Parameter Sweeping for Age-Structure Model Tutorial
by Matt Beasley

Model from Module 13.3, "Time after Time—Age- and Stage-Structured Models"

Introduction to Computational Science:
Modeling and Simulation for the Sciences, 2nd Edition

Angela B. Shiflet and George W. Shiflet
Wofford College

This tutorial details how to add MPI functionality to a relatively basic parameter sweeping C program. Each step has a corresponding spot or spots in the program *parameterSweep.c* where the code should be edited. These spots will be indicated by the following line(s) for each step *x*:

```
/************************ x ************************/
```

1. Include the MPI library header file and define global variables *SERVER* as 0 and *TAG* as 2.

2. Initialize variables *my_rank*, *world_size*, *source*, *destination*, *tag*, *length* and *status*. All of the types of these variables will be *int* except for *status*, which is of the *MPI_Status* type. Also, set *destination* to *SERVER* and *tag* to *TAG*.

3. Initialize MPI itself and the rank and size as *my_rank* and *world_size*.

4. Indicate to use a conditional for the section of the code will be performed by the process with rank *SERVER*. It is recommended to shift this section of code to the right for more clarity.

5. Use the MPI broadcast function to send *p*, *total*, *size*, *positions*, *combinations*, and *leslieMatrix* to the other processes.

6. Initialize a variable *eigenvalues* and create a nested *for* loop to receive the number of eigenvalues that each process is going to compute in the outer loop and then the combinations and eigenvalues in the inner loop.

7. Using a nested *for* loop, continue to print the remaining eigenvalues that were obtained from the other processes to the output file.

8. Before ending the server's section, close the input file and output file. Because the files do not need to be closed at the end of the code, the two lines that close the files towards the end can be commented out.

9. Using an *else* statement, indicate the section of code that will be performed by the processes that are not the server. It is recommended to shift this section of code to the right for more clarity.

10. Use the MPI broadcast function to receive all of the information sent by the server.

11. Initialize a variable *eigPerProcess* that approximates how many eigenvalues each process will calculate; the last process will have a smaller number if the number of processes does not evenly divide the total number of eigenvalues. Initialize variables *start* and *finish* to determine which sections of the total that process will calculate. A conditional should then be used to account for the final process.

12. Declare a variable *skip*, and initialize it to 0. Using a conditional, set *skip* to 1 if *start* becomes greater than *finish*. This accounts for the possibility of there being more processes than eigenvalues.

13. Declare a variable *numOfEigens*, and set it to the number of eigenvalues that the current process will calculate. Then, if *skip* has been set to 1, set *numOfEigens* to 0. Regardless of the value of *skip*, send *numOfEigens* to the server so that the server process will know how many eigenvalues to expect from this process.

14. Use a conditional to check if *skip* is still set to 0, because otherwise there is no need to calculate any eigenvalues. It is recommended to shift this section of code to the right for more clarity. Also, make sure that in the outermost for loop in this section, *i* will go through the correct section of the eigenvalues for each process.

15. Since the server process is now printing all the eigenvalues, there is no need for any of the other processes to print anything. This section can therefore be commented out.

16. After calculating its eigenvalues, each process will need to send the current combination number and the calculated eigenvalue to the server.

17. Finally, end the MPI section of code with the correct statement before the return statement of the main function.