# ECE26400 Programming Assignment #2

This assignment is related to PE03 and PE04. In particular, the numerical integration method in PE04 is one of the functions you have to implement and submit.

In this assignment, you will use the numerical integration method you have implemented in PE04 and an extension of it to calculate a Fourier series associated with a function (perhaps unknown to you).

The main learning goals are:

1. How to calculate the Fourier coefficients of a Fourier series associated with a function
2. How to define a structure required.
3. How to use `argc` and `argv` correctly in `main`.

## 1 Getting started

You should unzip on eceprog.ecn.purdue.edu the zip file `pa02_files.zip` using the following command:

```
unzip pa02_files.zip
```

The zip file `pa02_files.zip` contains a folder, named `PA02`, and seven files and a subfolder (with more files) within the folder:

1. `answer02.c`: This is the file that you have to modify and submit. It has the description of the numerical integration method in PE04, the description of an extension of it for computing the integrand needed for Fourier coefficients, and the description of a function to compute Fourier coefficients.
2. `answer02.h`: This is a "header" file and it declares the functions you will be writing for this assignment. You also have to define the structure required. *You have to submit this file.*
3. `pa02.c`: You should use this file to write the main function that would properly initialize the structure that would be passed into the function to compute Fourier coefficients. You will also submit this file.
4. `aux02.h`: This is an include file that declares a few unknown functions for which you have to calculate their associated Fourier coefficients.
5. `aux02.o`: This file provides the object code for the unknown functions declared in `aux02.h`.
6. `utility02.h`: This is an include file to declare utility functions for you to use in printing and also to plot functions in `matlab` format.
7. `utility02.o`: This file provides the object code for the utility functions declared in `utility02.h`.
8. `plots` subfolder: This subfolder contains 6 files which are plots for `unknown_function_3`: `data5.m`, `data10.m`, `data20.m`, `data40.m`, `data80.m`, and `data160.m`. These plots are in `matlab` format.

To get started, read this document in its entirety. You will be writing code in the `answer02.c` file. You will also write code in the `pa02.c` file to call the correct functions in `answer02.c` file. You should also read both `answer02.c` and `pa02.c` to figure out the structure that you have to define in `answer02.h`.

## 2 Fourier series

In the CompE and EE curricula in ECE, all students have to take the course ECE 30100 Signals and Systems. In ECE 30100, a Fourier series is typically expressed in the form of complex exponential. In this assignment, we will deal with the form that has terms that are more familiar to you: sine and cosine. While ECE 30100 will teach you the mathematics that ground the Fourier series, which requires many derivations, this assignment shows you how to perform the numerical calculation to obtain a Fourier series.

The following is based on the materials from taken

- Introduction to Partial Differential Equations by Professor Peter J. Olver (at University of Minnesota), Undergraduate Texts in Mathematics, Springer, New York, 2014
- http://www.stewartcalculus.com/data/CALCULUS%20Early%20Vectors/upfiles/FourierSeriesEV.pdf (late Dr. James Stewart, most recently Professor of Mathematics at McMaster University)
- http://mathworld.wolfram.com/FourierSeries.html
- https://en.wikipedia.org/wiki/Fourier_series

Why do we have to know Fourier Series? The following two paragraphs are lifted from Professor Peter Olver's write up:

> "Just before 1800, the French mathematician/physicist/engineer Jean Baptiste Joseph Fourier made an astonishing discovery. As a result of his investigations into the partial differential equations modeling vibration and heat propagation in bodies, Fourier was led to claim that "every" function could be represented by an infinite series of elementary trigonometric functions – sines and cosines. As an example, consider the sound produced by a musical instrument, e.g., piano, violin, trumpet, oboe, or drum. Decomposing the signal into its trigonometric constituents reveals the fundamental frequencies (tones, overtones, etc.) that are combined to produce its distinctive timbre. The Fourier decomposition lies at the heart of modern electronic music; a synthesizer combines pure sine and cosine tones to reproduce the diverse sounds of instruments, both natural and artificial, according to Fourier's general prescription."
>
> "Fourier analysis is an essential component of much of modern applied (and pure) mathematics. It forms an exceptionally powerful analytical tool for solving a broad range of partial differential equations. Applications in pure mathematics, physics and engineering are almost too numerous to catalogue – typing in "Fourier" in the subject index of a modern science library will dramatically demonstrate just how ubiquitous these methods are. Fourier analysis lies at the heart of signal processing, including audio, speech, images, videos, seismic data, radio transmissions, and so on. Many modern technological advances, including television, music CD's and DVD's, video movies, computer graphics, image processing, and fingerprint analysis and storage, are, in one way or another, founded upon the many ramifications of Fourier's discovery. In your career as a mathematician, scientist or engineer, you will find that Fourier theory, like calculus and linear algebra, is one of the most basic and essential tools in your mathematical arsenal. Mastery of the subject is unavoidable."

In reality, every piecewise continuous function $f$ over a range $[a,b]$ can have a Fourier series $F$ associated with it. However, that does not mean that $f$ is equal to $F$. However, if $f$ is a periodic with period $(b-a)$

and $f$ and its derivative are continuous over $[a,b]$, $f$ is equal to $F$ where $f$ is continuous. Where $f$ is discontinuous at $x$, $F(x)$ is the average of the right and left limits, i.e., $F(x) = [f(x^+) + f(x^-)]/2$ (the values of the function $f$ as we get closer and closer to $x$ from its right and left).

A function $f$ is said to be periodic if the function $f(x)$ repeats itself in regular intervals or periods. A sine wave or a cosine wave is periodic, with the period being $2\pi$, where $\pi$ is the mathematical constant that defines the ratio of a circle's circumference to its diameter. For this assignment, we use the macro `M_PI` to define the numeric value of $\pi$ in `answer02.h`. If we use a different standard in the options for `gcc`, `M_PI` may be defined in `math.h`.

Typically, the Fourier series is expressed over a range $[-\pi,\pi]$. We provide the general representation here, where $f$ is defined over a range $[a,b]$. If $f$ is assumed to be periodic, with $2L = (b-a)$ being the period,

$$F(x) = a_0/2 + \sum_{n=1}^{\infty} a_n \cos((n\cdot\pi\cdot x)/L) + \sum_{n=1}^{\infty} b_n \sin((n\cdot\pi\cdot x)/L)$$

where $\sum_{n=1}^{\infty}$ is the summation of terms with $n = 1, 2, ...$ till infinity, cos is the cosine function, and sin is the sine function. Therefore,

$$\sum_{n=1}^{\infty} a_n \cos((n\cdot\pi\cdot x)/L) = a_1 \cos((1\cdot\pi\cdot x)/L) + a_2 \cos((2\cdot\pi\cdot x)/L) + ...$$

$$\sum_{n=1}^{\infty} b_n \sin((n\cdot\pi\cdot x)/L) = b_1 \sin((1\cdot\pi\cdot x)/L) + b_2 \sin((2\cdot\pi\cdot x)/L) + ...$$

In the Fourier series, $a_0$, $a_1$, $a_2$, ..., and $b_1$, $b_2$, ... are the Fourier coefficients, which are defined as follows:

$$\begin{aligned} a_0 &= (1/L)\cdot\int_a^b f(x)dx \\ a_n &= (1/L)\cdot\int_a^b f(x)\cos((n\cdot\pi\cdot x)/L)dx, \;\; n = 1,2,... \\ b_n &= (1/L)\cdot\int_a^b f(x)\sin((n\cdot\pi\cdot x)/L)dx, \;\; n = 1,2,... \end{aligned}$$

where $\int_a^b$ is the integration over the bounded interval $[a,b]$. Your task in this assignment is to write two functions to compute $a_0$, $a_1$, $a_2$, ... and $b_1$, $b_2$, ...[1]

Why do we care about periodic functions? Many astronomical phenomena are periodic in nature. The rotation of the moon around Earth, our heartbeats, and vibrating strings are some examples. Even for man-made objects, we can find periodic behavior. We rely on a periodic clock signal to synchronize the operations of registers or flip-flops in an integrated circuit.

If we are to design a synthesizer that can play different types of musical instruments, we have to obtain the Fourier series of a particular instrument when a particular note is played so that we can use the Fourier series to re-construct the sound made by that instrument. All we have to do is make sure that we can generate sine and cosine waves and we can then combine (add) these waves (weighted by appropriated $a_1$, $a_2$, ..., and $b_1$, $b_2$, ...).

[1]When $[a,b] = [-\pi,\pi]$, we obtain the more familiar form of Fourier series, where $2L = 2\pi$, i.e., $L = \pi$: $F(x) = a_0/2 + \sum_{n=1}^{\infty} a_n \cos(n\cdot x) + \sum_{n=1}^{\infty} b_n \sin(n\cdot x)$, where $a_0 = (1/\pi)\cdot\int_{-\pi}^{\pi} f(x)dx$, $a_n = (1/\pi)\cdot\int_{-\pi}^{\pi} f(x)\cos(n\cdot x)dx$, $b_n = (1/\pi)\cdot\int_{-\pi}^{\pi} f(x)\sin(n\cdot x)dx$.

Of course, it is impossible to combine an infinite series. Therefore, we typically approximate the Fourier series with only the first $(k+1)$ terms (if we start the index at $a_0$ and end the index at $a_k$):

$$F(x) \approx a_0/2 + \sum_{n=1}^{k} a_n \cos((n \cdot \pi \cdot x)/L) + \sum_{n=1}^{k} b_n \sin((n \cdot \pi \cdot x)/L)$$

where

$$\begin{aligned} a_0 &= (1/L) \cdot \int_a^b f(x)dx \\ a_n &= (1/L) \cdot \int_a^b f(x)\cos((n \cdot \pi \cdot x)/L)dx, \;\; n = 1,2,...,k \\ b_n &= (1/L) \cdot \int_a^b f(x)\sin((n \cdot \pi \cdot x)/L)dx, \;\; n = 1,2,...,k \end{aligned}$$

The computation of $a_0$, $a_n$, and $b_n$ involves integration, a topic that you have dealt with in PE03 and PE04. In particular, we will use the Simpson's method in PE04 in this assignment.

## 3 Functions/Structure you have to define

### 3.1 Structure you have to define

In PE04, you were asked to define a structure called `integrand`. You have to do the same here. Please read up on PE04 on that. As in PE04, this is the only change you can make to `answer02.h`. You are not allowed to make other changes in `answer02.h`.

This structure will be used in another structure defined in `answer02.h` called `fourier`.

```
typedef struct _fourier {
   integrand intg;
   int n_terms;
   double *a_i;
   double *b_i;
} fourier;
```

In the structure `fourier`, we have a field that is of type `integrand`, which is defined by you. The field `n_terms` stores the number of terms in the truncated Fourier series. If the coefficient with the highest index is $k$, the field `n_terms` should have a value of $k+1$.

The field `a_i` stores an address pointing to a block of `n_terms` `double`'s, and `a_i[0]` corresponds to $a_0$, `a_i[1]` corresponds to $a_1$, ..., and `a_i[n_terms-1]` corresponds to $a_{(\texttt{n\_terms}-1)}$. Therefore, if `n_terms` has a value of $k+1$, `a_i[k]` corresponds to the coefficient with the highest index, i.e., $a_k$.

The field `b_i` stores an address pointing to a block of `n_terms` `double`'s. We do not use the first entry `b_i[0]`. Here, `b_i[1]` corresponds $b_1$, `b_i[2]` corresponds to $b_2$, ..., and `b_i[n_terms-1]` corresponds to $b_{(\texttt{n\_terms}-1)}$, i.e., $b_k$.

`pa02.c` contains code fragment that allocates the memory blocks to store the coefficients, and store the addresses of these memory blocks in `a_i` and `b_i`. It also contains code fragment to free the memory blocks. Do not modify them. Modification to these code fragments may lead to memory errors.

## 3.2 Functions you have to write in `answer02.c`

These three functions are declared in `answer02.h`:

```
double simpson_numerical_integration(integrand intg_arg);

double simpson_numerical_integration_for_fourier(integrand intg_arg, int n,
                                                 double (*cos_sin)(double));

void compute_fourier_coefficients(fourier fourier_arg);
```

You have to define these functions in `answer02.c`. If you have to write additional functions, please declare and define them as static functions in `answer02.c`. `Do not` declare your functions in `answer02.h`. Otherwise, we may not be able to evaluate your submission properly.

### 3.2.1 Function `simpson_numerical_integration`

The function `simpson_numerical_integration(integrand intg_arg)` computes

$$\int_a^b f(x)dx.$$

This function does not compute $a_0$ because term $(\frac{1}{L})$ is missing from the preceding expression. See Section 2 for the expression of $a_0$.

All information necessary for the integration should be contained in `intg_arg`. This should be the same as that in PE04. Simply copy that over.

### 3.2.2 Function `simpson_numerical_integration_for_fourier`

For the following integrations:

$$\int_a^b f(x)\cos((n\cdot\pi\cdot x)/L)dx, \;\; n = 1,2,...,k$$
$$\int_a^b f(x)\sin((n\cdot\pi\cdot x)/L)dx, \;\; n = 1,2,...,k$$

where $2L = (b-a)$, the function `simpson_numerical_integration_for_fourier(integrand intg_arg, int n, double (*cos_sin)(double))` should be used. Again, this function does not compute $a_n$ or $b_n$, for $n \neq 0$, because the term $(\frac{1}{L})$ is missing from the preceding expressions. See Section 2 for the expressions of $a_n$ and $b_n$ when $n \neq 0$.

Here, the function being integrated is

$$f(x)\cos((n\cdot\pi\cdot x)/L)$$

or

$$f(x)\sin((n\cdot\pi\cdot x)/L)$$

The function being integrated has two components: (i) the address of function $f(x)$ is contained in `intg_arg`, the first parameter of the C function, (ii) the address stored in the third parameter `cos_sin` should be either the address of the function `cos` or the address of the function `sin`, both of which are declared in `math.h` and available in the math library. The second parameter `n` (int) supplied to the C function is the $n$ in the cos or sin function. $\pi$ is defined as `M_PI` in `answer02.h`.

Depending on the Fourier coefficient being computed, the caller function sends the address of the appropriate function, `sin` or `cos`, to `simpson_numerical_integration_for_fourier`.

Both `simpson_numerical_integration` and `simpson_numerical_integration_for_fourier` are similar in their implementations. The main difference is one is computing the integrand of $f(x)$ and the other one is computing the integrand of $f(x)\cos((n\cdot\pi\cdot x)/L)$ or $f(x)\sin((n\cdot\pi\cdot x)/L)$.

#### 3.2.3 Function `compute_fourier_coefficients`

The function `compute_fourier_coefficients(fourier fourier_arg)` is the function that calls the two integration functions to compute the Fourier coefficients. The following coefficients should be computed and stored in the arrays whose addresses are stored in the `a_i` and `b_i` fields of `fourier_arg`:

$$\begin{aligned}
a_0 &= (1/L)\cdot\int_a^b f(x)dx \\
a_n &= (1/L)\cdot\int_a^b f(x)\cos((n\cdot\pi\cdot x)/L)dx,\ \ n=1,...,\texttt{n\_terms}-1 \\
b_n &= (1/L)\cdot\int_a^b f(x)\sin((n\cdot\pi\cdot x)/L)dx,\ \ n=1,...,\texttt{n\_terms}-1
\end{aligned}$$

where `n_terms` is a field in `fourier_arg`. `cos` function and `sin` function from the math library should be passed to `simpson_numerical_integration_for_fourier` as the third parameter depending which coefficient is being computed. Moreover, it passes `fourier_arg.intg` and appropriate `n` as the first parameter and second parameter, respectively, to `simpson_numerical_integration_for_fourier`.

### 3.3 `main` function in `pa02.c`

We provide three unknown functions in `aux02.h` and `aux02.o`. In `aux02.h`, we declare three functions: `unknown_function_1`, `unknown_function_2`, and `unknown_function_3`. `aux02.o` contains the object code for these three functions. If the executable is provided with correct arguments, it should compute the Fourier series of one of these three functions.

The executable of this assignment expects 6 arguments. If executable is not supplied with exactly 6 arguments, return `EXIT_FAILURE`.

The first argument specifies the function (declared in `aux02.h`) of which the Fourier series should be computed. If the first argument to the executable is `"1"`, you should compute the Fourier series of `unknown_function_1`. If the first argument is `"2"`, you compute the Fourier series of `unknown_function_2`. If the first argument is ”3”, you compute the Fourier series of of `unknown_function_3`.

If the first argument does not match `"1"`, `"2"`, or `"3"`, the executable returns `EXIT_FAILURE`.

The second argument and third argument specify the lower limit and upper limit of the period. You should use `strtod` to convert the second argument and third argument into `double`’s. (The function `strtod` is kind of similar to `strtol`.) If any of these two limits are of the wrong format, out of range, infinity (`inf` or `-inf`), or not a number (`nan` or `-nan`), the executable should return `EXIT_FAILURE`. **Moreover, if both limits are the same, you have to return `EXIT_FAILURE` since the period is not well-defined.**

The fourth argument provides the number of intervals (`int`) you should use for the integration. The argument should be provided in the base 10 format. You could use `strtol` to convert the fourth argument into a `long` (and then store in an `int`). If the argument is of the wrong format or out of range, the executable should return `EXIT_FAILURE`. Moreover, if the conversion of the fourth argument results in a value that is less than 1 or greater than `INT_MAX`, you should return `EXIT_FAILURE`.

The fifth argument provides the number of $a_i$ terms in the Fourier series to be computed. The argument should be provided in the base 10 format. You could use `strtol` to convert the fifth argument into a `long` (and then store in an `int`). If the argument is of the wrong format or out of range, the executable should return `EXIT_FAILURE`. Moreover, if the conversion of the fifth argument results in a value that is less than 1 or greater than `INT_MAX`, you should return `EXIT_FAILURE`.

You should declare a variable `fourier_arg` (of type `fourier`), and initialize its fields properly.

Assuming that you have initialized `fourier_arg.n_terms` properly, we provide the code to allocate memory for two arrays of `fourier_arg.n_terms` `double`'s. The addresses of the allocated arrays are stored in `fourier_arg.a_i` and `fourier_arg.b_i`.

`fourier_arg` should be passed to the function `compute_fourier_coefficients` for the computation of the coefficients of the Fourier series.

Upon the successful completion of the function `compute_fourier_coefficients`, you should save the coefficients into a file by calling the `save_fourier_coefficients` function from `utility02.o`. The name of the file should be specified by the sixth argument of the executable. The `save_fourier_coefficients` function saves the coefficients into a file using a binary format. A total of $2 * \texttt{n\_terms} - 1$ `double`'s are saved. The first `n_terms` `double`'s are the $a_i$ coefficients. The next $(\texttt{n\_terms} - 1)$ `double`'s are the $b_i$ coefficients. (The size of the file should be $(2 * \texttt{n\_terms} - 1) \times \texttt{sizeof(double)} = (2 * \texttt{n\_terms} - 1) \times 8$. You can verify the size of the file by using the terminal command "`ls -l`".)

It is difficult for a human being to interpret the data stored in a binary file. If you would like to know what those Fourier coefficients are, you can use the `print_fourier_coefficients` function to print the coefficients to `stderr`. The function prints `n_terms` $a_i$ coefficients (one coefficient per line), and then prints $(\texttt{n\_terms} - 1)$ $b_i$ coefficients (one coefficient per line).

You should not have any printout to `stdout`.

After saving, free the memory allocated for the arrays. The code is already provided. Return `EXIT_SUCCESS` from the `main` function.

You probably have to include more `.h` files.

### 3.4 Printing, helper functions and macros

All debugging, error, or log statements in `answer02.c` and `pa02.c` should be printed to `stderr`.

You may define your own helper functions in `pa02.c` and `answer02.c`. All these functions should be declared and defined as static functions. In other words, the scope of these functions are only within the files that they are found. Declaring and defining these functions as static eliminate the conflicts in function names.

You should modify and submit the `answer02.h`.

You should not use macros that start with `T_` (Upper case T and underscore) in their names. We will use such macros in the evaluation of your submissions. If you use such macros, we may not be able to evaluate your submission properly.

## 4 Compiling your program

We use the same flags introduced in PE03 and PE04 to compile the program for debugging purpose.

```
gcc -std=c99 -Wall -Wshadow -Wvla -Werror -g -pedantic -fstack-protector-strong \
    --param ssp-buffer-size=1 pa02.c answer02.c aux02.o utility02.o -o pa02 -lm
```

When we evaluate your program, we also use the optimization flag `-O3` (uppercase letter `O` and number three) instead of the `-g` flag as follows:

```
gcc -std=c99 -Wall -Wshadow -Wvla -Werror -O3 -pedantic -fstack-protector-strong \
    --param ssp-buffer-size=1 pa02.c answer02.c aux02.o utility02.o -o pa02 -lm
```

You are recommended to copy the `Makefile` from PE01 and modify it appropriately for PA02.

## 5 Running and testing your program

To run your program, supply six appropriate arguments to the executable. For example,

```
./pa02 1 0.0 10.0 20 10 output.fs
```

Note that `output.fs` is the name of the binary file that would store the coefficients. In this case, the file is not created because the `main` function is incomplete.

### 5.1 Testing your program

How do you know whether your implementation is correct when you have no idea what function, for which you are computing the Fourier coefficients, is? The nice thing is that for certain functions, you actually know what the Fourier coefficients should be. For example, if you are computing the Fourier coefficients for the cosine function, $cos(x)$, the Fourier coefficients will be all zero except for $a_1 = 1$. Of course, since you are performing numerical computation with roundoff errors, you may not get exactly zero and you may not get exactly one.

Similarly, if you are computing the Fourier coefficients for the sine function, $sin(x)$, the Fourier coefficients will be zero except for $b_1 = 1$.

Note that in both cases, I am assuming you are computing the coefficients over the range of $[-\pi, \pi]$ because the two functions have a period of $2\pi$.

Here, assume that you write your own `unknown_function_1`, `unknown_function_2`, and `unknown_function_3` in a file called `my_aux02.c`. Now, you compile with the following command:

```
gcc -std=c99 -Wall -Wshadow -Wvla -Werror -g -pedantic -fstack-protector-strong \
    --param ssp-buffer-size=1 pa02.c answer02.c my_aux02.c utility02.o -o pa02 -lm
```

Here, we assume that you are using some functions in `math.h`. Therefore, the `-lm` option is still being used so that we can link to the math library.

If you use a sine or a cosine function for your test, you should use a range whose lower limit is $-\pi$ and upper limit is $\pi$ (or any lower limit and an upper limit that is lower limit + $2\pi$). You may try different numbers of intervals to divide up the range for integration, and also different number of terms in the truncated Fourier series.

To also assist you on checking how closely the (truncated) Fourier series approximates the original function (whether the function is periodic or not), we provide in utility02.o a function for you to plot the original function (**in blue**) and the approximation obtained by Fourier series (**in red**) in `matlab` format.

The plot function is declared as follows:

```
void function_plot(double (*original_func)(double),
                   double lower_limit, double upper_limit,
                   double *a_i, double *b_i, int n_terms, char *filename);
```

The parameters are the address of the original function, the lower limit and upper limit defining the interval, the addresses of the arrays containing the coefficients, and the number of terms in the array, and the name of the file in which you want to store the plot. As a plot is in `matlab` format, you want to provide a filename that has a file extension `.m`.

After you have computed the Fourier coefficients, you can call this plot function to plot the original function and associated Fourier series. Let's assume that we name the file containing the plot `data.m`. To view the plot, invoke `matlab` from the directory that contains this file `data.m`. Within `matlab` and at the `matlab` prompt, type in `data` and hit return. A plot should be shown. You can examine the plot. The plot of the original function and that of the approximation should match very closely for the sine and cosine functions. For other more complicated functions, they would not match that closely for the following reasons:

1. The function may not be periodic. The computation of Fourier coefficients assume that it is periodic over the range provided as an argument.
2. The numerical integration performed is not exact. As the coefficients are computed based on numerical integration, accuracy loss is expected.
3. The number of terms you have specified is too small. When the number of terms is too small, you cannot capture the sharp changes in the original function. The problem does not go away by increasing the number of terms. Instead, you have an approximation that oscillates about the exact value. This is called the Gibbs phenomenon.

Among the three unknown functions, `unknown_function_1` is not periodic, `unknown_function_2` and `unknown_function_3` are both periodic with a period of $2\pi$.

In the subfolder `plots`, there are 6 plots: `data5.m`, `data10.m`, `data20.m`, `data40.m`, `data80.m`, and `data160.m`. They are obtained using `n_terms` = 5, 10, 20, 40, 80, and 160 for `unknown_function_3`. You use `data5`, `data10`, ... in `matlab` to show the plot.

The plot `data5.m` is obtained using the Fourier coefficients computed with the following command:

```
./pa02 3 -3.141593 3.141593 1000 5 output.fs
```

You can change the fifth argument to 10, 20, 40, 80, or 160 (from 5) to obtain the Fourier coefficients for other plots.

### 5.1.1 Optional seventh argument to the `main` function

To facilitate your use of the `function_plot` function for the debugging purpose, you are allowed to write your `main` function to accept an **optional** seventh argument. In such a version, your `main` function can still run successfully when we run the following command with `six arguments` to the executable:

```
./pa02 3 -3.141593 3.141593 1000 5 output.fs
```

Such a version will treat the seventh argument, if provided, as the name of the file into which you save the `matlab` plot.

```
./pa02 3 -3.141593 3.141593 1000 5 output.fs data5.m
```

Here, "`data5.m`" is provided as the seventh argument.

It is important that in such a version, your `main` function can handle six arguments or seven arguments properly.

## 6 Running `./pa02` under `valgrind`

You should also run `./pa02` with arguments under `valgrind`. To do that, you may use for example the following command:

```
valgrind --tool=memcheck --leak-check=yes --log-file=memcheck.log \
   ./pa02 1 0.0 10.0 20 10 output.fs
```

Note that you should use other input arguments to extensively test your function. If you follow the instructions and keep the `malloc` and `free` functions in the appropriate places, you should not have memory errors in this assignment.

It is possible to run `valgrind` with the simple command below.

```
valgrind ./pa02 1 0.0 10.0 20 10 output.fs
```

Please see PE01 description about `valgrind`.

## 7 Submission

You must submit a zip file called `PA02.zip`, which contains three files:

1. `answer02.h`
2. `answer02.c`
3. `pa02.c`

Assuming that you are in the folder that contains `answer02.h`, `answer02.c`, and `pa02.c`, use the following command to zip your files:

```
zip PA02.zip answer02.h answer02.c pa02.c
```

*Make sure that you name the zip file as `PA02.zip`. Moreover, the zip file should not contain any folders.* Submit `PA02.zip` through Brightspace. You may submit as many versions as you like. Brightspace will keep only the most recent submission, and we will grade only the final submission.

## 8 Grading

All debugging messages should be printed to `stderr`.

The only output we expect is a file that contains the Fourier coefficients.

We are not expecting you to plot the function. The function is provided only for the purpose of visualizing Fourier series. If you do that for your testing, remember to take that out before your submission.

Although you can make changes to `answer02.h` (since you are submitting the file), the only changes you are allowed to make is to define the type integrand in `answer02.h`. You should not make other changes to `answer02.h`.

It is important that if the instructor has a working version of `pa02.c`, it should be compilable with your `answer02.c` to produce an executable. For evaluation purpose, we will use different combinations of your submitted `.c` files and our `.c` files to generate different executables. If a particular combination does not allow an executable to be generated, you do not get any credit for the function(s) that the executable is supposed to evaluate. We use both `-g` and `-O3` flags (separately) to compile your submission. Your submission is evaluated only when compilation using each of the flags is successful.

Be aware that we set a time-limit for each test case based on the size of the test case. If your program does not complete its execution before the time limit for a test case, it is deemed to have failed the test case. We will not announce the time-limits that we will use. They will be very reasonable.

The `integrand` structure accounts for 20%, the `simpson_numerical_integration` function 20%, the `simpson_numerical_integration_for_fourier` function 20%, the `compute_fourier_coefficients` function 20%, and the `main` function 20%. In our evaluation of your implementation, some reasonable amount of roundoff error is acceptable. We will not announce the amount of roundoff error that we would tolerate.

**The occurrence of any memory issues (memory errors or memory leaks flagged in a `valgrind` report) will result in 50-point penalty.**

## 9 A few points to remember

All debugging messages should be printed to `stderr`.

You should not print anything to `stdout`. The Fourier coefficients should be saved to a file that is specified as one of the arguments to the executable.

You have to define the `integrand` structure in `answer02.h`, and you should not make other changes in the file.

Your executable should not plot the function (the original function and the Fourier series).

You are not submitting `aux02.h` and `utility.h`. Therefore, you should not make changes to `aux02.h` and `utility.h`.

You can declare and define additional static functions that you have to use in `pa02.c` and `answer02.c`.

You should not use macros that start with `T_`.

Grading of programming exercises and assignments is performed on machines with similar setup as eceprog.ecn.purdue.edu. You should perform testing of your work on eceprog.ecn.purdue.edu before submission. Correct output on your computer does not translate into correct output on eceprog.ecn.purdue.edu.