Approximate Bayesian Computation in Insurance

Patrick J. Laub and Pierre-Olivier Goffard

THE UNIVERSITY OF MELBOURNE

LABORATOIRE SAF
SCIENCES ACTUARIELLE & FINANCIÈRE

Motivation

Have a random number of claims $N \sim p_N(\,\cdot\,; \boldsymbol{\theta}_{\mathrm{freq}})$ and the claim sizes $U_1, \dots, U_N \sim f_U(\,\cdot\,; \boldsymbol{\theta}_{\mathrm{sev}})$.

We aggregate them somehow, like:

- aggregate claims: $X = \sum_{i=1}^{N} U_i$
- maximum claims: $X = \max_{i=1}^{N} U_i$
- stop-loss: $X = (\sum_{i=1}^{N} U_i - c)_+$.

Question: Given a sample $X_1, \dots, X_n$ of the summaries, what is the $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\mathrm{freq}}, \boldsymbol{\theta}_{\mathrm{sev}})$ which explains them?

E.g. a reinsurance contract

Likelihoods

For simple rv's we know their likelihood (normal, exponential, gamma, etc.).

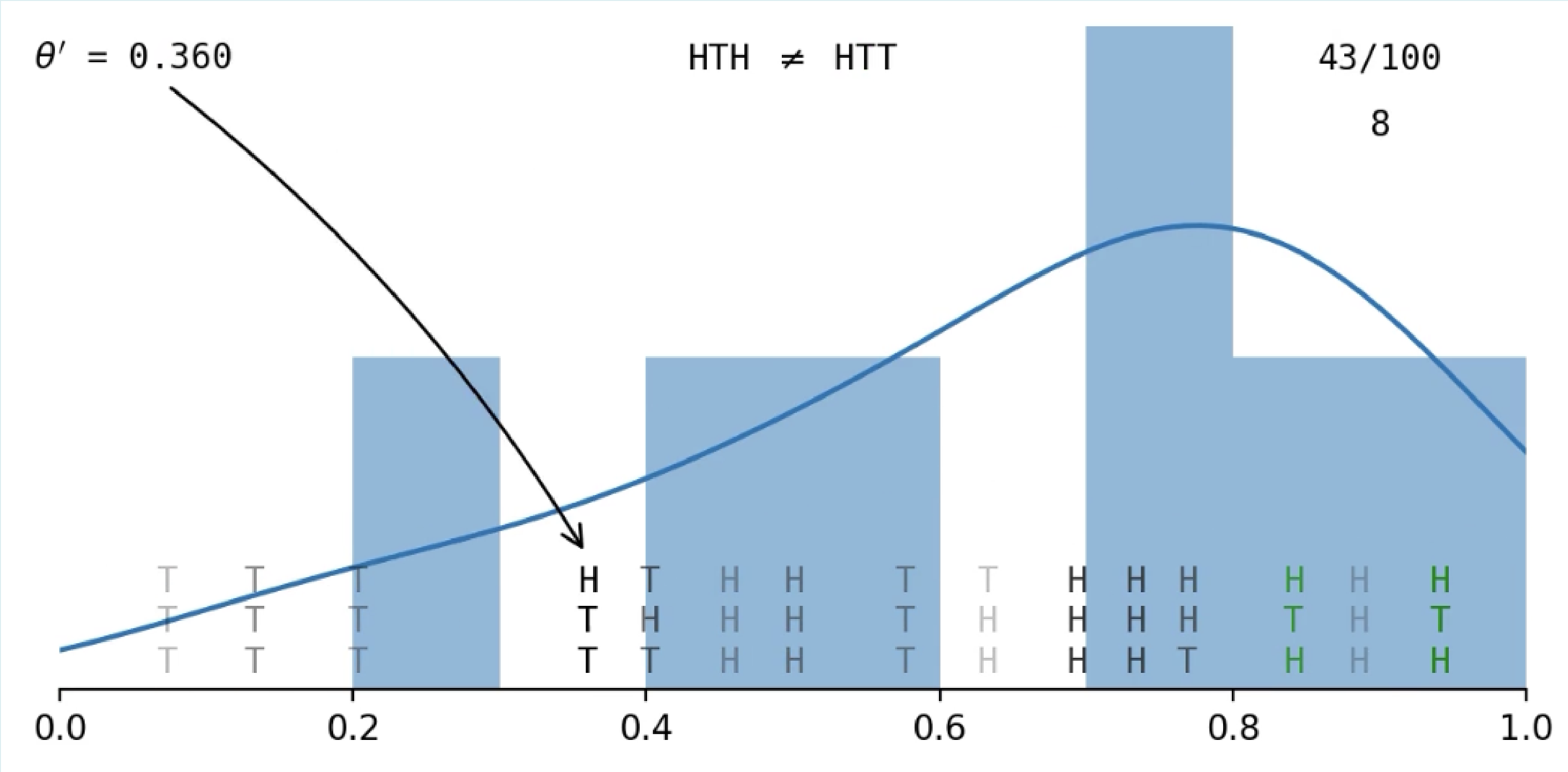
When simple rv's are combined, the resulting thing rarely has a tractable likelihood.

$$X_1, X_2 \overset{\text{i.i.d.}}{\sim} f_X(\,\cdot\,) \Rightarrow X_1 + X_2 \sim \texttt{\textbf{Intractable likelihood!}}$$

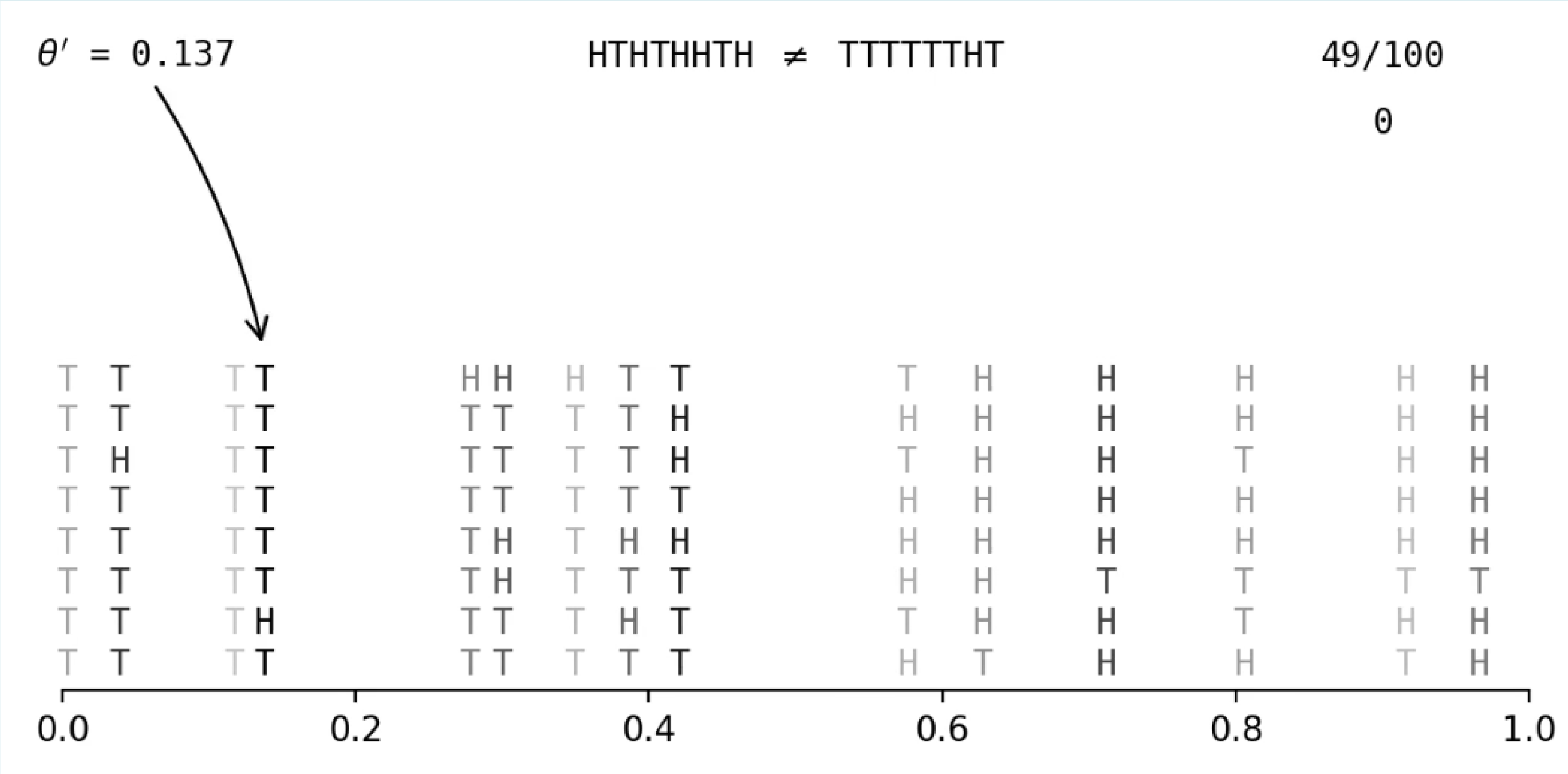
Usually it's still possible to simulate these things...

Approximate Bayesian Computation

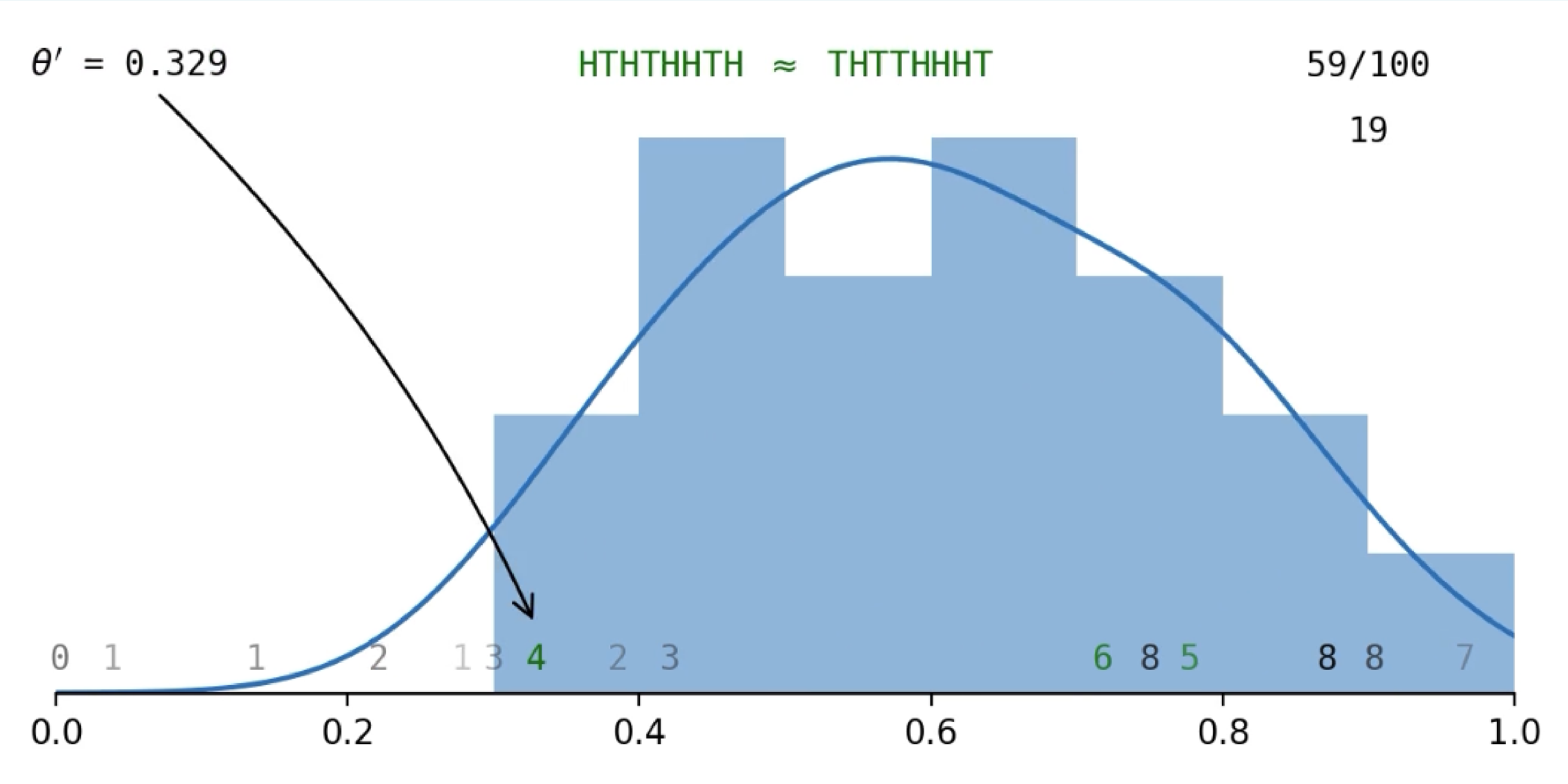
Example: Flip a coin a few times and get $(x_1, x_2, x_3) = (\mathrm{H}, \mathrm{T}, \mathrm{H})$; what is $\pi(\theta|\boldsymbol{x})$?



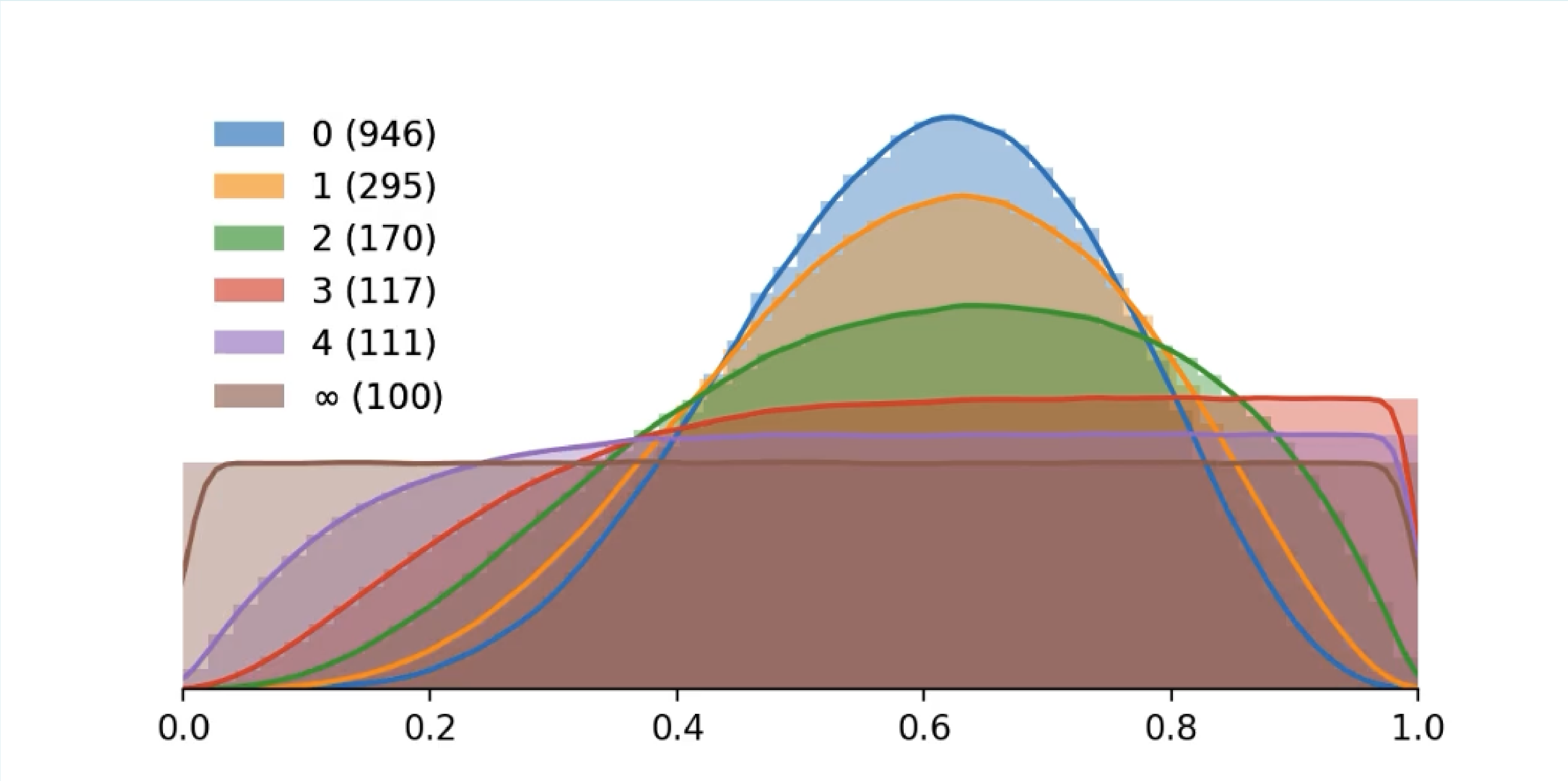
Getting an exact match of the data is hard...



Accept fake data that's close to observed data



The 'approximate' part of ABC



Does it work in theory?

We sample the *approximate / ABC posterior*

$$\pi_\epsilon(\boldsymbol{\theta} \mid \boldsymbol{x}_{\text{obs}}) \propto \pi(\theta) \times \mathbb{P}\big(\|\boldsymbol{x}_{\text{obs}} - \boldsymbol{x}^*\| \leq \epsilon \text{ where } \boldsymbol{x}^* \sim \boldsymbol{\theta}\big),$$

but we care about the true posterior $\pi(\boldsymbol{\theta} \mid \boldsymbol{x}_{\text{obs}})$.

Proposition: Say we have continuous data $\boldsymbol{x}_{\text{obs}}$, and our prior $\pi(\boldsymbol{\theta})$ has bounded support.

If for some $\epsilon > 0$ we have

$$\sup_{(\boldsymbol{z},\boldsymbol{\theta}) : \mathcal{D}(\boldsymbol{z},\boldsymbol{x}_{\text{obs}}) < \epsilon, \boldsymbol{\theta} \in \boldsymbol{\Theta}} \pi(\boldsymbol{z} \mid \boldsymbol{\theta}) < \infty$$

then for each $\boldsymbol{\theta} \in \boldsymbol{\Theta}$

$$\lim_{\epsilon \to 0} \pi_\epsilon(\boldsymbol{\theta} \mid \boldsymbol{x}_{\text{obs}}) = \pi(\boldsymbol{\theta} \mid \boldsymbol{x}_{\text{obs}}) \,.$$

Rubio and Johansen (2013), *A simple approach to maximum intractable likelihood estimation,* Electronic Journal of Statistics.

Mixed data

We filled in some of the blanks for *mixed data.*

Our data was mostly continuous data but had an atom at 0.

Get

$$\lim_{\epsilon \to 0} \pi_\epsilon(\boldsymbol{\theta} \mid \boldsymbol{x}_{\text{obs}}) = \pi(\boldsymbol{\theta} \mid \boldsymbol{x}_{\text{obs}})$$

when

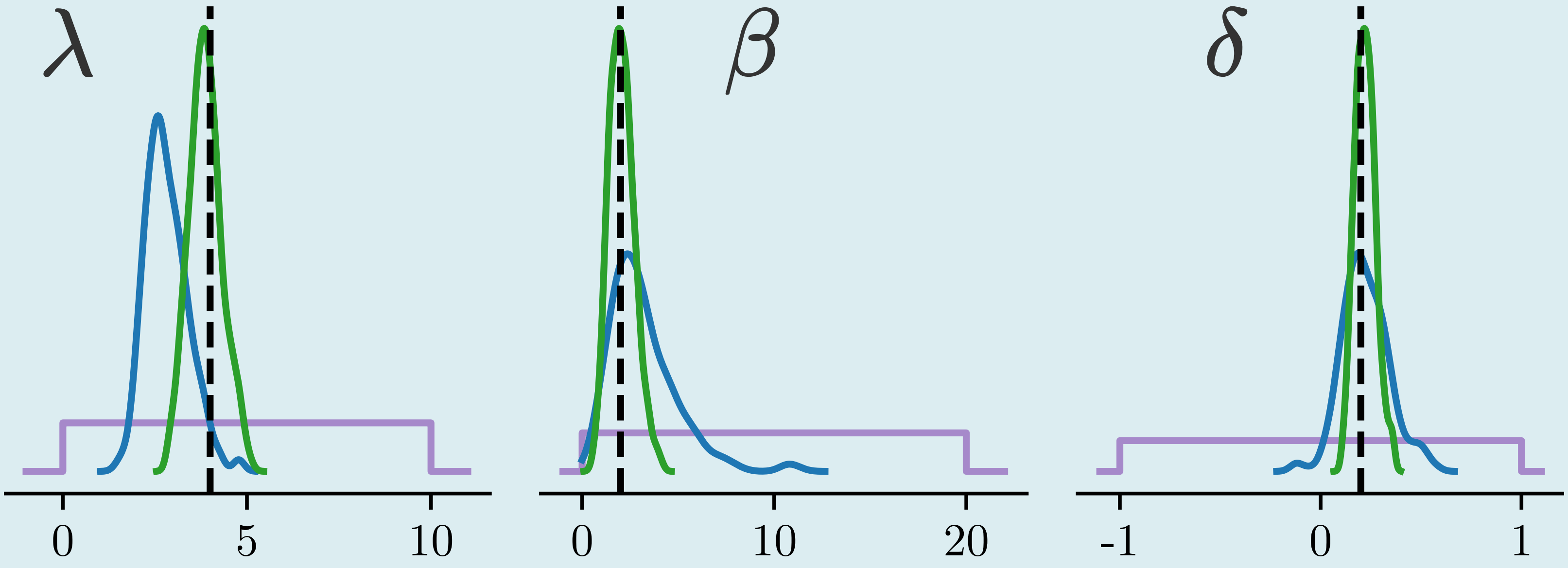
$$\mathcal{D}(\boldsymbol{z}, \boldsymbol{x}_{\text{obs}}) = \begin{cases} \mathcal{D}^+(\boldsymbol{z}^+, \boldsymbol{x}_{\text{obs}^+}) & \text{if } \#\text{Zeros}(\boldsymbol{z}) = \#\text{Zeros}(\boldsymbol{x}_{\text{obs}}), \\ \infty & \text{otherwise.} \end{cases}$$

Does it work in practice?

In other words, when does it break & how slow is it?

Dependent example

- frequency $N \sim \text{Poisson}(\lambda = 4)$,
- severity $U_i \mid N \sim \text{DepExp}(\beta = 2, \delta = 0.2)$, defined as $U_i \mid N \sim \text{Exp}(\beta \times \mathrm{e}^{\delta N})$,
- summation summary $X = \sum_{i=1}^{N} U_i$



ABC posteriors based on 50 X's and on 250 X's given uniform priors.

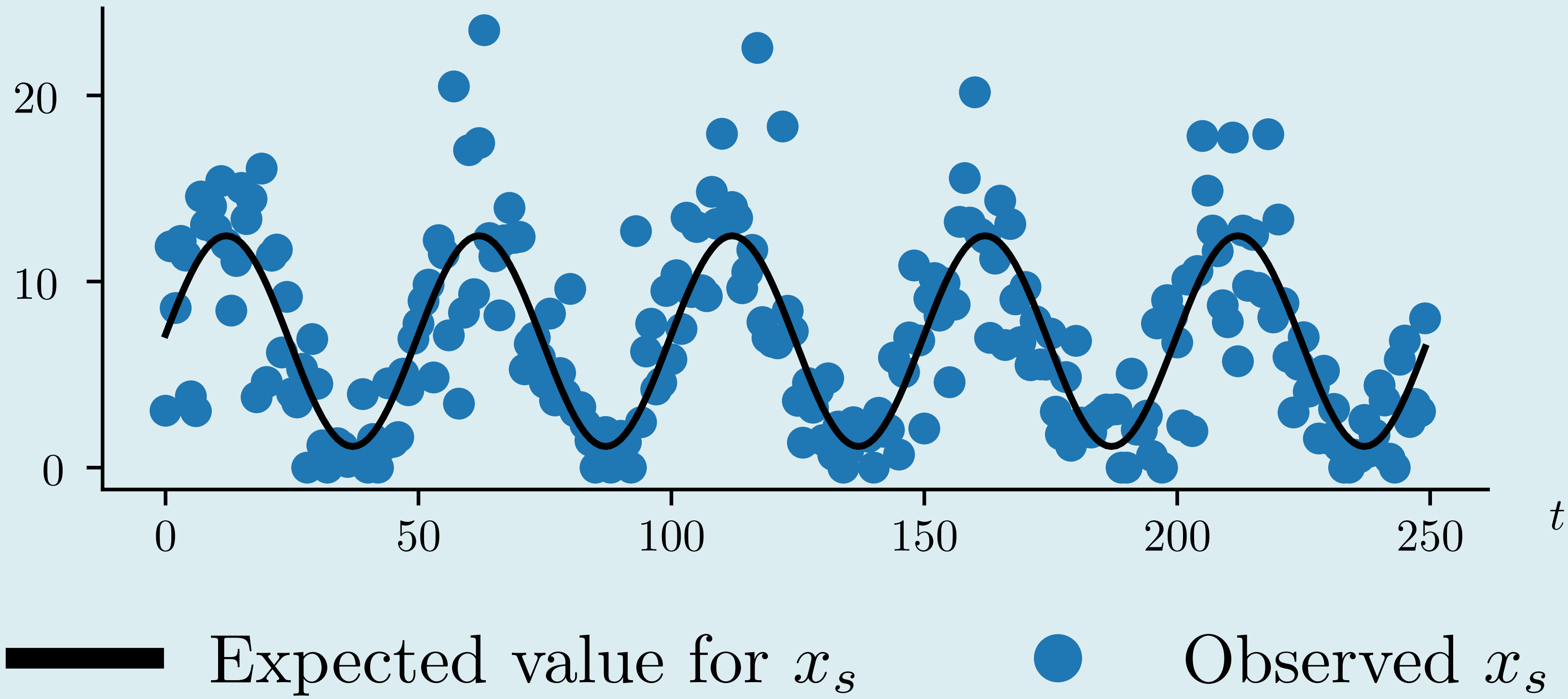
J. Garrido, C. Genest, and J. Schulz (2016), *Generalized linear models for dependent frequency and severity of insurance claims*, IME.

pip install approxbayescomp

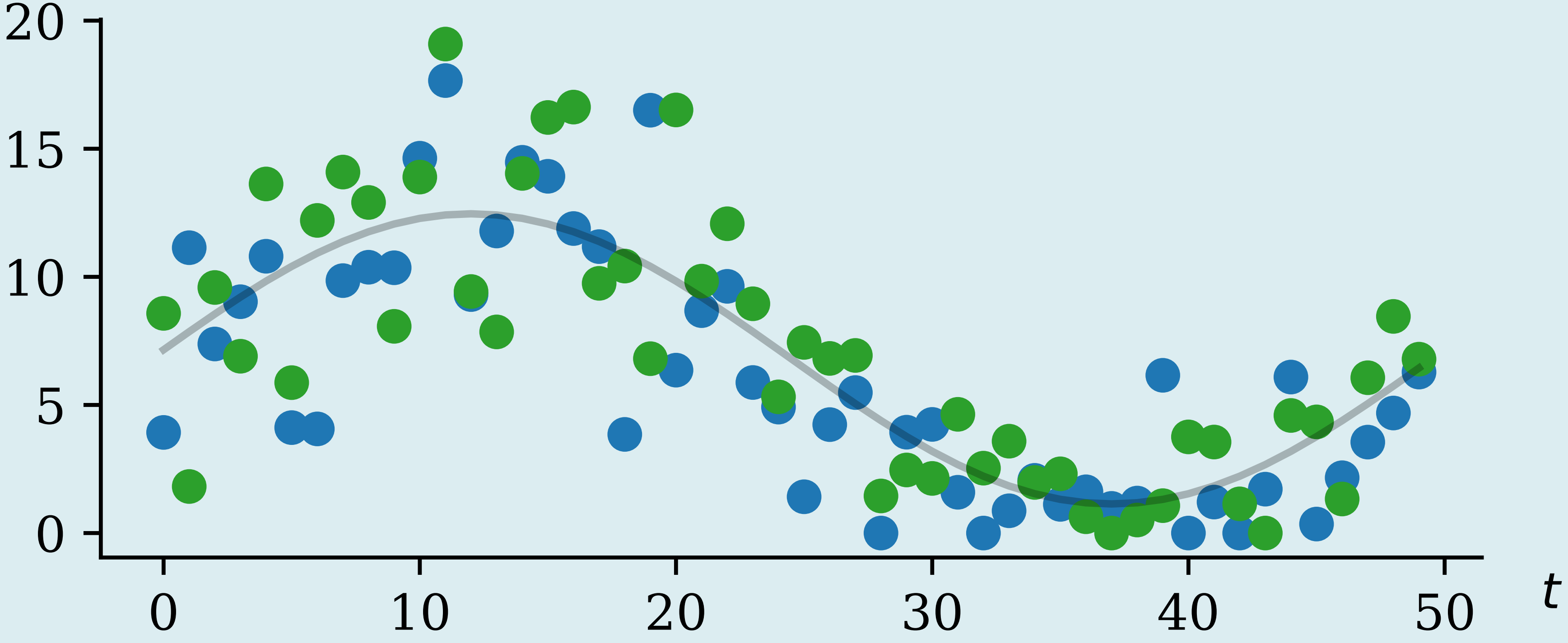
```python
import approxbayescomp as abc

# Load data to fit
obsData = ...

# Frequency-Loss Model
freq = "poisson"
sev = "frequency dependent exponential"
psi = abc.Psi("sum") # Aggregation process

# Fit the model to the data using ABC
prior = abc.IndependentUniformPrior([(0, 10), (0, 20), (-1, 1)])
model = abc.Model(freq, sev, psi, prior)
fit = abc.smc(numIters, popSize, obsData, model)
```

Time-varying example

- Claims form a Poisson process point with arrival rate $\lambda(t) = a + b[1 + \sin(2\pi ct)]$.
- The observations are $X_s = \sum_{i=N_{s-1}}^{N_s} U_i$.
- Frequencies are CPoisson$(a = 1, b = 5, c = \frac{1}{50})$ and sizes are $U_i \sim$ Lognormal$(\mu = 0, \sigma = 0.5)$.

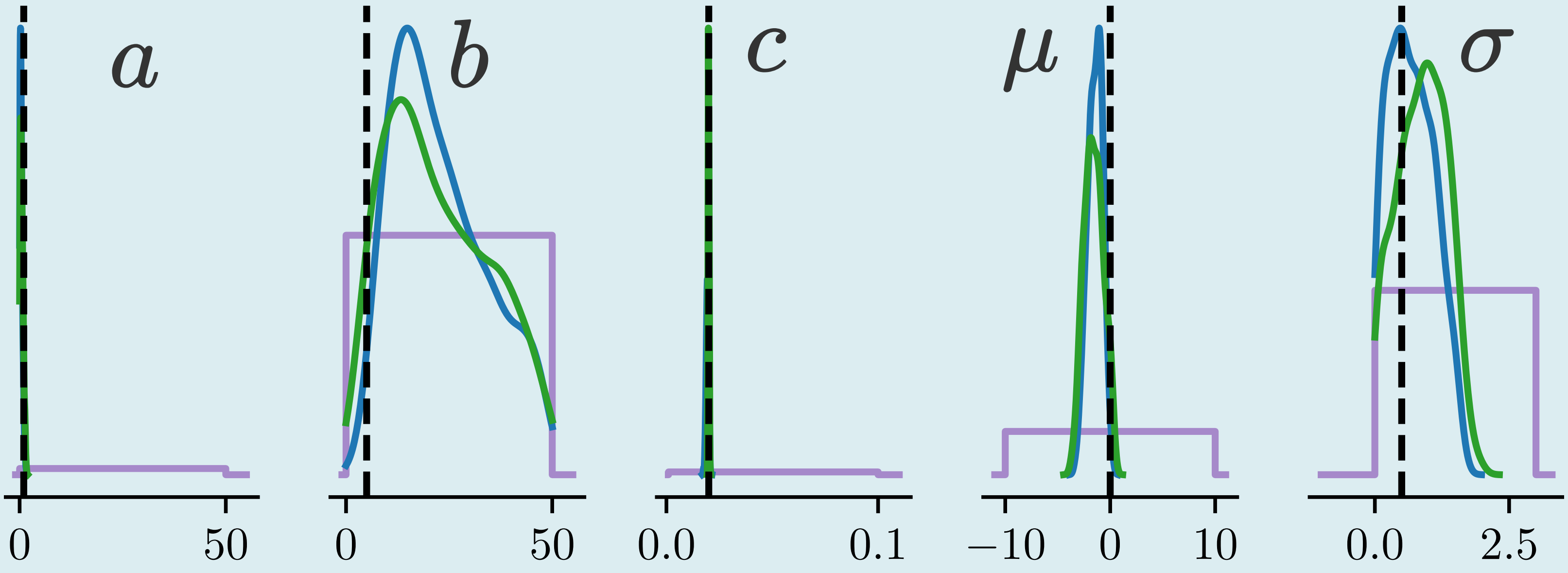


From the same $\boldsymbol{\theta}$ the data is **random**



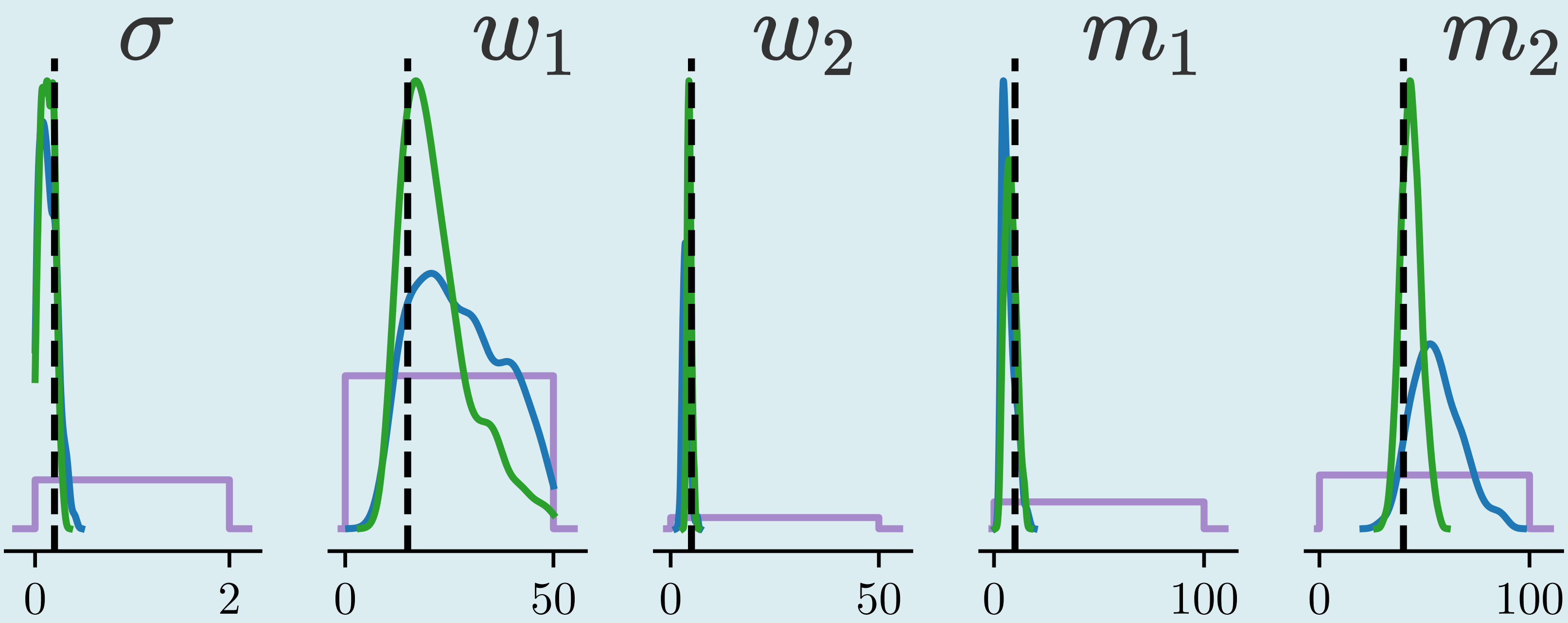
Time-varying example

- Claims form a Poisson process point with arrival rate $\lambda(t) = a + b[1 + \sin(2\pi c t)]$.
- The observations are $X_s = \sum_{i=N_{s-1}}^{N_s} U_i$.
- Frequencies are CPoisson$(a = 1, b = 5, c = \frac{1}{50})$ and sizes are $U_i \sim$ Lognormal$(\mu = 0, \sigma = 0.5)$.
- ABC posteriors based on 50 X's and on 250 X's with uniform priors.



Bivariate example

- Two lines of business with dependence in the claim frequencies.
- Say $\Lambda_i \sim$ **Lognormal**$(\mu \equiv 0, \sigma = 0.2)$.
- Claim frequencies $N_i \sim$ **Poisson**$(\Lambda_i w_1)$ and $M_i \sim$ **Poisson**$(\Lambda_i w_2)$ for $w_1 = 15$, $w_2 = 5$.
- Claim sizes for each line are **Exp**$(m_1 = 10)$ and **Exp**$(m_2 = 40)$.
- ABC posteriors based on 50 X's and on 250 X's with uniform priors.



Streftaris and Worton (2008), *Efficient and accurate approximate Bayesian inference with an application to insurance data*, Computational Statistics & Data Analysis

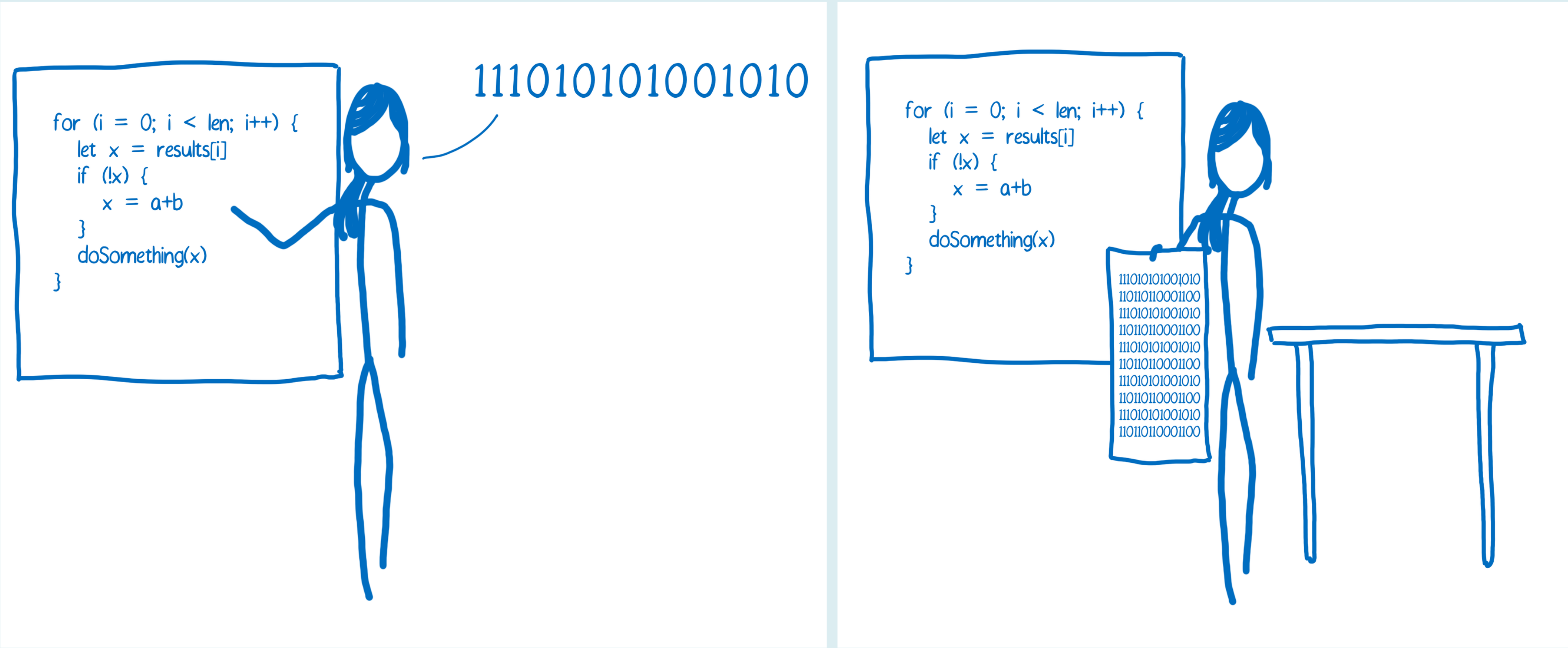
ABC turns a statistics problem into a programming problem

... to be used as a last resort

Compiled code

(use *numba*)

Interpreters & compilers



C.f. Lin Clark (2017), A crash course in JIT compilers

Interpreted version

```python
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
```

Compiled by *numba*

```python
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)

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.

Take-home messages

- What is ABC?

- ABC turns a *statistics problem* into a *programming problem*

- pip install approxbayescomp