# An AI-Driven Revolution in Scientific Computing

John Stachurski

2025

# Topics

Part 1: Slides

- AI-driven scientific computing
- Applications

Part 2: Hands on coding

`https://github.com/QuantEcon/rba_workshop_2024`

# 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....

Private AI investment in 2024:

- U.S. = $109 billion
- China $9.3 billion
- UK $4.5 billion

Investments in

- Data centers
- GPUs
- software development

What does this software do?

# Deep learning in two slides

Aim: approximate an unknown functional relationship

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

Examples.

- $x =$ cross section of returns, $y =$ return on oil futures tomorrow
- $x =$ weather sensor data, $y =$ max temp tomorrow

Problem:

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

Nonlinear regression: choose model $\{f_\theta\}_{\theta\in\Theta}$ and minimize the empirical loss

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

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

$$f_\theta = \sigma \circ A_m \circ \cdots \circ \sigma \circ A_2 \circ \sigma \circ A_1$$

where

- $A_j x = W_j x + b_j$ is an affine map
  - `output = dot(kernel, input) + bias`
- $\sigma$ is a nonlinear "activation" function

Nonlinear regression: choose model $\{f_\theta\}_{\theta\in\Theta}$ and minimize the empirical loss

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

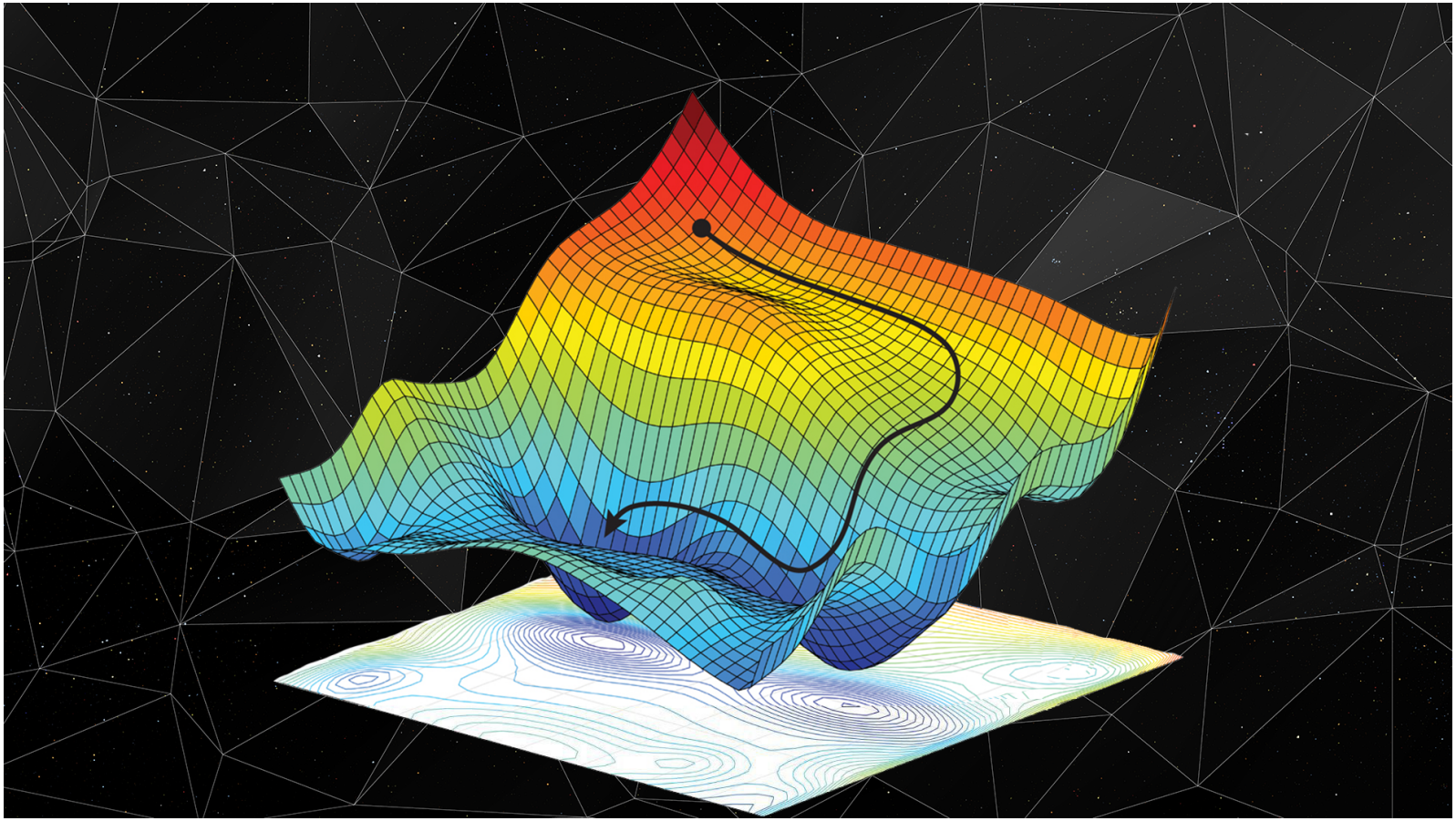
In the case of ANNs, we consider all $f_\theta$ having the form

$$f_\theta = \sigma \circ A_m \circ \cdots \circ \sigma \circ A_2 \circ \sigma \circ A_1$$

where

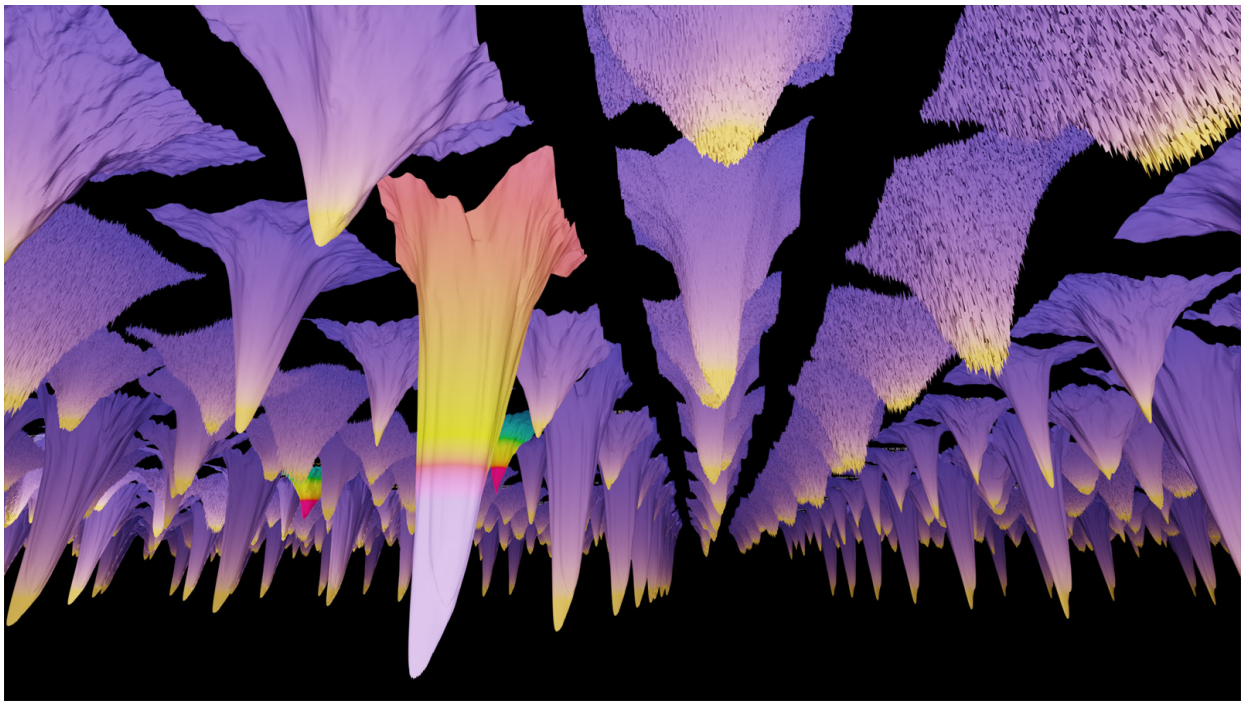
- $A_j x = W_j x + b_j$ is an affine map
  - `output = dot(kernel, input) + bias`
- $\sigma$ is a nonlinear "activation" function

# 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/

# How does it work?

Why is it possible to minimize over $\theta \in \mathbb{R}^d$ when $d = 10^{12}$ ?!?

Core elements

- automatic differentiation (for gradient descent)
- parallelization (GPUs or TPUs)
- Compilers / JIT-compilers

# How does it work?

Why is it possible to minimize over $\theta \in \mathbb{R}^d$ when $d = 10^{12}$ ?!?

Core elements

- automatic differentiation (for gradient descent)
- parallelization (GPUs or TPUs)
- Compilers / JIT-compilers

# Automatic differentiation

"Exact numerical" differentiation

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

loss_gradient = grad(loss)
```

Now use gradient descent...

# Parallelization

```
outputs = pmap(f, data)
```

- multithreading over GPU cores (how many?)
- multiprocessing over accelerators in a GPU farm / supercomputing cluster (how many?)

# Just-in-time compilers

```
@jit
def f(x):
    return jnp.sin(x) - jnp.cos(x**2)
```

Advantages over AOT compilers:

- cleaner code
- more portable
- automatic parallelization (same code for CPUs / GPUs)

Advantages over NumPy / MATLAB

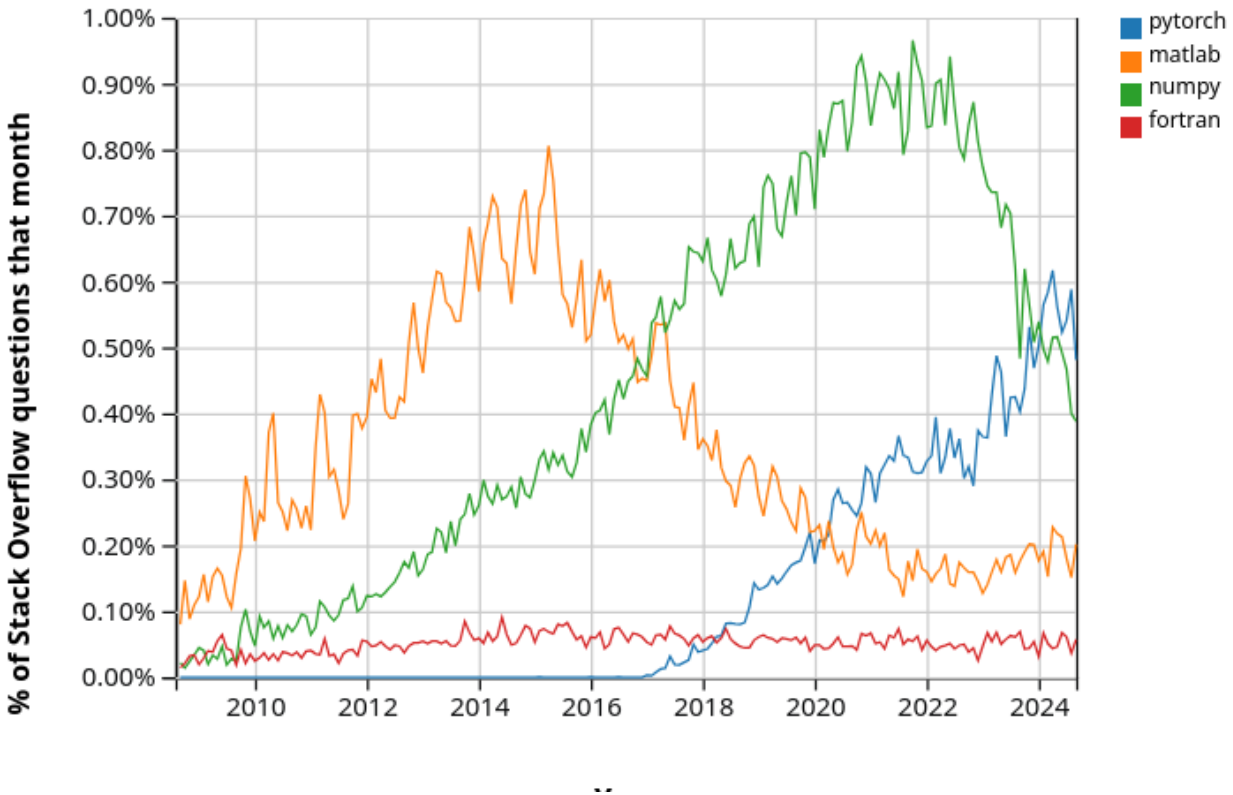
- can specialize machine code based on parameter types / shapes
- automatically matches tasks with accelerators (GPU / TPU)
- fuses array operations for speed and memory efficiency

# Platforms

Platforms that support AI / deep learning:

- Tensorflow
- PyTorch (Llama, ChatGPT)
- Google JAX (Gemini, DeepMind)
- Keras (backends = JAX, PyTorch)
- Mojo? (Modular (Python))
- MATLAB?

# Popularity


% 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
pytorch
matlab
numpy
fortran

# AI tools for economic modeling

Let's say that you want to do computational macro rather than deep learning

Can these new AI tools be applied?

Yes!

- fast matrix algebra
- fast solutions to linear systems
- fast nonlinear system solvers
- fast optimization, etc.

# AI tools for economic modeling

Let's say that you want to do computational macro rather than deep learning

Can these new AI tools be applied?

Yes!

- fast matrix algebra
- fast solutions to linear systems
- fast nonlinear system solvers
- fast optimization, etc.

Advantages of JAX (vs PyTorch / Numba / etc.) for economists:

- exposes low level functions
- elegant functional programming style – close to maths
- elegant autodiff tools
- array operations follow standard NumPy API
- automatic parallelization
- same code, multiple backends (CPUs, GPUs, TPUs)

# Case Study

The CBC uses the "overborrowing" model of Bianchi (2011)

- credit constraint loosens during booms
- bad shocks $\rightarrow$ sudden stops

CBC implementation in MATLAB

- runs on \$10,000 mainframe with 356 CPUs and 1TB RAM
- runtime $=$ 12 hours

Rewrite in Python + Google JAX

- runs on \$400 gaming GPU with 10GB RAM
- runtime $=$ 4.17 seconds

# Case Study

The CBC uses the "overborrowing" model of Bianchi (2011)

- credit constraint loosens during booms
- bad shocks $\rightarrow$ sudden stops

CBC implementation in MATLAB

- runs on \$10,000 mainframe with 356 CPUs and 1TB RAM
- runtime $=$ 12 hours

Rewrite in Python $+$ Google JAX

- runs on \$400 gaming GPU with 10GB RAM
- runtime $=$ 4.17 seconds

# Live coding

See notebooks in

`https://github.com/QuantEcon/rba_workshop_2024`

Steps

1. Go to Google Colab (`https://colab.google/`)
2. Open notebook → GitHub → quantecon → `rba_workshop_2024` → select notebook
3. Edit → Notebook settings → select a GPU
4. Shift-enter to run each cell

# AI for Coding

Intro

Claude:

"I'm definitely stronger with Python than MATLAB.

While I'm knowledgeable about both languages, my capabilities with Python are more comprehensive. I have deeper familiarity with Python's extensive ecosystem of libraries, frameworks, and modern development practices. I can more confidently help with advanced Python topics, debugging complex Python code, and implementing Python best practices."

Claude:

"I'm definitely stronger with Python than Julia.

Python is one of my most proficient languages - I have deep familiarity with its syntax, libraries, frameworks, and best practices across many domains including data science, web development, machine learning, and general-purpose programming.

While I understand Julia's syntax and core concepts, my expertise with it isn't as comprehensive as with Python."