

# Recent Trends in Scientific Computing

John Stachurski

November 2023

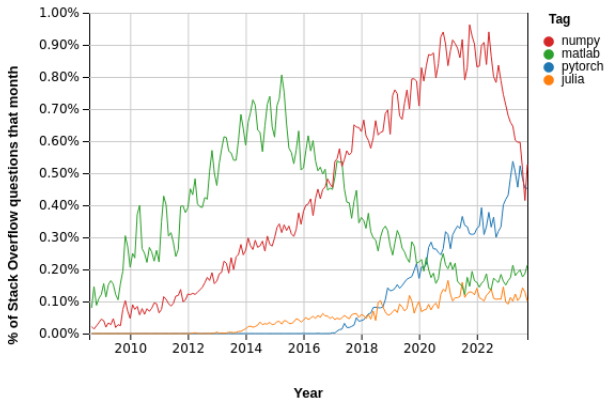
# Topics

- Traditional compiled languages
- Modern JIT compilers
- AI-driven scientific computing
- Where are we heading?
- Economic applications

Sides, code:

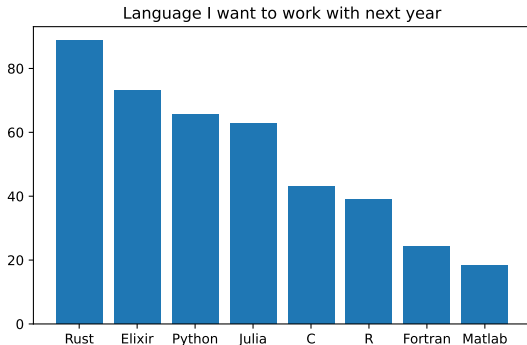
[https://github.com/QuantEcon/treasury\\_2023](https://github.com/QuantEcon/treasury_2023)

Some trends:



Source: Stackoverflow Trends

## Stack Overflow 2023 Developer Survey (50 languages)



— <https://survey.stackoverflow.co/2023/>

# A review of some scientific computing environments

General purpose scientific computing environments:

1. Fortran / C / Rust
2. MATLAB ( $\approx$  Python + NumPy)
3. Julia ( $\approx$  Python + Numba)
4. Python + Google JAX ( $\approx$  Python + PyTorch)

# Fortran / C / Rust — static types and AOT compilers

**Example.** Suppose we want to compute the sequence

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

from some given  $k_0$

Let's write a function in C that

1. implements the loop
2. returns the last  $k_t$

---

```
#include <stdio.h>
```

```
#include <math.h>
```

```
int main() {  
    double k = 0.2;  
    double alpha = 0.4;  
    double s = 0.3;  
    double delta = 0.1;  
    int i;  
    int n = 1000;  
    for (i = 0; i < n; i++) {  
        k = s * pow(k, alpha) + (1 - delta) * k;  
    }  
    printf("k = %f\n", k);  
}
```





## Pros

- fast

## Cons

- time consuming to write
- lack of portability
- hard to debug
- hard to parallelize
- low interactivity

For comparison, the same operation in Python:

---

```
 $\alpha$  = 0.4  
s = 0.3  
 $\delta$  = 0.1  
n = 1_000  
k = 0.2  
  
for i in range(n-1):  
    k = s * k** $\alpha$  + (1 -  $\delta$ ) * k  
  
print(k)
```

---

## Pros

- easy to write
- high portability
- easy to debug
- high interactivity

## Cons

- slow

So how can we get

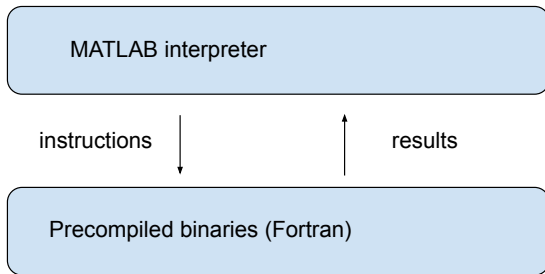
good execution speeds **and** high productivity / interactivity?

# MATLAB

```
A = [2.0, -1.0  
      5.0, -0.5];
```

```
b = [0.5, 1.0]';
```

$$x = \text{inv}(A) * b$$



# Python + NumPy

---

```
import numpy
```

```
A = ((2.0, -1.0),  
      (5.0, -0.5))
```

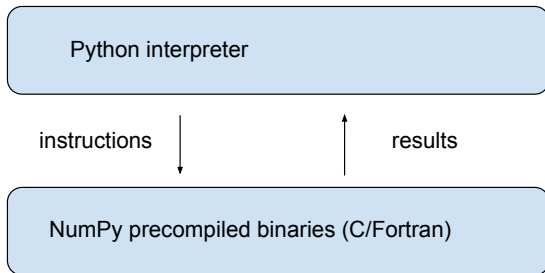
```
b = (0.5, 1.0)
```

```
A, b = np.array(A), np.array(b)
```

```
x = np.inv(A) @ b
```

---







But also has fast loops via an efficient JIT compiler

**Example.** Suppose, again, that we want to compute

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

from some given  $k_0$

- Iterative, not easily vectorized

---

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

solow(0.2)
```

---

Julia accelerates `solow` at runtime via a JIT compiler

# Python + Numba copy Julia

---

```
from numba import jit

@jit(nopython=True)
def solow(k0,  $\alpha=0.4$ ,  $\delta=0.1$ , n=1_000):
    k = k0
    for i in range(n-1):
        k = s * k** $\alpha$  + (1 -  $\delta$ ) * k
    return k

solow(0.2)
```

---

Runs at same speed as Julia / C / Fortran

# AI-driven scientific computing

## Key players

- OpenAI (ChatGPT, Whisper), Microsoft
- Google Research, Google DeepMind
- Meta
- Apple?

## Platforms / libraries

- PyTorch (Meta, OpenAI, Stable Diffusion)
- Google JAX (Google, Apple Ajax)
- Tensorflow, Mojo?

# Lightening introduction to deep learning

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

$$y = f(x)$$

- $x$  is the input and  $y$  is the output

## Examples.

- $x$  = unfinished sentence,  $y$  = next word
- $x$  = weather sensor data,  $y$  = max temp tomorrow

Relevant to economics?

“ECMWF’s weather forecasting model is considered the gold standard for medium-term weather forecasting...Google DeepMind claims in an non-peer-reviewed paper to have beat it 90% of the time...”

“Traditional forecasting models are big, complex computer algorithms based on atmospheric physics and take hours to run. AI models can create forecasts in just seconds.”

Source: MIT Technology Review July 2023



# Training

Nonlinear regression: Take data set  $(x_i, y_i)_{i=1}^n$  and solve

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

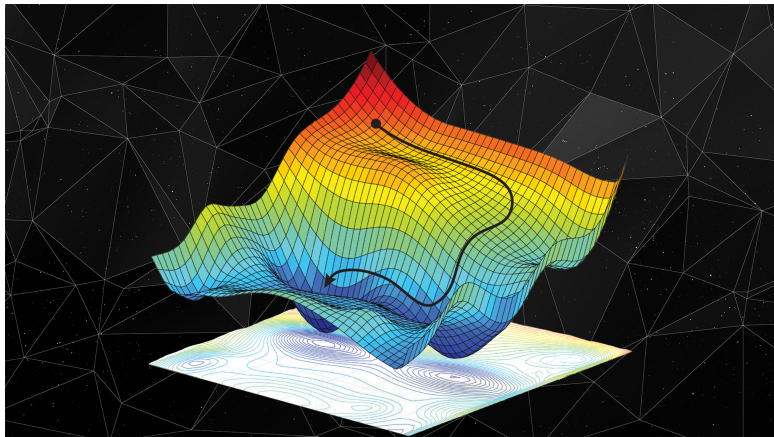
In the case of ANNs, we consider all  $f_{\theta}$  having the form

$$f_{\theta} = \sigma \circ A_1 \circ \cdots \circ \sigma \circ A_k$$

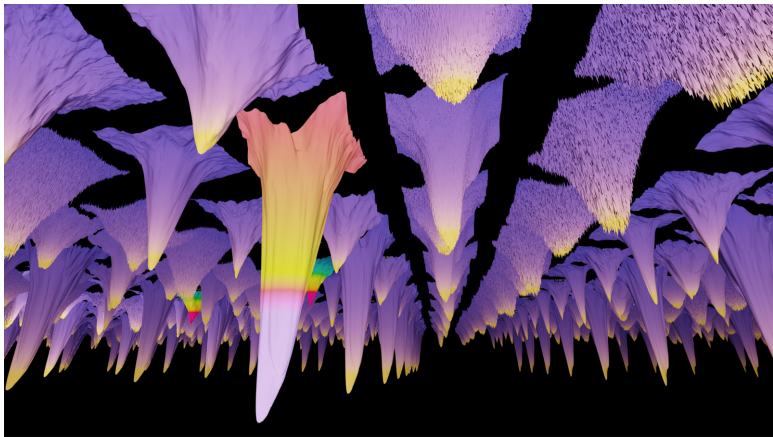
where

- $A_i x = W_i x + b_i$  is an affine map
- $\sigma$  is a nonlinear “activation” function

## Minimizing a smooth loss functions



Source: <https://danielkhv.com/>



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

## Core elements

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

---

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

def predict(params, x):
    for W, b in params:
        y = jnp.dot(W, x) + b
        x = jnp.tanh(y)
    return y

def loss(params, x, targets):
    preds = predict(params, x)
    return jnp.sum((preds - targets)**2)

grad_loss = jit(grad(loss))
```

---

Sample code

[https://github.com/QuantEcon/treasury\\_2023/notebooks](https://github.com/QuantEcon/treasury_2023/notebooks)