2], c_sub_i(SP)             ; temp <- C(PIVJ), new C(I)
0000    290          MOVL    c_sub_0(SP)[R0], c_sub_0(SP)[R2]         ; C(PIVJ) <- C(I)
0000    291          MOVL    c_sub_i(SP), c_sub_0(SP)[R0]             ; C(I) <- temp
0000    292
0000    293 ;+
0000    294 ; If pivot element is 0, then array appears to be singular
0000    295 ;-
0000    296
0000    297          TST'array_dtype'        pivot(SP)        ; is pivot 0?
0000    298          BNEQ    10$                              ; no, continue
0000    299          JMP     ERR_CANINVMAT                    ; yes, error
0000    300
0000    301 ;+
0000    302 ; Calculate determinant.
0000    303 ; Switch sign of determinant if \??????????????????????\
0000    304 ;-
0000    305
0000    306 10$:     MUL'array_dtype'2       pivot(SP), determinant(SP) ; det = det * PIVOT
0000    307
0000    308          .IF     IDN     array_dtype, D
0000    309          DIVD2   scale(SP), determinant(SP)       ; get rid of extra scale on det
0000    310          CMPD    scale(SP), #1                    ; is the scale factor 0 (10**0)
0000    311          BEQL    11$                              ; yes, don't integerize
0000    312          MOVD    determinant(SP), R0              ; no, integerize det
0000    313          JSB     G^MTH$DINT_R4
0000    314          MOVD    R0, determinant(SP)              ; and store the result
0000    315          .ENDC
0000    316
0000    317 ;+
0000    318 ; Now store 1 in A(R(I), C(I)) and divide each element of the row
0000    319 ; by the old value of A(R(I), C(I)), ie PIVOT.
0000    320 ;-
0000    321
0000    322 11$:     .IF     IDN     array_dtype, H
0000    323          MOVH    #1, R0
0000    324          MOVL    R9, R4                           ; pointer to output array
```

```
                                                                I  2
0000    325          MOVL    r_sub_i(SP), R5                    ; current row
0000    326          MOVL    c_sub_i(SP), R6                    ; current col
0000    327          MOVH    R0, data(SP)
0000    328          STORE   H
0000    329          .IFF
0000    330          .IF     IDN     array_dtype, G
0000    331          MOVG    #1, R0
0000    332          MOVL    R9, R2                             ; pointer to output array
0000    333          MOVL    r_sub_i(SP), R3                    ; current row
0000    334          MOVL    c_sub_i(SP), R4                    ; current col
0000    335          MOVG    R0, data(SP)
0000    336          STORE   G
0000    337          .IFF
0000    338          .IF     IDN     array_dtype, D
0000    339          MOV'array_dtype'        #1, R0             ; data to be stored
0000    340          MULD2   scale(SP), R0                      ; scale the 1
0000    341          MOVL    R9, R2                             ; pointer to output array
0000    342          MOVL    r_sub_i(SP), R3                    ; current row
0000    343          MOVL    c_sub_i(SP), R4                    ; current col
0000    344          MOV'array_dtype'        R0, data(SP)
0000    345          STORE   'array_dtype'                      ; store data into output array
0000    346          .IFF
0000    347          MOV'array_dtype'        #1, R0             ; data to be stored
0000    348          MOVL    R9, R1                             ; pointer to output array
0000    349          MOVL    r_sub_i(SP), R2                    ; current row
0000    350          MOVL    c_sub_i(SP), R3                    ; current col
0000    351          MOV'array_dtype'        R0, data(SP)
0000    352          STORE   'array_dtype'                      ; store data into output array
0000    353          .ENDC
0000    354          .ENDC
0000    355          .ENDC
0000    356
0000    357          MOVL    maximum_sub(SP), R10               ; max value for J (row) loop
0000    358 20$:     MOVL    R9, R0                             ; pointer to output array
0000    359          MOVL    r_sub_i(SP), R1                    ; current row
0000    360          INDEX   R10, #1, maximum_sub(SP), #4, #0, R11 ; to fetch C(J)
0000    361          MOVL    c_sub_0(SP)[R11], R2               ; current column
0000    362          FETCH   'array_dtype'                      ; fetch data from output array
0000    363
0000    364          .IF     IDN     array_dtype, D
0000    365          MULD2   scale(SP), R0                      ; Prescale the element
0000    366          .ENDC
0000    367
0000    368          DIV'array_dtype'2       pivot(SP), R0
0000    369                                  ; A(R(I),C(J)) <- A(R(I),C(J)) / A(R(I),C(I))
0000    370          .IF     IDN     array_dtype, D
0000    371          CMPD    scale(SP), #1                      ; is the scale factor 0 (10**0)
0000    372          BEQL    25$                                ; yes, don't integerize
0000    373          JSB     G^MTH$DINT_R4                      ; no, integerize
0000    374          .ENDC
0000    375
0000    376 ;+
0000    377 ; Now store the quotient in A(R(I), C(J))
0000    378 ;-
0000    379
0000    380 25$:     .IF     IDN     array_dtype, H
0000    381          MOVL    R9, R4                             ; pointer to output array
```

```
0000    382          MOVL    r_sub_i(SP), R5                    ; current row
0000    383          MOVL    c_sub_0(SP)[R11], R6               ; current col
0000    384          MOVH    R0, data(SP)
0000    385          STORE   H
0000    386          .IFF
0000    387          .IF     IDN     array_dtype, G
0000    388          MOVL    R9, R2                             ; pointer to output array
0000    389          MOVL    r_sub_i(SP), R3                    ; current row
0000    390          MOVL    c_sub_0(SP)[R11], R4               ; current col
0000    391          MOVG    R0, data(SP)
0000    392          STORE   G
0000    393          .IFF
0000    394          .IF     IDN     array_dtype, D
0000    395          MOVL    R9, R2                             ; pointer to output array
0000    396          MOVL    r_sub_i(SP), R3                    ; current row
0000    397          MOVL    c_sub_0(SP)[R11], R4               ; current col
0000    398          MOV'array_dtype'        R0, data(SP)
0000    399          STORE   'array_dtype'                      ; store data into output array
0000    400          .IFF
0000    401          MOVL    R9, R1                             ; pointer to output array
0000    402          MOVL    r_sub_i(SP), R2                    ; current row
0000    403          MOVL    c_sub_0(SP)[R11], R3               ; current col
0000    404          MOV'array_dtype'        R0, data(SP)
0000    405          STORE   'array_dtype'                      ; store data into output array
0000    406          .ENDC
0000    407          .ENDC
0000    408          .ENDC
0000    409
0000    410 ;+
0000    411 ; Loop through all elements of that row
0000    412 ; (note that SOBGTR will not work here)
0000    413 ;-
0000    414
0000    415          DECL    R10
0000    416          BLEQ    26$
0000    417          BRW     20$                                ; next element in row
0000    418 ;+
0000    419 ; Now loop through all elements of the array except the pivot, continuing to
0000    420 ; compute inverse.  In the area of the array beyond the row and column of the
0000    421 ; pivot find the largest element to select the next pivot.
0000    422 ;-
0000    423
0000    424 26$:     MOVL    SP, R11                            ; point to arg list for subr
0000    425          SUBL3   #1, current_i(SP), R10             ; upperbound for K loop
0000    426          BEQL    30$                                ; I is 1, start with 2
0000    427          MOVL    #1, current_k(SP)                  ; initialize K loop
0000    428          BSBW    LOOP_K 'array_dtype'               ; loop K = 1, I-1
0000    429 30$:     CMPL    current_i(SP), maximum_sub(SP)     ; is I upperbound?
0000    430          BEQL    40$                                ; yes, finished I loop
0000    431          ADDL3   #1, current_i(SP), current_k(SP)   ; initialize K loop
0000    432          MOVL    maximum_sub(SP), R10               ; upperbound for K loop
0000    433                                                     ; pointer to arg list still set
0000    434          BSBW    LOOP_K 'array_dtype'               ; loop K = I+1, N
0000    435          MOVL    R2, pivi(SP)                       ; new pivot row
0000    436          MOVL    R3, pivj(SP)                       ; new pivot column
0000    437          MOV'array_dtype'        R4, pivot(SP)      ; new pivot element
0000    438
```

```
0000    439 ;+
0000    440 ; Continue outer (I) loop
0000    441 ;-
0000    442
0000    443         ADDL2   #1, current_i(SP)                   ; get next I
0000    444         BRW     LOOP_I_'array_dtype'                ; continue loop, loop escape is
0000    445                                                     ;  inside loop
0000    446
0000    447 ;+
0000    448 ; Now store the determinant.  The determinant is stored in double precision
0000    449 ; unless the matrix is g or h floating.  In these cases, the determinant will
0000    450 ; be stored in g and h floating, respectively.
0000    451 ;-
0000    452
0000    453 40$:    .IF     IDN     array_dtype, H
0000    454                                                     ;
0000    455         CVTDH   scale(SP), temp                     ; cvt scale factor to H_float
0000    456         MULH2   temp, determinant(SP)               ; scale it
0000    457                                                     ;
0000    458         PUSHAH  determinant(SP)                     ; pass det by ref (stored as h)
0000    459         CALLS   #1, G^BAS$$STORE_DET_H              ; store the hfloat det
0000    460         .IFF
0000    461         .IF     IDN     array_dtype, G
0000    462                                                     ;
0000    463         CVTDH   scale(SP), temp                     ; cvt scale factor to G_float
0000    464         CVTHG   temp, scale(SP)                     ; in 2 steps because no CVTDG
0000    465         MULG2   scale(SP), determinant(SP)          ; scale it
0000    466                                                     ;
0000    467         PUSHAG  determinant(SP)                     ; pass det by ref (stored as g)
0000    468         CALLS   #1, G^BAS$$STORE_DET_G              ; store the gfloat det
0000    469         .IFF
0000    470
0000    471         .IF     DIF     array_dtype, D              ; if not double
0000    472         CVT'array_dtype'D       determinant(SP), determinant(SP) ; make double
0000    473         MULD2   scale(SP), determinant(SP)          ; scale it
0000    474         PUSHAD  determinant(SP)                     ; pass det by ref (stored as d)
0000    475         CALLS   #1, G^BAS$$STORE_DET                ; store the dbl det
0000    476
0000    477         .IFF
0000    478         PUSHAD  determinant(SP)                     ; pass det by ref (stored as d)
0000    479         CALLS   #1, G^BAS$$STORE_DET                ; store the dbl det
0000    480         .ENDC
0000    481         .ENDC
0000    482         .ENDC
0000    483
0000    484
0000    485 ;+
0000    486 ; Put the rows and columns in the correct order.
0000    487 ;-
0000    488
0000    489         MOVL    #1, R10                             ; initialize I
0000    490 INVERT_ROWS_'array_dtype':
0000    491         INDEX   R10, #1, maximum_sub(SP), #4, #0, R11 ; to fetch R(I)
0000    492         MOVL    r_sub_0(SP)[R11], r_sub_i(SP)       ; fetch R(I)
0000    493         MOVL    c_sub_0(SP)[R11], c_sub_i(SP)       ; fetch C(I)
0000    494         INDEX   r_sub_i(SP), #1, maximum_sub(SP), #4, #0, R11
0000    495                                                     ; to fetch loc(R(I))
```

```
0000    496          MOVL    loc_sub_0(SP)[R11], loc_sub_r_sub_i(SP) ; fetch loc(R(I))
0000    497          CMPL    loc_sub_r_sub_i(SP), c_sub_i(SP) ; if loc(R(I)) = C(I)
0000    498          BNEQ    10$
0000    499          BRW     20$                                 ; don't switch
0000    500 10$:     MOVL    #1, R11                             ; initialize L
0000    501
0000    502 ;+
0000    503 ; Switch 2 rows
0000    504 ;-
0000    505
0000    506 10$:     MOVL    R9, R0                              ; pointer to output array
0000    507          MOVL    loc_sub_r_sub_i(SP), R1             ; current row
0000    508          MOVL    R11, R2                             ; current column
0000    509          FETCH   'array_dtype'                       ; fetch A(loc(R(I)), L)
0000    510          MOV'array_dtype'        R0, save_element(SP) ; save it in temp
0000    511
0000    512          MOVL    R9, R0                              ; pointer to output array
0000    513          MOVL    c_sub_i(SP), R1                     ; current row
0000    514          MOVL    R11, R2                             ; current column
0000    515          FETCH   'array_dtype'                       ; fetch A(C(I), L)
0000    516
0000    517          .IF     IDN     array_dtype, H
0000    518          MOVL    R9, R4                              ; pointer to output array
0000    519          MOVL    loc_sub_r_sub_i(SP), R5             ; current row
0000    520          MOVL    R11, R6                             ; current col
0000    521          .IFF
0000    522          .IF     IDN     array_dtype, G
0000    523          MOVL    R9, R2                              ; pointer to output array
0000    524          MOVL    loc_sub_r_sub_i(SP), R3             ; current row
0000    525          MOVL    R11, R4                             ; current col
0000    526          .IFF
0000    527          .IF     IDN     array_dtype, D
0000    528          MOVL    R9, R2                              ; pointer to output array
0000    529          MOVL    loc_sub_r_sub_i(SP), R3             ; current row
0000    530          MOVL    R11, R4                             ; current col
0000    531          .IFF
0000    532          MOVL    R9, R1                              ; pointer to output array
0000    533          MOVL    loc_sub_r_sub_i(SP), R2             ; current row
0000    534          MOVL    R11, R3                             ; current col
0000    535          .ENDC
0000    536          .ENDC
0000    537          .ENDC
0000    538          MOV'array_dtype'        R0, data(SP)
0000    539          STORE   'array_dtype'                       ; store data into output array
0000    540
0000    541          MOV'array_dtype'        save_element(SP), R0 ; get element from temp
0000    542          .IF     IDN     array_dtype, H
0000    543          MOVL    R9, R4                              ; pointer to output array
0000    544          MOVL    c_sub_i(SP), R5                     ; current row
0000    545          MOVL    R11, R6                             ; current col
0000    546          .IFF
0000    547          .IF     IDN     array_dtype, G
0000    548          MOVL    R9, R2                              ; pointer to output array
0000    549          MOVL    c_sub_i(SP), R3                     ; current row
0000    550          MOVL    R11, R4                             ; current col
0000    551          .IFF
0000    552          .IF     IDN     array_dtype, D
```

```
0000    553          MOVL    R9, R2                                  ; pointer to output array
0000    554          MOVL    c_sub_i(SP), R3                         ; current row
0000    555          MOVL    RT1, R4                                 ; current col
0000    556          .IFF
0000    557          MOVL    R9, R1                                  ; pointer to output array
0000    558          MOVL    c_sub_i(SP), R2                         ; current row
0000    559          MOVL    RT1, R3                                 ; current col
0000    560          .ENDC
0000    561          .ENDC
0000    562          .ENDC
0000    563          MOV'array_dtype'        R0, data(SP)
0000    564          STORE   'array_dtype'                           ; store data into output array
0000    565
0000    566
0000    567 ;+
0000    568 ; Formerly AOBLEQ was used here, but the code grew too big and now the
0000    569 ; destination is out of range.
0000    570 ;-
0000    571
0000    572          ACBL    maximum_sub(SP), #1, R11, 10$   ; next L
0000    573
0000    574 20$:     INDEX   c_sub_i(SP), #1, maximum_sub(SP), #4, #0, R0 ; to get tag(C(I))
0000    575          INDEX   loc_sub_r_sub_i(SP), #1, maximum_sub(SP), #4, #0, R1
0000    576                                                          ; to get tag(loc(R(I)))
0000    577          MOVL    tag_sub_0(SP)[R0], tag_sub_0(SP)[R1] ; tag(loc(R(I)))=tag(C(I))
0000    578          MOVL    r_sub_i(SP), tag_sub_0(SP)[R0]  ; tag(C(I))=R(I)
0000    579          INDEX   tag_sub_0(SP)[R1], #1, maximum_sub(SP), #4, #0, R2
0000    580                                                          ; to get loc(tag(loc(R(I))))
0000    581          INDEX   r_sub_i(SP), #1, maximum_sub(SP), #4, #0, R3
0000    582                                                          ; to get loc(R(I))
0000    583          MOVL    loc_sub_0(SP)[R2], loc_sub_0(SP)[R3]
0000    584                                                          ; loc(R(I))=loc(tag(loc(R(I))))
0000    585          MOVL    loc_sub_r_sub_i(SP), loc_sub_0(SP)[R2]
0000    586                                                          ; loc(tag(loc(R(I))))=loc(R(I))
0000    587          ACBW    maximum_sub(SP), #1, R10, INVERT_ROWS 'array_dtype'
0000    588                                                          ; next I
0000    589
0000    590 ;+
0000    591 ; Reinitialize the tag and loc vectors
0000    592 ;-
0000    593
0000    594          MOVL    #1, R10                                 ; initialize I
0000    595 50$:     INDEX   R10, #1, maximum_sub(SP), #4, #0, R11 ; to fetch Ith element
0000    596          MOVL    R10, loc_sub_0(SP)[R11]                 ; initialize loc
0000    597          MOVL    R10, tag_sub_0(SP)[R11]                 ; initialize tag
0000    598          AOBLEQ  maximum_sub(SP), R10, 50$               ; next element
0000    599
0000    600 ;+
0000    601 ; Put the columns in the correct order.
0000    602 ;-
0000    603
0000    604          MOVL    #1, R10                                 ; initialize I
0000    605 INVERT_COLS 'array_dtype':
0000    606          INDEX   R10, #1, maximum_sub(SP), #4, #0, R11 ; to fetch C(I)
0000    607          MOVL    r_sub_0(SP)[R11], r_sub_i(SP)   ; fetch R(I)
0000    608          MOVL    c_sub_0(SP)[R11], c_sub_i(SP)   ; fetch C(I)
0000    609          INDEX   c_sub_i(SP), #1, maximum_sub(SP), #4, #0, R11
```

```
0000    610                                                     ; to fetch loc(C(I))
0000    611             MOVL    loc_sub_0(SP)[R11], loc_sub_c_sub_i(SP) ; fetch loc(C(I))
0000    612             CMPL    loc_sub_c_sub_i(SP), r_sub_i(SP) ; if loc(C(I)) = R(I)
0000    613             BNEQ    102$
0000    614             BRW     20$                             ; don't switch
0000    615 102$:       MOVL    #1, R11                         ; initialize L
0000    616
0000    617 ;+
0000    618 ; Switch 2 cols
0000    619 ;-
0000    620
0000    621 10$:        MOVL    R9, R0                          ; pointer to output array
0000    622             MOVL    loc_sub_c_sub_i(SP), R2         ; current column
0000    623             MOVL    R11, R1                         ; current row
0000    624             FETCH   'array_dtype'                   ; fetch A(L, loc(C(I)))
0000    625             MOV'array_dtype'        R0, save_element(SP) ; save it in temp
0000    626
0000    627             MOVL    R9, R0                          ; pointer to output array
0000    628             MOVL    r_sub_i(SP), R2                 ; current column
0000    629             MOVL    R11, R1                         ; current row
0000    630             FETCH   'array_dtype'                   ; fetch A(L, R(I))
0000    631
0000    632             .IF     IDN     array_dtype, H
0000    633             MOVL    R9, R4                          ; pointer to output array
0000    634             MOVL    loc_sub_c_sub_i(SP), R6         ; current col
0000    635             MOVL    R11, R5                         ; current row
0000    636             .IFF
0000    637             .IF     IDN     array_dtype, G
0000    638             MOVL    R9, R2                          ; pointer to output array
0000    639             MOVL    loc_sub_c_sub_i(SP), R4         ; current col
0000    640             MOVL    R11, R3                         ; current row
0000    641             .IFF
0000    642             .IF     IDN     array_dtype, D
0000    643             MOVL    R9, R2                          ; pointer to output array
0000    644             MOVL    loc_sub_c_sub_i(SP), R4         ; current column
0000    645             MOVL    R11, R3                         ; current row
0000    646             .IFF
0000    647             MOVL    R9, R1                          ; pointer to output array
0000    648             MOVL    loc_sub_c_sub_i(SP), R3         ; current col
0000    649             MOVL    R11, R2                         ; current row
0000    650             .ENDC
0000    651             .ENDC
0000    652             .ENDC
0000    653             MOV'array_dtype'        R0, data(SP)
0000    654             STORE   'array_dtype'                   ; store data into output array
0000    655
0000    656             MOV'array_dtype'        save_element(SP), R0 ; get element from temp
0000    657             .IF     IDN     array_dtype, H
0000    658             MOVL    R9, R4                          ; pointer to output array
0000    659             MOVL    r_sub_i(SP), R6                 ; current col
0000    660             MOVL    R11, R5                         ; current row
0000    661             .IFF
0000    662             .IF     IDN     array_dtype, G
0000    663             MOVL    R9, R2                          ; pointer to output array
0000    664             MOVL    r_sub_i(SP), R4                 ; current col
0000    665             MOVL    R11, R3                         ; current row
0000    666             .IFF
```

```
0000    667          .IF     IDN     array_dtype, D
0000    668          MOVL    R9, R2                          ; pointer to output array
0000    669          MOVL    r_sub_i(SP), R4                 ; current column
0000    670          MOVL    RT1, R3                         ; current row
0000    671          .IFF
0000    672          MOVL    R9, R1                          ; pointer to output array
0000    673          MOVL    r_sub_i(SP), R3                 ; current col
0000    674          MOVL    RT1, R2                         ; current row
0000    675          .ENDC
0000    676          .ENDC
0000    677          .ENDC
0000    678          MOV'array_dtype'        R0, data(SP)
0000    679          STORE   'array_dtype'                   ; store data into output array
0000    680
0000    681
0000    682 ;+
0000    683 ; AOBLEQ will not work here because the destination is out of range.
0000    684 ;-
0000    685
0000    686          ACBL    maximum_sub(SP), #1, R11, 10$   ; next L
0000    687
0000    688 20$:     INDEX   r_sub_i(SP), #1, maximum_sub(SP), #4, #0, R0 ; to get tag(R(I))
0000    689          INDEX   loc_sub_c_sub_i(SP), #1, maximum_sub(SP), #4, #0, R1
0000    690                                                  ; to get tag(loc(C(I)))
0000    691          MOVL    tag_sub_0(SP)[R0], tag_sub_0(SP)[R1] ; tag(loc(C(I)))=tag(R(I))
0000    692          MOVL    c_sub_i(SP), tag_sub_0(SP)[R0]  ; tag(R(I))=C(I)
0000    693          INDEX   tag_sub_0(SP)[R1], #1, maximum_sub(SP), #4, #0, R2
0000    694                                                  ; to get loc(tag(loc(C(I))))
0000    695          INDEX   c_sub_i(SP), #1, maximum_sub(SP), #4, #0, R3
0000    696                                                  ; to get loc(C(I))
0000    697          MOVL    loc_sub_0(SP)[R2], loc_sub_0(SP)[R3]
0000    698                                                  ; loc(C(I))=loc(tag(loc(C(I))))
0000    699          MOVL    loc_sub_c_sub_i(SP), loc_sub_0(SP)[R2]
0000    700                                                  ; loc(tag(loc(C(I))))=loc(C(I))
0000    701          ACBW    maximum_sub(SP), #1, R10, INVERT_COLS_'array_dtype'
0000    702                                                  ; next I
0000    703
0000    704
0000    705
0000    706
0000    707          RET                                     ; yes, finished
0000    708
0000    709
0000    710 ;+
0000    711 ; Input arguments to subroutines
0000    712 ; R9     pointer to array
0000    713 ; R10    upper bound on loop
0000    714 ; R11    pointer to callers stack argument list
0000    715 ; Output from subroutine, always returned, only picked up on the 2nd call
0000    716 ; R4-R7 new pivot
0000    717 ; R2     new pivot row
0000    718 ; R3     new pivot column
0000    719 ;-
0000    720
0000    721 LOOP_K_'array_dtype':
0000    722
0000    723 ;+
```

```
0000    724 ; Redefine the following symbols, since the stack will be changed.  This is
0000    725 ; necessary for FETCH and STORE, which use them to access the stack.
0000    726 ;-
0000    727
0000    728 value_desc = 124
0000    729 data = 132
0000    730 pointer = 128
0000    731 dtype = 126
0000    732 class = 127
0000    733 str_len = 124
0000    734
0000    735         SUBL2   #56, SP                         ; space for 2 pivots & row & col
0000    736         CLR'array_dtype'        temp_pivot_abs(SP) ; initialize new abs pivot
0000    737
0000    738 ;+
0000    739 ; Now store in temps elements that are invariant in inner loop
0000    740 ;-
0000    741
0000    742 START_K_LOOP 'array_dtype':
0000    743         INDEX   current_k(R11), #1, maximum_sub(R11), #4, #0, R0 ; to fetch R(K)
0000    744         MOVL    r_sub_0(R11)[R0], r_sub_k(R11)  ; store R(K) temp
0000    745         MOVL    R9, R0                          ; pointer to output array
0000    746         MOVL    r_sub_k(R11), R1                ; current row
0000    747         MOVL    c_sub_i(R11), R2                ; current column
0000    748         FETCH   'array_dtype'                   ; fetch data from output array
0000    749         MOV'array_dtype'        R0, a_sub_rkci(R11) ; store A(R(K), C(I)) temp
0000    750
0000    751 ;+
0000    752 ; Store 0 in A(R(K), C(I)) so that computation can be made in J loop
0000    753 ;-
0000    754
0000    755         CLR'array_dtype'        R0              ; zero to be stored
0000    756         .IF     IDN     array_dtype, H
0000    757         MOVL    R9, R4                          ; pointer to output array
0000    758         MOVL    r_sub_k(R11), R5                ; current row
0000    759         MOVL    c_sub_i(R11), R6                ; current col
0000    760         .IFF
0000    761         .IF     IDN     array_dtype, G
0000    762         MOVL    R9, R2                          ; pointer to output array
0000    763         MOVL    r_sub_k(R11), R3                ; current row
0000    764         MOVL    c_sub_i(R11), R4                ; current col
0000    765         .IFF
0000    766         .IF     IDN     array_dtype, D
0000    767         MULD2   scale(R11), R0                  ; scale the 0
0000    768         MOVL    R9, R2                          ; pointer to output array
0000    769         MOVL    r_sub_k(R11), R3                ; current row
0000    770         MOVL    c_sub_i(R11), R4                ; current col
0000    771         .IFF
0000    772         MOVL    R9, R1                          ; pointer to output array
0000    773         MOVL    r_sub_k(R11), R2                ; current row
0000    774         MOVL    c_sub_i(R11), R3                ; current col
0000    775         .ENDC
0000    776         .ENDC
0000    777         .ENDC
0000    778         MOV'array_dtype'        R0, data(SP)
0000    779         STORE   'array_dtype'                   ; store data into output array
0000    780
```

```
0000    781 ;+
0000    782 ; Now loop through the cols (J in algorithm) of row K that are greater than I
0000    783 ; Look for new pivot element at the same time.
0000    784 ;-
0000    785
0000    786         MOVL    maximum_sub(R11), current_j(R11) ; initialize J
0000    787         CMPL    current_i(R11), maximum_sub(R11) ; is I = max sub?
0000    788         BNEQ    10$                              ; no, continue
0000    789         BRW     LOOP_J 'array_dtype'             ; yes, do loop J <= I
0000    790 10$:    INDEX   current_j(R11), #1, maximum_sub(R11), #4, #0, R0 ; to fetch C(J)
0000    791         MOVL    c_sub_0(R11)[R0], -(SP)          ; current column
0000    792
0000    793 ;+
0000    794 ; A(R(K), C(J)) = A(R(K), C(J)) - A(R(I), C(J)) * A(R(K), C(I))
0000    795 ;-
0000    796
0000    797         MOVL    R9, R0                           ; pointer to output array
0000    798         MOVL    r_sub_i(R11), R1                 ; current row
0000    799         MOVL    (SP), R2                         ; current column is on stack
0000    800         FETCH   'array_dtype'                    ; fetch data from output array
0000    801
0000    802         MUL'array_dtype'2       a_sub_rkci(R11), R0 ; R0=A(R(I),C(J))*A(R(K),C(I))
0000    803
0000    804         .IF     IDN     array_dtype, D
0000    805         DIVD2   scale(R11), R0                   ; get rid of extra scale
0000    806         CMPD    scale(R11), #1                   ; if scale factor is 0
0000    807         BEQL    15$                              ; don't integerize
0000    808         JSB     G^MTH$DINT_R4                    ; otherwise, integerize
0000    809         .ENDC
0000    810
0000    811 15$:    MOVL    (SP), R5                         ; prepare for next fetch
0000    812         MOV'array_dtype'        R0, -(SP)        ; save partial computation
0000    813         MOVL    R9, R0                           ; pointer to output array
0000    814         MOVL    r_sub_k(R11), R1                 ; current row
0000    815         MOVL    R5, R2                           ; move saved current col
0000    816         FETCH   'array_dtype'                    ; fetch data from output array
0000    817
0000    818         SUB'array_dtype'2       (SP)+, R0        ; R0=A(R(K),C(J))-temp
0000    819
0000    820 ;+
0000    821 ; Now store the computed result in A(R(K), C(J)).  If the computed result
0000    822 ; is larger than pivot, make this element the new pivot.
0000    823 ;-
0000    824
0000    825         .IF     IDN     array_dtype, H
0000    826         MOVL    (SP)+, R4                        ; save col here for now
0000    827         .IFF
0000    828         .IF     IDN     array_dtype, G
0000    829         MOVL    (SP)+, R4                        ; put current col for store
0000    830         .IFF
0000    831         .IF     IDN     array_dtype, D
0000    832         MOVL    (SP)+, R4                        ; put current col for store
0000    833         .IFF
0000    834         MOVL    (SP)+, R3                        ; put current col for store
0000    835         .ENDC
0000    836         .ENDC
0000    837         .ENDC
```

```
0000    838
0000    839          MOV'array_dtype'        R0, save_result(SP) ; move computed result
0000    840          BGEQ    20$                                 ; computed is pos, continue
0000    841          MNEG'array_dtype'       save_result(SP), save_result(SP) ; make neg result p
0000    842 20$:     CMP'array_dtype'        save_result(SP), temp_pivot_abs(SP) ; is element les
0000    843          BLSS    30$                                 ; yes, continue
0000    844          MOV'array_dtype'        save_result(SP), temp_pivot_abs(SP) ; no, make new a
0000    845          MOV'array_dtype'        R0, temp_pivot(SP) ; new pivot
0000    846          MOVL    current_k(R11), temp_pivi(SP)   ; new pivot row
0000    847          MOVL    current_j(R11), temp_pivj(SP)   ; new pivot col
0000    848
0000    849 30$:     .IF     IDN     array_dtype, H
0000    850          MOVL    R4, R6                          ; move current col
0000    851          MOVL    R9, R4                          ; pointer to output array
0000    852          MOVL    r_sub_k(R11), R5                ; current row
0000    853          .IFF
0000    854          .IF     IDN     array_dtype, G
0000    855          MOVL    R9, R2                          ; pointer to output array
0000    856          MOVL    r_sub_k(R11), R3                ; current row
0000    857          .IFF
0000    858          .IF     IDN     array_dtype, D
0000    859          MOVL    R9, R2                          ; pointer to output array
0000    860          MOVL    r_sub_k(R11), R3                ; current row
0000    861          .IFF
0000    862          MOVL    R9, R1                          ; pointer to output array
0000    863          MOVL    r_sub_k(R11), R2                ; current row
0000    864          .ENDC
0000    865          .ENDC
0000    866          .ENDC
0000    867          MOV'array_dtype'        R0, data(SP)
0000    868          STORE   'array_dtype'                   ; store data into output array
0000    869
0000    870          SUBL2   #1, current_j(R11)              ; decrement loop counter
0000    871          CMPL    current_j(R11), current_i(R11)  ; continue looping?
0000    872          BLEQ    LOOP_J_'array_dtype'            ; no
0000    873          BRW     10$                             ; yes
0000    874
0000    875 ;+
0000    876 ; Now loop through the cols (J in algorithm) of row K that are less than or
0000    877 ; equal to I.  Don't look for new pivot in this area of the array.
0000    878 ;-
0000    879
0000    880 LOOP_J_'array_dtype':
0000    881          INDEX   current_j(R11), #1, maximum_sub(R11), #4, #0, R0 ; to fetch C(J)
0000    882          MOVL    c_sub_0(R11)[R0], -(SP)         ; current column
0000    883
0000    884 ;+
0000    885 ; A(R(K), C(J)) = A(R(K), C(J)) - A(R(I), C(J)) * A(R(K), C(I))
0000    886 ;-
0000    887
0000    888          MOVL    R9, R0                          ; pointer to output array
0000    889          MOVL    r_sub_i(R11), R1                ; current row
0000    890          MOVL    (SP), R2                        ; current column is on stack
0000    891          FETCH   'array_dtype'                   ; fetch data from output array
0000    892
0000    893          MUL'array_dtype'2       a_sub_rkci(R11), R0 ; R0=A(R(I),C(J))*A(R(K),C(I))
0000    894
```

```
0000    895          .IF     IDN     array_dtype, D
0000    896          DIVD2   scale(R11), R0                  ; get rid of extra scale
0000    897          CMPD    scale(R11), #1                  ; if scale factor is 0
0000    898          BEQL    15$                             ; don't integerize
0000    899          JSB     G^MTH$DINT_R4                   ; otherwise, integerize
0000    900          .ENDC
0000    901
0000    902 15$:     MOVL    (SP), R5                        ; prepare for next fetch
0000    903          MOV'array_dtype'        R0, -(SP)       ; save partial computation
0000    904          MOVL    R9, R0                          ; pointer to output array
0000    905          MOVL    r_sub_k(R11), R1                ; current row
0000    906          MOVL    R5, R2                          ; move saved current col
0000    907          FETCH   'array_dtype'                   ; fetch data from output array
0000    908
0000    909          SUB'array_dtype'2       (SP)+, R0       ; R0=A(R(K),C(J))-temp
0000    910
0000    911 ;+
0000    912 ; Now store the computed result in A(R(K), C(J)).
0000    913 ;-
0000    914
0000    915          .IF     IDN     array_dtype, H
0000    916          MOVL    (SP)+, R6                       ; put current col for store
0000    917          MOVL    R9, R4                          ; pointer to output array
0000    918          MOVL    r_sub_k(R11), R5                ; current row
0000    919          .IFF
0000    920          .IF     IDN     array_dtype, G
0000    921          MOVL    (SP)+, R4                       ; put current col for store
0000    922          MOVL    R9, R2                          ; pointer to output array
0000    923          MOVL    r_sub_k(R11), R3                ; current row
0000    924          .IFF
0000    925          .IF     IDN     array_dtype, D
0000    926          MOVL    (SP)+, R4                       ; put current col for store
0000    927          MOVL    R9, R2                          ; pointer to output array
0000    928          MOVL    r_sub_k(R11), R3                ; current row
0000    929          .IFF
0000    930          MOVL    (SP)+, R3                       ; put current col for store
0000    931          MOVL    R9, R1                          ; pointer to output array
0000    932          MOVL    r_sub_k(R11), R2                ; current row
0000    933          .ENDC
0000    934          .ENDC
0000    935          .ENDC
0000    936          MOV'array_dtype'        R0, data(SP)
0000    937          STORE   'array_dtype'                   ; store data into output array
0000    938
0000    939
0000    940 ;+
0000    941 ; SOBGTR will not work here because the destination is out of range.
0000    942 ;-
0000    943
0000    944          DECL    current_j(R11)
0000    945          BLEQ    103$
0000    946          BRW     LOOP_J_'array_dtype'            ; continue J loop
0000    947
0000    948 103$:    ACBW    R10, #1, current_k(R11), START_K_LOOP_'array_dtype'
0000    949                                                  ; continue K loop
0000    950
0000    951          MOV'array_dtype'        temp_pivot(SP), R4 ; return new pivot
```

```
0000    952          MOVL    (SP)+, R2                       ; return new pivot row
0000    953          MOVL    (SP)+, R3                       ; return new pivot column
0000    954          ADDL2   #48, SP                         ; pop off 2 pivot temps
0000    955
0000    956 ;+
0000    957 ; Restore symbols
0000    958 ;-
0000    959
0000    960 value_desc = 64
0000    961 data = 72
0000    962 pointer = 68
0000    963 dtype = 66
0000    964 class = 67
0000    965 str_len = 64
0000    966
0000    967          RSB                                     ; return
0000    968
0000    969          .ENDM
0000    970
```

```
                     0000   972          .SBTTL  BAS$MAT_INV  - Invert a matrix
                     0000   973 ;++
                     0000   974 ; FUNCTIONAL DESCRIPTION:
                     0000   975 ;
                     0000   976 ;       Invert the contents of and input matrix and put the result in a
                     0000   977 ;       destination matrix.  Signal an error if the upper and
                     0000   978 ;       lower bounds (excluding 0) for rows and columns in input_matrix
                     0000   979 ;       does not equal the  pper and lower bounds (excluding 0) for rows
                     0000   980 ;       and columns respect vely in output_matrix.
                     0000   981 ;       An error will also be signalled if either matrix does not
                     0000   982 ;       have a DIMCT of 2.  Copy the input_matrix to the output_matrix using
                     0000   983 ;       BAS$MAT_ASSIGN.  The assign routine will redimension
                     0000   984 ;       the output matrix to have a lower bound of 0 for both dimensions,
                     0000   985 ;       and an upper bound for rows equal to the upper bound for rows for
                     0000   986 ;       input_matrix, and an upper bound for columns equal to the upper bound
                     0000   987 ;       for columns for input_matrix.  Initialize all the necessary
                     0000   988 ;       looping information on the stack.  To support different data types,
                     0000   989 ;       divide the looping portion according to the data types.
                     0000   990 ;
                     0000   991 ; CALLING SEQUENCE:
                     0000   992 ;
                     0000   993 ;       CALL BAS$MAT_INVERT (input_matrix.rx.da, output_matrix.wx.da)
                     0000   994 ;
                     0000   995 ; INPUT PARAMETERS:
                     0000   996 ;
00000004             0000   997 ;       input_matrix = 4
                     0000   998 ;
                     0000   999 ; IMPLICIT INPUTS:
                     0000  1000 ;
                     0000  1001 ;       Scale from the callers frame to scale double precision.
                     0000  1002 ;
                     0000  1003 ; OUTPUT PARAMETERS:
                     0000  1004 ;
00000008             0000  1005 ;       output_matrix = 8
                     0000  1006 ;
                     0000  1007 ; IMPLICIT OUTPUTS:
                     0000  1008 ;
                     0000  1009 ;       NONE
                     0000  1010 ;
                     0000  1011 ; FUNCTION VALUE:
                     0000  1012 ; COMPLETION CODES:
                     0000  1013 ;
                     0000  1014 ;       NONE
                     0000  1015 ;
                     0000  1016 ; SIDE EFFECTS:
                     0000  1017 ;
                     0000  1018 ;       This routine calls the matrix copy routine which calls
                     0000  1019 ;       the redimensioning routine and the array element
                     0000  1020 ;       fetch and store routines and therefore may signal any of their errors.
                     0000  1021 ;       It may also signal any of the errors listed in the externals section.
                     0000  1022 ;       It may also cause the destination array to have different dimensions.
                     0000  1023 ;
                     0000  1024 ;--
                     0000  1025
                 4FFC 0000  1026          .ENTRY BAS$MAT_INV, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,IV>
                     0002  1027
                     0002  1028 ;+
```

```
                                0002  1029 ;       REGISTER USAGE
                                0002  1030 ;       R0 - R8 destroyed by store routines
                                0002  1031 ;       R9      place to store element of 1st array while fetching 2nd element
                                0002  1032 ;       R10     pointer to dest matrix descriptor (except for double in which
                                0002  1033 ;               case R10 is part of double value R9-R10)
                                0002  1034 ;       R11     current value of inner subscript
                                0002  1035 ;-
                                0002  1036
                                0002  1037 ;+
                                0002  1038 ; Put routine arguments into registers for ease of use.
                                0002  1039 ; If block 2 of array descriptor (multipliers) is not present then error.
                                0002  1040 ;-
                                0002  1041
    52    04 AC    D0          0002  1042         MOVL    input_matrix(AP), R2                ; ptr to input array descr
27 0A A2    07    E1          0006  1043         BBC     #DSC$V_FL_BOUNDS, DSC$B_AFLAGS(R2), ERR_ARGDONMAT
                                000B  1044                                                 ; exit if block 3 not
                                000B  1045                                                 ; present in descriptor
    02    0B A2    91          000B  1046         CMPB    DSC$B_DIMCT(R2), #2                 ; 2 dimensional?
                14    12        000F  1047         BNEQU   ERR_MATDIMERR                       ; if not, error
28 A2    20 A2    D1          0011  1048         CMPL    dsc$l_u1_2(R2), dsc$l_u2_2(R2)      ; see if matrix to be inverted
                                0016  1049                                                 ;  is square, since inversion
                                0016  1050                                                 ;  will be done in output matrx
                                0016  1051                                                 ;  and it will have lower bnds
                                0016  1052                                                 ;  of 0, only check upper bnds
                34    13        0016  1053         BEQL    INIT_TWO_SUBS                       ; square matrix, continue
                                0018  1054                                                 ;  else fall into error
                                0018  1055
                                0018  1056 ERR_ARRMUSSQU:
     00000000'8F    DD          0018  1057         PUSHL   #BAS$K_ARRMUSSQU                    ; Signal error, array to be
00000000'GF    01    FB          001E  1058         CALLS   #1, G^BAS$$STOP                     ;  inverted is not square
                                0025  1059
                                0025  1060 ERR_MATDIMERR:
     00000000'8F    DD          0025  1061         PUSHL   #BAS$K_MATDIMERR                    ; Signal error, input matrix
00000000'GF    01    FB          002B  1062         CALLS   #1, G^BAS$$STOP                     ;  is not 2 dimensional
                                0032  1063
                                0032  1064 ERR_ARGDONMAT:
     00000000'8F    DD          0032  1065         PUSHL   #BAS$K_ARGDONMAT                    ; signal error,
00000000'GF    01    FB          0038  1066         CALLS   #1, G^BAS$$STOP                     ;  block 2 or 3 absent
                                003F  1067
                                003F  1068 ERR_CANINVMAT:
     00000000'8F    DD          003F  1069         PUSHL   #BAS$K_CANINVMAT                    ; signal error, matrix
00000000'GF    01    FB          0045  1070         CALLS   #1, G^BAS$$STOP                     ;  is singular
                                004C  1071
                                004C  1072 ;+
                                004C  1073 ; There are 2 subscripts.  Copy the input matrix to the output matrix so
                                004C  1074 ; that the inversion may be done in place.  The copy will do the necessary
                                004C  1075 ; redimensioning of the output array.
                                004C  1076 ;-
                                004C  1077
                                004C  1078 INIT_TWO_SUBS:
        08 AC    DD          004C  1079         PUSHL   output_matrix(AP)                   ; destination of copy
           52    DD          004F  1080         PUSHL   R2                                  ; source for copy
00000000'GF    02    FB          0051  1081         CALLS   #2, G^BAS$MAT_ASSIGN                ; copy to output matrix
                                0058  1082
                                0058  1083 ;+
                                0058  1084 ; Initialize 4 vectors to keep track of row and column permutations during
                                0058  1085 ; the invert.  The 4 vectors are interwoven so that addressing one vector
```

```
                                0058  1086 ; won't be dependent on the length of the other vectors.  Note that the vectors
                                0058  1087 ; will put 4 times the width of the array of longwords on the stack.  Perhaps
                                0058  1088 ; this should be done in heap storage.  however, space for arrays is usually
                                0058  1089 ; allocated on the stack, and the 4n for the vectors is minor when compared to
                                0058  1090 ; the n**2 space used for the array for large n.
                                0058  1091 ;-
                                0058  1092
          50    20 A2    DO   0058  1093          MOVL    dsc$l_u1_2(R2), R0              ; get # of rows and cols
                50    DD   005C  1094 1$:       PUSHL   R0                              ; nth element of row vector
                50    DD   005E  1095           PUSHL   R0                              ; nth element of col vector
                50    DD   0060  1096           PUSHL   R0                              ; nth element of tag vector
                50    DD   0062  1097           PUSHL   R0                              ; nth element of loc vector
             F5 50    F5   0064  1098           SOBGTR  R0, 1$                          ; do for all elements of row
                                0067  1099                                             ;  and cols
                                0067  1100
                                0067  1101 ;+
                                0067  1102 ; Initialize remaining stack temporaries
                                0067  1103 ;-
                                0067  1104
       5E    0000005C 8F    C2   0067  1105          SUBL2   #92, SP                 ; save space on stack for
                                006E  1106                                             ;  7 temps, some may be hfloat
                                006E  1107                                             ;  also data and value_desc
                01    DD   006E  1108          PUSHL   #1                              ; initialize pivj
                01    DD   0070  1109          PUSHL   #1                              ; initialize pivi
                7E    7C   0072  1110          CLRQ    -(SP)                           ; space for 2 longword temps
                01    DD   0074  1111          PUSHL   #1                              ; initialize current_i
             20 A2    DD   0076  1112          PUSHL   dsc$l_u1_2(R2)                  ; initialize upperbnd for loops
          50    0C AD    D0   0079  1113          MOVL    SF$L_SAVE_FP(FP), R0            ;  get caller's frame and
       00000000'GF    16   007D  1114          JSB     G^BAS$$SCALE_R1                 ;  get scale in R0&R1
        28 AE    50    70   0083  1115          MOVD    R0, scale(SP)                   ;  save it for computations
                                0087  1116
                                0087  1117 ;+
                                0087  1118 ; Algorithm now differs according to data types
                                0087  1119 ;-
                                0087  1120
                                0087  1121 SEPARATE_DTYPES:
          59    08 AC    D0   0087  1122          MOVL    output_matrix(AP), R9           ; get pointer to output array
             58    59    D0   008B  1123          MOVL    R9, R8                          ; local copy which may change
    05    06    02 A8    8F   008E  1124 5$:       CASEB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                   0037' 0093  1125 2$:       .WORD   BYTE-2$                         ; code for byte dtype
                   15AD' 0095  1126           .WORD   WORD-2$                         ; code for word dtype
                   2B23' 0097  1127           .WORD   LONG-2$                         ; code for long dtype
                   002A' 0099  1128           .WORD   ERR_DATTYPEKR-2$                ; quad not supported
                   409C' 009B  1129           .WORD   FLOAT-2$                        ; code for float dtype
                   5612' 009D  1130           .WORD   DOUBLE-2$                       ; code for double dtype
                         009F  1131
                         009F  1132 ;+
                         009F  1133 ; G and h floating fall outside the range of the CASEB statement, so check
                         009F  1134 ; for them separately.
                         009F  1135 ;-
                         009F  1136
          1B    02 A8    91   009F  1137          CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_G
                03    12   00A3  1138          BNEQ    3$
                6BBF    31   00A5  1139          BRW     GFLOAT
                         00A8  1140
          1C    02 A8    91   00A8  1141 3$:       CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_H
                03    12   00AC  1142          BNEQ    4$
```

```
                 407B   31  00AE  1143          BRW     TO_HFLOAT
                                00B1  1144
              18 02 A8   91  00B1  1145 4$:      CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_DSC
                    06   12  00B5  1146          BNEQ    ERR_DATTYPERR
              58 04 A8   D0  00B7  1147          MOVL    4(R8), R8
                    D1   11  00BB  1148          BRB     5$
                                00BD  1149
                                00BD  1150 ERR_DATTYPERR:
          00000000'8F    DD  00BD  1151          PUSHL   #BAS$K_DATTYPERR                ; Signal error, unsupported
      00000000'GF    01    FB  00C3  1152          CALLS   #1, G^BAS$$STOP                 ; dtype in array desc
```

```
                        ;+
                        ; Input array is a byte array.  Use the macro to generate code.
                        ;-

00CA  1159 BYTE:    $BAS$MAT_INVERT B

                        ;+
                        ; Input array is a word array.  Use the macro to generate code.
                        ;-

1640  1165 WORD:    $BAS$MAT_INVERT W
```

```
                        1168 ;+
                        1169 ; Input array is a longword array.  Use the macro to generate code
                        1170 ;-
                        1171
2BB6                    1172 LONG:    $BAS$MAT_INVERT L
412C                    1173
```

```
                414F   31  412C  1175 TO_HFLOAT:      BRW     HFLOAT
```

```
                        1177 ;+
                        1178 ; Source1 array is a floating array.  Use the macro to generate code
                        1179 ;-
                        1180
                        1181 FLOAT:  $BAS$MAT_INVERT F
                        1182
```

```
                        ;+
                        ; Source1 array is a double array.  Use the macro to generate code
                        ;-
                        
56A5  1188 DOUBLE:      $BAS$MAT_INVERT D
                        
                        ;+
                        ; Input array is a gfloat array.  Use the macro to generate code.
                        ;-
                        
6C67  1194 GFLOAT:      $BAS$MAT_INVERT G
                        
                        ;+
                        ; Input array is an hfloat array.  Use the macro to generate code.
                        ;-
                        
827E  1200 HFLOAT:      $BAS$MAT_INVERT H
                        
9898  1202              .END
```

Symbol table

```
A_SUB_RKCI                      = 00000064
BAS$$SCALE_R1                     ********    X   00
BAS$$STOP                         ********    X   00
BAS$$STORE_DET                    ********    X   00
BAS$$STORE_DET_G                  ********    X   00
BAS$$STORE_DET_H                  ********    X   00
BAS$FETCH_BFA                     ********    X   00
BAS$FET_FA_B_R8                   ********    X   00
BAS$FET_FA_D_R8                   ********    X   00
BAS$FET_FA_F_R8                   ********    X   00
BAS$FET_FA_G_R8                   ********    X   00
BAS$FET_FA_H_R8                   ********    X   00
BAS$FET_FA_L_R8                   ********    X   00
BAS$FET_FA_W_R8                   ********    X   00
BAS$K_ARGDONMAT                   ********    X   00
BAS$K_ARRMUSSQU                   ********    X   00
BAS$K_CANINVMAT                   ********    X   00
BAS$K_DATTYPERR                   ********    X   00
BAS$K_MATDIMERR                   ********    X   00
BAS$MAT_ASSIGN                    ********    X   00
BAS$MAT_INV                       00000000 RG     03
BAS$STORE_BFA                     ********    X   00
BAS$STO_FA_B_R8                   ********    X   00
BAS$STO_FA_D_R8                   ********    X   00
BAS$STO_FA_F_R8                   ********    X   00
BAS$STO_FA_G_R8                   ********    X   00
BAS$STO_FA_H_R8                   ********    X   00
BAS$STO_FA_L_R8                   ********    X   00
BAS$STO_FA_W_R8                   ********    X   00
BYTE                              000000CA R      03
CURRENT_I                       = 00000004
CURRENT_J                       = 00000008
CURRENT_K                       = 0000000C
C_SUB_0                         = 00000064
C_SUB_I                         = 00000060
DETERMINANT                     = 00000030
DOUBLE                            000056A5 R      03
DSC$A_A0                        = 00000010
DSC$B_AFLAGS                    = 0000000A
DSC$B_CLASS                     = 00000003
DSC$B_DIMCT                     = 0000000B
DSC$B_DTYPE                     = 00000002
DSC$K_CLASS_BFA                 = 000000BF
DSC$K_DTYPE_B                   = 00000006
DSC$K_DTYPE_D                   = 0000000B
DSC$K_DTYPE_DSC                 = 00000018
DSC$K_DTYPE_G                   = 0000001B
DSC$K_DTYPE_H                   = 0000001C
DSC$L_L1_1                      = 00000018
DSC$L_L1_2                      = 0000001C
DSC$L_L2_2                      = 00000024
DSC$L_M1                        = 00000014
DSC$L_M2                        = 00000018
DSC$L_U1_1                      = 0000001C
DSC$L_U1_2                      = 00000020
DSC$L_U2_2                      = 00000028
DSC$V_FL_BOUNDS                 = 00000007
DSC$W_LENGTH                    = 00000000
ERR_ARGDONMAT                     00000032 R      03
ERR_ARRMUSSQU                     00000018 R      03
ERR_CANINVMAT                     0000003F R      03
ERR_DATTYPERR                     000000BD R      03
ERR_MATDIMERR                     00000025 R      03
FIND_1ST_PIVOTB                   000000CA R      03
FIND_1ST_PIVOTD                   000056A5 R      03
FIND_1ST_PIVOTF                   0000412F R      03
FIND_1ST_PIVOTG                   00006C67 R      03
FIND_1ST_PIVOTH                   0000827E R      03
FIND_1ST_PIVOTL                   00002BB6 R      03
FIND_1ST_PIVOTW                   00001640 R      03
FLOAT                             0000412F R      03
GFLOAT                            00006C67 R      03
HFLOAT                            0000827E R      03
INIT_TWO_SUBS                     0000004C R      03
INPUT_MATRIX                    = 00000004
INVERT_COLS_B                     00000A59 R      03
INVERT_COLS_D                     0000605C R      03
INVERT_COLS_F                     00004ABE R      03
INVERT_COLS_G                     00007645 R      03
INVERT_COLS_H                     00008C5A R      03
INVERT_COLS_L                     00003545 R      03
INVERT_COLS_W                     00001FCF R      03
INVERT_ROWS_B                     0000063D R      03
INVERT_ROWS_D                     00005C40 R      03
INVERT_ROWS_F                     000046A2 R      03
INVERT_ROWS_G                     00007211 R      03
INVERT_ROWS_H                     00008826 R      03
INVERT_ROWS_L                     00003129 R      03
INVERT_ROWS_W                     00001BB3 R      03
LOC_SUB_0                       = 00000070
LOC_SUB_C_SUB_I                 = 00000058
LOC_SUB_R_SUB_I                 = 00000058
LONG                              00002BB6 R      03
LOOP_I_B                          000002CE R      03
LOOP_I_D                          000058AE R      03
LOOP_I_F                          00004333 R      03
LOOP_I_G                          00006E7D R      03
LOOP_I_H                          00008495 R      03
LOOP_I_L                          00002DBA R      03
LOOP_I_W                          00001844 R      03
LOOP_J_B                          00001353 R      03
LOOP_J_D                          0000696A R      03
LOOP_J_F                          000053B8 R      03
LOOP_J_G                          00007F7C R      03
LOOP_J_H                          00009596 R      03
LOOP_J_L                          00003E3F R      03
LOOP_J_W                          000028C9 R      03
LOOP_K_B                          00000E5B R      03
LOOP_K_D                          0000645E R      03
LOOP_K_F                          00004EC0 R      03
LOOP_K_G                          00007A5F R      03
LOOP_K_H                          00009074 R      03
LOOP_K_L                          00003947 R      03
LOOP_K_W                          000023D1 R      03
```

```
MAXIMUM_SUB                     = 00000000
MTH$DINT_R4                       ********    X   00
OUTPUT_MATRIX                   = 00000008
PIVI                            = 00000010
PIVJ                            = 00000014
PIVOT                           = 00000018
R_SUB_0                         = 00000068
R_SUB_I                         = 0000005C
R_SUB_K                         = 00000058
SAVE_ELEMENT                    = 00000064
SAVE_RESULT                     = 00000028
SCALE                           = 00000028
SEPARATE_DTYPES                   00000087 R      03
SF$L_SAVE_FP                    = 0000000C
START_K_LOOP_B                    00000E61 R      03
START_K_LOOP_D                    00006464 R      03
START_K_LOOP_F                    00004EC6 R      03
START_K_LOOP_G                    00007A65 R      03
START_K_LOOP_H                    0000907B R      03
START_K_LOOP_L                    0000394D R      03
START_K_LOOP_W                    000023D7 R      03
TAG_SOB_0                       = 0000006C
TEMP                              00000000 R      02
TEMP_PIVI                       = 00000000
TEMP_PIVJ                       = 00000004
TEMP_PIVOT                      = 00000008
TEMP_PIVOT_ABS                  = 00000018
TO_HFLOAT                         0000412C R      03
WORD                              00001640 R      03
```

## Psect synopsis

| PSECT name | Allocation | PSECT No. | Attributes |
|---|---|---|---|
| . ABS . | 00000000 ( 0.) | 00 ( 0.) | NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE |
| $ABS$ | 00000000 ( 0.) | 01 ( 1.) | NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE |
| _BAS$DATA | 00000010 ( 16.) | 02 ( 2.) | PIC USR CON REL LCL NOSHR NOEXE RD WRT NOVEC BYTE |
| _BAS$CODE | 00009898 (39064.) | 03 ( 3.) | PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG |

## Performance indicators

| Phase | Page faults | CPU Time | Elapsed Time |
|---|---|---|---|
| Initialization | 29 | 00:00:00.08 | 00:00:00.53 |
| Command processing | 118 | 00:00:00.61 | 00:00:02.60 |
| Pass 1 | 1372 | 00:00:59.84 | 00:02:13.54 |
| Symbol table sort | 5 | 00:00:02.54 | 00:00:05.77 |
| Pass 2 | 876 | 00:00:13.31 | 00:00:28.96 |
| Symbol table output | 19 | 00:00:00.13 | 00:00:00.25 |
| Psect synopsis output | 4 | 00:00:00.02 | 00:00:00.11 |
| Cross-reference output | 0 | 00:00:00.00 | 00:00:00.00 |
| Assembler run totals | 2426 | 00:01:16.53 | 00:02:51.76 |

The working set limit was 1200 pages.
424912 bytes (830 pages) of virtual memory were used to buffer the intermediate code.
There were 70 pages of symbol table space allocated to hold 297 non-local and 1223 local symbols.
1202 source lines were read in Pass 1, producing 114 object records in Pass 2.
69 pages of virtual memory were used to define 11 macros.

+----------------------------+
! Macro library statistics !
+----------------------------+

| Macro library name | Macros defined |
|---|---|
| _$255$DUA28:[BASRTL.OBJ]BASRTL.MLB;1 | 2 |
| _$255$DUA28:[SYSLIB]STARLET.MLB;2 | 5 |
| TOTALS (all libraries) | 7 |

493 GETS were required to define 7 macros.

There were no errors, warnings or information messages.

```
MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS$:BASMATINV/OBJ=OBJ$:BASMATINV MSRC$:BASMATINV/UPDATE=(ENH$:BASMATINV)+LI
```

BASMATMUL
LIS

BASMATIO
LIS

BASMATRED
LIS

BASMATNUL
LIS

BASMATINV
LIS