CMPT 423/820 ¶

Assignment 1 Question 3

- Seyedeh Mina Mousavifar
- 11279515
- sem311

In this question, we'll introduce Numpy, a very powerful and popular Python module for numerical computation.

Numpy adds a very fast, very flexible data structure called NDArray to our tool-box. A NDArray stores numerical data much more efficiently than Python lists, and its methods are optimized for array-level tasks. The powerful thing about Numpy is that you can express array-level calculations with simple algebraic expressions, and usually without loops. The Numpy module also provides very useful libraries for array-based and matric-based computations.

In [1]:
```
# almost always imported like this
import numpy as np
```

Task 1

The Numpy module includes libraries for numerical calculations. There are array-based calculations, and matrix-based calculations. For example, the cell below defines a small 2×2 array, and then does two kinds of multiplication.

In [2]:
```
# a small array
twobytwo = np.array([[1, 2], [3, 4]])

# array-based
array_product = 2 * twobytwo * twobytwo

# matrix-based
matrix_product = 2 * np.matmul(twobytwo, twobytwo)

# print them both
print(array_product)
print(matrix_product)
```

```
[[ 2  8]
 [18 32]]
[[14 20]
 [30 44]]
```

Task 1 Questions

1. (Easy) Explain the difference between the two products.

> The array_product is the element-wise multiplication of twobytwo matrix, multiplied by two. However, the matrix_product is the matrix multiplication, multiplied by two.

2. (Deeper) Python uses the multiplication operator `*` for normal Python numbers, but here we see it being used for NDArrays. How does Python know which kind of multiplication to do?

> Python uses duck typing, which means that instead of determining functions based on the object's type, it examines the presence of specific methods and properties rather than the type of the object itself. In case of absence, it will raise an error during runtime. Consequently, in this case, it calls * operator on NDArrays object, which does the elementwise multiplication.

Task 2. Scripts using NDArrays

Here's a cure little result. It turns out you can calculate Fibonacci numbers using matrix multiplication. Suppose we want the Nth Fibonacci Number, F_N:

$$\begin{pmatrix} F_{N+1} \\ F_N \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^N \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Task 2 Questions.

1. Write a few lines of Python, using NDArrays, Numpy, and matrix multiplication to calculate F_{30} (which we saw in Question 2). There's a cell below for this script. It's only a few lines of code!
2. In Question 2, we could calculate F_N for very large N, and it would be exact. Explain what happens if we try to do the same thing here. Answer by discussing what your script produces for large N, and then explain why it's wrong.

> The difference between results is due to overflow in numpy-arrays because the operations in the pydata stack (NumPy/pandas) have C-style fixed-precision integers. In this situation, as we can see, the array is stored in the int64 format, which its max value is 9223372036854775807. So, the overflow occurs, and we couldn't obtain the previous result.

In [3]:
```
def fibonacci(number):
    """
    :purpose: This function calculates nth Fibonacci number, which is Fn
= Fn-1 + Fn-2 based on array multiplication
    :param number: nth Fibonacci number to be calculated
    :return: value of nth number in Fibonacci sequence
    """
    nth_number = None
    if number == 0:
        return 0
    elif number == 1:
        return 1
    else:
        first_matrix = np.array([[1, 1], [1, 0]])
        second_matrix = np.array([[1], [0]])

        # for saving nth power of first_matrix
        power_matrix = np.array([[1, 1], [1, 0]])

        # calculating first_matrix nth power
        for j in range(number-1):

            power_matrix = np.matmul(power_matrix, first_matrix)

        # multiplicating given formula
        nth_number = np.matmul(power_matrix, second_matrix)

        return nth_number

N = 30
f_N = fibonacci(N)

print('The', N, 'th Fibonacci number is', str(f_N[1]).lstrip('[').rstrip
(']'))
```

```
The 30 th Fibonacci number is 832040
```

In [4]:
```
N = 111
f_N = fibonacci(N)

print('The', N, 'th Fibonacci number is', str(f_N[1]).lstrip('[').rstrip
(']'))

print('The type of NDarray is:', f_N.dtype)

f_N = int(str(f_N[1]).lstrip('[').rstrip(']'))

pre_f_N = 7049252476708912581 4114

print('The difference between matrix calculation and loop calculation i
s:', pre_f_N - f_N)
```

```
The 111 th Fibonacci number is 7515661444929089378
The type of NDarray is: int64
The difference between matrix calculation and loop calculation is: 7048
5009105644196724736
```

What to hand in

Your version of this notebook named A1Q3.pdf, containing completed work above, and your name and student number at the top.

Evaluation:

- 1 mark: Your answer two Task 1 Part 1 was correct.
- 2 marks: Your answer two Task 1 Part 2 was correct.
- 4 marks: Your code cell for Task 2 Part uses Python and Numpy to calculate the 30th Fibonacci number correctly.
- 3 marks: Your explanation for the behaviour of this script for large N is correct.