XIII.A. DYNAMIC ALLOCATION OF ARRAY STORAGE

Because SAMMY was developed prior to the advent of FORTRAN 90, which supports dynamic storage allocation, SAMMY uses its own scheme for dynamic allocation of array storage. This scheme should be effectively invisible to the most SAMMY users. On porting the code to a new machine, minor modification may be needed: If the total memory on your machine is larger or smaller than the default value, you may need to change the total array size during the "configure" part of the installation process. See Section XIII.F to learn how this is done.

Historically there have been many advantages to using dynamic allocation of array storage. First, core requirements are kept to a minimum, since only the array length actually needed is allocated and temporary arrays are released when no longer needed. Secondly, because allocation is made during the execution of a program, substantial changes in the dimensions for a specific case do not require recompilation of the program. Finally, when recompilation is needed (when the maximum array space requirement must be changed), only a few routines need to be recompiled.

The SAMMY scheme for dynamic allocation of array storage is best illustrated with a simple example. Let us suppose that two vectors V1 and V2, both of length N, are to be initialized, added, and stored in V1, after which V2 is no longer required. A program to perform these operations is given in Table XIII A.1. Mnemonic names can be used in the usual manner in all subroutines, provided arrays are input to the subroutines through argument listings.

Table XIII A.1. Illustration of dynamic allocation of array storage

```
      PROGRAM MAIN
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(-12345:12345)
      NSIZE = 12345
      N = 50
      IV1 = IDIMEN (N, 1, 'V1  N, 1')        #Allocate storage for V1
      IV2 = IDIMEN (N, 1, 'V2  N, 1')        #Allocate storage for V1
      CALL SET (A(IV1), A(IV2), N)           #Initialize V1 and V2
      CALL ADD (A(IV1), A(IV2), N)           #Add V1 = V1 + V2
      I = IDIMEN (IV2, -1, 'I   I, -1')      #Release storage for V2
      ...
      I = IDIMEN (0, 0, 'I    0, 0')         #Request that IDIMEN print out the
      STOP                                   #           maximum length used
      END
```

```
      SUBROUTINE SET (V1, V2, N)
      DIMENSION V1(N), V2(N)
      READ (11) V1
      READ (11) V2
      RETURN
      END
```

```
      SUBROUTINE ADD (V1, V2, N)
      DIMENSION V1(*), V2(*)
      DO I=1,N
          V1(I) = V1(I) + V2(I)
      END DO
      RETURN
      END
```

What follows is a brief description of the scheme currently in use in SAMMY for dynamic allocation of array storage. Most users of the code will not need to be concerned with the remainder of this section. The information is included here for completeness' sake, and for the benefit of anyone who may need to modify the code.

The current scheme is essentially the same as in previous releases of the SAMMY code, though different in detail. All arrays required by the computer program are stored in one array A, dimensioned A(-*n*:*n*) where *n* is a large number. [Use of this form, rather than the more common A(*n*), makes use of the negative integers and thus effectively doubles the maximum size of the array available with the largest possible 8-byte (32-bit) integer (2^{31}-1 = 2 147 483 647, since the 32nd bit is used for the sign). For a very few applications, this large size is needed and can be accommodated on computers with sufficient memory.] The precise value to be used for *n* is machine dependent. The SAMMY default is 65 million, but this can be overwritten when the code is configured for a specific machine, as described in Section XIII.F.

As shown in Table XIII A.1, allocation of space in this array is accomplished via a call to FUNCTION IDIMEN, which keeps track of the last location allocated, and appends the new array to that position. When an array is no longer needed, its space is released via another call to IDIMEN. At the end of each segment, IDIMEN is called one more time to report the maximum size actually used in that segment. IDIMEN also issues a warning if more than 2*n* words are required in the array A(-*n*:*n*), and then aborts the run.

Three arguments are used for IDIMEN (m1, m2, a3): The third, a3, is an arbitrary alphanumeric phrase used by the author for debugging. The second, m2, tells IDIMEN what to do with the first, m1. If m2 > 0, an array of size m1 is allocated. If m2 < 0, everything in A(-*n*:*n*) beyond position A(m1) is released for reuse. If m2 = 0, the total amount of storage used since the previous such call is reported (written into the SAMMY.LPT file).

Users familiar with the array storage system in earlier versions of SAMMY will note some differences in the 7.0.0 release of the code, especially with respect to the array sizes printed in the SAMMY.LPT file.

First, the storage requirement for segment XCT will generally be twice the size for the same segment in previous versions of the code; nevertheless, the largest requirement (for all segments) remains approximately the same. At the beginning of segment XCT, the new array-storage system allots all of the storage (for theory and derivatives, original and Doppler broadened, for example) needed to store intermediate results. SAMMY reuses those same storage locations as it goes from segment to segment. For a typical run including both Doppler and resolution broadening, after completion of segment FGM (Doppler), the "original" array positions will be re-used to hold the resolution-broadened values.

In the old version of the code, only the arrays needed in segment XCT were defined in XCT; these were then written to a temporary file and the space released. Then in FGM, space was allocated for the original arrays and for the Doppler-broadened arrays; the original values were read in from the temporary file. When the FGM calculations finished, the Doppler-broadened values were written to a temporary file and the space (for all these arrays) was released. This process was repeated in every module of the code.

Hence, the new system uses essentially the same total storage space but eliminates the reads and writes to the temporary files. Total time for a SAMMY run is therefore decreased. (Future development will likely eliminate nearly all of the temporary files. For more on the use of temporary files, see Section XIII.B.)

As indicated above, SAMMY allocates the array space for *storing* the values and derivatives at the beginning of segment XCT. However, it does not allocate all array space needed for all of the processes involved in *calculating* those values and derivatives. Thus, for example, the SSM segment (which calculates self-shielding and multiple-scattering corrections) requires many auxiliary calculations, for which it needs extra array space.