Approximate Bayesian Computation and Insurance

Patrick Laub

UNSW Sydney, School of Risk and Actuarial Studies

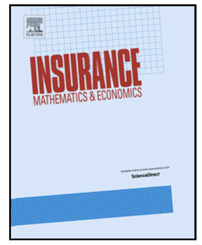
February 3, 2023

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

What is the bias?

We flip a coin three times and get:

Heads

Tails

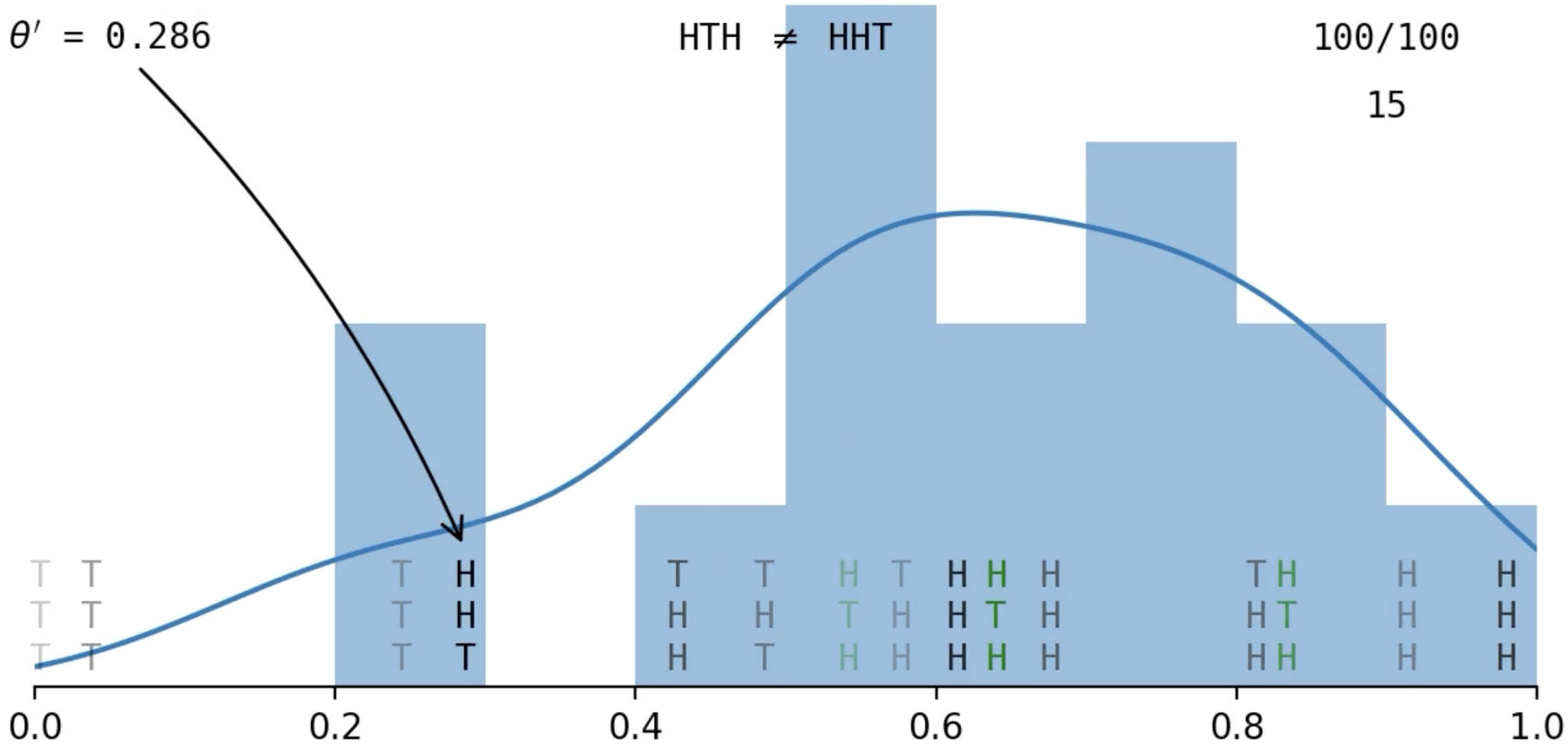
Heads

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

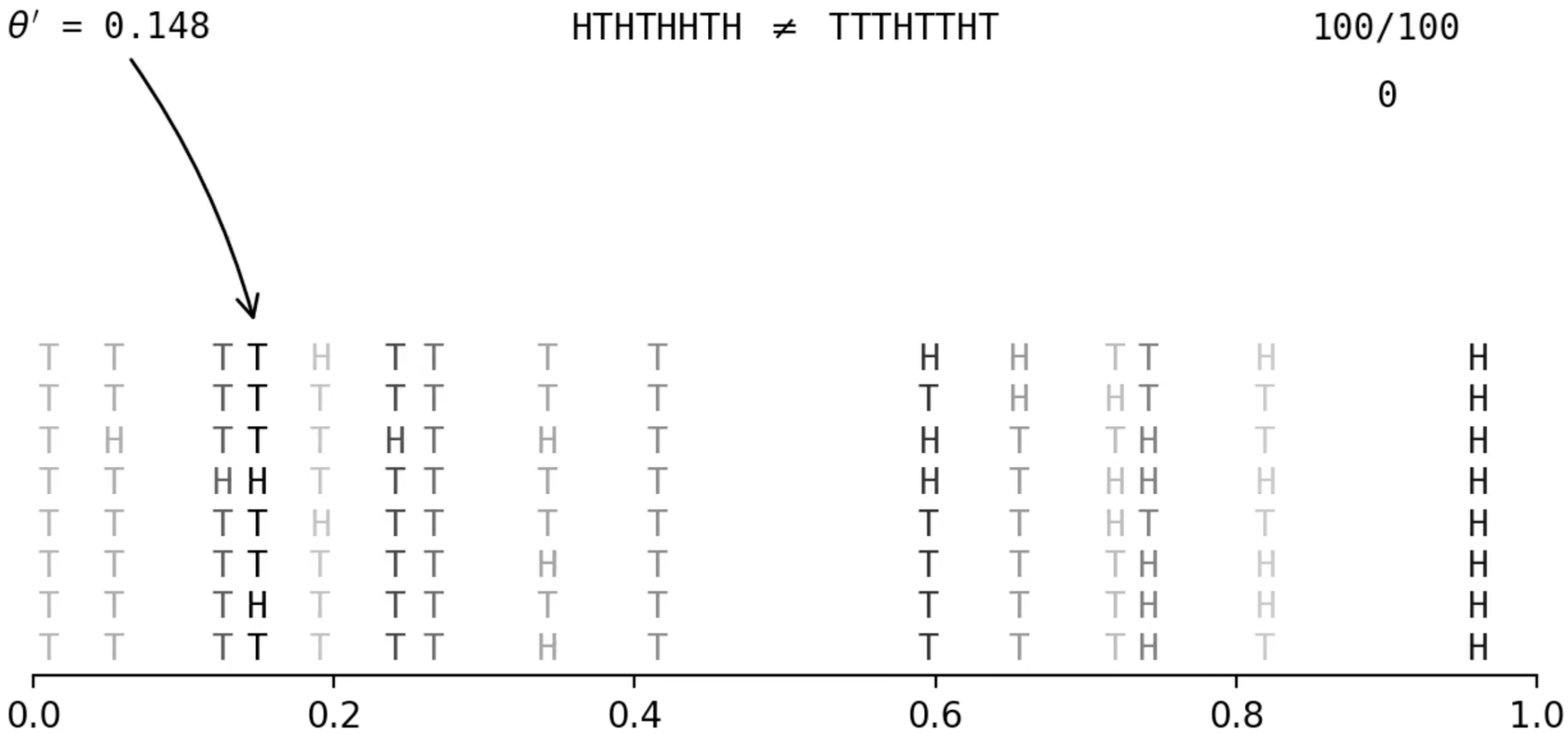
Let’s flip coins

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

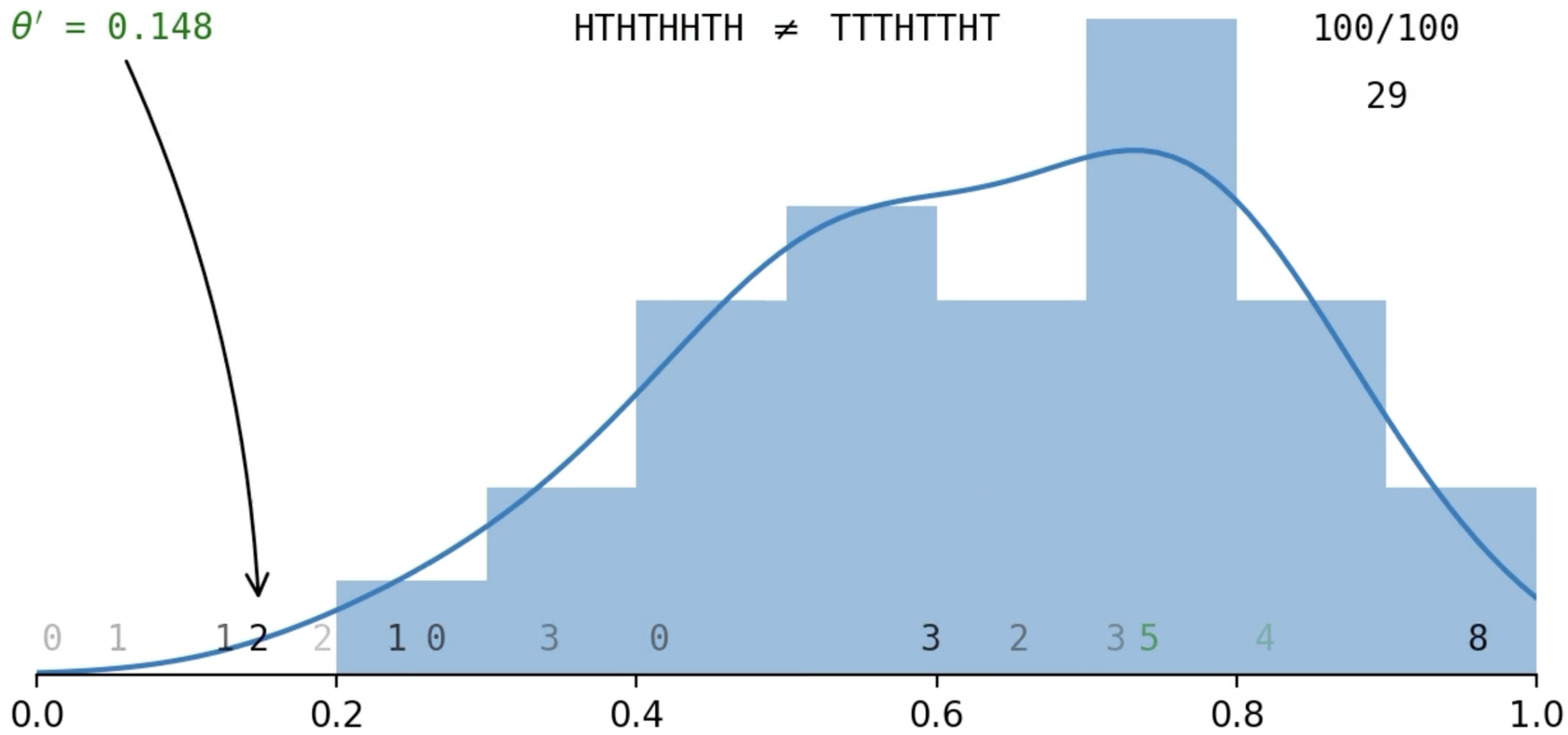
θ-biased coin gives (H, T, H)



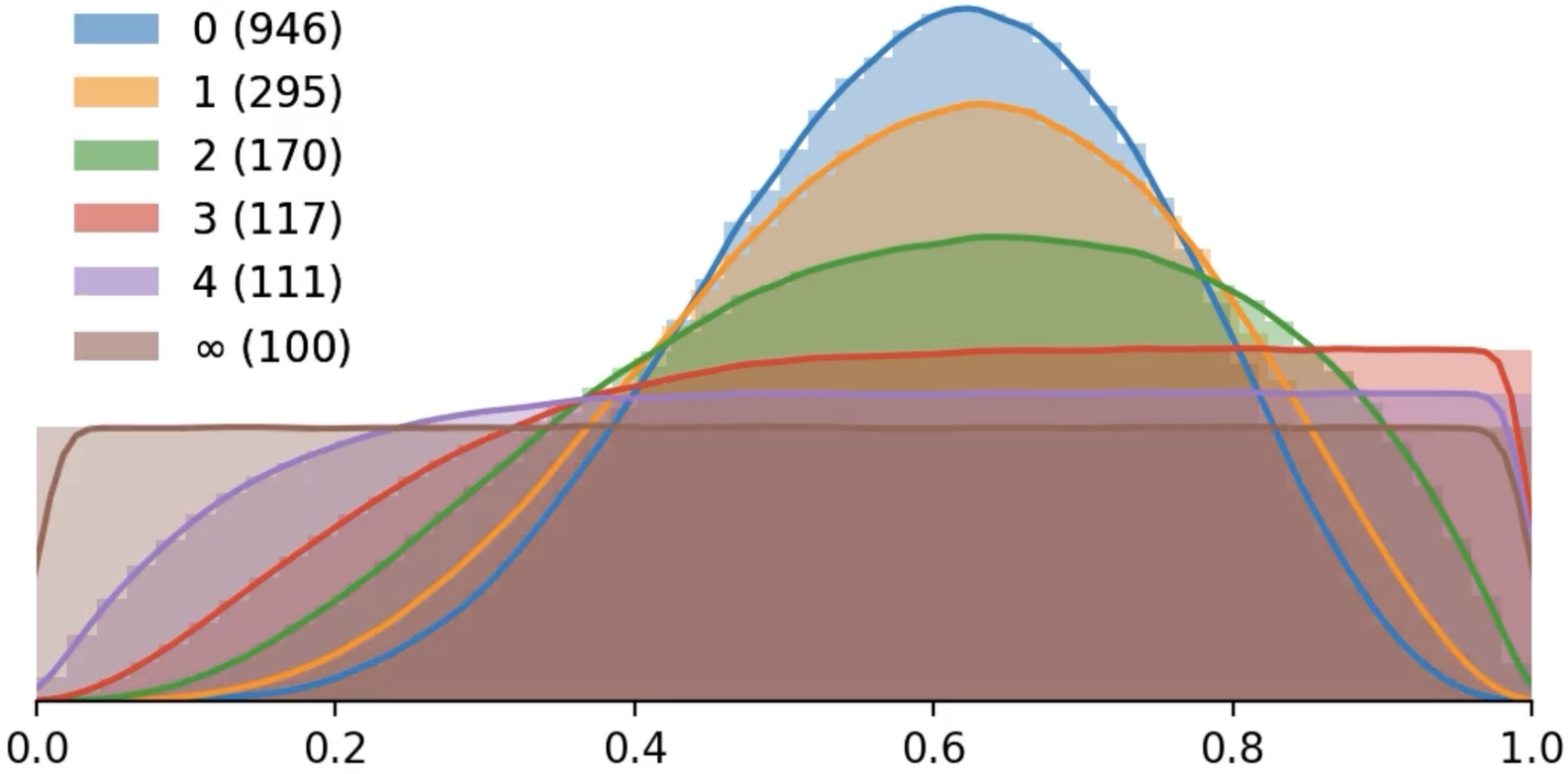
Getting an exact match is hard…



Accept close enough



The ‘approximate’ part of ABC



Statistics with likelihoods

We can only find the likelihood for simple models.

$$X_1, X_2 \overset{\text{i.i.d.}}{\sim} f_X(\,\cdot\,)$$

$$\Rightarrow X_1 + X_2 \sim \texttt{Intractable likelihood!}$$

Have a sample of n i.i.d. observations. As n increases,

$$p_{\boldsymbol{X}}(\boldsymbol{x} \mid \boldsymbol{\theta}) = \prod \text{Small things} \overset{\dagger}{=} 0,$$

or just takes a long time to compute, then **`Intractable Likelihood`**!

Usually it's still possible to simulate these things…

Insurance Motivation

Have a random number of claims $N \sim p_N(\,\cdot\,; \boldsymbol{\theta}_{\text{freq}})$.

Random claim sizes $U_1, \ldots, U_N \sim f_U(\,\cdot\,; \boldsymbol{\theta}_{\text{sev}})$.

We aggregate them somehow, like:

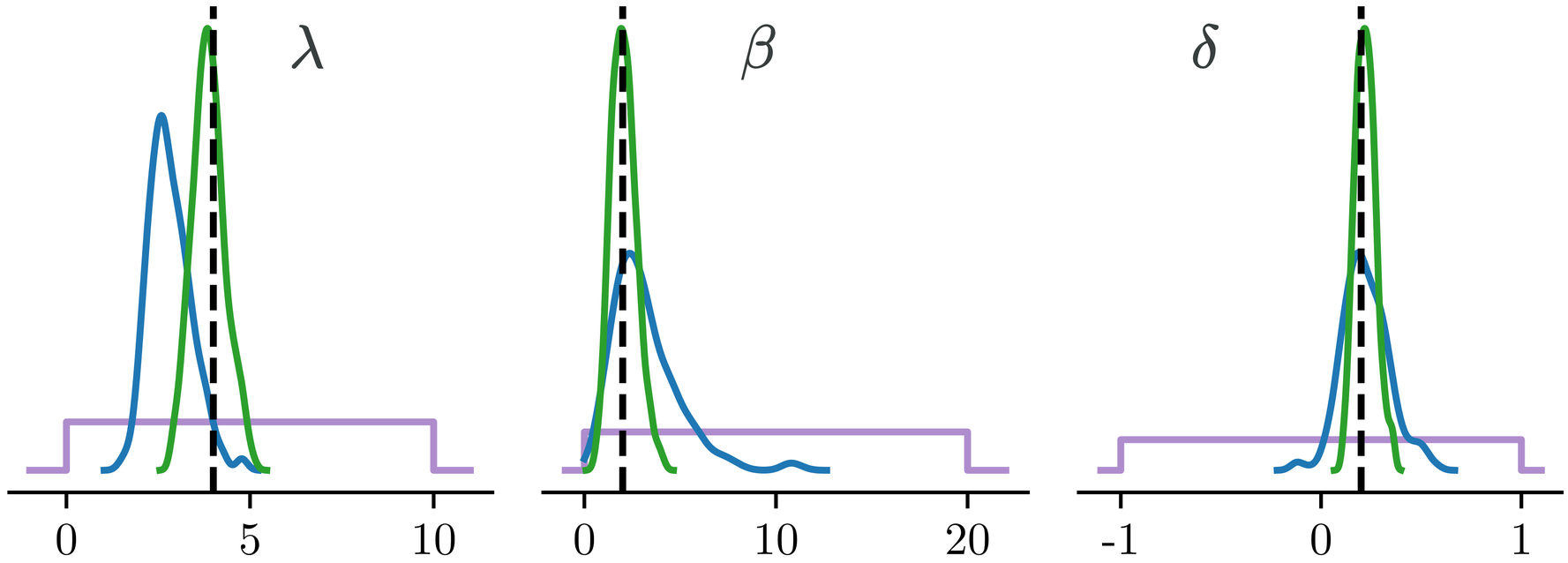
- aggregate claims: $X = \sum_{i=1}^{N} U_i$
- maximum claims: $X = \max_{i=1}^{N} U_i$
- stop-loss: $X = (\sum_{i=1}^{N} U_i - c)_+$.

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

E.g. a reinsurance contract

Dependent random sums

$$N \sim \text{Poisson}(\lambda), \quad U_i \mid N \sim \text{Exp}(\beta \times \mathrm{e}^{\delta N}), \quad X = \sum_{i=1}^{N} U_i.$$

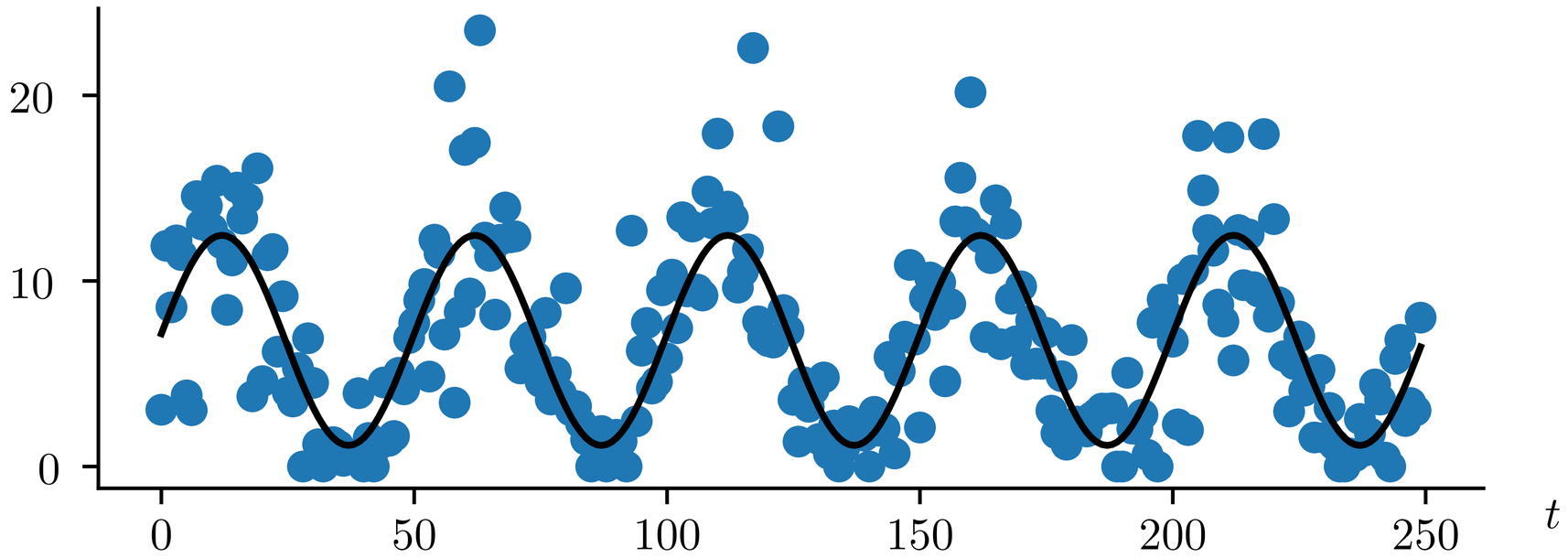


Posteriors for λ, β, and δ with 50 sums and 250 sums.

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

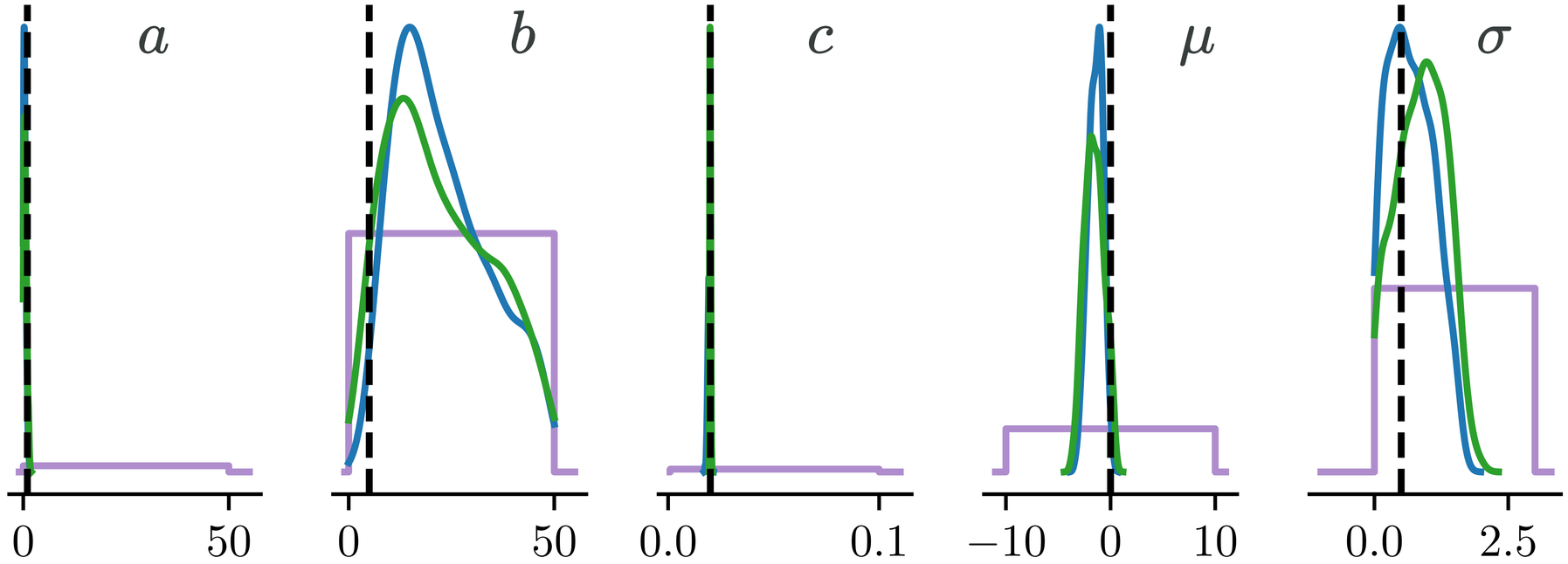
Time-varying arrival rate

$$\lambda(t) = a + b[1 + \sin(2\pi c t)]$$



Time-varying example

Sizes are $U_i \sim \mathsf{Lognormal}(\mu, \sigma)$, observe $X_s = \sum_{i=N_{s-1}}^{N_s} U_i$.



Posteriors for a, b, c, μ, and σ with 50 sums and 250 sums.

Python package

Approximate Bayesian Computation Python Package

Package Description

Approximate Bayesian Computation (ABC) is a statistical method to fit a Bayesian model to data when the likelihood function is hard to compute. The `approxbayescomp` package implements an efficient form of ABC — the *sequential Monte Carlo (SMC)* algorithm. While it can handle any general statistical problem, we built in some models so that fitting insurance loss distributions is particularly easy.

Installation

To install simply run

```
pip install -U approxbayescomp
```

Soon, it will be possible to install using `conda`; at that point the preferred method will be to run

```
conda install -c conda-forge approxbayescomp
```

The source code for the package is available on Github.

R package

approxbayescomp 1.0 Reference Articles Search for

approxbayescomp

The `approxbayescomp` R package implements a series of *Approximate Bayesian Computation* tools that can be used for Bayesian analysis with intractible likelihoods.

Installation

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

```
# install.packages("devtools")
devtools::install_github("Pat-Laub/approxbayescomp-r")
```

Python Package

This package is an R port of our approxbayescomp Python package.

Developed by Patrick Laub, Pierre-Olivier Goffard.

Site built with pkgdown 2.0.7.

License

What license is it under?

Citation

Citing approxbayescomp

Developers

Patrick Laub
Author, maintainer

Pierre-Olivier Goffard
Author

Dev status

lifecycle experimental
CRAN not published

Stats problem → computer problem

Table 3
Runtimes (in seconds or minutes) and the associated server rental costs (in USD or cents) for the ABC fits showcased in the figures in this section on a 'c6g.16xlarge' (64 ARM Neoverse cores) instance. The Fig. 11 times correspond to the curve-matching ABC fits.

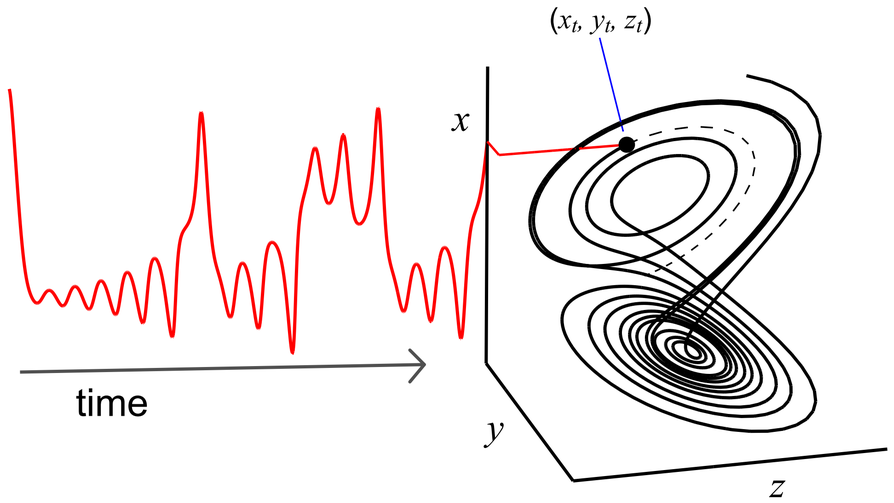
Num. obs.	Figure Number								Total
	3	4	5	6	7	8	9	11	
50	64 s (3.9 ¢)	22 s (1.3 ¢)	31 s (1.9 ¢)	40 s (2.4 ¢)	51 s (3.1 ¢)	19 s (1.1 ¢)	21 s (1.3 ¢)	29 s (1.8 ¢)	11 m
250	106 s (6.4 ¢)	34 s (2.1 ¢)	32 s (1.9 ¢)	92 s (5.6 ¢)	67 s (4.0 ¢)	19 s (1.1 ¢)	19 s (1.1 ¢)	67 s (4.0 ¢)	($0.43)

> "… you can just throw more and more computers at the problem, and that's much much easier than throwing more brains at a problem." Hadley Wickham

Get involved!

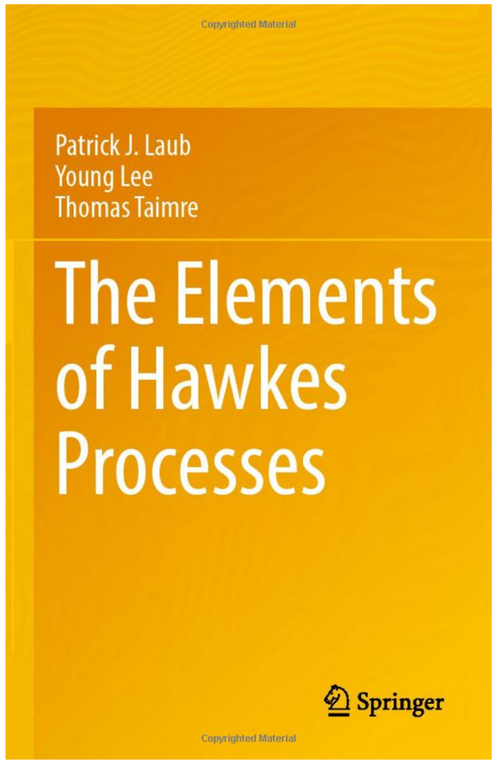
😊 Give it a try, feedback would be very welcome.

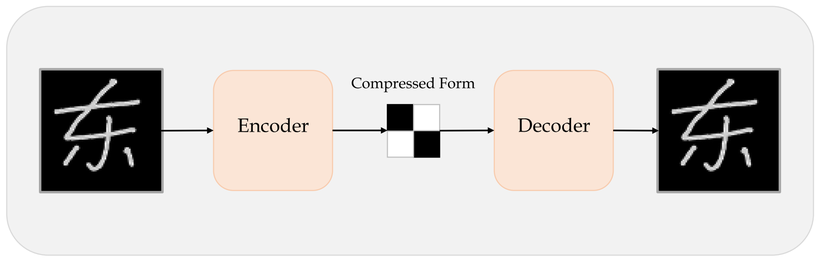
😍 We'd love contributions!



Empirical Dynamic Modelling

Cf. https://pat-laub.github.io.





Deep Learning for Actuaries

Appendix

Dependent random sums code

```
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)
```

Deep Learning for Actuaries

Lecture slides from UNSW's ACTL3143 & ACTL5111 courses

AUTHOR
Dr Patrick Laub

Overview

These are the lecture slides from my recent "Deep Learning for Actuaries" courses (coded ACTL3143 & ACTL5111) at UNSW. They can be used to see what topics I covered in these courses. The slides are not intended to be used to learn deep learning from scratch. For that, you need to attend the lectures & complete the assessment.

Lecture slides

- Course overview slides, source
- Artificial intelligence & machine learning slides, source
- Python slides, source
- Making a chess-playing AI slides, source
- Deep Learning with Keras slides, source
- Mathematics of Deep Learning slides, source
- Network Architectures for Tabular Data slides, source

Does it work in theory?

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

If for some $\epsilon \geq 0$ 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 $\boldsymbol{\theta} \in \boldsymbol{\Theta}$

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

□

We sample the *approximate/ABC posterior*.

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

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

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

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

ABC Sequential Monte Carlo

- Input: data $\boldsymbol{x}_{\text{obs}}$, prior $\pi(\boldsymbol{\theta})$, distance $\mathcal{D}(\cdot, \cdot)$, # of generations G, # of particles K
- Start with $\epsilon_0 = \infty$, $\pi_0(\boldsymbol{\theta} \mid \boldsymbol{x}_{\text{obs}}) = \pi(\boldsymbol{\theta})$
- For each generation $g = 1$ to G
  - For each particle $k = 1$ to K
    - Repeatedly:
      - Generate a guess $\boldsymbol{\theta}^*$ from $\pi_{g-1}(\boldsymbol{\theta} \mid \boldsymbol{x}_{\text{obs}})$
      - Generate fake data $\boldsymbol{x}^*$ from $\boldsymbol{\theta}^*$
      - Stop when $\mathcal{D}(\boldsymbol{x}^*, \boldsymbol{x}_{\text{obs}}) < \epsilon_{g-1}$
    - Store $\boldsymbol{\theta}_k^g = \boldsymbol{\theta}^*$
  - Create a new threshold $\epsilon_g \leq \epsilon_{g-1}$ and a new population by discarding particles with $\mathcal{D}(\boldsymbol{x}_k^g, \boldsymbol{x}_{\text{obs}}) \geq \epsilon_g$ until the effective sample size is $K/2$
  - Weight each particle by $w_k^g \propto \pi(\boldsymbol{\theta}_k^g) / \pi_{g-1}(\boldsymbol{\theta}_k^g \mid \boldsymbol{x}_{\text{obs}})$
  - Create a KDE $\pi_g(\boldsymbol{\theta} \mid \boldsymbol{x}_{\text{obs}})$ based on the surviving $(\boldsymbol{\theta}_k^g, w_k^g)$ particles