Computational Economics Workshop

The University of Melbourne

John Stachurski

August 2024

Topics

Part 1: Workshop

- Background on scientific computing
- Current and future trends: AI-driven scientific computing
- Applications (hands on, using Python)

Part 2: Computational economics in action

- Joint with James Hansen and Yong Song

Key questions:

- What computational skills should economists learn in 2024?
- What are some interesting applications?

Please feel free to question / debate / share your experiences

Flow

- 1:00 – 2:30 Lecture 1
- 2:30 – 3:00 afternoon tea (staff lounge)
- 3:00 – 4:00 Lecture 2
- 4:00 – 4:15 break
- 4:15 – 5:00 Computational Economics in Action

Slides, code:

https://github.com/QuantEcon/melbourne_2024

Quick poll:

- Python programmers?
    - NumPy? Numba? PyTorch? JAX?
- Julia?
- MATLAB?
- C?
- Fortran?

Regular GPU users?

Old school: static types & AOT compilers

Example. Consider the Solow–Swan growth dynamics

$$k_{t+1} = s k_t^\alpha + (1 - \delta) k_t$$

Task: compute k_n given

1. $n \in \mathbb{N}$
2. initial condition k_0
3. parameter values

Fortran code:

```fortran
program main
 implicit none
 integer, parameter :: dp=kind(0.d0)
 integer :: n=1000
 real(dp) :: s=0.3_dp
 real(dp) :: delta=0.1_dp
 real(dp) :: alpha=0.4_dp
 real(dp) :: k=0.2_dp
 integer :: i
 do i = 1, n - 1
  k = s * k**alpha + (1 - delta) * k
 end do
 print *,'k = ', k
end program main
```

Relative merits of Fortran / C / other static type AOT compiled languages?

Pros

- fast loops / arithmetic

Cons

- low interactivity
- time consuming to write / read / debug
- hard to parallelize

For comparison, the same operation in Python:

```
α = 0.4
s = 0.3
δ = 0.1
n = 1_000
k = 0.2

for i in range(n):
    k = s * k**α + (1 - δ) * k

print(k)
```

Often this will be written using a function

```
def solow(k0, α=0.4, δ=0.1, s=0.3, n=1_000):
    k = k0
    for i in range(n-1):
        k = s * k**α + (1 - δ) * k
    return k

print(solow(0.2))
```

Pros

- high interactivity
- easy to write / read / debug

Cons

- slow loops / arithmetic

Why is pure Python slow?

Problem 1: Type checking

```
x, y = 1, 2
z = x + y

x, y = 1.0, 2.0
z = x + y

x, y = 'foo', 'bar'
z = x + y
```

How does Python know which operation to perform?

Answer: Python checks the type of the objects first

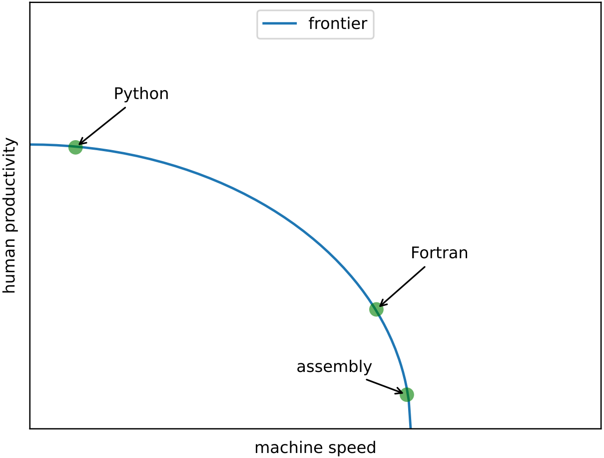
```
>> x = 1
>> type(x)
int
```

```
>> x = 'foo'
>> type(x)
str
```

In a large loop, this type checking generates massive overhead

Problem 2: Memory management

```
>>> import sys
>>> x = [1.0, 2.0]
>>> sys.getsizeof(x) * 8          # number of bits
576                               # whaaaat???
```

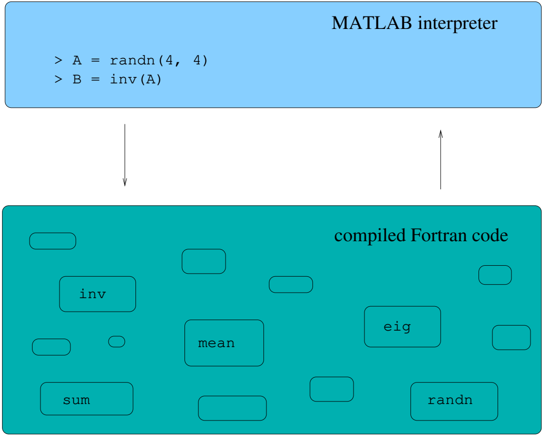

frontier
Python
Fortran
assembly
human productivity
machine speed

Shifting the frontier

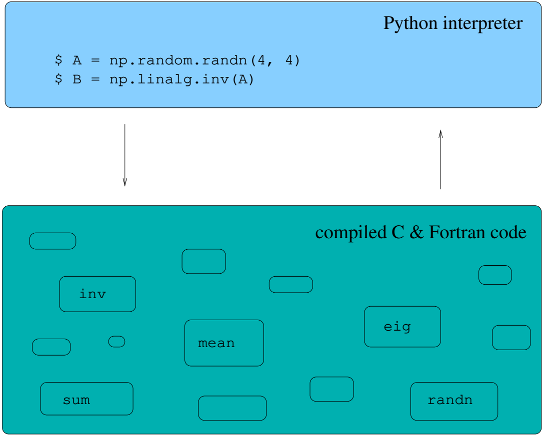
How can we get

good execution speeds **and** high productivity / interactivity?

Gen 1: MATLAB's vectorization trick



Python + NumPy – MATLAB workalike



Vectorization: pros and cons

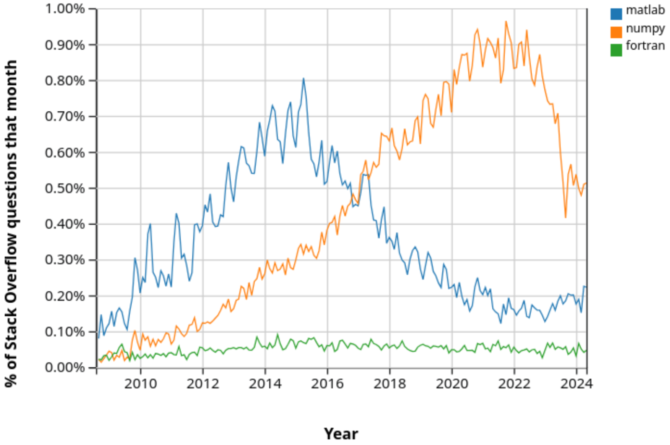
Pros

- high interactivity

Cons

- some tasks cannot be efficiently vectorized
- cannot adapt flexibly to function arguments / hardware

Some trends:



% of Stack Overflow questions that month
1.00%
0.90%
0.80%
0.70%
0.60%
0.50%
0.40%
0.30%
0.20%
0.10%
0.00%
2010
2012
2014
2016
2018
2020
2022
2024
Year
matlab
numpy
fortran

New trend — rise of the JIT compilers

- Code is optimized by the JIT compiler at runtime
- Optimization is between function boundaries
- Optimization specializes on input types

Example: Python + Numba

```
from numba import jit

@jit
def solow(k0, α=0.4, δ=0.1, s=0.3, n=1_000):
    k = k0
    for i in range(n-1):
        k = s * k**α + (1 - δ) * k
    return k

solow(0.2)
```

Runs at same speed as C / Fortran

Why are JIT compilers on the rise?

- JIT + interpreter retains interactivity
- high degree of flexibility
- compiler can adapt on the fly to available hardware

AI-driven scientific computing

AI is changing the world

- image processing / computer vision
- speech recognition, translation
- scientific knowledge discovery
- forecasting and prediction
- generative AI

Plus killer drones, skynet, etc....

AI-driven scientific computing

AI is changing the world

- image processing / computer vision
- speech recognition, translation
- scientific knowledge discovery
- forecasting and prediction
- generative AI

Plus killer drones, skynet, etc....

Projected spending on AI in 2024:

- Google: $48 billion
- Microsoft: $60 billion
- Meta: $40 billion
- etc.

Deep learning in two slides

Supervised deep learning: find a good approximation to an unknown functional relationship

$$y = f(x) \qquad (x \in \mathbb{R}^d,\ y \in \mathbb{R})$$

Examples.

- $x =$ sequence of words, $y =$ next word
- $x =$ weather sensor data, $y =$ max temp tomorrow

Problem:

- observe $(x_i, y_i)_{i=1}^n$ and seek f such that $y_{n+1} \approx f(x_{n+1})$

Training: minimize the empirical loss

$$\ell(\theta) := \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$

But what is $\{f_\theta\}_{\theta \in \Theta}$?

In the case of ANNs, we consider all f_θ having the form

$$f_\theta = \sigma \circ A_1 \circ \cdots \circ \sigma \circ A_{k-1} \circ \sigma \circ A_k$$

where

- $A_i x = W_i x + b_i$ is an affine map
- σ is a nonlinear "activation" function

Training: minimize the empirical loss

$$\ell(\theta) := \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$

But what is $\{f_\theta\}_{\theta\in\Theta}$?

In the case of ANNs, we consider all f_θ having the form

$$f_\theta = \sigma \circ A_1 \circ \cdots \circ \sigma \circ A_{k-1} \circ \sigma \circ A_k$$

where

- $A_i x = W_i x + b_i$ is an affine map
- σ is a nonlinear "activation" function

Training: minimize the empirical loss

$$\ell(\theta) := \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$

But what is $\{f_\theta\}_{\theta \in \Theta}$?

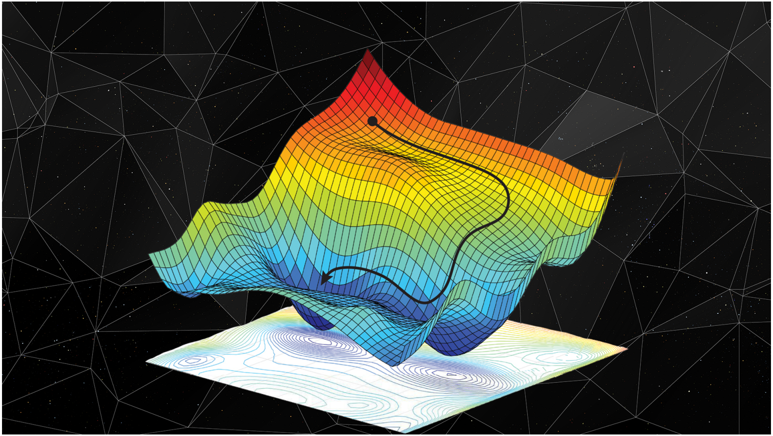
In the case of ANNs, we consider all f_θ having the form

$$f_\theta = \sigma \circ A_1 \circ \cdots \circ \sigma \circ A_{k-1} \circ \sigma \circ A_k$$

where

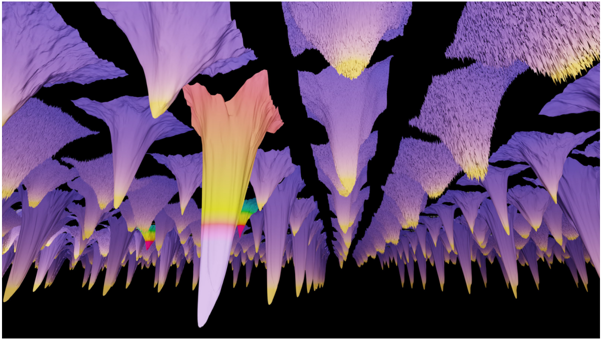
- $A_i x = W_i x + b_i$ is an affine map
- σ is a nonlinear "activation" function

Minimizing a smooth loss functions – what algorithm?



Source: https://danielkhv.com/

Deep learning: $\theta \in \mathbb{R}^d$ where $d = ?$



Source: https://losslandscape.com/gallery/

Hardware

"NVIDIA supercomputers are the factories of the AI industrial revolution." – Jensen Huang

Software

Core elements

- automatic differentiation (for gradient descent)
- parallelization (GPUs! — how many?)
- Compilers / JIT-compilers

Crucially, these components must be well integrated

Popular platforms with these features

- Pytorch
- Google JAX (with Python)

```python
import jax.numpy as jnp
from jax import grad, jit

def f(θ, x):
  for W, b in θ:
    w = W @ x + b
    x = jnp.tanh(w)
  return x

def loss(θ, x, y):
  return jnp.sum((y - f(θ, x))**2)

grad_loss = jit(grad(loss))  # Now use gradient descent
```

Source: JAX readthedocs

JAX for economists

How about mathematical modeling / optimization / simulation?

For these tasks, I recommend JAX

- more consistent with other Python libraries
- exposes lower level building blocks vis-a-vis PyTorch
- automated parallelization
- JIT compiler
- integrated autodiff
- automatically / transparently supports CPUs / GPUs / TPUs