AM 129 Homework 3 Report Deliverables

Owen Morehead

Nov 3 2020

Overall I found success in writing the Fortran code which approximates the value of pi using partial sums of the series:

$$\pi = \sum_{n=0}^{\infty} 16^{-n}\left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6}\right) \tag{1}$$

The code needed to accomplish this was in essence a loop that continues to add each summation term together for increasing values of n until the difference between π_{appx} and $\pi_{true} = acos(-1.d0)$ (where the -1.d0 gives us the real number calculated in double precision) is less than the declared error threshold.

For all four threshold values used, the sum converged fairly quickly and the number of summation terms needed to meet the threshold is shown in the table below.

Table of π Approximations Using Series Summation

Error Threshold	1.0e-04	1.0e-08	1.0e-12	1.0e-16
Number summation terms	3	5	8	11
Numerical value of π	3.1415873903465816	3.1415926454603365	3.1415926535889729	3.1415926535897931

We see that only 11 summation terms are needed to get a value of π_{appx} accurate up to 1.0e-16. For most practical purposes this is an accurate enough value and shows that when using double precision, the value of pi can indeed be calculated using the series above.

I chose to use primarily the same compiler flags that were used for the debugging section of this homework. The one that came in handy for me was -fdefault-real-8 and -fdefault-double-8, which converts all numbers to double precision which are not explicitly defined as so. This was useful because at first my values of π_{appx} were not as accurate as they should be and this was because I did not declare all real numbers in **Equation 1** above to be stored in 8 bytes. When using the Fortran flags stated, we can just leave these numbers as

typed in the equation (without .d0 at the end for example) and the value of π_{appx} is calculated correctly in double precision.
Utilizing a makefile is helpful in that we can specify how exactly we want to compile our code. It allows for quicker and easier compilation since I specified object and compiler targets. To compile the code all I need to call is make pi, which compiles pimod.f90 and pi.f90. The *phony* target *clean* is also used to get rid of unnecessary files created after compiling.

In terms of debugging the gaussian elimination code part of this assignment, everything went well and gdb was helpful when it came to debugging a couple different runtime errors. The advised list of compiler flags are the right combination for this task and provide useful debugging checks and backtrace printing (helpful for more runtime errors). Some syntax errors were simple enough to catch by eye and the flags point these out as well. These included things like the print statement errors (no * was used which is needed for proper syntax), syntax errors found at multiple variable declarations (such as proper *intent* and matrix declarations), and misspellings. I don't think it's necessary to go over every error found but I will say that using gdb was helpful with a couple of the runtime errors since it can show more specifically what the compiler was accessing when the error occurred. The segmentation fault - invalid memory reference error I got was pointed out to be from the fact that variables A and b are declared allocatable but are not allocated anywhere with specific dimensions. Also gdb pointed out in a runtime error the fact that the subroutines were not being called correctly and must either be in a contains or module statement or be after the program and not before. These were a couple of the important errors I dealt with. To get rid of the warning caused by PRINTINFO it was necessary to change the file extension to .F90 to enable the C-style preprocessor. I didn't feel the need to utilize this debugging tool which just allows the ability to debug and ignore code and the errors between #ifdef and #endif statements. Once there were no more errors or warnings the code ran successfully. I then solved the system $A^T x = b$ and obtained the solution:
$x = -2.5$, $y = 1.5$, $z = 0$.

I enjoyed learning to use and debug Fortran in more depth in this assignment. This concludes my deliverables report.