Empirical Dynamic Modelling

Automatic Causal Inference and Forecasting

Patrick Laub

UNSW Sydney, School of Risk and Actuarial Studies

February 7, 2023

About Me & Outline

- Software engineer & maths (UQ)
- PhD in probability (Aarhus)
- Post-doc (ISFA Lyon)
- RSE (Uni Melbourne)
- Lecturer (UNSW)

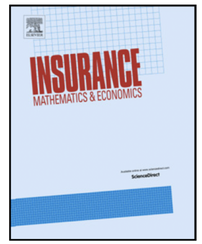
1. Brief ABC game
2. Empirical Dynamic Modelling
3. [Time permitting] Performant code

Insurance: Mathematics and Economics 100 (2021) 350–371

Contents lists available at ScienceDirect

Insurance: Mathematics and Economics

www.elsevier.com/locate/ime



Approximate Bayesian Computations to fit and compare insurance loss models

Pierre-Olivier Goffard [a,b,*], Patrick J. Laub [a,b]

[a] Univ Lyon, Université Lyon 1, LSAF EA2429, France
[b] University of Melbourne, Australia

Our IME paper

What is the bias?

We flip a coin three times and get:

Heads

Tails

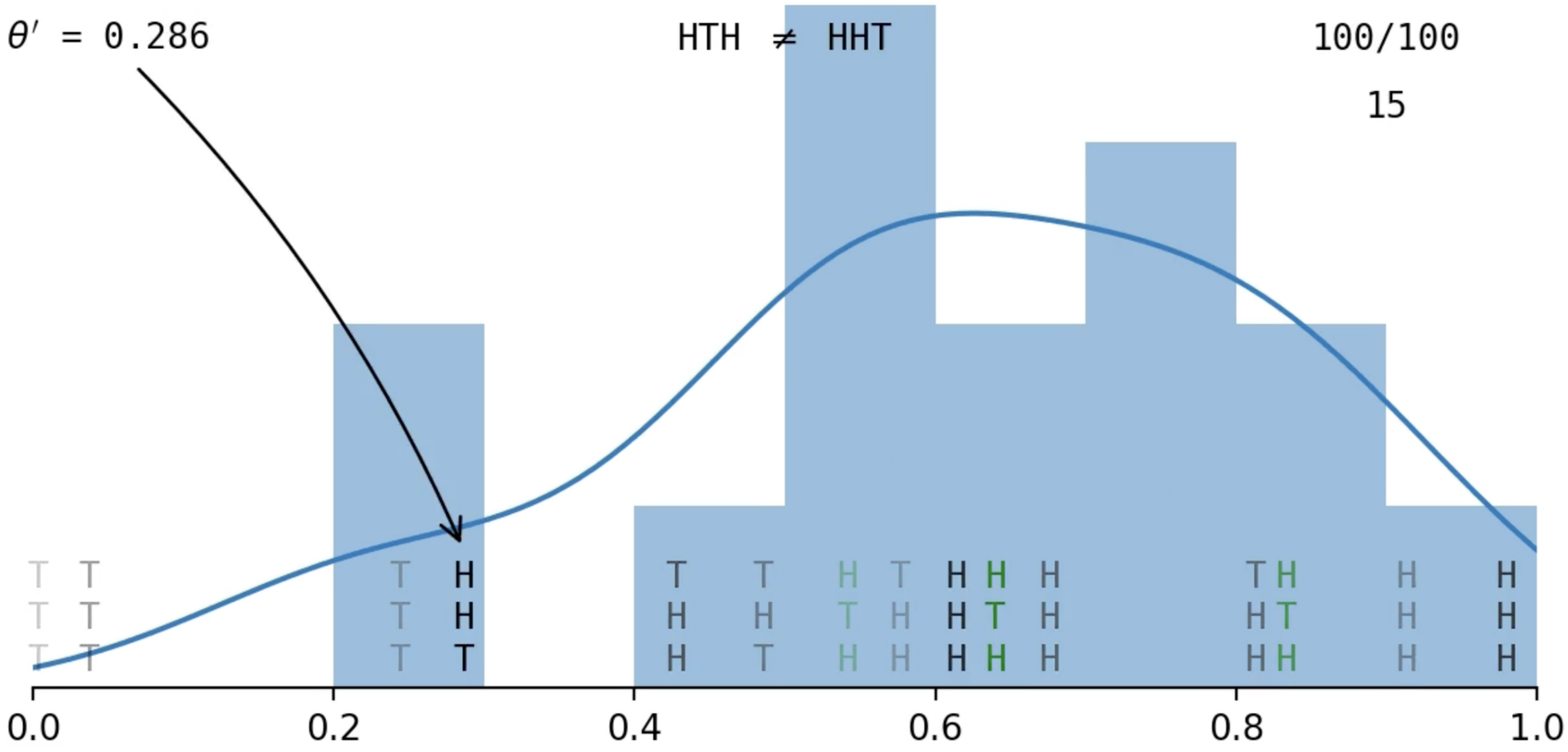
Heads

What is $\theta = \mathbb{P}(\text{Heads})$?

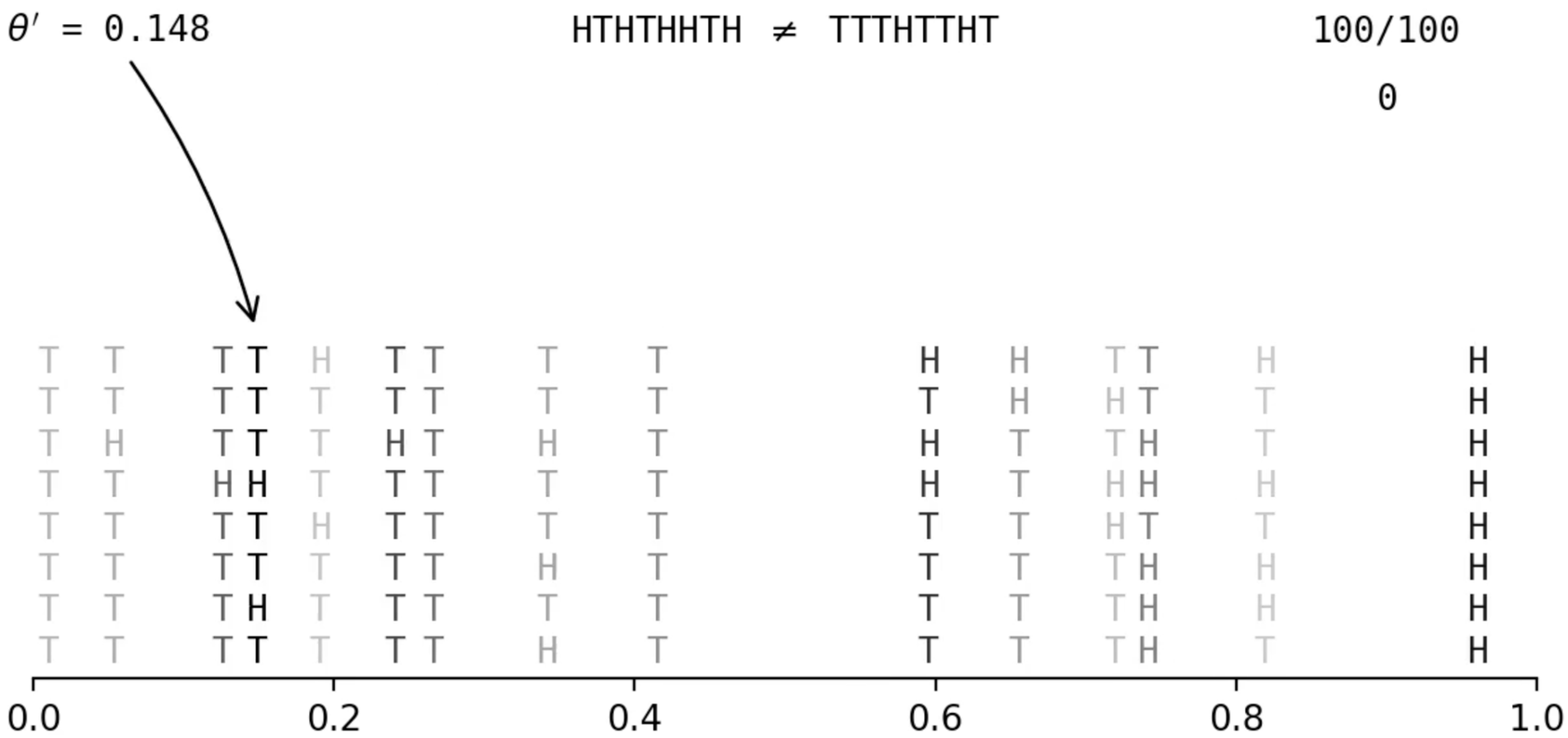
Let's flip coins

https://pat-laub.github.io/abc-party/game

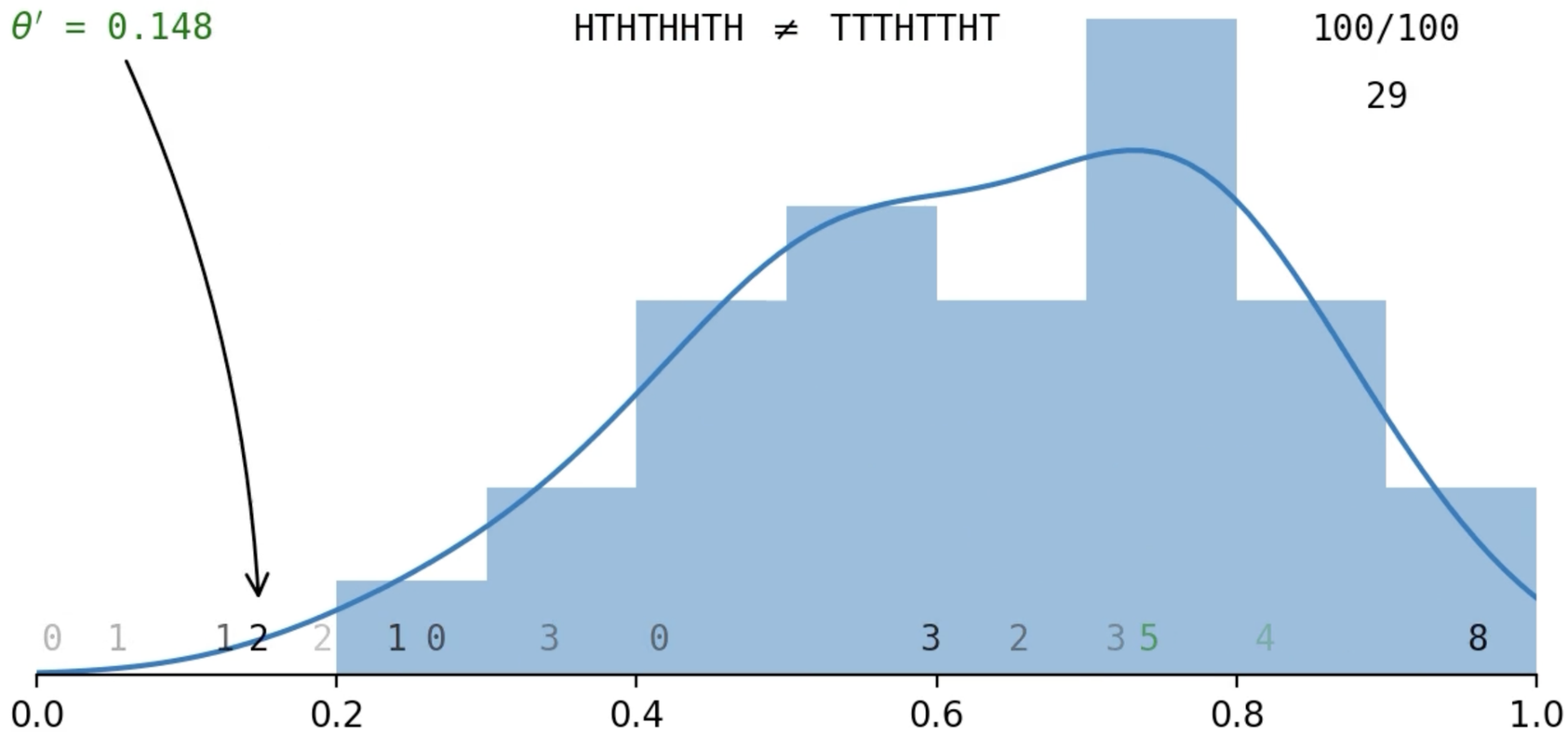
θ-biased coin gives (H, T, H)



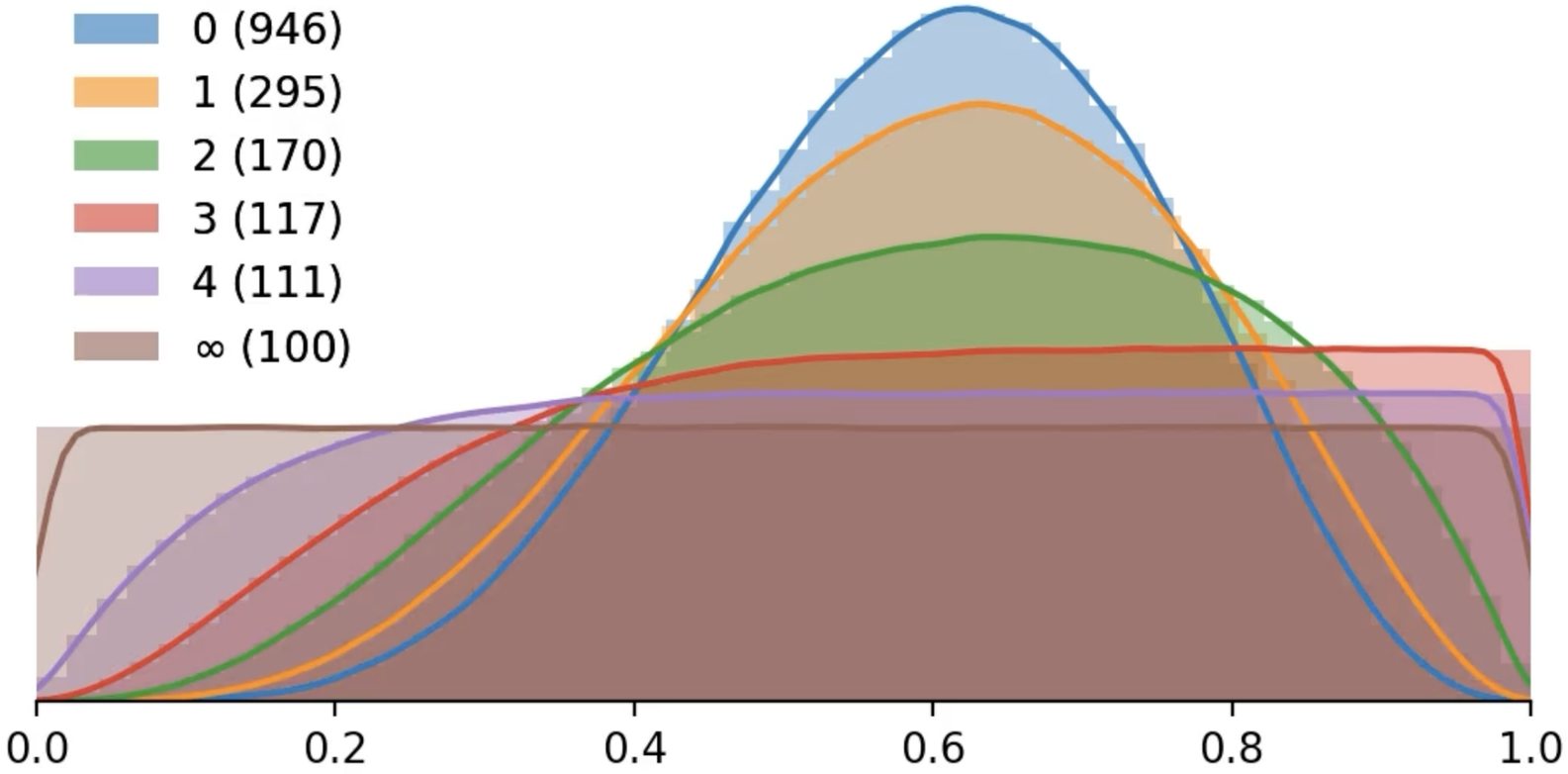
Getting an exact match is hard…



Accept close enough



The ‘approximate’ part of ABC



Causal Inference

Introduction

Goal: automatic causal inference

```
df <- read.csv("chicago.csv")
head(df)
#>   Time Temperature Crime
#> 1    1       24.08  1605
#> 2    2       19.04  1119
#> 3    3       28.04  1127
#> 4    4       30.02  1154
#> 5    5       35.96  1251
#> 6    6       33.08  1276

library(fastEDM)

crimeCCMCausesTemp <- easy_edm("Crime", "Temperature", data=df)
#> ✕ No evidence of CCM causation from Crime to Temperature found.

tempCCMCausesCrime <- easy_edm("Temperature", "Crime", data=df)
#> ✓ Some evidence of CCM causation from Temperature to Crime found.
```

The Stata Journal (2021)
21, Number 1, pp. 220–258 DOI: 10.1177/1536867X211000030

Beyond linearity, stability, and equilibrium: The edm package for empirical dynamic modeling and convergent cross-mapping in Stata

Jinjing Li
University of Canberra

Michael J. Zyphur
University of Queensland

George Sugihara
University of California San Diego

Patrick J. Laub
UNSW

Acknowledgments

Discovery Project DP200100219 and Future Fellowship FT140100629.

Mirage correlation

$$X(t+1) = X(t)\,[3.8 - 3.8\,X(t) - 0.02\,Y(t)]$$
$$Y(t+1) = Y(t)\,[3.5 - 3.5\,Y(t) - 0.1\,X(t)]$$

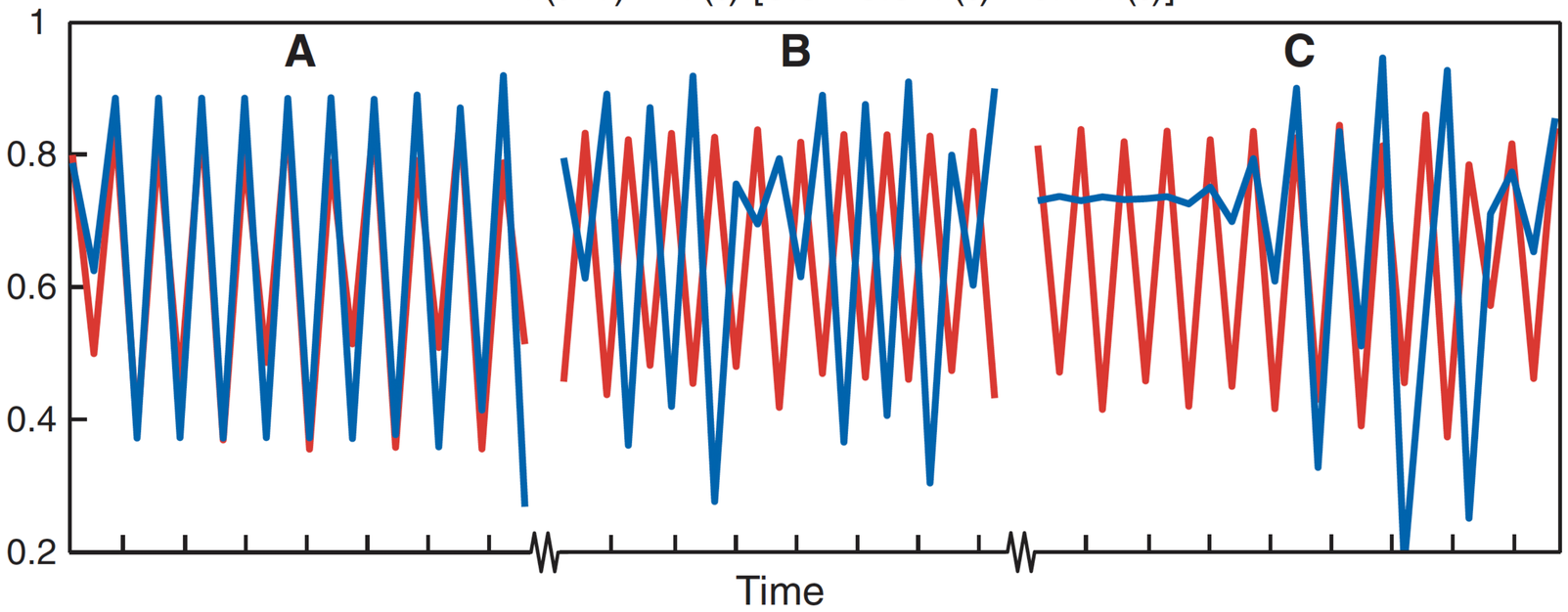


Fig. 1. Mirage correlations. (**A** to **C**) Three samples from a single run of a coupled two-species nonlinear logistic difference system with chaotic dynamics. Variables X (blue) and Y (red) appear correlated in the first time segment (A), anticorrelated in the second time segment (B), and lose all coherence in the third time segment (C) with alternating interspersed periods of positive, negative, and zero correlation. Although the system is deterministic and dynamically coupled, there is no long-term correlation ($n = 1000$, $\rho = 0.0054$, $P = 0.864$).

Source: Sugihara et al. (2012), Detecting causality in complex ecosystems.

A different view of causality

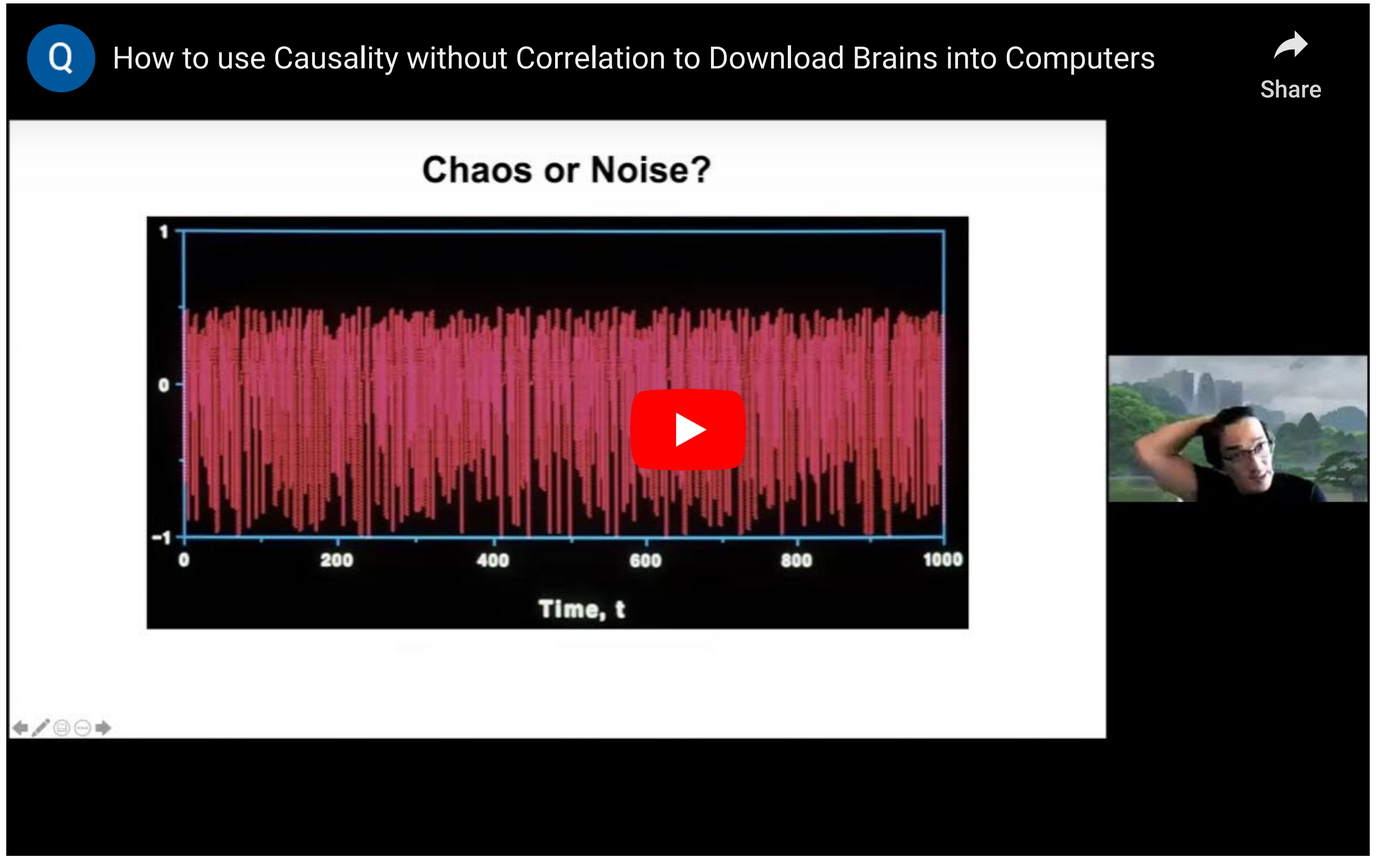
Imagine x_t, y_t, z_t are interesting time series…

If the data is generated according to the nonlinear system:

$$x_{t+1} = \sigma(y_t - x_t)$$
$$y_{t+1} = x_t(\rho - z_t) - y_t$$
$$z_{t+1} = x_t y_t - \beta z_t$$

then $y \Rightarrow x$, both $x, z \Rightarrow y$, and both $x, y \Rightarrow z$.

Fish brains



Gerald Pao (2021), “How to use Causality without Correlation to Download Brains into Computers”, QMNET Seminar.

Linear/nonlinear dynamical systems

Say $\mathbf{x}_t = (x_t, y_t, z_t)$, then if:

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t$$

we have a linear system.

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t)$$

we have a nonlinear system.

> Using a term like nonlinear science is like referring to the bulk of zoology as the study of non-elephant animals. (Stanisław Ulam)

We don't fit a model for f, non-parametrically use the data. Hence the name *empirical* dynamic modelling.

Unobserved variables

Takens' theorem to the rescue, though…

> Takens' theorem is a deep mathematical result with far-reaching implications. Unfortunately, to really understand it, it requires a background in topology. (Munch et al. 2020)

Takens's theorem

From Wikipedia, the free encyclopedia

In the study of dynamical systems, a **delay embedding theorem** gives the conditions under which a chaotic dynamical system can be reconstructed from a sequence of observations of the state of a dynamical system. The reconstruction preserves the properties of the dynamical system that do not change under smooth coordinate changes (i.e., diffeomorphisms), but it does not preserve the geometric shape of structures in phase space.

Contents [hide]

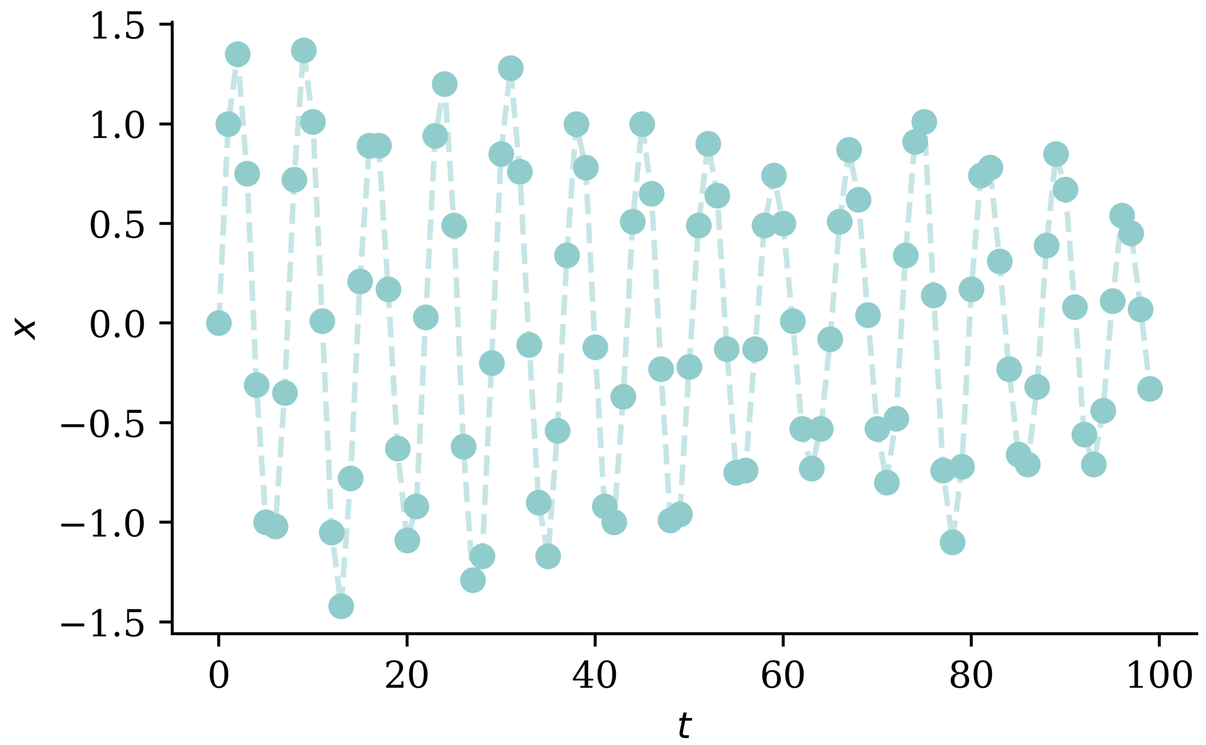
Simplified, slightly inaccurate version [edit]

Suppose the d-dimensional state vector x_t evolves according to an unknown but continuous and (crucially) deterministic dynamic. Suppose, too, that the one-dimensional observable y is a smooth function of x, and "coupled" to all the components of x. Now at any time we can look not just at the present measurement $y(t)$, but also at observations made at times removed from us by multiples of some lag τ : $y_{t+\tau}, y_{t+2\tau}$, etc. If we use k lags, we have a k-dimensional vector. One might expect that, as the number of lags is increased, the motion in the lagged space will become more and more predictable, and perhaps in the limit $k \to \infty$ would become deterministic. In fact, the dynamics of the lagged vectors become deterministic at a finite dimension; not only that, but the deterministic dynamics are completely equivalent to those of the original state space (More exactly, they are related by a smooth, invertible change of coordinates, or diffeomorphism.) The magic embedding dimension k is at most $2d + 1$, and often less.[1]

Source: Munch et al. (2020), Frequently asked questions about nonlinear dynamics and empirical dynamic modelling, ICES Journal of Marine Science.

Simplex Algorithm

Toy problem



t	x_t
0	0.00
1	1.00
2	1.35
...	...
98	0.07
99	-0.33

Create the embeddings

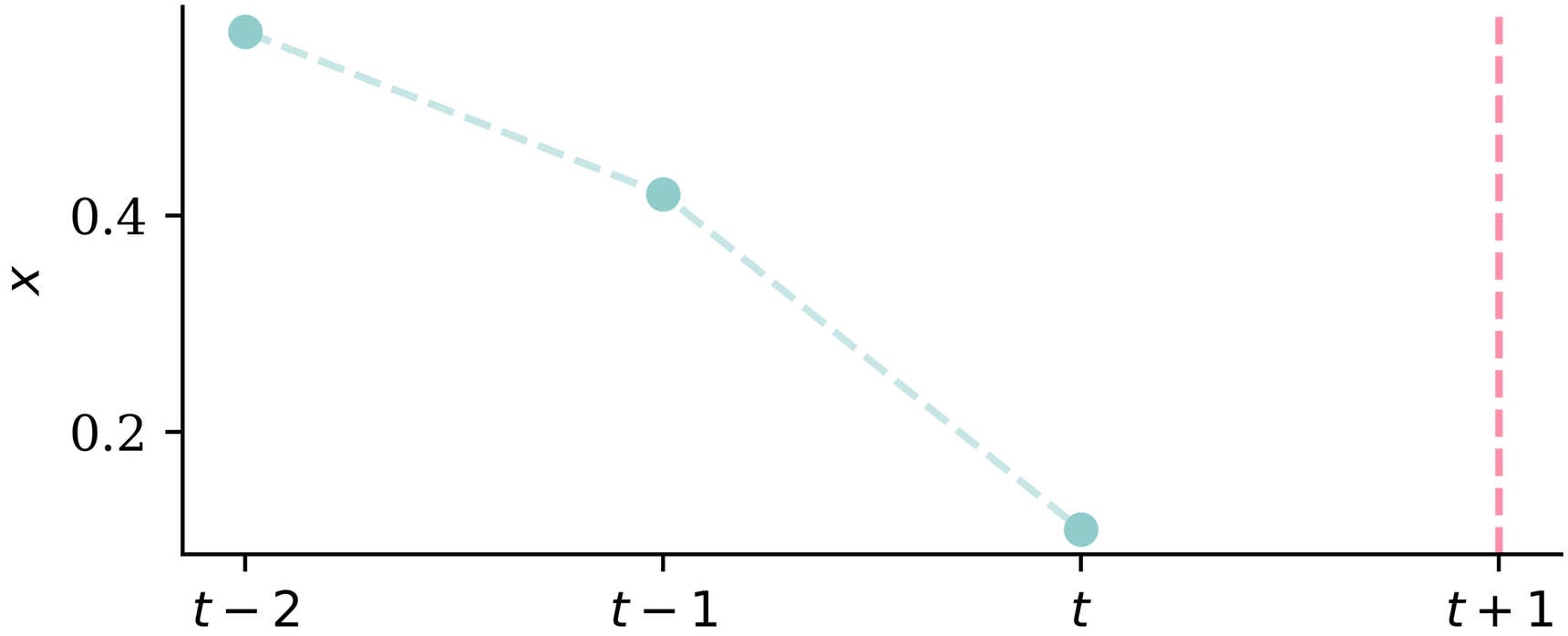
t	x_t
0	0.00
1	1.00
2	1.35
...	...
98	0.07
99	-0.33

i	x_t	x_{t-1}	x_{t-2}
0	1.35	1.00	0.00
1	0.75	1.35	1.00
2	-0.31	0.75	1.35
...	...	...	...
95	0.45	0.54	0.11
96	0.07	0.45	0.54

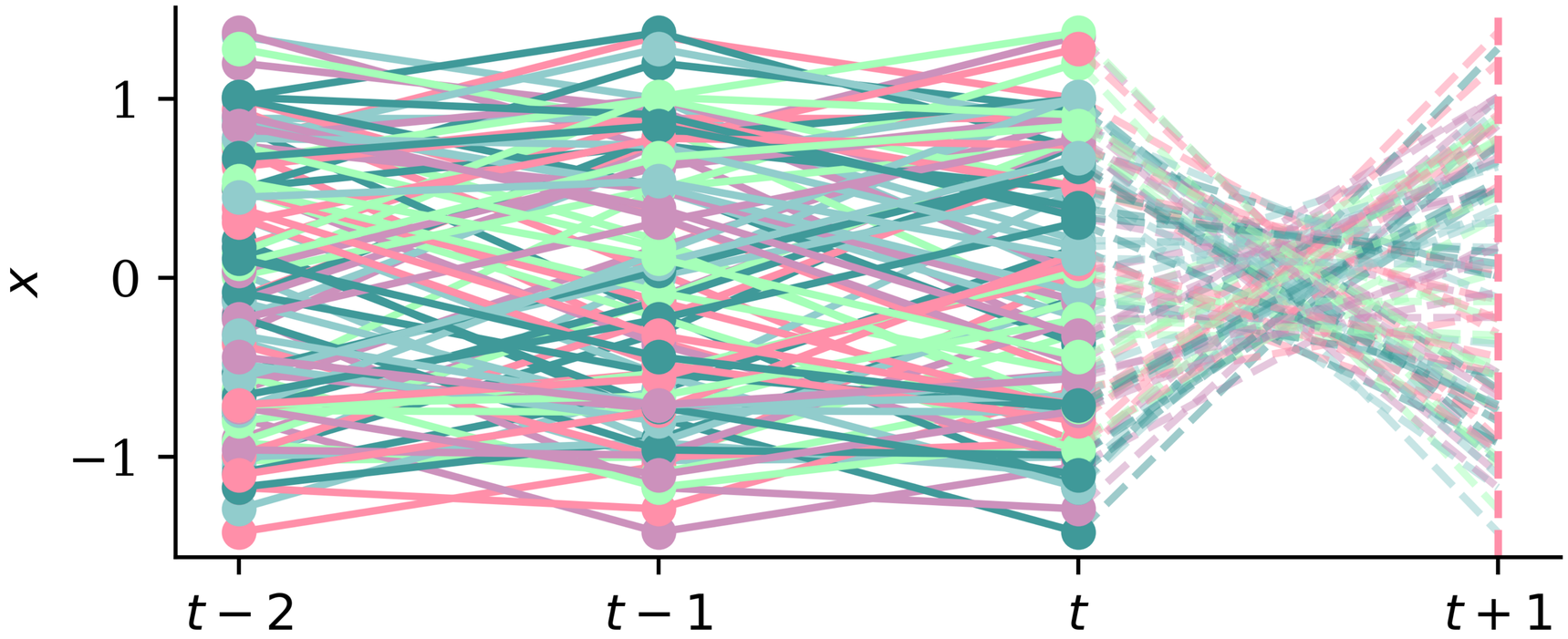
i	x_{t+1}
0	0.75
1	-0.31
2	-1.00
...	...
95	0.07
96	-0.33

Making a prediction

$$\boldsymbol{x}^* = (0.11, 0.42, 0.57)$$



Look at the manifold



Calculate the distances to the point

i	x_t	x_{t-1}	x_{t-2}	x_{t+1}	$d(\mathbf{x}_i, \mathbf{x}^*)$
0	1.35	1.00	0.00	0.75	0.98
1	0.75	1.35	1.00	-0.31	1.30
2	-0.31	0.75	1.35	-1.00	1.56
...	...	...	...	...	...
95	0.45	0.54	0.11	0.07	0.17
96	0.07	0.45	0.54	-0.33	0.66

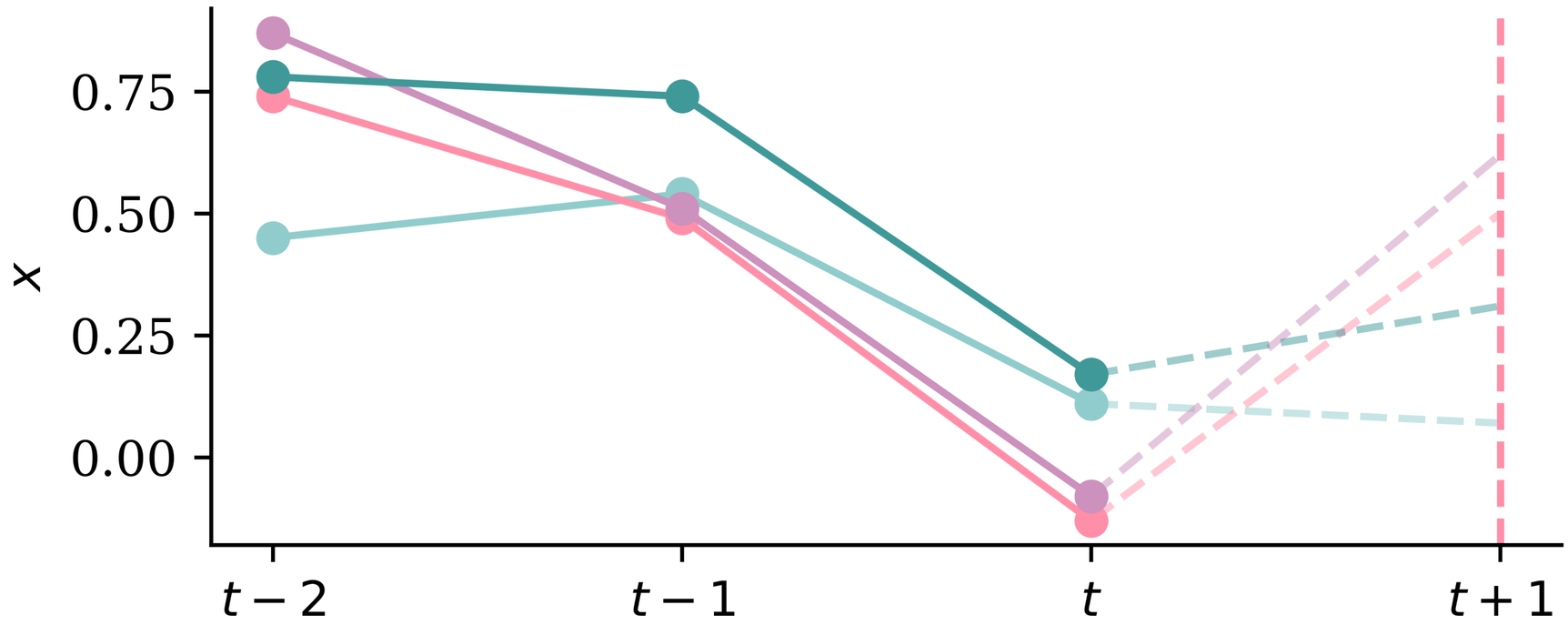
Find the nearest neighbours

i	x_t	x_{t-1}	x_{t-2}	x_{t+1}	$d(\mathbf{x}_i, \mathbf{x}^*)$
95	0.45	0.54	0.11	0.07	0.17
57	0.74	0.49	-0.13	0.50	0.30
65	0.87	0.51	-0.08	0.62	0.37
...	...	...	...	...	...
26	-1.17	-1.29	-0.62	-0.20	2.55
12	-0.78	-1.42	-1.05	0.21	2.56

Find the nearest neighbours

i	x_t	x_{t-1}	x_{t-2}	x_{t+1}	$d(\mathbf{x}_i, \mathbf{x}^*)$
95	0.45	0.54	0.11	0.07	0.17
57	0.74	0.49	-0.13	0.50	0.30
65	0.87	0.51	-0.08	0.62	0.37
80	0.78	0.74	0.17	0.31	0.39

Plot those trajectories

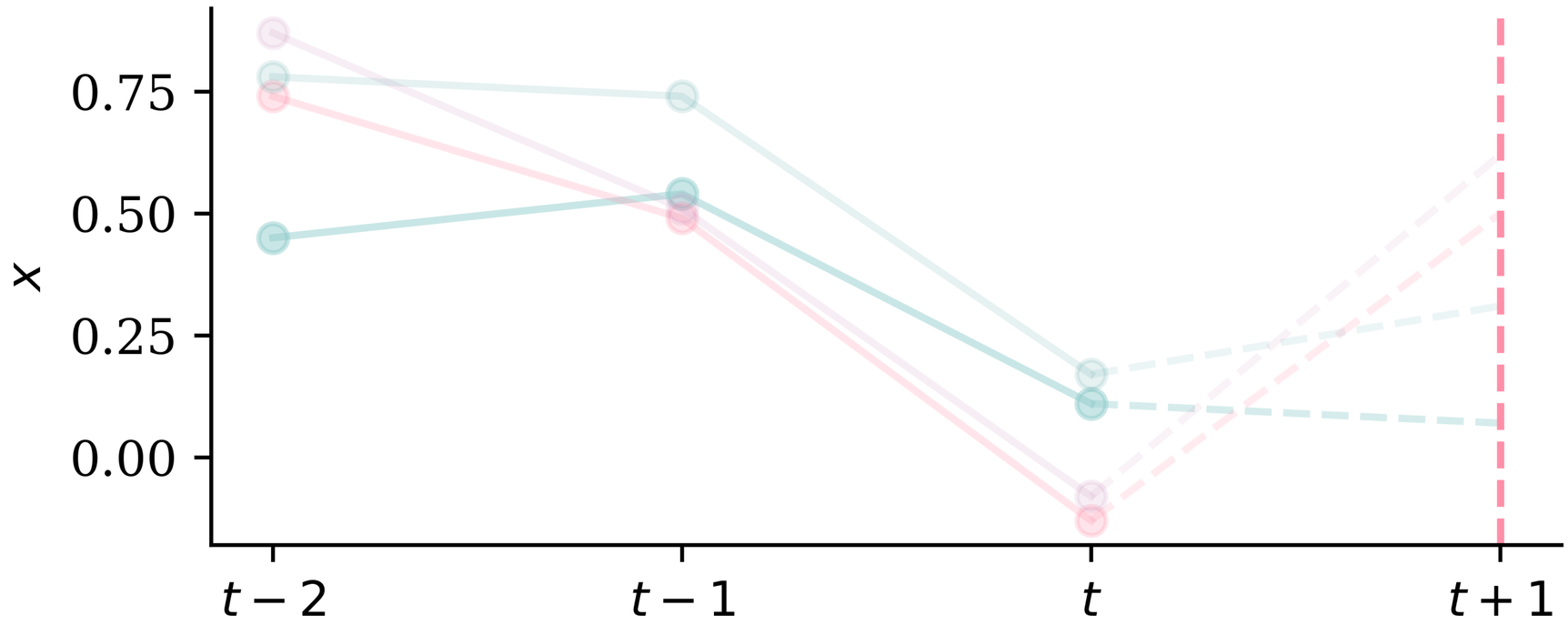


Make a prediction

i	x_{t+1}	$d(\mathbf{x}_i, \mathbf{x}^*)$	$\tilde{d}(\mathbf{x}_i, \mathbf{x}^*)$	w_i
95	0.07	0.17	1.00	0.49
57	0.50	0.30	1.76	0.23
65	0.62	0.37	2.18	0.15
80	0.31	0.39	2.29	0.13

$$w_i = \frac{\exp\{-\theta\, \tilde{d}(\mathbf{x}_i, \mathbf{x}^*)\}}{\sum_{j=1}^{k} \exp\{-\theta\, \tilde{d}(\mathbf{x}_j, \mathbf{x}^*)\}}$$

Plot those trajectories (weighted)

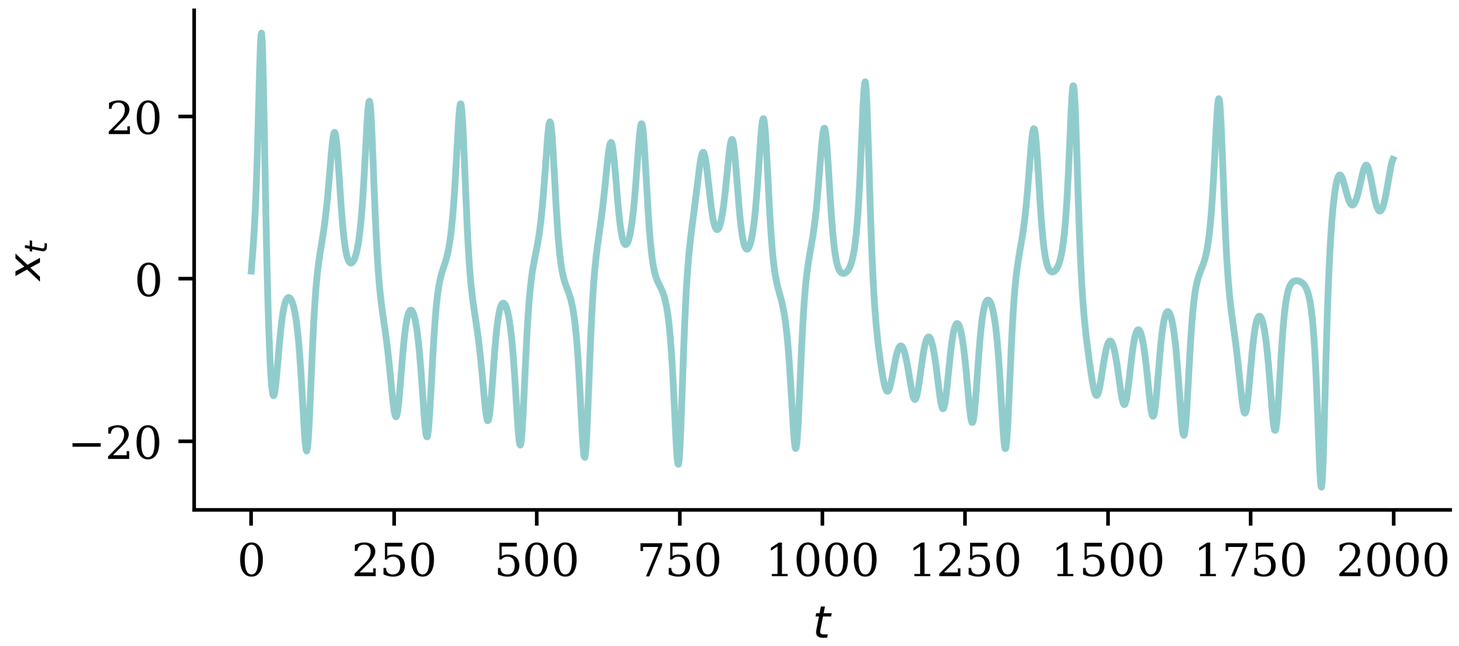


S-map Algorithm

Sequential Locally Weighted Global Linear Maps

Lorenz system

Given x_t from the Lorenz system:



predict x_{t+10} given (x_t, x_{t-20}).

Time series to embedding

t	x_t
0	0.95
1	1.85
2	2.75
...	...
1998	14.52
1999	14.69

i	x_t	x_{t-20}
0	27.53	0.95
1	24.75	1.85
2	21.34	2.75
...	...	...
1768	-16.59	-4.76
1769	-17.32	-4.90

i	x_{t+10}
0	-4.75
1	-6.82
2	-8.63
...	...
1768	-15.64
1769	-14.56

Calculate normalised distances

i	x_t	x_{t-20}	x_{t+10}	$d(\mathbf{x}_i, \mathbf{x}^*)$	$\hat{d}(\mathbf{x}_i, \mathbf{x}^*)$
0	27.53	0.95	-4.75	18.32	0.91
1	24.75	1.85	-6.82	15.45	0.77
2	21.34	2.75	-8.63	12.06	0.60
...	...	...	...	...	...
1768	-16.59	-4.76	-15.64	31.18	1.55
1769	-17.32	-4.90	-14.56	31.90	1.58

The average of the distances is:

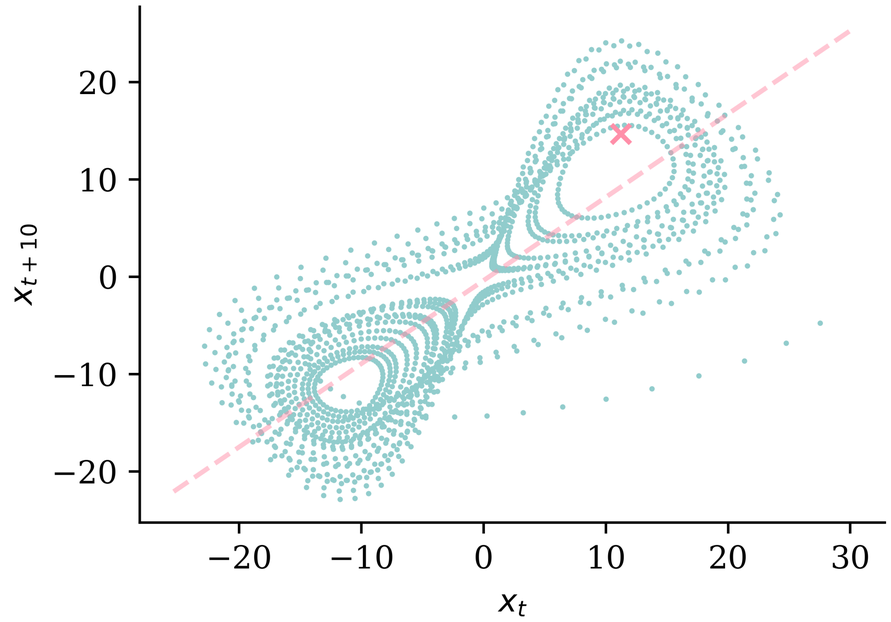
```
20.13
```

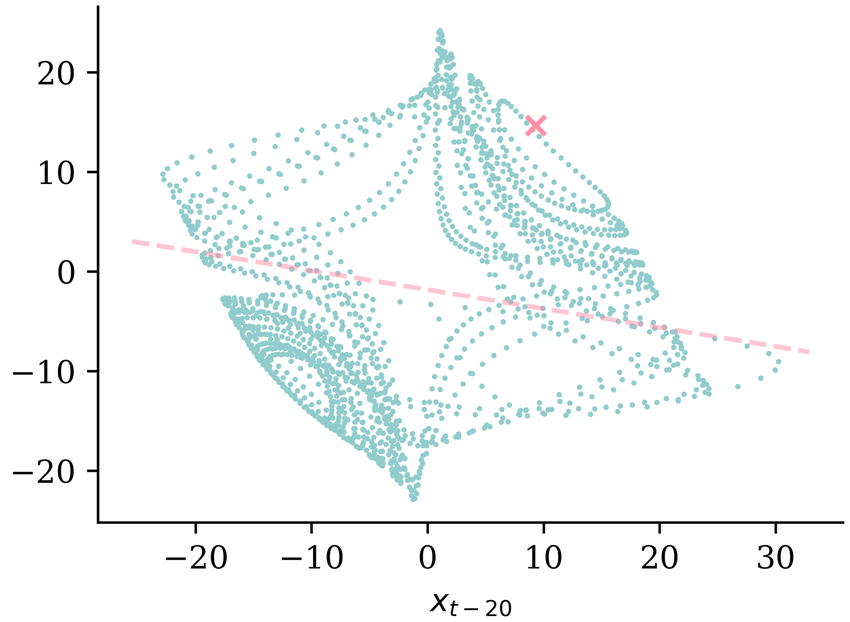
Weights

i	x_t	x_{t-20}	$d(\mathbf{x}_i, \mathbf{x}^*)$	w_i
0	27.53	0.95	18.32	0.01
1	24.75	1.85	15.45	0.02
2	21.34	2.75	12.06	0.05
...	...	...	...	...
1768	-16.59	-4.76	31.18	0.00
1769	-17.32	-4.90	31.90	0.00

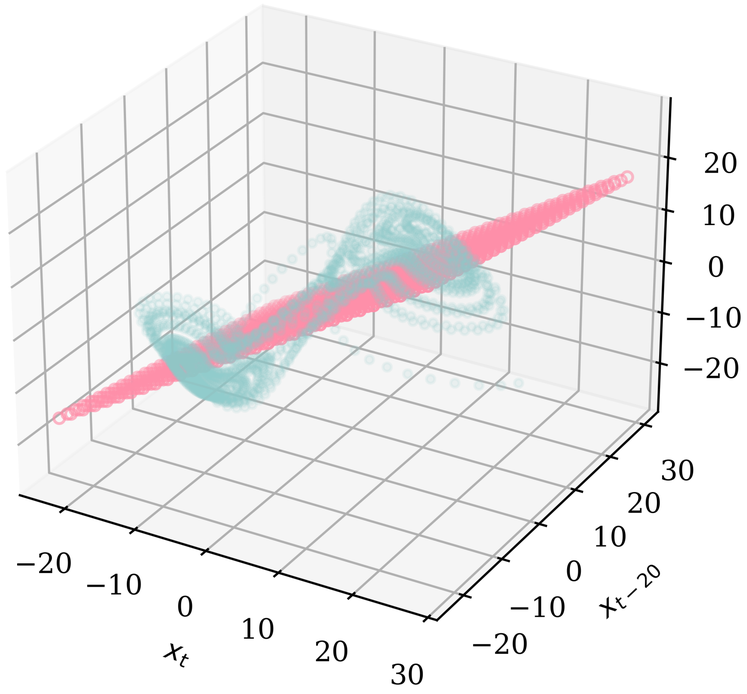
$$w_i = \exp\left\{-\theta\, \hat{d}\left(\mathbf{x}_i, \mathbf{x}^*\right)\right\}$$

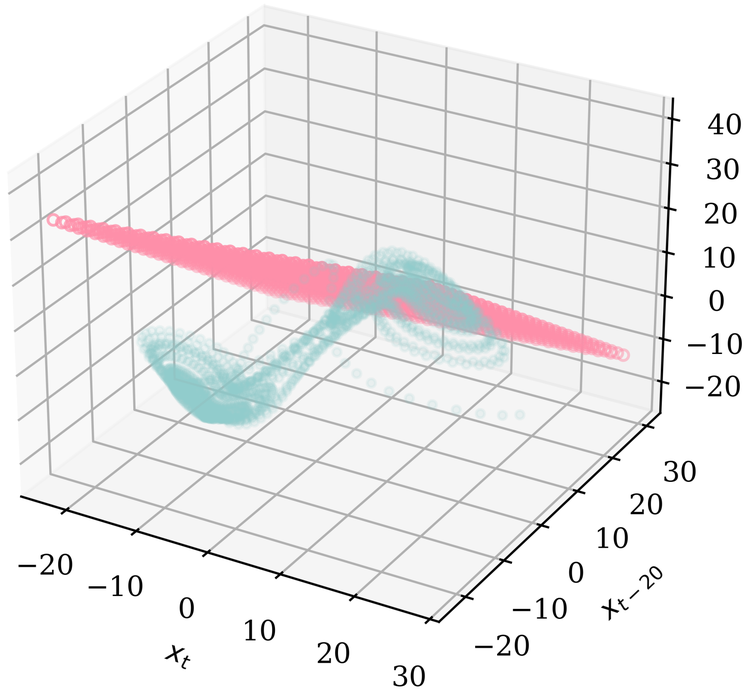
Linear auto-regression



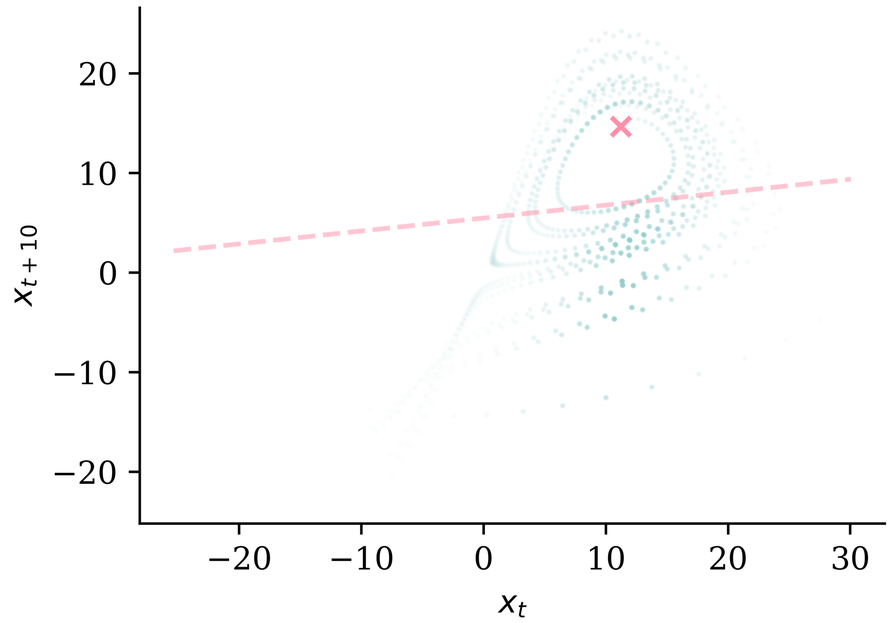


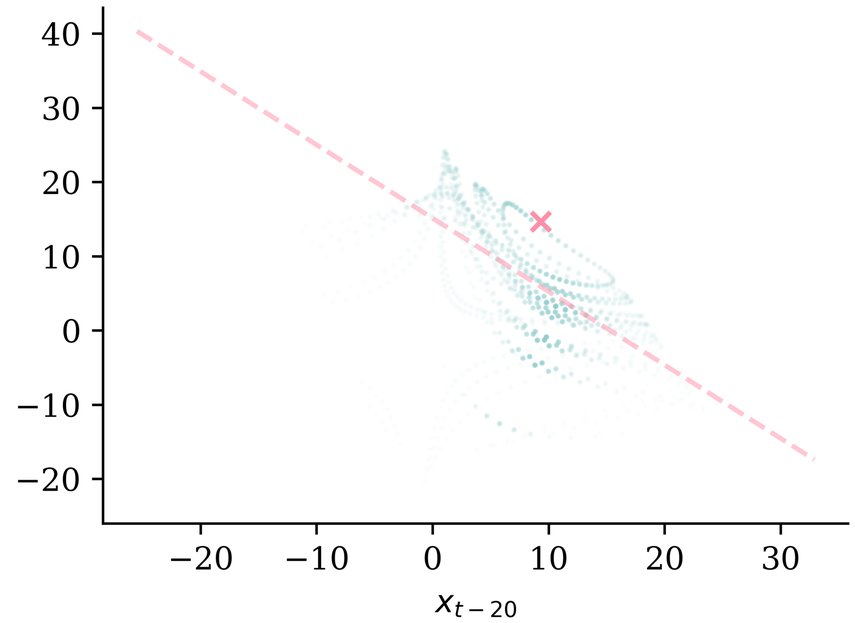
Linear regression plane





Weighted linear regression





Empirical Dynamic Modelling (EDM)

Create lagged embeddings

Given two time series, create E-length trajectories

$$\mathbf{x}_t = (\text{Temp}_t, \text{Temp}_{t-1}, \ldots, \text{Temp}_{t-(E-1)}) \in \mathbb{R}^E$$

and targets

$$y_t = \text{Crime}_t.$$

Note

The $\mathbf{x}_t$'s are called *points* (on the shadow manifold).

Split the data

- $\mathcal{L} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ is *library set,*
- $\mathcal{P} = \{(\mathbf{x}_{n+1}, y_{n+1}), \ldots, (\mathbf{x}_T, y_T)\}$ is *prediction set.*

For point $\mathbf{x}_s \in \mathcal{P}$, pretend we don't know y_s and try to predict it.

$$\forall\, \mathbf{x} \in \mathcal{L} \quad \text{find} \quad d(\mathbf{x}_s, \mathbf{x})$$

This is computationally demanding.

Convergent cross mapping

- If Temp_t causes Crime_t, then information about Temp_t is somehow embedded in Crime_t.
- By observing Crime_t, we should be able to forecast Temp_t.
- By observing more of Crime_t (more "training data"), our forecasts of Temp_t should be more accurate.

Example: Chicago crime and temperature.

Software

Stata package

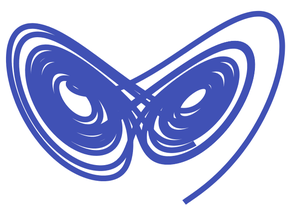
Empirical Dynamic Modeling Stata Package

Package Description

Empirical Dynamic Modeling (EDM) is a way to perform *causal analysis on time series data*. The `edm` Stata package implements a series of EDM tools, including the convergent cross-mapping algorithm.

Key features of the package:

- powered by a fast multi-threaded *C++ backend*,
- able to process panel data, a.k.a. *multispatial EDM*,
- able to handle *missing data* using new `dt` algorithms or by dropping points,
- *factor variables* can be added to the analysis,
- *multiple distance functions* available (Euclidean, Mean Absolute Error, Wasserstein),
- *GPU acceleration* available.



R package

Thanks to Rishi Dhushiyandan for his hard work on `easy_edm`.

fastEDM 0.1 Reference Articles Changelog Search for

fastEDM

The `fastEDM` R package implements a series of *Empirical Dynamic Modeling* tools that can be used for *causal analysis of time series* data.

Key features of the package:

- powered by a fast multi-threaded *C++ backend*,
- able to process panel data, a.k.a. *multispatial EDM*,
- able to handle *missing data* using new `dt` algorithms or by dropping points.

Installation

You can install the development version of fastEDM from GitHub with:

License

MIT + file LICENSE

Citation

Citing fastEDM

Developers

Patrick Laub
Author, maintainer

Jinjing Li
Author

Michael Zyphur
Author

More about authors...

Python package

fastEDM Python Package

Package Description

Empirical Dynamic Modeling (EDM) is a way to perform *causal analysis on time series data*. The `fastEDM` Python package implements a series of EDM tools, including the convergent cross-mapping algorithm.

Key features of the package:

- powered by a fast multi-threaded *C++ backend*,
- able to process panel data, a.k.a. *multispatial EDM*,
- able to handle *missing data* using new `dt` algorithms or by dropping points.

Installation

To install the latest version from Github using `pip` run:

Modern engineering

- Open code (9,745 LOC) on MIT License,
- unit & integration tests (5,342 LOC),
- documentation (5,042 LOC),
- Git (1,198 commits),
- Github Actions (11 tasks),
- vectorised, microbenchmarking, ASAN, linting,
- all C++ compilers, WASM, all OSs.

Profiling

> “Premature optimization is the root of all evil.” Donald Knuth

> “… one of the key things is always measure, never trust your instincts because no matter how many years you’ve been doing this, and I’ve been doing this for nearly three decades on and off, you’re always wrong.” Matt Godbolt

Random sums example

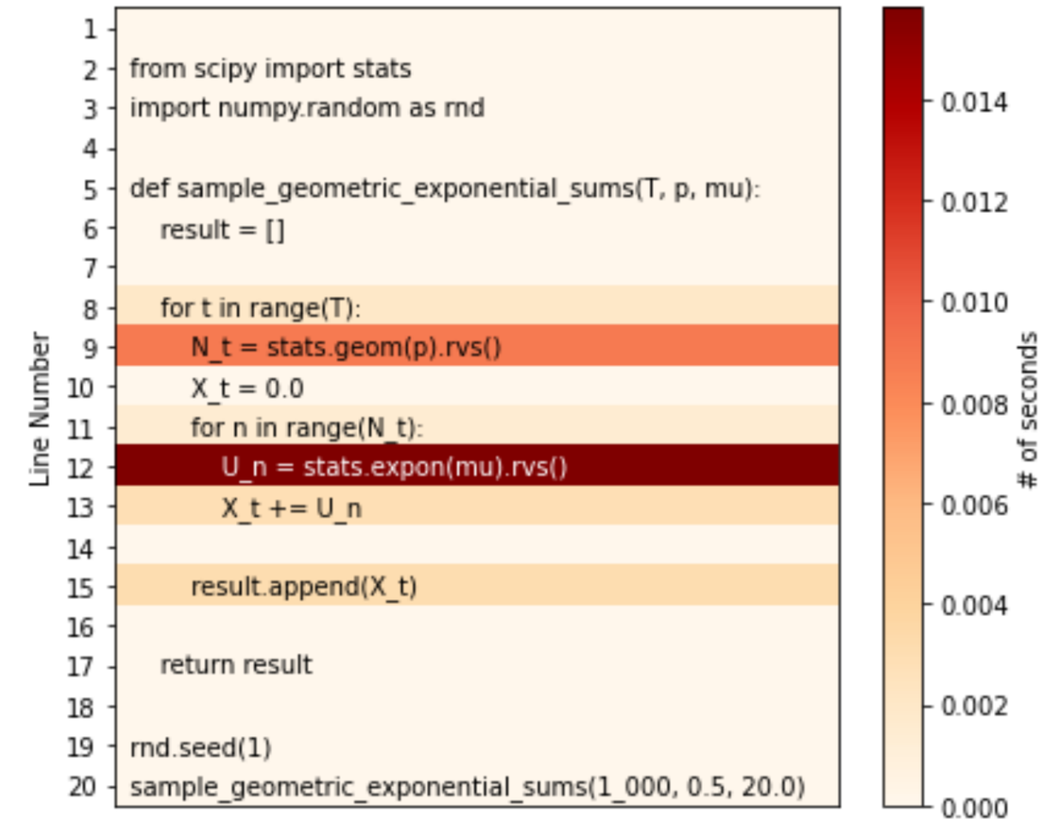
$$U_n \sim \mathsf{Exp}(\mu) \qquad N_t \sim \mathsf{Poisson}(\lambda)$$

$$X_t = \sum_{n=1}^{N_t} U_n \qquad \mathbf{X} = (X_t)_{t=1,\ldots,T} \qquad \text{Need } R \approx 10^7$$

```
def sample_geometric_exponential_sums(T, p, μ):
  result = []

  for t in range(T):
    N_t = stats.geom(p).rvs()
    X_t = 0.0
    for n in range(N_t):
      U_n = stats.expon(μ).rvs()
      X_t += U_n

    result.append(X_t)

  return np.array(result)

sample_geometric_exponential_sums(1_000, 0.5, 20.0)
```

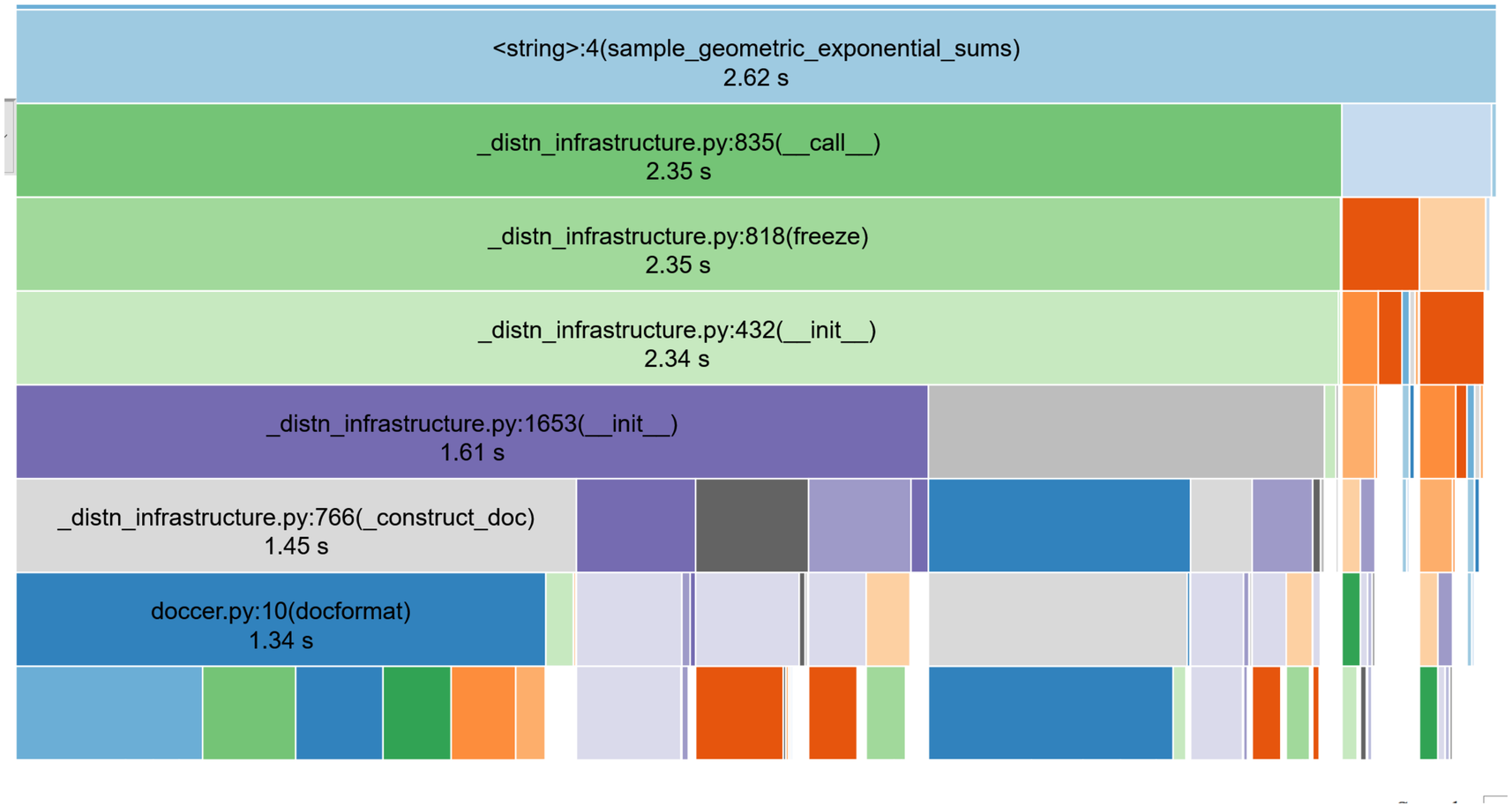
```
array([21.27, 20.36, 20.1 , ..., 87.  , 20.46, 20.59])
```

Line profiler



Line profiler

“3D” profiler (snakeviz)



Snakeviz

Timing

```
def sample_geometric_exponential_sums(T, p, μ):
  result = []

  for t in range(T):
    N_t = stats.geom(p).rvs()
    X_t = 0.0
    for n in range(N_t):
      U_n = stats.expon(μ).rvs()
      X_t += U_n

    result.append(X_t)

  return result

rnd.seed(1)
%time rvs = sample_geometric_exponential_sums(1_000, 0.5, 20.0)
```

```
CPU times: total: 547 ms
Wall time: 1.21 s
```

Timing

```
def sample_geometric_exponential_sums_alternative(T, p, μ):
  result = []

  for t in range(T):
    N_t = stats.geom.rvs(p)
    X_t = 0.0
    for n in range(N_t):
      U_n = stats.expon.rvs(μ)
      X_t += U_n

    result.append(X_t)

  return result

rnd.seed(1)
%time rvs = sample_geometric_exponential_sums_alternative(1_000, 0.5, 20.0)
```

```
CPU times: total: 31.2 ms
Wall time: 48 ms
```

Speedup: 24.91x

```
N_t = stats.geom(p).rvs()
```

```
N_t = stats.geom.rvs(p)
```

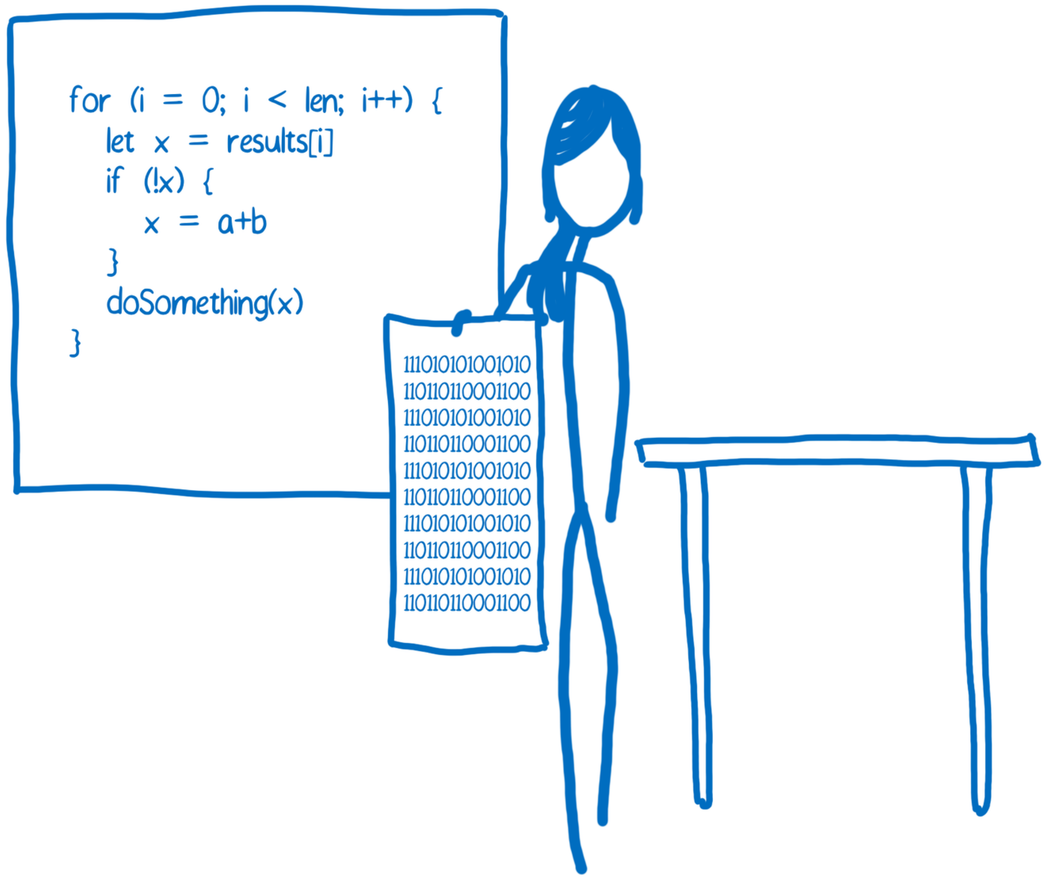
Compiled code

Talking to a computer

Arrival

Source: Lin Clark (2017), A crash course in JIT compilers, Mozilla blog.

Interpreters & compilers

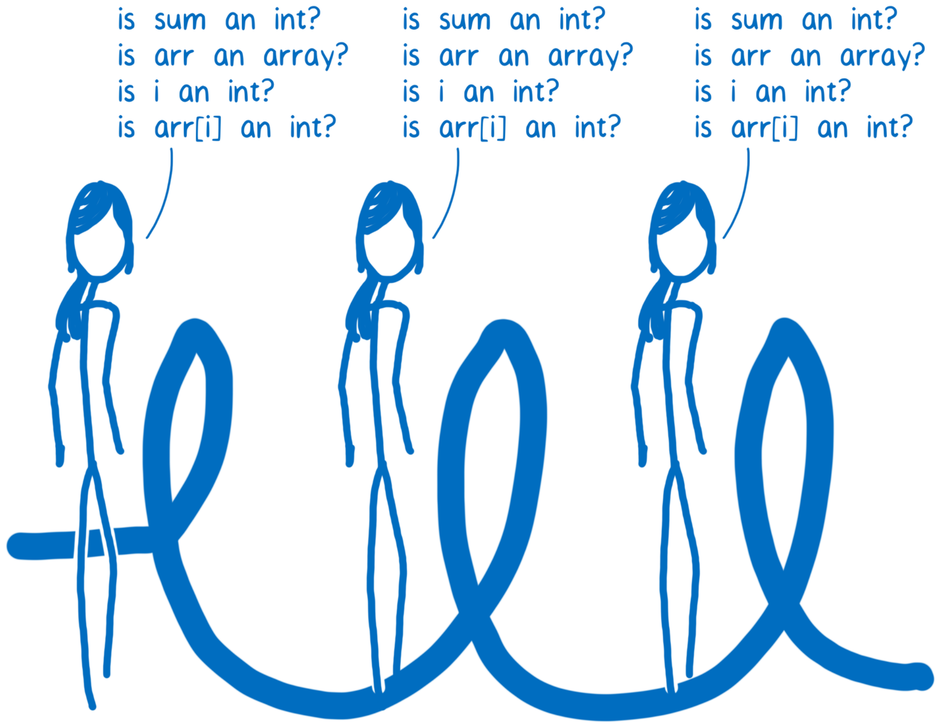


Source: Lin Clark (2017), A crash course in JIT compilers, Mozilla blog.

Type checking

```
function arraySum(arr) {
  sum = 0
  for (i in 1:length(arr)) {
    sum = sum + arr[i]
  }
  return(sum)
}
```

Compiler Explorer:
https://godbolt.org/z/dTv78M



Type checking

Theoretical limits

Amdahl's Law (or Law of Diminishing Returns)

P is the proportion of a program that can be made parallel
$1 - P$ remains serial.

Theoretical maximum speedup of achieved by N processor is

$$S(N) = 1/[(1 - P) + P/N]$$

Hence with unlimited processor count

$$S(\infty) = 1/(1 - P)$$

E.g. $P = 0.9$ then $S(\infty) = 10$.

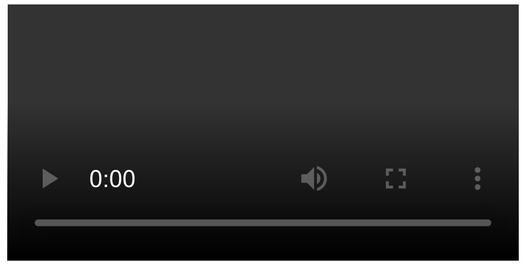
Get involved!

😊 Give it a try, feedback would be very welcome.

😍 If you're talented in causal inference or programming (Stata/Mata, R, Javascript, C++, Python), we'd love contributions!

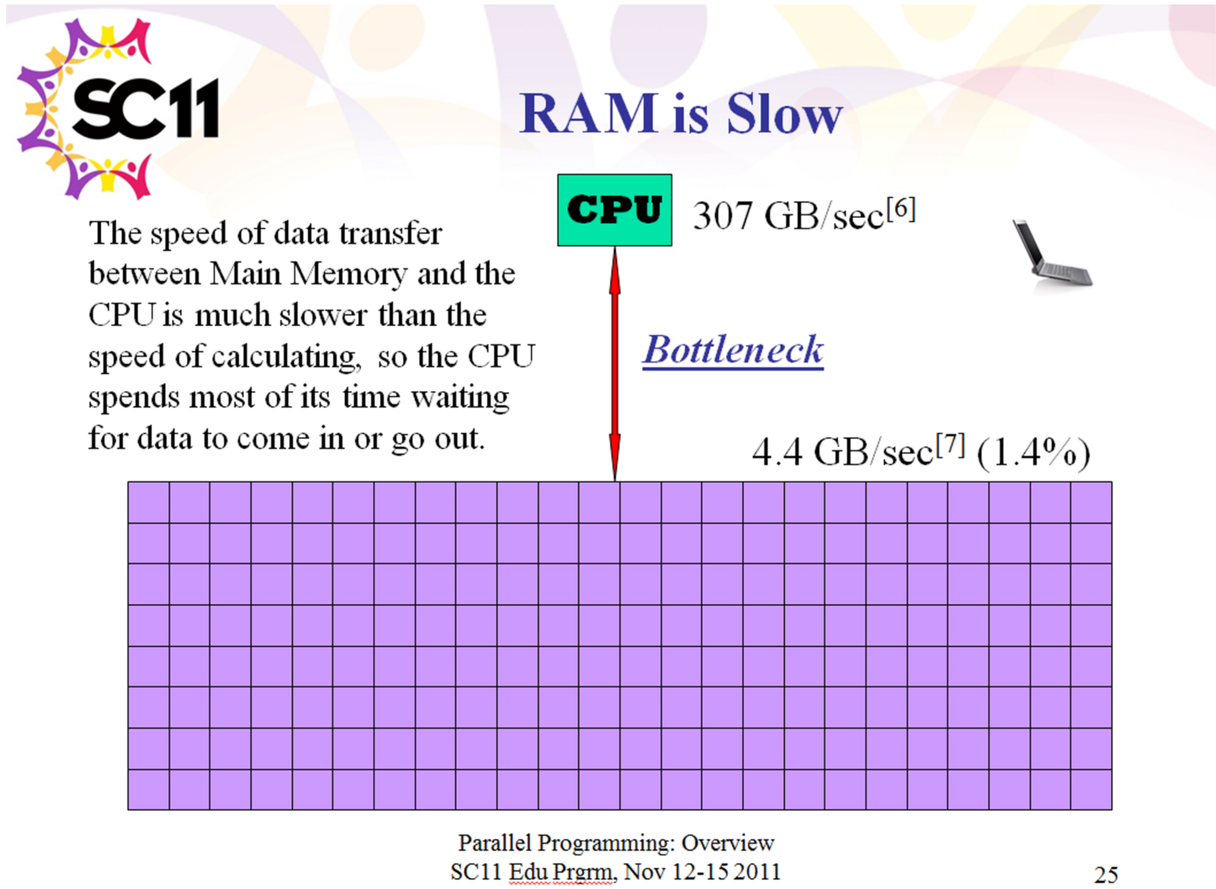
Appendix

My computer



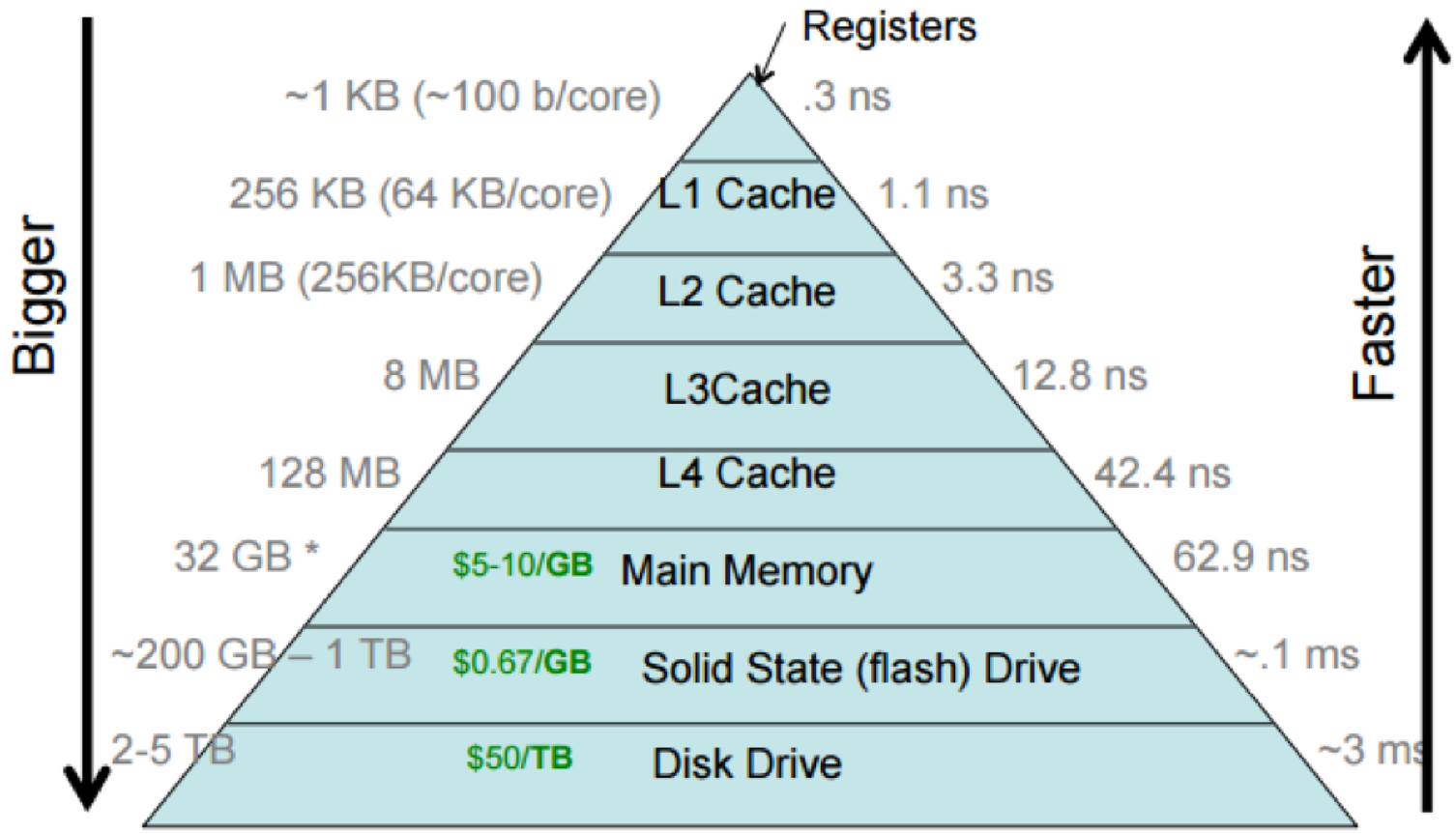
My work machine (for deep learning experiments).

Speeds of computing



CPU vs RAM speeds.

Cache system



Hierarchy of caches

Impact on matrix order

```
n = 10_000
matrix = np.array(range(n*n)).reshape(n, n)
```

```
%%time
col_sums = np.zeros(n)
for j in range(n):
  col_sums += matrix[:, j]
```

```
CPU times: total: 281 ms
Wall time: 721 ms
```

```
%%time
row_sums = np.zeros(n)
for i in range(n):
  row_sums += matrix[i, :]
```

```
CPU times: total: 15.6 ms
Wall time: 63.7 ms
```

Compiled code

What does a machine understand?

1100 0111 01000101 1111 1000 0101 1111 1000 0000 0001 0000 000 0000 0000 0000 0000 0000 0000 0000

function arg1 arg2

```
movl $0x1, -0x8(%rbp)
```

```
int x = 1;
```

Instruction set architecture (ISA), e.g. x86-64

Intel i5 (2010): add (0101), subtract (1111), …

Intel i7 (2020): add (0101), subtract (1111), add_vectors (1011), …

Apple M1 (2020): add (100), subtract (111), …

Try Compiler Explorer https://godbolt.org/z/dTv78M

```
def sample_geometric_exponential_sums(T, p, μ):
  X = np.zeros(T)

  N = rnd.geometric(p, size=T)
  U = rnd.exponential(μ, size=N.sum())

  i = 0
  for t in range(T):
    X[t] = U[i:i+N[t]].sum()
    i += N[t]

  return X
```

Interpreted version

```
cpdef sample_geometric_exponential_sums(long T, double p, double mu):
  cdef double[:] X = np.zeros(T)
  cdef long[:] N = rnd.geometric(p, size=T)
  cdef double[:] U = rnd.exponential(mu, size=np.sum(N))

  cdef int i = 0
  cdef int t
  for t in range(T):
    X[t] = np.sum(U[i:i+N[t]])
    i += N[t]
    return X
```

Compiled by *Cython* (not recommended)

```
def sample_geometric_exponential_sums(T, p, μ):
  X = np.zeros(T)

  N = rnd.geometric(p, size=T)
  U = rnd.exponential(μ, size=N.sum())

  i = 0
  for t in range(T):
    X[t] = U[i:i+N[t]].sum()
    i += N[t]

  return X
```

Interpreted version

```
from numba import njit

@njit()
def sample_geometric_exponential_sums(T, p, μ):
  X = np.zeros(T)

  N = rnd.geometric(p, size=T)
  U = rnd.exponential(μ, size=N.sum())

  i = 0
  for t in range(T):
    X[t] = U[i:i+N[t]].sum()
    i += N[t]

  return X
```

First run is compiling (500 ms), but after we are down from 2.7 ms to 164 µs (16x speedup)

Compiled by *numba*

Original code: 1.7 s

Basic profiling with **snakeviz**: 5.5 ms, 310x speedup

- Vectorisation/preallocation with **numpy**: 2.7 ms, 630x speedup
- Compilation with **numba**: 164 μs, 10,000x speedup

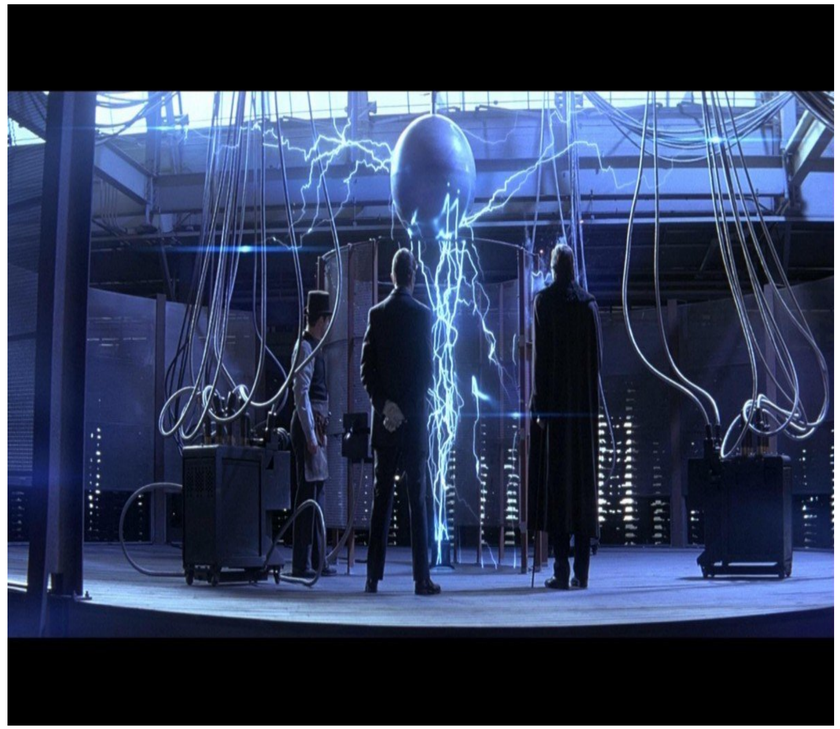
And potentially:

- Parallel over 80 cores: say another 50x improvement, so overall 50,000x speedup.

Parallel code

Threads or processes?

Use multiple processes in Python, probably should use multiple threads elsewhere.

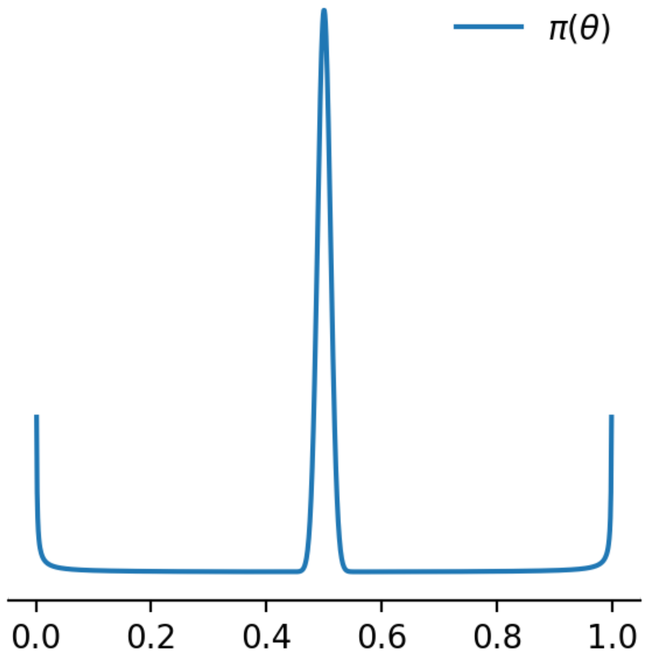


Baumann et al. (2019), A fork() in the road 17th Workshop on Hot Topics in Operating Systems

```cpp
#include <iostream>
#include <thread>
#include <vector>

int count = 0;

int numThreads = 1000;
int numIncrements = 1000;

void increase_count() {
        for (int i = 0; i < numIncrements; i++) {
                count += 1;
        }
}

int main() {

        std::vector<std::thread> threads;
        for (int t = 0; t < numThreads; t++) {
                threads.push_back(std::thread(increase_count));
        }

        for (auto& t : threads) {
                t.join();
        }
        std::cout << "Count is " << count;
        std::cout << ", expected " << numThreads*numIncrements << std::endl;

        return 0;
}
                                                                16,0-1        Bot
```

```
(base) plaub@mrc ~/thread-fail-test c++ test.cpp -std=c++20 -lpthread -o test
(base) plaub@mrc ~/thread-fail-test ./test
Count is 1000000, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 996867, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 999231, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 1000000, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 998393, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 997000, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 998895, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 994588, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 999565, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 1000000, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 996976, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 1000000, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 999413, expected 1000000
(base) plaub@mrc ~/thread-fail-test ./test
Count is 998327, expected 1000000
(base) plaub@mrc ~/thread-fail-test
```

Bayesian approach



Prior