Coding the Simpson's Rules for data input

Ashley Gannon

Fall 2020

Recalling How to Make Arrays

Creating and passing arrays into a function

Recall how to declare/define an array

```
int myarr[4] = {4, 1, 8, 13}
```

Or by

```
int myarr[] = {4,1,8,13}
```

Passing an array into a function is easy.

```
double myfunc(int myarr[])
{
    ...
}
```

We intend to pass arrays into our function for x and f given below.

i	0	1	2	3	4	5	6	7
x_i	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7
$f(x_i)$	1.543	1.669	1.811	1.971	2.151	2.352	2.577	2.828

Writing the `DataSimpsonsRules` Code

Modifying our pseudocode for Simpson's 1/3 Rule

For the case where the number of segments, n, is even, we only need to apply Simpson's 1/3 Rule. The total integration using Simpson's 1/3 Rule can be represented as

$$I = \frac{h}{3}(f(x_0) + f(x_n)) + \frac{h}{3}\left(4\sum_{i=1,3,5\ldots}^{n-1} f(x_i) + 2\sum_{j=2,4,6,\ldots}^{n-2} f(x_j)\right)$$

Looking at the pseudocode from last class without the recursion,

```
declare function that takes in a, b, n
 declare/define h = (b-a)/n;
declare/define sum = (h/3)*(f(a)+f(b));
declare/define xi = a+h;

loop over i = 1,...,n-1
   if(i % 2 == 0)
      add 2*h/3*f(xi) to sum
   else
      add 4*h/3*f(xi) to sum
   update xi = xi +h;
```

We need to edit our code so that our function takes in arrays of values and function values and re-write it to use the values of the arrays.

Modifying our pseudocode for Simpson's 1/3 Rule

For the case where the number of segments, n, is even, we only need to apply Simpson's 1/3 Rule. The total integration using Simpson's 1/3 Rule can be represented as

$$I = \frac{h}{3}(f(x_0) + f(x_n)) + \frac{h}{3}\left(4\sum_{i=1,3,5\ldots}^{n-1} f(x_i) + 2\sum_{j=2,4,6,\ldots}^{n-2} f(x_j)\right)$$

Looking at the pseudocode from last class without the recursion,

```
declare function that takes in a, b, n

declare/define h = (x[n]-x[0])/n;
declare/define sum = (h/3)*(f[0]+f[n]);

loop over i = 1,...,n-1
   if(i % 2 == 0)
      add 2*h/3*f[i] to sum
   else
      add 4*h/3*f[i] to sum
```

We need to edit our code so that our function takes in arrays of values and function values and re-write it to use the values of the arrays.

Pseudocode for Simpson's 3/8 Rule

For the case where the number of segments, n, is odd AND divisible by 3, we only need to apply Simpson's 3/8 Rule. The total integration using Simpson's 3/8 Rule can be represented as

$$I = \frac{3h}{8}(f(x_0) + f(x_n)) + \frac{3h}{8}\left(\sum_{i=1}^{n-1} 3f(x_i)\right)$$

```
declare function that takes in a, b, n

declare/define h = (x[n]-x[0])/n;
declare/define sum = (3*h/8)*(f[0]+f[n]);

loop over i = 1,...,n-1
   add (3*h/8)*3*f[i] to the sum
```

Pseudocode for a combination of Simpson's 1/3 and 3/8 Rules

For the case where n is odd and NOT divisible by 3, we have to apply a combination of Simpson's Rules. Recall the example from last class where we were integrating the data set

i	0	1	2	3	4	5	6	7
x_i	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7
$f(x_i)$	1.543	1.669	1.811	1.971	2.151	2.352	2.577	2.828

We split up the domain

1. Use Simpson's 1/3 rule on the interval [1.0,1.4].
2. Use Simpson's 3/8 rule on the interval [1.4, 1.7].
3. Add them together.

Pseudocode for a combination of Simpson's 1/3 and 3/8 Rules

For the case where n is odd and NOT divisible by 3, we have to apply a combination of Simpson's Rules. Recall the example from last class where we were integrating the data set

i	0	1	2	3	4	5	6	7
x_i	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7
$f(x_i)$	1.543	1.669	1.811	1.971	2.151	2.352	2.577	2.828

We split up the domain

1. Use Simpson's 1/3 rule on the interval [1.0,1.4].
2. Use Simpson's 3/8 rule on the interval [1.4, 1.7].
3. Add them together.

Generally speaking,

1. Use Simpson's 1/3 rule on the interval [0, n-3].
2. Use Simpson's 3/8 rule on the interval [n-3, n].
3. Add them together.

Pseudocode for a combination of Simpson's 1/3 and 3/8 Rules

For the case where *n* is odd and NOT divisible by 3, we have to apply a combination of Simpson's Rules.

1. Use Simpson's 1/3 rule on the interval [0, n-3].
2. Use Simpson's 3/8 rule on the interval [n-3, n].
3. Add them together.

```
declare function that takes in a, b, n

declare/define h = (x[n]-x[0])/n;
//start with Simpson's 1/3 rule, leaving the last 3 segments for
 Simpson's 3/8 rule
declare/define sum = (h/3)*(f[0]+f[n-3]);
loop over i = 1,...,n-2
   if(i % 2 == 0)
      add 2*h/3*f[i] to sum
   else
      add 4*h/3*f[i] to sum

//Now we will apply Simpson's 3/8 rule to the last three segments
add (3*h/8)*(f[n-3]+f[n]) to the sum
loop over i =n-2,n-1
   add (3*h/8)*3*f to the sum
```

Let's code it!

We are going to write a function `DataSimpsonsRules` that takes in two arrays and the number of segments, n, and returns the sum evaluated by one of the three methods above.

```
double DataSimpsonsRules(double data[], double x[], double n)
double h = (data[n]-data[0])/n
double sum = 0.0
if (n % 2 == 0)
   Code for Case 1.
if (n % 3 == 0)
   Code for Case 2.
else
   Code for Case 3.
```

Testing this on our problem

i	0	1	2	3	4	5	6	7
x_i	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7
$f(x_i)$	1.543	1.669	1.811	1.971	2.151	2.352	2.577	2.828

We need to write our main

```
int main()
double data[] = {1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6,1.7}
double f[] = { 1.543, 1.669, 1.811, 1.971, 2.151, 2.352, 2.577,2.828
unsigned int n = sizeof(data) / sizeof(data[0]) - 1
double sum = DataSimpsonsRules(data, f,n)
```

Note, you don't have to use

```
unsigned int n = sizeof(data) / sizeof(data[0]) - 1
```

to define the number of segments, you can hard code it for a small example like this one. We will eventually learn how to import data, so defining it the way I have done here allows for n to be defined based on the length of any array. Meaning, if you have a very long data set, you won't have to count each point to determine how many segments you will have, the computer does that for you!