Class 1 (Monday 10 October)

Contents

These tasks are designed to be worked on in the practical class on Monday 10 October.

Getting to know Numpy

Practice using Numpy by doing the following tasks:

```
import numpy as np
```

- Create the vectors $\mathbf{a} = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}$ and $\mathbf{b} = \begin{pmatrix} -3 \\ \frac{3}{2} \\ 1 \end{pmatrix}$.

```
a = np.array([1, 2, 0])
b = np.array([-3, 3/2, 1])
```

- Create the matrix $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix}$.

```
A = np.array([[1, 0, 0], [0, 2, 0], [1, 0, 1]])
```

- Compute the matrix-vector product Aa.

```
print(A @ a)
```

- Compute the dot product $\mathbf{a} \cdot \mathbf{b}$.

```
print(np.dot(a, b))
print(a.dot(b))
```

- Find a vector that is perpendicular to both $\mathbf{a}$ and $\mathbf{b}$.

```
print(np.cross(a, b))
```

- Find the vector $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{a}$

```
print(np.linalg.solve(A, a))

# This is slower for larger matrices but will give the same result:
print(np.linalg.inv(A) @ a)
```

The following snippet of code defines a function that computes a matrix-vector product. The last two lines should print the same result: this can be used to check that the function is behaving as expected.

```python
import numpy as np

def slow_matvec(matrix, vector):
    assert matrix.shape[1] == vector.shape[0]
    result = []
    for r in range(matrix.shape[0]):
        value = 0
        for c in range(matrix.shape[1]):
            value += matrix[r, c] * vector[c]
        result.append(value)
    return np.array(result)


# Example of using this function
matrix = np.random.rand(3, 3)
vector = np.random.rand(3)
print(slow_matvec(matrix, vector))
print(matrix @ vector)
```

Using this code as a template, write your own function called `faster_matvec` that computes matrix-vector products by taking the dot product of the matrix with each row. Check that your function also gives the same result as these two functions.

Testing with asserts

When writing a code that you want to run on a large HPC system, it is important to test that the code is correct before setting it off. There are better ways to do this than manually comparing outputs like you did above.

Probably the best way to test your code for correctness is to write some `assert` statements to assert that your function gives the correct result for some small problems that you know the answer to. For example, the following code tests a function that's been written to add two integers.

```python
def add(a, b):
    return a + b


assert add(1, 1) == 2
assert add(4, 5) == 9
```

When using floating point numbers, asserts using `==` can fail even when the numbers should be the same (due to differences around the size of machine precision). For example, the following assert fails (or at least, it fails on my computer using Python 3.10.4):

```python
assert 100 / 3 * 30 == 1000.0
```

To avoid this issue, the function `np.isclose` should be used, eg:

```python
import numpy as np
assert np.isclose(100 / 3 * 30, 1000.0)
```

For vectors and matrices, `np.allclose` can be used.

Write some Python code that test your `faster_matvec` function by computing that matrix-vector product of a random matrix and vector and compares the result to the result when using `@`.

(Python libraries commonly use the `pytest` library to carry out automated testing. Use of `pytest` is beyond the scope of this course.)

Timing a function

You're now going to measure the time it takes to compute matrix-vector products with the two functions `slow_matvec` and `fast_matvec`. The following code snippet measures the time taken to run a function `f`. This runs the function 1000 times and prints the total time taken.

```
from timeit import timeit


def f():
    # contents of the function go here


t = timeit(f, number=1000)
print(t)
```

Write some Python code that measures the time taken to compute a matrix-vector product of an $n \times n$ matrix and a vector with n entries for $n = 2, 10, 100$ for both `slow_matvec` and `fast_matvec`. How much is your function faster than `slow_matvec`?

Plotting with matplotlib

The Python library matplotlib is commonly used to draw plots of data. Matplotlib gives you a lot of freedom to do whatever you want, but this means that it has a lot of options/functions you can use.

(When making plots with matplotlib, you may want to bear in mind that around 4% of people have some form of colourblindness. It can be very helpful to use different line styles and/or markers as well as colour differences.)

The following example code makes a plot with the curves $y = x$, $y = x^2$ and $y = 3x^2$ between $x = 1$ and $x = 3$.

```
import matplotlib.pylab as plt
import numpy as np

x = np.linspace(1, 3, 20)
y0 = x
y1 = x ** 2
y2 = 3 * x ** 2

plt.plot(x, y0, "ro-")
plt.plot(x, y1, "g^-")
plt.plot(x, y2, "b-")

plt.xlabel("Values of x")
plt.ylabel("Values of y")
plt.legend(["$y=x$", "$y=x^2$", "$y=3x^2$"])
```

(If you're using a Jupyter notebook, you may need to put the magic command `%matplotlib inline` at the start of the cell to display the plot. If you're running Python from the command line, you'll need to put `plt.show()` at the end to show the plot.)

Make an alternative version of this plot with both axes using a log scale (hint: `plt.xscale("log")`). (Personally, I think log-log plots are much clearer if the x and y axes use equal tick sizes: you can do this with `plt.axis("equal")`.) What do you notice about your curves on this log-log plot? (Pen and paper task: starting with $y = ax^b$, work out why what you've observed happens.)

Using matplotlib, make a plot showing the timings you measures in the previous part. Add some more data to your plot to make it more informative.

Saving data to a file

If your problem takes a long time to solve, you don't want to have to resolve it to get the data when you want to tweak a matplotlib plot. It's therefore a good idea to save your data to a file and make a pretty plot with the data separately.

In Jupyter notebooks, this can be acheived by running cells selectively: once you're run the cell with your solver in once, you can tweak your plotting cell and re-run it as many times as you like without having to re-solve your problem.

When running through a command line interface, you can use the functions `numpy.save` and `numpy.load` to save and load Numpy objects. For example, the following snippet saves a random matrix to a `.npy` file.

```
import numpy as np

data = np.random.rand(10, 10)

numpy.save("my_results.npy", data)
```

This matrix can then be loaded in a different file:

```
import numpy as np

data = numpy.load("my_results.npy")
```

By Timo Betcke & Matthew Scroggs