Notebook tips

September 13, 2020

1 IPython magic commands

When using Python with Jupyter, we are actually using the *IPython kernel* to run code. IPython has what are known as *magic commands* that help us interact with Jupyter. Magic commands are prefixed by '%'. Magic commands are *not* part of the Python language; they are specific to IPython. In a plain Python program, magic commands would not be recognised and would lead to an error.

We summarise below the magic functions that are used in the Activity notebooks. The full documentation for IPython magic commands is available here.

1.1 Matplotlib

To display plots inline in a notebook, we use the magic command:

[1]:
```
%matplotlib inline
```

The full documentation for Matplotlib magic functions is here.

1.2 Timing programs

We can use magic commands to time our programs. This is particularly useful when investigating the performance of different implementations.

1.2.1 Simple timing

The magic command `%time` is used to time parts of a program. We just add

`%time`

in front of the function call we wish to time, and the time taken will be displayed. Below is an example:

[2]:
```
def f(x):
    s = ""
    for i in range(x):
        s += " "
    return s

%time p = f(100000)
```

```
CPU times: user 7.99 ms, sys: 0 ns, total: 7.99 ms
Wall time: 7.94 ms
```

Usually we are interested in the '`Wall time`', which is the real (wall clock) time elapsed to run the function.

1.2.2 Detailed timing

Sometimes we want to get the time as a variable, for example to produce a plot of time versus problem size. In this case we use `%timeit`.

`%timeit` has a number of options, including:

- `-o`: Return a `TimeitResult` variable that we can query
- `-q`: Quiet (supress output)
- `-n`: Number of times to run code
- `-r`: How many times to run `timeit`

The return value can be queried in several ways. Below are examples.

[3]:
```
# Problem size to test
N = 1000000

# Time the command 'p = f(N)' once, suppressing output (-q).
t = %timeit -o -n1 -r1 -q p = f(N)

# Get best (only) timing
print("Best time: {}".format(t.best))

# Time the command 'p = f(N)' twice, calling three times each (not quiet)
t = %timeit -o -n3 -r2 p = f(N)

# Get results of all runs as a list (length will be 2 since we used -r2)
print("Time for all runs: {}".format(t.all_runs))

# Best time will be 1/3 (since we used of lowest value in t.all_runs
print("Time for best runs: {}".format(t.best))
```

```
Best time: 0.05672190198674798
58.4 ms ± 869 µs per loop (mean ± std. dev. of 2 runs, 3 loops each)
Time for all runs: [0.17265189500176348, 0.17786804199567996]
Time for best runs: 0.05755063166725449
```