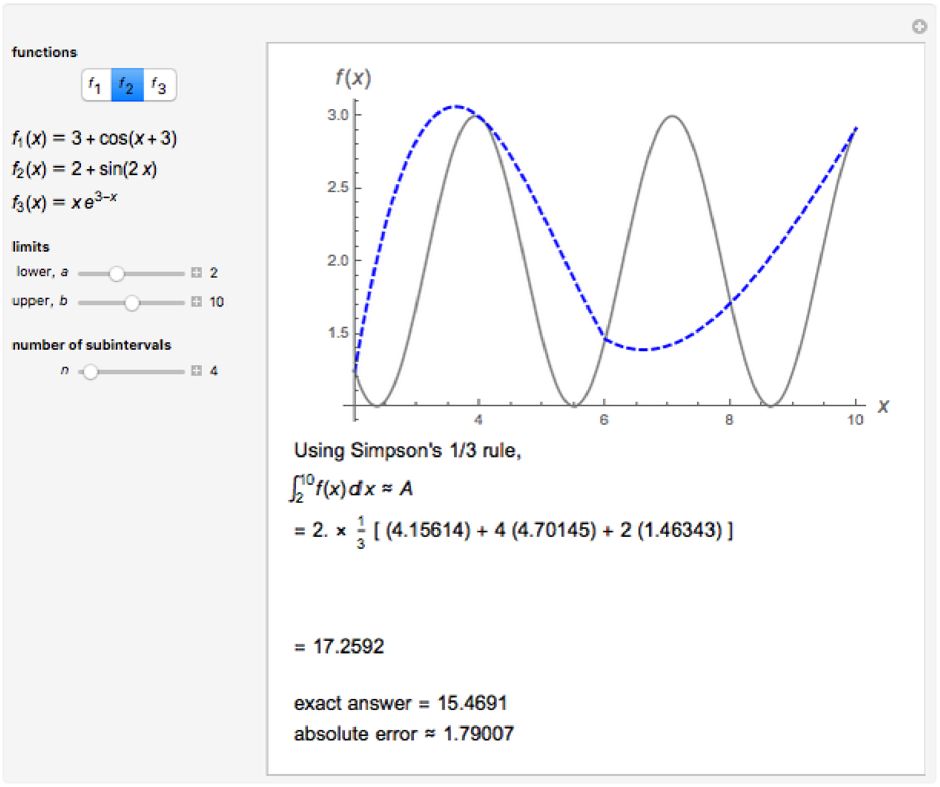
Assignment 2 Design Document

Program Requirements

When this function is run, it prints out the value of the integral of a certain function, within given bounds, at a given accuracy. When calling this program, one must provide four arguments. The first argument specifies which function will be integrated, the second specifies the lower bound, the third specifies the upper bound, while the last specifies the number of partitions that will be made to the area under the curve (the greater the number of partitions, the more accurate the integral).

Our first objective is to write the file mathlib.c which will house all the functions that may be integrated. Every function must perform the operation of its namesake (eg. sin(x), e^x, log(x), etc.). Our second objective is to write the file integrate.c, which will perform two roles. One of the roles will be performed in a function called ‘integrate()’ and it is to calculate the area under the curve of a given function within given bounds, using the Simpson’s ⅓ rule, to the accuracy of a specified even number of partitions. The other role will be performed by the main function and it is to match the first argument of a program call to a function within the given file functions.c (this will be accomplished using function pointers), and also to assign the values of the other arguments to variables that can be called by the ‘integrate()’ function.

This visual demonstrates how Simpson’s ⅓ rule is supposed to work:



$f_1(x) = 3 + \cos(x+3)$
$f_2(x) = 2 + \sin(2x)$
$f_3(x) = x e^{3-x}$

lower, a 2
upper, b 10
n 4

Using Simpson's 1/3 rule,

$\int_2^{10} f(x)\,dx \approx A$

$= 2. \times \frac{1}{3}$ [(4.15614) + 4 (4.70145) + 2 (1.46343)]

= 17.2592

exact answer = 15.4691
absolute error ≈ 1.79007

integrate.c Pseudocode

#define OPTIONS “abcdefghijp:q:n:H”

/*

As can be clearly seen, this function follows along the same tracks as those of the given implementation for the 3/8 rule. The necessary changes have been made to alter the process to calculate the integral according to the 1/3 rule.

*/

```
double integrate(double (*f)(double), double a, double b, uint32_t n) {
    Set h to be a double variable. Define it per the example given for the Simpson's ⅜ rule.
    Same as above for the variable sum.

    For every value i from 1 to n …
        If the value i is odd,
            sum += 4 * f(a + i * h)
        Else,
            sum += 2 * f(a + i * h)

    Multiply the value of sum by h/3.
    Return the value of sum.
}
```

```
/*
The usage prompt provides details on every possible command. The parameter here is a pointer
to a character in order to allow for passing the first argument of argv in the main function.
*/
void usage(char *exec_name) {
    Print the format in which the program must be called.
    Describe the purpose of the program.
    Show which functions correspond to the letters a-j.
    Specify that -H reveals this help message.
    Specify that -p sets the lower limit for the integration.
    Specify that -q sets the upper limit for the integration.
    Specify that -n sets the number of partitions to be used.
    Specify that -n should be a positive even integer.
    return;
}

int main(int argc, char **argv) {
    Assign 0 to the integer opt.
    Set a function pointer called fn to return the type double and assign it the value NULL.
    Declare the double variables low and high.
    Declare num_parts (the number of partitions) to be of the type uint32_t.
    // For our purposes, it would be more apt to think of the following as an array of character arrays.
    Create an array of pointers to characters called func_names[], which will contain the typed version of the functions in functions.c.
    Declare a variable of type uint8_t called func_name_index, which will later be used to specify which function in the array above has been selected, so as to be printed in the form it exists as in the array.

    Use getopt to parse the input arguments.
```

```
while ((opt  = getopt(argc, argv, OPTIONS)) != -1) {
    switch(opt):
    /*
    In the following cases a-j, the function pointer fn is assigned the address of the
    function in functions.c whose name corresponds to the case selected.
    Additionally, func_name_index is assigned the position of that corresponding
    function in func_names.
    */
    case(a):
        fn = a
        break
    Repeat the above for b - j.
    case(p):
        Read the next argument. Save it as a double variable named low.
        break
    case(q):
        Read the next argument. Save it as a double variable named high.
        break
    case(n):
        Read the next argument. Save it as a double variable named
num_partitions.
        break
    default:
```

```
                Print usage.

                Exit

        }


        Print the function that has been called, followed by the values the user has given as
arguments.

        For every value i from 2 to the upper bounds, incremented by 2 with every iteration:


            Call the integrate function using the parameters fn, low, high, and num_partitions.

        Return the value 0.

}
```

mathlib.c Pseudocode

The pseudocode for this file is already *de facto* given as part of the assignment document in the form of Python code for all functions, except for Cos(). Consequently, that is the function for which my own pseudocode is displayed below.

```
double Cos(double x) {

        Assign the value 1.0 to the double variable s.

        Assign the value 1.0 to the double variable v.

        Assign the value 1.0 to the double variable t.

        Assign the value 2.0 to the double variable k.
        While the absolute value of t is greater than the imported value of EPSILON …
            $t = (t * x^2) / (k * (k - 1))$
            Invert the sign of s.
            Increase the value of v by (s * t).
            Increment the value of k by 2.0.

        Return the value of v.
    }
```