1) Real (fp) can be used throughout **boundaryValueProblem.f90** and **discreteFourierTransform.f90** because it is defined in our **utility.f90** file that's in the same directory, and both of these files **use** utility and specify the variables pi and fp.

2) **discreteFourierTransform.f90** initializes a function to multiply a matrix by a vector, and 2 more functions for matrix transformation and inverse matrix transformation, which approximate integrals of sine and cosine functions used to find fourier series coefficients. The integrals are incremented finitely (discretized) with even and odd coefficients of x, and are also incremented along their rows and columns.

boundaryValueProblem.f90 sets the forcing function. It then runs the functions initialized in discreteFourierTransform.f90, which solve for the 3 coefficients defined in a fourier transform. Those coefficients are multiplied by a wavenumber array (scaled by (1/1+k^2)) and then re-transformed through multiplication with the inverse matrix (tuned into u(x)) to put our fourier transform in its final form. This file also calculates an error rate, writes data to new files in our created data directory, and allocates and deallocates variable attributes whose size can be adjusted.

3) The variables at the top of boundaryValueProblem.f90 that aren't imported are marked *allocatable* because their sizes are not necessary to know at compile time. This helps with adjusting the size of the problem (can do this for different sizes of N) and staying organized with memory usage for large arrays/matrices. These variables are allocated and deallocated in their respective subroutines at the bottom of the file.

4) Elemental functions are defined to take scalar arguments. When they're called to take in an array, they're applied to that array element by element. Scalars defined in elemental functions can also only be intent(in).

num_points vs error (selected real kind 15)

N	Err
21	9.8179922227248362**E-003**
41	5.7127791075473056**E-007**
61	1.2797260318109238**E-007**

81	1.7118912509772599**E-008**
101	3.6223864352535884**E-008**

5) The lowest error value I could find was at 81 points. Beyond that, values stay in the **E-007** range unless you use a very large number, like 10,000. This might indicate a round off error with our floating point calculations.

num_points vs error (selected real kind 6)

N	**Err**
21	9.82248783**E-03**
41	8.22544098**E-06**
61	2.38418579**E-06**
81	4.52995300**E-06**
101	3.57627869**E-06**

6) This error seems more consistent , but the larger numbers indicate less accuracy (**E-06** range, not **E-007)**. This makes sense as single-precision values drop in accuracy after the 6th-7th decimal place, so our error values are not as precise. I looked at the plots in python as well and both versions look identical to me.

7) We're given a second order ODE system periodic with 2π, with a forcing function setting up a BVP. To approximate the forcing function, we use a fourier series. The fourier coefficients required for this are solved by using discretized integrals whose values are put in a matrix(alternating between odd and even values). The coefficients are then scaled and transformed by an inverse matrix and a wavenumber array which puts the series into its final BVP form. The matrices are of size N, which is the number of points we use to approximate the forcing function in our fourier series. Low values of N make a really bad approximation while higher values get better and better.