AM 129 Homework 2 Report Deliverables

Owen Morehead

Oct 26 2020

Overall I found success in writing and implementing the code at hand. There were minimal unexpected issues and my primary debugging strategy was to use print statements throughout the code and run individual parts to ensure things were going correctly. This included debugging the matrix vector product function with print statements and by following the error messages until it ran properly. And doing the same with a couple of the other subroutines created.

The reason we can use real(dp) throughout the files dft.f90 and dftmod.f90 is because it is initialized inside the utility.f90 module. It is communicated there that:

integer, parameter :: dp = kind(0.d0).

Inside the other files we can call use utility, only : dp, pi to use these commands wherever we wish.

So when we compile these three files (utility.o, dftmod.o, dft.o) into our executable file, both dftmod.f90 and dft.f90 files will accept what was created in the utility.f90 file. The utility file simply defines useful content, which in our case was real(dp) and the value of pi.

The file dftmod.f90 essentially holds the problem specific parameters; the function and subroutines that get called on variables declared in the dft.f90 file. This file is where the pieces get connected. The matrices and vectors needed will be built here and then passed through functions to find the real Fourier cosine series directly and solve the boundary value problem.

The included functions at the bottom of dft.f90 did not have to be declared external at the top of the program block because they are written inside the dft.f90 program and the compiler knows that those are functions and not variables. Notice that those functions are inside the end program statement. The elemental keyword is used for elementwise operations that are applied on an array. The input array to these elemental functions defined in dft.f90 is x.

Table of Max Errors for Different N Values for dp = kind(0.d0):

N	20	40	60	80	100
Max Error	1.4503e-02	9.9623e-07	5.6431e-10	2.6645e-15	3.1086e-15

Comments on behavior: We see that as the number of grid points, the max error of our numerical boundary value solution gets smaller and starts to asymptote around an error of 4e-015 which is very small and shows the preciseness of our fortran implementation. I tested N = 200 and obtained an error of 4.1078e-015. We see that error can be decreased significantly by increasing N even if it's only to 40 or 60 data points. And as we continue to exponentially increase N, the maximum error stops decreasing as we are using 8 bit real numbers and can only get to certain high values of precision.

In utility.f90, change dp = kind(0.d0) to dp = kind(0.e0). Call *make clean* and *make dft*, and repeat the error table:

Table of Max Errors for Different N Values for dp = kind(0.e0):

N	20	40	60	80	100
Max Error	1.4503e-02	3.6955e-06	2.1458e-06	1.1920e-06	2.3842e-06

Comments on behavior: We can see that the smallest max error value is much smaller using 4-byte real numbers, as expected. Only 32 bits are used here to represent floating point numbers rather than 64 bits, and so output error values are larger and not as precise. In essence, double precision provides greater range and precision which is why we use it primarily.

Inside the *makefile* flags are specified that control the compilation. These flags are essentially rules that the program will follow. The first is FC = gfortran which is just a shortcut statement for gfortran. Compiler flags are also specified with the FFLAGS = ..., in our makefile there are flags set for debugging and optimizing. The macro OBJ = ... is just the list of files we wish to compile together and is written so we only have to write this list of files once. This macro is called in the line below, dft:..., where we create the .o files from .f90 files. The .o:... recipe is used to compile these files. Lastly the phony target clean removes files created when compiling the code in order to clean things up and delete unnecessary files. Clean is the action, and the keyword phony is used because there is no new file being created rather just the action, clean, described below as removing all .o, .mod, and ~ files after compiling.

Concluding Statements:

All in all things went well throughout this assignment. As someone who is new to the fortran language, I felt like I've learned a lot and put together a lot of useful information that is helping me learn the language. Although most of the code was already written for us, it was nice to look through it to see how the program is set up and implemented, how discretizing integrals can be very useful, and how to add the final bits of code which allows us to solve the given BVP using a spectral collocation method. Essentially we choose to write the forcing function as a sum of sines and cosines by finding its discrete fourier transform. The fourier coefficients can be found and we can write out our solution to the boundary value problem. We utilize matrices in our code so we can store the discrete integrals that makeup our transform. The coefficients are then scaled and rearranged into our solution u(x). The final output after compiling our files is the error between our numerical solution and the exact solution for the BVP. When storing our numbers in 8 bytes, the two solutions agree very well. It is shown that the larger the integral matrix is made, or the more we discretize our transform, the more accurate our solution gets to the exact value.