Week 10: Integration and local variables

Intended learning outcomes

By the end of this class, you will be able to:

- calculate integrals numerically using different approximations;
- use local variables in functions;
- define recursive functions.

1 A first look at numerical integration

The calculation of integrals is a task that crops up in many computer applications of mathematics. However, not all functions can be integrated in closed form. For example, you can't express the integral $\int_a^b e^{x^2}\,\mathrm{d}x$ in closed form in terms of elementary functions. This means that numerical techniques are used to obtain approximate answers. As you have seen in MAS110, you can approximate a definite integral (thought of as the area under a graph) by using a number of rectangles. We will do this here using the 'midpoint-rectangle approximation'.

The idea is that to approximate $\int_a^b f(x)\,\mathrm{d}x$ you split the interval from a to b up into N equal subintervals. You use the midpoint of each subinterval to determine the height of a rectangle, and add up the areas of all the rectangles to obtain an approximation to the area under the graph of $f(x)$. This is illustrated in Figure 1.

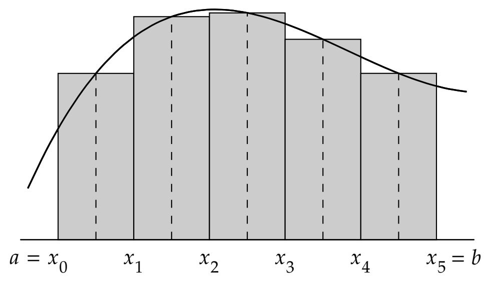

Figure 1: Midpoint-rectangle approximation to $\int_a^b f(x)\,\mathrm{d}x$, using $N = 5$.

Exercise 10.1. Find an approximation to the integral $\int_0^1 x^2\,\mathrm{d}x$ using the midpoint-rectangle approximation with two rectangles ($N = 2$). Compare this with the answer obtained by calculus.

As the N rectangles have the same width as each other, their width is $\frac{b-a}{N}$. If we label the intervals and rectangles in Python fashion, from 0 to $N-1$, then the left-hand end-point of the interval i is $x_i = a + i\frac{b-a}{N}$ and its midpoint is $a + (i+1/2)\frac{b-a}{N}$. The height of rectangle i is the value of the function at its midpoint, $f\left(a + (i+1/2)\frac{b-a}{N}\right)$. Thus the area of rectangle i is $\frac{b-a}{N} f\left(a + (i+1/2)\frac{b-a}{N}\right)$, and the total area of the rectangles is given by

$$\sum_{i=0}^{N-1} \frac{b-a}{N} f\left(a + (i+1/2)\frac{b-a}{N}\right).$$

This is straightforward to implement in Python, as shown by the program below. The main part of this program is the function `midpoint_approximation()`, which evaluates the above sum given a function `f`, integer `N`, and floats `a` and `b`.

```
1  import math
2
3
4  def midpoint_approximation(f, N, a, b):
5      """
6      Calculate an approximation to int_a^b f(x)
7      with N midpoint-rectangles
8      """
9      approximation = 0
10     for i in range(N):
11         approximation += (b - a)/N * f(a + (i + 1/2)*(b - a)/N)
12     return approximation
13
14
15 def f(x):
16     """Calculate x squared."""
17     return x**2
18
19
20 def g(x):
21     """Calculate e**(x**2)."""
22     return math.exp(x**2)
23
24
25 print("\int_0^1 x**2 dx is roughly {0:.6f}".format(
26       midpoint_approximation(f, 2, 0, 1)))
27 print("\int_0^1 e**(x**2) dx is roughly {0:.6f}".format(
28       midpoint_approximation(g, 1000, 0, 1)))
```

Line 11 includes some notation we have not used before. The syntax `x += 1` is a short way to tell Python `x = x + 1`. You can think of it as "to `x` add one". Similarly you can write `x *= 3` meaning "multiply `x` by three", in place of writing `x = x * 3`.

Run the above program and make sure that the $N = 2$ approximation it gives for $\int_0^1 x^2\,\mathrm{d}x$ is the same as that which you calculated by hand. Change the `2` on line 26 to `10` then `100` then `1000`, and compare the resulting approximations to the actual integral.

Exercise 10.2. The trapezoidal approximation to an integral is done by using trapeziums rather than rectangles. (See the wikipedia article on the Trapezoidal Rule.) Using N trapeziums, the formula for the approximation to $\int_a^b f(x)\mathrm{d}x$ is

$$\sum_{i=0}^{N-1} \frac{b-a}{2N}\left(f\left(a+i\frac{b-a}{N}\right)+f\left(a+(i+1)\frac{b-a}{N}\right)\right).$$

Add a function `trapezoidal_approximation()` to the above program which calculates the trapezoidal approximation, similar to the midpoint approximation function. Use it to compare the midpoint and trapezoidal approximations to $\int_0^1 x^2\,\mathrm{d}x$ and $\int_0^1 e^{x^2}\,\mathrm{d}x$.

2 Local variables

An important, but confusing, concept is the notion of **local variables**, which are defined only inside Python functions. Look at the program below and write down what you think it will print when run. Now type it in and run it.

```
def test_function():
    i = 7
    print("Inside the function: i =", i)


i = 2
print("Before the function is called: i =", i)
test_function()
print("After the function is called: i =", i)
```

Clearly something odd is going on! The variable seems to have switched back to its old value with without being told to do so.

Many sheets back, functions were described as mini-programs. Basically, each time they are called, they have their own variables. So the variable `i` in one function will be 'local' to that function and what you do to it will not affect variables called `i` in other functions. This has many advantages, not least of which is that you don't have to think about using different variable names in all of your functions, and when you import modules you don't have to worry that some of the functions might have the same variable names as those you've used in your program.

The way that you communicate variables between your main program and functions (or functions and other functions!) is by using parameters for the function and by using **`return`** statements.

Now think about the following program. What does it do? Write down what you think the output will be.

```
def test_function(i):
    i = i + 2
    print("Inside the function: i =", i)


i = 2
print("Before the function is called: i =", i)
test_function(i)
print("After the function is called: i =", i)
```

Now run the program. Did it do what you expected? Let's go through what Python does.

It sets the variable `i` in the main part of the program to be 2. It prints the value of `i`. Then it sends the value of `i`, i.e. 2, to the function `test_function(i)`. The function sets its local variable `i` to be this number. It adds 2 to the local variable `i`, prints it, then exits the function. The main program then prints its variable `i`, which has been unchanged, and so is still 2.

In operation, the above program is identical to the following one:

```
def test_function(x):
    x = x + 2
    print("Inside the function: x =", x)


i = 2
print("Before the function is called: i =", i)
test_function(i)
print("After the function is called: i =", i)
```

We have just renamed what `test_function()` calls its variable, from `i` to `x`, and this makes absolutely no difference to the running of the program.

If we had wanted to get the function to increase the main program's variable `i` by 2, we would have to explicitly have done it using a **`return`** command:

```
def test_function(x):
    x = x + 2
    print("Inside the function: x =", x)
    return x


i = 2
print("Before the function is called: i =", i)
i = test_function(i)
print("After the function is called: i =", i)
```

Here we could rename the x in the function back to i without changing the way the program runs, because it is local to the function and nothing outside the function can actually see what it is called. Anything outside the function will just know that the function accepts one value as a parameter and returns one value.

Note that this is a first introduction to local variables and things are actually slightly more complicated! But this is enough for now.

Exercise 10.3. The notion of local variable is what makes the following function work. It is an example of a **recursive function**. Figure out what the function is doing! This usually makes people scratch their head the first time they come across it. Which familiar mathematical function is this Python function? Why does it work?

```
def function(n):
    """Calculate some mysterious function recursively."""
    if n == 0:
        return 1
    return n*function(n-1)


print(function(5))
```

Homework

1. Finish off the sheet.

2. (*Quick review.*) What is the output of the following? Type it in and check.

```
def square(x):
    x = x**2
    print("x =", x)
    return(x)

x = 3
print("x =", x)
a = square(x)
print("x =", x, "; a =", a)
```

3. (*Assessed homework.*) This week's homework involves writing a program implementing **Simpson's rule**, which is another method of approximating an integral. To access the homework, you must navigate to the following webpage:

   `https://aim.shef.ac.uk/moodle/mod/quiz/view.php?id=432`

   Your code will be run with some test functions. If it gives the correct answer you will get a mark, if it does not then you will get no mark. You can submit your homework anytime from now til 2pm on Monday of Week 11 (6th December).