MATHS 2104 Numerical Methods
Practical 2: Numerical integration and differentiation

Contents

1 Functions of functions and midpoint integration

The midpoint integration rule for equispaced data is

$$\int_a^b f(x)\,\mathrm{d}x \approx h\sum_{j=1}^{n} f_j, \tag{1}$$

where $f_j = f(x_j)$, $x_j = a + (j - 1/2)h$ for $j = 1, \ldots, n$ and $h = (b - a)/n$.

In this exercise, we will create a MATLAB function that will integrate another function. In order to do so, we will learn how to create and use functions of functions.

1.1. Create a MATLAB function with the interface

```
function I = imidpoint(func,a,b,n)
```

and save it as `imidpoint.m`. In this function, `func` is a function specified by the user when they call `imidpoint`, `a` and `b` are the integration limits and `n` is the number of subintervals to be used in calculating the integral. The output `I` is the approximate value of the integral of `func` from `a` to `b`.

1.2. Add a command that calculates the width of the subintervals `h`. You can use the formula given below (1).

1.3. Add commands that will generate a row vector `xj` of `n` subinterval midpoints. You can use the formula given below (1).

1.4. Add a command that evaluates the given function `func` at each of the points in `xj`, that is,

```
fj = func(xj);
```

Of course, we are assuming that `func` accepts a single vector argument.

1.5. Finally, add a command that calculates the midpoint integration rule approximation of the integral and stores the result in `I`. Don't use a `for` loop!

1.6. Test your function by calculating the midpoint integration rule approximation of

$$\int_0^{\pi} \sin x \, \mathrm{d}x \tag{2}$$

using $n = 20$ subintervals. To do this, type

```
imidpoint(@sin,0,pi,20)
```

in the command window. Notice the `@` symbol. This symbol creates a 'handle' or 'pointer' to the function `sin`, which is passed to `imidpoint` through the argument `func`. When `imidpoint` calls `func`, it will call whatever function is associated with that handle.

Does your function give the correct answer? What can you do to obtain a more accurate answer?

1.7. You can pass functions defined in `m`-files as well. For example, create the function

```
function u = sin2cos3(x)
u = sin(x).^2.*cos(x).^3;
end
```

and save it as `sin2cos3.m`. Notice the use of element-by-element operators, which allows this function to accept vector arguments. Calculate the integral

$$\int_0^{\pi/2} \sin^2 x \cos^3 x \, \mathrm{d}x \tag{3}$$

by typing

```
imidpoint(@sin2cos3,0,0.5*pi,20)
```

at the command line.

1.8. Simple one-line functions can also be defined in scripts or on the command line using the syntax

```
fhandle = @(arguments) expression
```

For example, in the command window type

```
expsin = @(x) exp(sin(x))
```

This creates a function handle to the function $\exp(\sin(x))$. Functions defined in this way are known as anonymous functions. You can evaluate the function in the normal way by typing `expsin(1)`, for example. Calculate the integral

$$\int_0^{2} e^{\sin x} \, \mathrm{d}x \tag{4}$$

by typing

```
imidpoint(expsin,0,2,20)
```

Note that the @ symbol is not needed here because expsin is already a function handle.

1.9. Sometimes, you might want to integrate a function that has more than one argument. For example, create the function

```
function u = monomial(x,n)
u = x.^n;
end
```

and save it as monomial.m. Suppose we wish to use this function to calculate the integral

$$\int_0^1 x^5 \, dx. \tag{5}$$

We cannot pass the function monomial to imidpoint because it has two arguments and imidpoint assumes there is only one. In this case, we can create an anonymous function in the call to imidpoint in which we specify the parameter n=5. To do this, type

```
imidpoint(@(x) monomial(x,5),0,1,20)
```

2 Differentiation

Differentating the quadratic polynomial that passes through (x_{j-1}, f_{j-1}), (x_j, f_j) and (x_{j+1}, f_{j+1}), or using Taylor's theorem, yields the approximate numerical differentiation formulae

$$f_1' = \frac{-f_3 + 4f_2 - 3f_1}{2h}, \tag{6a}$$

$$f_j' = \frac{f_{j+1} - f_{j-1}}{2h}, \quad j = 2, \ldots, n-1 \tag{6b}$$

$$f_n' = \frac{3f_n - 4f_{n-1} + f_{n-2}}{2h}, \tag{6c}$$

where $f_j = f(x_j)$, $f_j' = f'(x_j)$ for $j = 1, \ldots, n$ and x_j are equispaced grid points, with spacing $h = x_{j+1} - x_j$.

In this exercise, we will use these finite-difference formulae to estimate the derivative of a specified function.

2.1. Write a script that creates a vector x of n equispaced points from 0 to 2π, then creates a vector f containing the corresponding values of $f(x) = \sin x$. Start with a small value for n. Add a command to calculate the grid spacing h.

2.2. Add vectorised code to calculate approximate derivatives of f at each point in x. For each of the interior points calculate the derivative using (6b). At the first point, use (6a). At the last point, use (6c).

2.3. Add code that plots the numerical derivative and the exact derivative $f'(x) = \cos x$ on a single plot. Check that the accuracy improves as you increase n.

2.4. Using code from your script, create a MATLAB function called `ddx` that uses (6) to numerically differentiate a vector of discrete data `f`. The function must have the interface

```
function dfdx = ddx(f, h)
```

where `f` is a row or column vector of data values $f_j = f(x_j)$, `dfdx` is a vector of the same size as `f` containing the derivatives f'_j, and `h` is the grid spacing. Save your function as `ddx.m`.

2.5. Modify your script to use your new function.

3 Submission

When you are happy with your functions `imidpoint` and `ddx`, you should submit it to Mathworks Cody Coursework. We will also be using Cody Coursework for assignment submission.

3.1. Login into Mathworks Cody Coursework using the email address and password that you registered with.

3.2. Once you have logged in, click on "Practical 2" and then on the problem you want to solve. To submit your code click on "Solve" and copy and paste your function into the text box provided.

3.3. Test your function by clicking on "Test". This will run a number of tests on the output of your function and give you feedback on whether it has passed those tests.

The feedback will take the form of a sequence of red or green squares, each of which indicate that the function failed or passed a particular test, respectively. In addition, you will be able to view the output of the test, as well as any messages generated by the tests.

3.4. When you are happy, click "Submit" to save your function. Note that you can submit as often as you like by clicking on "Improve your solution". Cody Coursework saves all of your submissions.

3.5. You will notice a graph which records the results of the tests for all of your submissions. Submissions that pass all the tests appear as green circles, whereas submissions that fail any of the tests appears as red crosses. The best solution is marked by a blue diamond.

The "Size" gives you an idea of how concise your code is (not including documentation). Broadly speaking, we are aiming for concise, well-vectorised code. But your first priority is to write code that works!

3.6. For more information, please consult the Mathworks Cody Coursework documentation.

3.7. You do not have to submit your scripts.