Integration with Unequal Segments

Ashley Gannon

Fall 2020

Integration with Unequal Segments

Integration with Unequal Segments

To this point, all formulas for numerical integration have been based on equispaced data.

In practice, there are many situations where this assumption does not hold and we must deal with unequal-sized segments. For example, experimentally derived data are often of this type.

For these cases, we need to change the way we define h.

Let's write a Trapezoid Rule function that takes in non equispaced data.

Thinking about this in regards to the Trapezoid Rule

Recall the approximate value for the integral with equispaced data was

$$I = \frac{h}{2}(f(x_0) + f(x_n)) + h\left(\sum_{i=1}^{n-1} f(x_i)\right)$$

Since our h value isn't the same for each segment, we can no longer write it this way. Instead,

$$I = \frac{h_1}{2}(f(x_0) + f(x_1)) + \frac{h_2}{2}(f(x_1) + f(x_2)) + ... + \frac{h_n}{2}(f(x_{n-1}) + f(x_n))$$

Where h_1 is the width of the first segment, h_2 is the width of the second segment and so on. So, instead of defining h once outside out loop, we need to calculate it each time we iterate.

The Composite Trapezoid Rule Pseudocode

Our old pseudocode:

```
function takes in a, b, n

declare/define h = (b-a)/n;
declare/define sum =
(h/2)*(f(a)+f(b));
declare/define xi = a+h;

loop over i = 1,...,n-1
add h*f(xi) to sum
update xi = xi +h;
```

Pseudocode for non equispaced data

```
function takes in x, fx, n

declare/define h = 0;
declare/define sum = 0;

loop over i = 1,...,n
calculate new h = x[i]-x[i-1]
add (h/2)*(fx[i-1]+fx[i]) to sum
```

Importing Data

Reading in data

In order to write and read files we need to include header file `<fstream>`.

Reading from file can be done by declaring `ifstream` in the following manner:

```
// create a binary file object and open a file that contains the data
we want
ifstream xdata("xdata.dat", ifstream::binary);;

// write data to file
for (int i = 0; i < N; i++)
{
// this takes the data on one line and stores in x[i]
inputFile » x[i];
}

// close file
inputFile.close();
```

Reading in data

In order to write and read files we need to include header file `<fstream>`.

Reading from file can be done by declaring `ifstream` in the following manner:

```
// create a binary file object and open a file that contains the data
we want
ifstream xdata("xdata.dat", ifstream::binary);;

// write data to file
for (int i = 0; i < N; i++)
{
// this takes the data on one line and stores in x[i]
inputFile » x[i];
}

// close file
inputFile.close();
```

But what if we don't know how many elements our data has?

C++ Vectors

In order to use vectors, we need to include header file `<vector>`.

Vectors are sequence containers representing arrays that can change in size.

Internally, vectors use a dynamically allocated array to store their elements.

They are declared as follows:

```
vector<double> x
```

C++ Vectors

In order to use vectors, we need to include header file `<vector>`.

Vectors are sequence containers representing arrays that can change in size.

Internally, vectors use a dynamically allocated array to store their elements.

They are declared as follows:

```
vector<double> x
```

If we want to save each element of the data file to the vector, we can use the `push_back` member function. This member function adds a new element to the end of the vector, `x`.

```
while (xdata » n)
    x.push_back(n);
xdata.close()
```

Reading in x and fx

Download "fdata.dat" and "xdata.dat" from Canvas, you'll need them for this.

```
ifstream xdata("xdata.dat", ifstream::binary)
ifstream fdata("fdata.dat", ifstream::binary)

vector<double> x,fx;
double n;
while (xdata » n)
    x.push_back(n)
xdata.close()

while (fdata » n)
    fx.push_back(n)
fdata.close()
```

To get the size of the vector, we use the public member function `size`.

```
int nseg = x.size() - 1
```