numint

Nicolás A. Ortega

December 1, 2016

1 Usage

`numint` is a quick program written in C that calculates the area underneath a function using Left Rectangle, Right Rectangle, Middle-Point Rectangle, Trapezoidal, and Simpson part integrations. The usage is simple, first define the function you wish to integrate in the `` `func' `` function found at the end of the `` `main.c' `` file. After doing so compile and run the program. By default `numint` uses the CMake build system, therefore with CMake installed you could run the following commands from the root directory of the project:

```
$ cd build/
$ cmake ..
$ make
```

This would generate a file, `` `build/numint' `` which is the program's executable file. On UNIX systems you can run this file directly from the command-line from the `` `build/' `` directory by running `` `./numint' ``. At this point `numint` will prompt you with `` `Enter "min, max":' ``, which is asking you to enter the range in which you want to calculate the integral (e.g. `` `4, 6' ``), these numbers may contain decimals. After this you will be prompted `` `Number of parts:' `` which is asking for the number parts for the range previously given should be divided into (this number *cannot* contain a decimal).

If all has been done correctly then the program should print the results of the integrations for the formula in all five methods.

2 Examples

In order make sure that the program works the way we think it does we'd have to test it first. For these examples let's use the integral $\int_2^5 x^3 + 4x - 2$.

The results taken by hand for this integral in one part for all forms are as follows:

- Left Rectangle: $f(a) \cdot (b-a) = (2^3 + 4 \cdot 2 - 2) \cdot (5-2) = 42$
- Right Rectangle: $f(b) \cdot (b-a) = (5^3 + 4 \cdot 5 - 2) \cdot (5-2) = 429$
- Middle-Point Rectangle: $f(\frac{a+b}{2}) \cdot (b-a) = (3.5^3 + 4 \cdot 3.5 - 2) \cdot (5-2) = 164.625$
- Trapezoid: $\frac{f(a)+f(b)}{2} \cdot (b-a) = \frac{(2^3+4\cdot 2-2)+(5^3+4\cdot 5-2)}{2} \cdot (5-2) = 235.5$
- Simpson: $\frac{b-a}{6} \cdot (f(a) + f(b) + 4 \cdot f(\frac{a+b}{2})) = \frac{5-2}{6} \cdot ((2^3 + 4 \cdot 2 - 2) + (5^3 + 4 \cdot 5 - 2) + 4 \cdot (3.5^3 + 4 \cdot 3.5 - 2)) = 188.25$

If we run this in `numint` we get the following output:

```
Enter "min, max": 2, 5
Number of parts: 1
Left Rectangle: 42.000000
Right rectangle: 429.000000
Middle Point Rectangle: 164.625000
Trapezoidal: 235.500000
Simpson: 188.250000
```

As you can see you get the exact same results (with trailing zeros due to the `` `double' `` data type).

We have now seen how `numint` works for integrals in one part, but we should also test it for multiple parts. Let's use the same integral but test for 2 parts, which should give us slightly more accuracy. The results are as follows:

- Left Rectangle: $\sum_{i=1}^{2} f(x_i) \cdot (x_{i+1} - x_i) = 103.3125$
- Right Rectangle: $\sum_{i=1}^{2} f(x_{i+1}) \cdot (x_{i+1} - x_i) = 296.8125$
- Middle-Point Rectangle: $\sum_{i=1}^{2} f(\frac{x_i + x_{i+1}}{2}) \cdot (x_{i+1} - x_i) = 182.34375$
- Trapezoidal: $\sum_{i=1}^{2} \frac{f(x_i) + f(x_{i+1})}{2} \cdot (x_{i+1} - x_i) = 200.0625$

- Simpson: $\sum_{i=1}^{2} \frac{x_{i+1} - x_i}{6} \cdot (f(x_i) + f(x_{i+1}) + 4 \cdot f(\frac{x_i + x_{i+1}}{2})) = 188.25$

If we then run this in `numint` we get the following output:

```
Enter "min, max": 2, 5
Number of parts: 2
Left Rectangle: 103.312500
Right rectangle: 296.812500
Middle Point Rectangle: 182.343750
Trapezoidal: 200.062500
Simpson: 188.250000
```

Again the same exact results.

3 Technical Analysis

Now let's analyze how the code works. To begin let's look at the function $f(x)$ which is defined in the `` `func(double x)' `` C function. The function is defined as follows:

```
double func(double x) {
    return pow(x, 3.0) + 4.0 * x - 2.0;
}
```

It's a simple function that takes in `` `double' `` as a parameter and returns the result in the form of a `` `double' `` as well. The function itself uses the `` `pow()' `` from the C Math library which is what allows us to easily use exponents (in this case `` `pow(x, 3.0)' `` is equivalent to x^3). If you want to change the function you'll have to modify the `` `return' `` statement in this function.

Later, after retrieving the `` `min' ``, `` `max' ``, and `` `parts' `` variables that were prompted to the user we enter the main loop of the program for the calculations.

```
for(unsigned int i = 0; i < parts; ++i) {
    ...
}
```

This loop will create a variable `` `i' `` which will be useful later. This variable (as stated by the loop) will increment while it is less than the `` `parts' `` variable.

After this are the following lines:

```
double p0 = a + (i * (b - a) / parts);
double p1 = p0 + ((b - a) / parts);
```

Here we create two variables: `p0' and `p1'. In regards to the previous calculations done in the 'Examples' section, `p0' and `p1' would be equivalent to x_i and x_{i+1} respectively. So, let's imagine that we're looking at the first part of two for this integral that goes from 2 to 5 (like in the example). In this case, mathematically, the equation would look like this: $p0 = 2 + \frac{0 \cdot (5-2)}{2}$ and $p1 = p0 + \frac{5-2}{2}$. Now, why multiply by 0? Doesn't the loop start at 1? Well, in programming, luckily, we always start with 0 (remember in the `for' loop where it said `unsigned int i = 0'?). Since this is the first part we multiply by 0 because we want to start at 2. This is all being setup for the calculation of the integrals with parts.

Finally comes the calculation of the integrals, which is as follows:

```
lRect += func(p0) * (p1 - p0);
rRect += func(p1) * (p1 - p0);
mRect += func((p0 + p1) / 2) * (p1 - p0);
trap += (func(p0) + func(p1)) / 2 * (p1 - p0);
simp += (p1 - p0) / 6 * (func(p0) + func(p1) + 4 * func((p0 + p1) / 2));
```

All these variables (`lRect', `rRect', etc.) have all been assigned to 0 before entering the loop. Therefore, on each of these lines the integral for a given part of the function (where `p0' is the beginning of the part and `p1' is the end of the part) which is then added to all the other parts because of the `for' loop seen previously.

After this all the variables are printed to `stdout' (Standard Output) for the user to see the results.

4 Modifications

Currently `numint` is limited to a pre-coded function (as previously mentioned, this function can be modified in `func(double x)' found at the end of the file `main.c'). Modifications to the source for this project are permitted and encouraged, however all distributions of said changes must comply with the project license, which is the GNU General Public License version 3.

5 License

This document is licensed under a Creative Commons Attribution-Share Alike 4.0 International License.

Copyright © 2016 Nicolás A. Ortega <deathsbreed@themusicinnoise.net>