Lesson 0: Preparing Your Computer for the Course

Aaron A. King Edward L. Ionides Kunyang He

Tuesday, March 10, 2026

Introduction

This course uses state-of-the-art computational tools. We use `pypomp`,a Python package for modeling and data analysis with partially observed Markov process (POMP) models. `pypomp` uses `jax` to provide high-performance computing.

- `jax` can be thought of as an extension to `numpy` that permits just-in-time (JIT) compilation, GPU computing and automatic differentiation (AD).
- Basic JAX, on a CPU, without AD, is enough for this course.
- JIT is managed by `pypomp` and you won't necessarily need to look under the hood.
- As model complexity and data size grow, you can start taking advantage of the power of JAX.

Additionally, the course emphasizes reproducible data analysis using Quarto

What you will need

To get started:

- Python 3.10 or higher.
- Basic familiarity with Python programming
- A text editor or IDE (e.g., VS Code, PyCharm, Jupyter, Positron). The notes are in Quarto (qmd) format. VS Code (with the Quarto extension) and Positron are recommended for working with qmd files.
- If you run Windows, the WSL2 Linux virtual machine is recommended for `jax`.

Python Installation

First, check if you have Python installed and verify the version:

```
python --version
```

or

```
python3 --version
```

You should have Python 3.10 or higher. If not, make a fresh installation. Various Python distributions exist, but we recommend using the official Python distribution:

1. Download Python from https://www.python.org/downloads/
2. Follow the installation instructions for your operating system
3. Make sure to check "Add Python to PATH" during installation (Windows)

Setting Up a Virtual Environment

It is highly recommended to use a virtual environment to avoid conflicts with other Python projects. We use the built-in `venv`

```
# Create a virtual environment in the
# working directory for your project
python -m venv .venv

# Activate the environment
source .venv/bin/activate
```

Installing Required Packages

The main package we will use is `pypomp`. Install it using `pip`:

```
pip install pypomp
```

This will also install various dependencies, including `jax`, `numpy`, `pandas`, `scipy`, `matplotlib`.

The following additional packages are commonly used in the course:

```
pip install jupyter
```

An introduction to JAX for pypomp

We cover enough to get started. The JAX documentation is good for learning more. Here, we focus on CPU hardware. GPUs are useful for scalability.

- `jax` is a package in the JAX library. The partner package `jaxlib` is needed for compilation.

```
pip install jax jaxlib # for CPU
pip install jax[cuda12] # for GPU
```

Customary imports are

```
import jax.numpy as jnp
import numpy as np
```

You can mix `np` with `jnp`. Howver, JIT and AD require `jnp`. Specifically, `np` variables are considered constants by JIT and AD.

- For `pypomp`, it is usually better to use `jnp`.

What is JIT and what do I need to know about it?

- Python is a relatively slow language, interpreted rather than compiled.
- JIT in JAX can lead to large acceleration, say, 10 times on a CPU
- JIT enables code to run on a GPU if your machine has one. This can lead to much greater acceleration, if your GPU has 10^4 cores.
- Just-in-time compilation means that functions are compiled only when called. Different values of constants, or different dimensions of arguments, trigger re-compilation. Excessive re-compilation can severly hurt performance.

Vectorization and GPU acceleration with JAX

`vmap` is a `jax` function that **vectorizes** code.

`vmap` instructs JAX that multiple calls to a function can be done in parallel.

- A scalar function applied to a vector argument is vectorized by appyling the function to each component of the vector.

`jit` can compile a `vmap` to make use of multiple cores on a CPU or GPU.

What is automatic differentiation (AD) and what do I need to know about it?

AD is an elegant algorithm providing a gradient for not much more computational effort than evaluating the function.

- AD amounts to recursive application of the chain rule for differentiation.
- Access to gradients can speed up optimization and other numeric calculations.
- For this course, we don't need to know much about AD. It can be helpufl, and some `pypomp` algorithms use the AD capabilities of `jax`.

Setting Up Jupyter

Jupyter provides an alternative to Quarto for running Python code. If you haven't already installed Jupyter:

```
pip install jupyter notebook jupyterlab
```

To start Jupyter Notebook:

```
jupyter notebook
```

To start JupyterLab, a more modern interface:

```
jupyter lab
```

Registering your virtual environment as a Jupyter kernel

If you created a virtual environment, register it with Jupyter:

```
# Make sure your virtual environment is activated
# First, cd to the directory where you built .venv
source .venv/bin/activate
pip install ipykernel
python -m ipykernel install --user --name=.venv \
    --display-name="SBIED Course"
```

Test basic pypomp functionality

Create a simple test to verify pypomp is working:

```
import pypomp as pp
import jax
import matplotlib.pyplot as plt

d = pp.dacca() # constructs an example POMP model
key = jax.random.key(42)
n_sims = 1
X_sims, Y_sims = d.simulate(key,nsim=n_sims)
sim_df = Y_sims.pivot_table(
    index="time", columns="sim", values="obs_0"
)
```

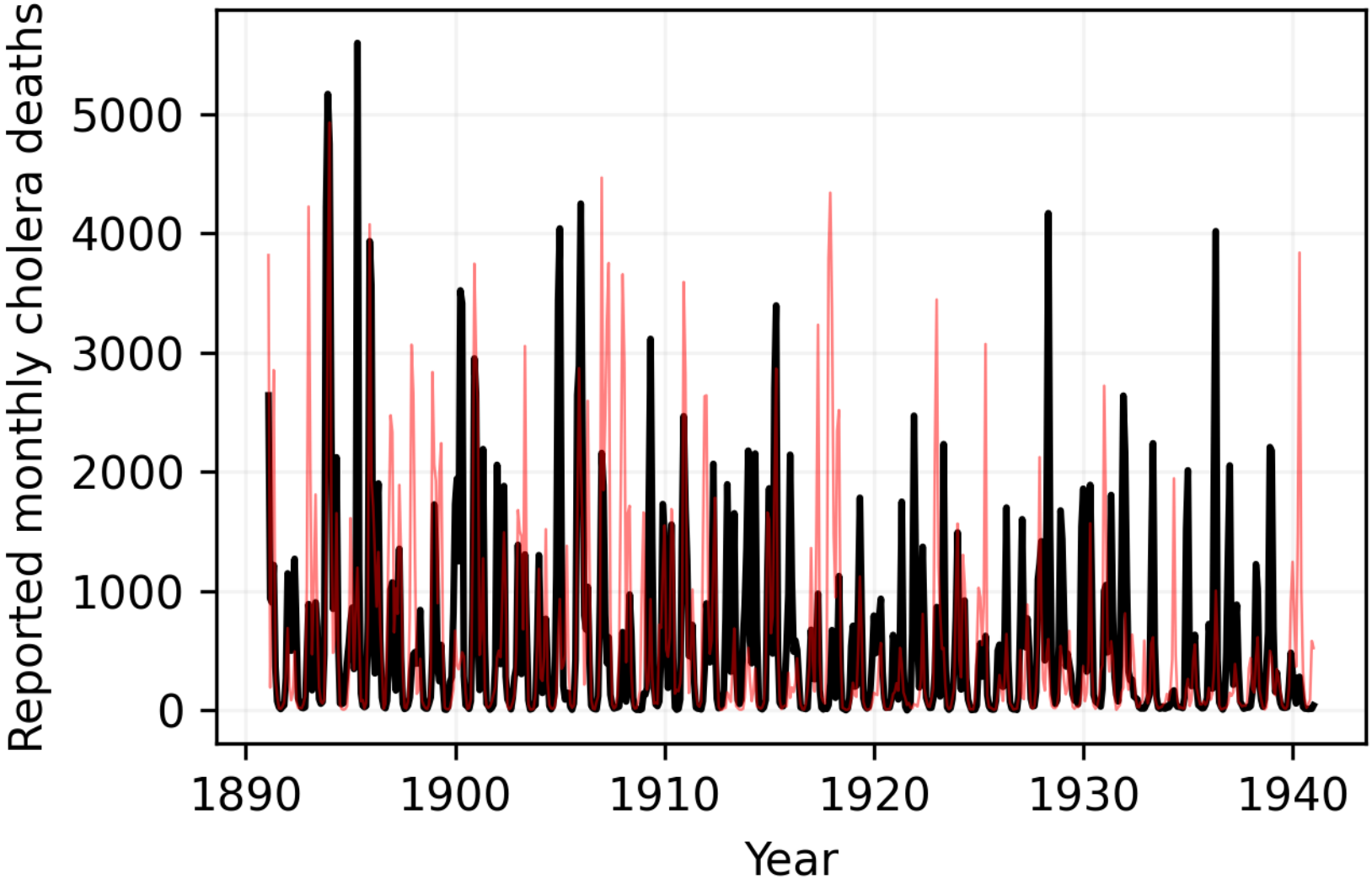

Figure 1: Dhaka cholera: data (black) and a simulation (red).

Using Git to make a local copy of SBIED materials

```
git clone https://github.com/pypomp/tutorials.git
cd tutorials/sbied
```

Then, you can use

```
git pull
```

to get updated files as they arrive.

Troubleshooting: Common issues and solutions I

"pip: command not found"

- Make sure Python is installed and added to your PATH
- Try using `python -m pip` instead of `pip`

"Permission denied" errors

- This should not happen when using a virtual environment as recommended

Import errors

- Make sure your virtual environment is activated
- Verify package installation: `pip freeze`
- Try reinstalling: `pip uninstall package_name && pip install package_name`

Troubleshooting: Common issues and solutions II

Jupyter kernel issues

- Make sure ipykernel is installed in your environment
- Re-register the kernel (see "Registering your virtual environment" above)
- Restart Jupyter after adding a new kernel

Getting help

If you encounter issues:

1. Ask your favorite AI
2. Check the pypomp documentation: https://pypomp.readthedocs.io/
3. Search for error messages online
4. Ask for help from the instructors

Summary

By now, you should have:

- Python 3.10+ installed
- A virtual environment created and activated
- pypomp and required packages installed
- Quarto and/or Jupyter set up
- Verified your installation with test code

You are now ready to begin computing with POMP models!

Additional Resources

- **Python Tutorial**: https://docs.python.org/3/tutorial/
- **NumPy Documentation**: https://numpy.org/doc/
- **Matplotlib Tutorial**: https://matplotlib.org/stable/tutorials/index.html
- **Jupyter Documentation**: https://jupyter.org/documentation
- **pypomp Repository**: https://github.com/pypomp/pypomp
- **JAX Documentation**: https://docs.jax.dev