

CS319: Scientific Computing**Week 12: Wrap Up**

Dr Niall Madden

2nd and 4th April, 2025

- 1 News and Updates
- 2 Why C++
 - Python and C++
 - jit
- 3 Module review
 - THE END!!!!!!!!!!

Slides and examples:

<https://www.niallmadden.ie/2425-CS319>

1. News and Updates

- ▶ Class test: grades were posted (right?).
- ▶ Lab 6: tomorrow.
- ▶ Lab 8: Friday.
- ▶ Presentations: tomorrow and Friday. See <https://www.niallmadden.ie/2425-CS319/2324-CS319-Projects.pdf>

2. Why C++

In CS319, we have mainly used C++. Although it is not so popular in academic settings it (and its older sibling C) remain important languages in industry. This is mainly for 2 reasons

- ▶ They are **VERY** fast, compared with interpreted languages such as Python and R. This is because the compiler is highly optimised.
- ▶ They are **low-level**: since they can directly access memory addresses, they are used for hardware programming. (E.g., Arduino).

2. Why C++

Of course, C and C++ have disadvantages compared to (say) Python.

- ▶ They have a steeper learning curve: it is harder to get started.
- ▶ No good notebook solution (and I doubt there ever will be).
- ▶ Can have security risks if improperly used.
- ▶ If you use Python **carefully**, then it can be quite fast.
- ▶ Python has many more modules and libraries.

Is there a “best of both worlds”? In a sense, there are several.

- ▶ Python is written in C, and so it is possible to integrate C/C++ libraries with it.
- ▶ For example, when you use `numpy` in Python, you are using an interface to libraries written in C and C++.
- ▶ You can write C++ code that can be run from Python, in one of two ways:
 - Write a Python module in C++ using the `PyBind1` library.
 - Use the `ctypes` module in Python to access a C++ library.
 - Use just-in-time (`jit`) compilation of your Python code.

The first two of these are not too complicated, but are beyond what we can cover in this module. However, using `jit` is easy...

Older languages, such as C and C++, are “**compiled**”: the source code is converted to a machine language before it is run. (Easy to optimised; not really interactive).

Many more modern languages are “**interpreted**”, where the code is executed at run-time, usually line-by-line. (Hard to optimise, can be interactive).

The newest languages, such as Julia, try to combine both ideas by compiling only as needed.

In Python, we can do this with the `jit` decoration, which is part of the `numba` module.

```
1 import numpy as np
   import time
3 from numba import jit
```

Now define a function for multiplying two matrices:

```
1 def MatMat(A,B):
   N = len(A)
3   C = np.zeros((N,N), dtype=A.dtype)
   for i in range(N):
5       for j in range(N):
           for k in range(N):
7               C[i][j] += A[i,k]*B[k,j]
   return C
```

Let's test this:

```
2 N = 400;
  A = np.random.rand(N,N)
  start = time.time()
4 C1 = MatMat(A,A)
  MatMat_time = time.time() - start
6 print(f"MatMat for {N}-by-{N} matrix took {MatMat_time
    :.3f} seconds")
```

The output I get from this is

MatMat for 400-by-400 matrix took 33.387 seconds

However, if you load the `numba` module, we can “decorate” the cell defining the function with `@jit`

This causes Python to call a compiler the first time this function is used.

So, for example, when I try

```
@jit(parallel=True, fastmath=True)
2 def MatMatjit(A,B):
    N = len(A)
4    C = np.zeros((N,N), dtype=A.dtype)
    for i in range(N):
6        for j in range(N):
            for k in range(N):
8                C[i][j] += A[i,k]*B[k,j]
    return C
```

And when I run the above code (or similar) first time, I get

```
MatMatjit for 400-by-400 matrix took 0.792 seconds
```

Already this is much faster. But some of that time including compilation. So any subsequent time I run it, I get something like:

```
MatMatjit for 400-by-400 matrix took 0.075 seconds
```

3. Assessment for CS319

The assessment for CS319 is based on

1. 40% based on lab assignments: 10% for each of Labs 2, 4, 6 and 8.
2. 20% from Class Test.
3. 40% for Project
 - (a) Initial Project Plan/Discussion [**5 Marks**]
 - (b) Project Proposal [**5 Marks**]
 - (c) Presentation [**5 Marks**]
 - (d) Project code [**15 Marks**]
 - (e) Project report [**10 Marks**]

4. Module review

The topics we have covered (not necessarily in order) are:

- (a) Basic I/O (`cin`, `cout`), and manipulators (`endl`, `setw`, ...);
- (b) Flow of control and looping (`if`, `for`, `while`, etc.)
- (c) Fundamental data-types (`int`, `float`, `double`, `char`, `bool`, ...). Arrays.
- (d) `string`, and C-style strings.
- (e) Computer representation of numbers (underflow, overflow, machine epsilon, ...)

4. Module review

- (f) Dynamic memory allocation, including of multidimensional arrays
- (g) Functions: default parameter values, overloading, functions as arguments to other functions, recursion, pass by reference/value.
- (h) Classes, including the `private`, `public` and `friend` access specifiers.
- (i) Classes: constructors and destructors, including copy constructors.
- (j) `templates`.

4. Module review

- (k) Function and operator overloading (syntax, precedence, implicit/explicit arguments, the `this` pointer, unary/binary operators, assignment operators, ...).
- (l) The C++ preprocessor
- (m) The Standard Template Library (STL), including containers (especially `set`, `multiset` and `vector`), iterators, algorithms (but not functors); range-based loops.
- (n) Optimisation and bisection;
- (o) Sorting;
- (p) Stacks;

4. Module review

- (q) Numerical Integration;
- (r) Solving linear systems by the Jacobi and Gauss-Seidel methods;
- (s) Solving diagonal and triangular systems (back-substitution);
- (t) Sparse matrix representation, especially triplet and CCS; Matrix-vector multiplication
- (u) Reading to and writing from files;

I hope you have enjoyed CS319, and have learned something (I did!).

