Introduction to HPC with Python: History, Trends and Future Directions

John Stachurski

October 2023

Outline

- Trends in scientific computing
- Likely future directions
- Hands-on computing with Python + NumPy + Numba + JAX

A (very) short history of scientific computing

General purpose scientific computing environments:

1. Fortran & C / C++
2. MATLAB & (Python + NumPy)
3. Julia & (Python + Numba)
4. Python + Google JAX

Fortran & C — static types and AOT compilers

```c
#include <stdio.h>
int main() {
    int x = 1 + 1;
    printf("1 + 1 = %d\n", x);
    return 0;
}
```

```fortran
PROGRAM ONE_PLUS_ONE
INTEGER :: X = 1 + 1
PRINT *, '1 + 1 = ', X
END PROGRAM ONE_PLUS_ONE
```

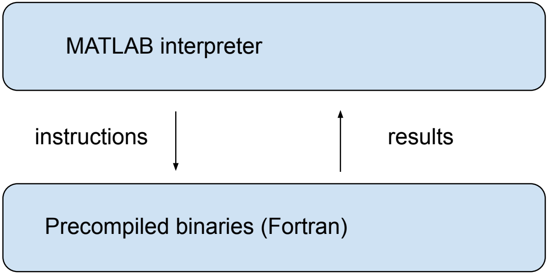
Pros

- fast — on a single thread

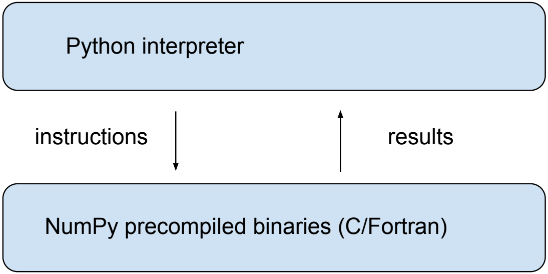
Cons

- tedious to write
- lack of portability
- hard to debug
- hard to parallelize
- low interactivity

Phase 2: MATLAB



Phase 2A: Python + NumPy



Phase 3: Julia — rise of the JIT compilers

```julia
function quad(x0, α, n)
    x = x0
    for i in 1:(n-1)
        x = α * x * (1 - x)
    end
    return x
end

quad(0.2, 4.0, 10_000_000)
```

Phase 3 continued: Python + Numba copy Julia

```
from numba import jit

@jit
def quad(x0, α, n):
    x = x0
    for i in range(n-1):
        x = α * x * (1 - x)
    return x

quad(0.2, 4.0, 10_000_000)
```

Phase 4: AI-driven scientific computing

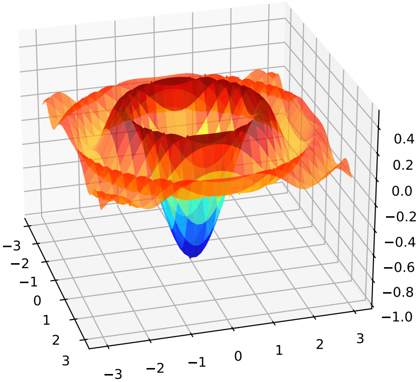
Core elements

- JIT-compilers
- automatic differentiation
- parallelization (CPUs / GPUs / TPUs)

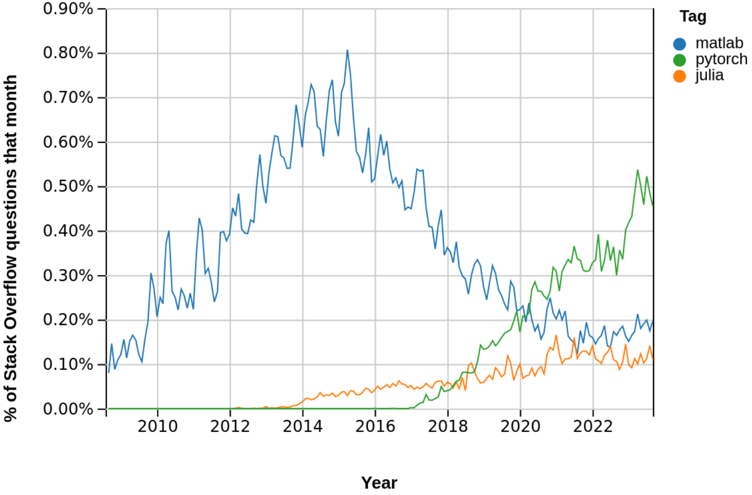
Key players

- TensorFlow, PyTorch
- Google JAX
- Mojo?

AI / machine learning: minimizing differentiable loss functions



Stack Overflow Trends



Sample code

https://github.com/QuantEcon/columbia_2023/notebooks