Euler's method

In previous weeks we have covered some of the basic steps involved in creating a Python program. This time we'll try to bring it all together by coding up a specific mathematical application.

Euler's method

Rather than calculate the exact (analytical) solution to an equation, Euler's method is part of a collection of numerical methods which are used to find approximations.

Euler's method uses the gradient of the tangent at a particular point to approximate the value at a nearby point. Then it uses that approximation to step a little further and approximate the next point, and so on in a process of iteration.

This does not give the exact solution, but it can give a good approximation and is especially useful for equations that we cannot solve exactly.

We are going to solve a first-order Ordinary Differential Equation using Euler's method. Here is the basic idea.

A first-order Ordinary Differential Equation is one involving an independent variable, a dependent variable and the first derivative of that dependent variable.

Here we are trying to solve a first-order ODE of the form:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x,y).$$

The gradient of a straight line is given by the equation

$$\text{gradient} = \frac{\text{change in } y}{\text{change in } x}.$$

The gradient of the tangent to a curve $y = f(x)$ at a point with coordinates (x, y) can be approximated by moving a small distance away from that point to $(x + \delta x, y + \delta y)$ and calculating the gradient of the line between those points. δx just means a small change in x, and δy similarly means a small change in y.

The gradient of the line between (x, y) and $(x + \delta x, y + \delta y)$ is therefore

$$\text{gradient} = \frac{\text{change in } y}{\text{change in } x} = \frac{(y + \delta y) - y}{(x + \delta x) - x}.$$

We would normally define the derivative to be the limit:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \lim_{\delta x \to 0} \frac{(y + \delta y) - y}{(x + \delta x) - x}.$$

In cases where a derivative cannot be found analytically, we have a number of techniques for solving ODEs, the simplest of which is known as Euler's method.

In Euler's method, the derivative is approximated using a first-order difference equation:

$$\frac{\mathrm{d}y}{\mathrm{d}x} \approx \frac{y_{n+1} - y_n}{x_{n+1} - x_n}.$$

In terms of the gradient equation above, this is like moving along in steps of length δx and considering $y_{n+1} = y_n + \delta y$ and $x_{n+1} = x_n + \delta x$.

For equally-spaced values of x, say $x_{n+1} - x_n = h$, this becomes

$$\frac{\mathrm{d}y}{\mathrm{d}x} \approx \frac{y_{n+1} - y_n}{h}.$$

Remembering the definition $\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y)$ above, we can rearrange this to give

$$y_{n+1} \approx y_n + hf(x_n, y_n).$$

Essentially, we are stepping a small distance h along the tangent and approximating the value of the function at that point.

Suppose we know one value of the function, say $(x, y) = (x_0, y_0)$. We know the value of the gradient of the tangent at any particular (x, y), because this is given by the ODE itself $\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y)$. We can use this to predict values of (x, y) near to (x_0, y_0).

So for example we would say $x_1 = x_0 + h$ and

$$y_1 \approx y_0 + hf(x_0, y_0).$$

The next value will be $y_2 \approx y_1 + hf(x_1, y_1)$, and so on. This is a process of iteration.

This process produces approximate solutions and depends on the extent to which the slope at the starting point represents the average slope over the interval between that point and the next.

Exercise

We wish to write a program to solve

$$\frac{\mathrm{d}y}{\mathrm{d}x} = 2x^2y$$

using the above method.

(Note: this is a simple ODE we can solve exactly, we are only using it to illustrate how Euler's method works.)

We can write a function to return values of $f(x, y)$:

```
def f(x,y):
    return 2*x**2*y
```

Your program should store the initial parameters:

- The value of x_0;
- The value of y_0;
- The size of each step, h, and the number of steps.

For example, you could start at $(x_0, y_0) = (0, 1)$ and proceed in 20 steps of length $h = 0.05$ to estimate the value of y at $x = 1$.

Having set up the initial parameters, your program now needs a loop to do the iteration. Within this loop:

- calculate the new value of $x_{n+1} = x_n + h$;
- calculate the new value of $y_{n+1} = y_n + hf(x_n, y_n)$ using your function defined above;
- print the values of x_{n+1} and y_{n+1};
- save the value of x_{n+1} and y_{n+1} as x_n and y_n ready for the next iteration of the loop.

To help you get started, here is an iteration loop which generates numbers $t_{n+1} = 2t_n + 1$ with $t_0 = 1$. Hopefully this will give you an idea of how to write a loop to generate numbers iteratively.

```
# store initial parameters
num_steps = 15
t0 = 1

tn = t0 # first value of t n
for i in range(0,num_steps):
    tnext = 2 * tn + 1 # calculate t n+1
    print(tnext)
    tn = tnext # store t n+1 as t n ready for next iteration
```

Some extensions

Here are some examples of things you could try to extend your code.

Plotting your approximation

- Rather than printing the values, store the x values in a list and the y values in another list.
- Use the lists to plot the data.

  Here is an example program which uses `matplotlib` to plot some data. You might adapt this to plot your solution from Euler's Method.

  ```
  from matplotlib import pyplot as plt # import commands
  ```

```
# define some data
t = [0,1,2,3,4,5]
y = [10,56,34,109,84,318]

# command to plot t against y
plt.plot(t,y,label="Plot of t against y")

# add a legend then display the plot
plt.legend()
plt.show()
```

Plotting the exact solution

In this case, we can solve the ODE exactly given $y(0) = 1$. The solution is

$$y = e^{\frac{2}{5}x^3}.$$

We can use this to see how far the approximation is from the exact value.

- Write a function `y(x)` that returns values of y using the exact solution given above.
- Use your function to generate exact y values for the x points (the x points should already be stored in a list in your program, so you could loop through the x coordinates you have already stored).
- Plot these on the same chart as your approximation.

Error

The percentage error is given by

$$\epsilon = 100\left(\frac{y_{\text{estimate}} - y_{\text{true}}}{y_{\text{true}}}\right),$$

where y_{estimate} is the numerical estimate and y_{true} is the analytical solution.

- Use the exact solution to calculate and display the percentage error at $x = 1$.