Effective Programming Practices for Economists

Scientific Computing

Writing fast code with numpy

Janoś Gabler and Hans-Martin von Gaudecker

Why numpy can be fast

- Numbers are stored efficiently in arrays
  - dtype of all numbers is known
  - it is fast to read numbers from memory into registers
- numpy functions are implemented very efficiently
  - in a low-level language like C
  - by experts who really know what they are doing
- Python overhead is incurred once per array, not once per number

Implications for writing fast code

- Vectorize everything!
- Use broadcasting where possible
- Prefer few large arrays over many small arrays

Why the example is slow

```
def array_cobb_douglas(factors, weights, a):
    out = np.empty(len(factors))
    for i in range(len(factors)):
        out[i] = _cobb_douglas(factors[i], weights, a)
    return out

def _cobb_douglas(factors, weights, a):
    return a * np.prod(factors**weights)

# (inputs as before)

%timeit array_cobb_douglas(factors, weights, a)

25.1 ms ± 488 µs per loop
```

- This uses array operations (e.g. np.prod) inside a loop
- This is typically slow and full vectorization should be faster
- Even writing out everything as a loop might be faster than the mix!

Full vectorization

```
def vectorized_array_cobb_douglas(factors, weights, a):
    return a * np.prod(factors**weights, axis=-1)
```

```
%timeit vectorized_array_cobb_douglas(factors, weights, a)
```

```
215 µs ± 1.63 µs per loop
```

- From ~25 milliseconds to ~215 microseconds
- Speedup of more than 110×
- Code is actually more readable
- Need to get good with `axis` argument!

Limits of performance with numpy

- Numpy is not a compiler, so it cannot
  - Fuse multiple operations into one (for more speed)
  - Eliminate intermediate results (for using less memory)
- Creating arrays is slower than creating list
  - Only relevant if you create many tiny arrays
- Calling array operations in a loop is typically slow but it is hard to detect this inefficiency in a profiler