MET3220C
Computational Statistics

Programming – AS #6
Dr. Mark Bourassa

Topics:
Dynamic Memory Allocation
Array Operations

Turn in your program and discuss chi squared results.
Due in one week.

Dynamic Memory Allocation

- Dynamic Array Allocation is used to create variables (space in memory) anywhere in the code. This is nice because
  - You can tailor array sizes to match changing conditions
    - This make code versitile.
  - You can deallocate it when you are done with it.
    - This allows memory (which is sometimes a limiting consideration) to be used form any applications.
    - Program size can drop a great deal!
- The number of dimensions must be specified with the regularly declared variables, but the size of dimensions can be specified later.
- REAL, ALLOCATABLE, dimension(:,:,:) : array_name
  - Would be a three dimensional array.
  - Each ':' indicates a dimension
  - array_name is the name of the variable.

Example of Dynamic Allocation

```
 REAL, ALLOCATABLE, DIMENSION (:) :: qscat_spd_array
 max_num_spd = 1E6
 ALLOCATE (qscat_spd_array(max_num_spd), STAT=status)
```

You can check to see if the allocation worked:

```
IF (allocated(qscat_spd_array)) THEN
 PRINT *, 'Array allocated: status= ', status
 qscat_spd_array = 0.0
ELSE
 PRINT *, 'Array NOT allocated: status= ', status
END IF
```

A status of zero is good. If status is not used the program will exit if allocation fails

Freeing Dynamic Memory

- The DEALLOCATE command is used to free up memory from old dynamically allocated variables
- DEALLOCATE (allocated_array_name [, stat=status])
  - The terms in itallics are variables
  - allocated_array_name is the name of a variable that was setup through dynamic memory allocation
  - If stat is not used, the code will exit if the deallocation fails

Array Operations

- FORTRAN 90 (and FORTRAN 95 and FORTRAN 03) allow array manipulations.
  - This is something really cool that to the best of my knowledge cannot be done in C or C++.
- We have used such operations to set whole arrays to the same value
  - E.g., qscat_spd_array = 0.0
  - This sets every value in the array to zero
- Similarly, we could square every value in an array:
  - qscat_spd_array = qscat_spd_array**2
- We can also take sums in one line:
  - sum_qscat_spd = SUM (qscat_spd_array)
  - OR a mean:
  - sum_qscat_spd = SUM (qscat_spd_array) / n_good_data
- Why would the above mean be correct if the array was allocated for more speed values than n_good_data?

Array Operations

- The previous calculation of a mean (or a sum of squares) works because the extra array values are all zero, and they do not contribute to a sum.
- What happens if we try this for the sum of the log of the speeds?
  - sum_log_spd = SUM (LOG (qscat_spd_array + small))
- The above would be rather bad because the natural log of the variable small is substantial!
- To deal with this we much use only the part of the array that contains meaningful data (that is speeds).
  - sum_log_spd = SUM (LOG (qscat_spd_array(1:n_good_data) + small))
- In general, we can specify a portion of the array as follows.
  - Array_name(starting_index : ending_index)

Assignment #6

- Copy the code from assignment #5 to assignment #6.
- Add code to dynamically allocate an array of speed data. See previous example.
- Store the speed data is the dynamically allocated array.
- Delete the sums that are inside the loop that reads the data.
- After the loop that reads the data, use array operations to calculate these sums.
- Near the end of your code, calculate chi-squared values for the Gaussian distribution and the log-normal distribution.
  - Write the two values to the screen, identifying which is which.
- Comment on which fit is better, and if either is indistinguishable from the observed histogram.