Probabilistic Programming and Inference

Large-Scale Data Processing and Optimisation
13 Nov 2024

Hong Ge
hg344@cam.ac.uk

Machine Learning Group, Department of Engineering, University of Cambridge

slides courtesy of Tor Fjelde

Bayesian inference: probability view

Given some "real world" data $\mathcal{D}$

Want to determine parameters θ
given "real world" data $\mathcal{D}$, i.e.

$$p(\theta \mid \mathcal{D})$$

Bayes' rule tells us that

$$p(\theta \mid \mathcal{D}) \propto p(\mathcal{D} \mid \theta) p(\theta)$$

Which we can write as *generative model*

$$\theta \sim p(\theta)$$

$$\mathcal{D} \sim p(\mathcal{D} \mid \theta)$$

Bayesian inference: simulator view

Have simulator which, given parameters theta, can generate new data $\tilde{\mathcal{D}}$

Have data $\mathcal{D}$ from "real world"

Have prior knowledge regarding values of theta are *probable*, giving rise to a <u>prior</u> distribution $\mathrm{p}(\theta)$

Aim: determine parameters θ such that the *generated* data $\tilde{\mathcal{D}}$ is close to *true* data $\mathcal{D}$, i.e. sample from $\mathrm{p}(\theta \mid \mathcal{D})$.

$$\theta \sim \mathrm{p}(\theta)$$

$$\mathcal{D} \sim \mathrm{p}(\mathcal{D} \mid \theta)$$

Bayesian inference: biased coin

Data $\mathcal{D}$ of coin flips from a **biased** coin

$$\mathcal{D} = (\mathrm{H}, \mathrm{H}, \mathrm{H}, \mathrm{T}, \ldots)$$

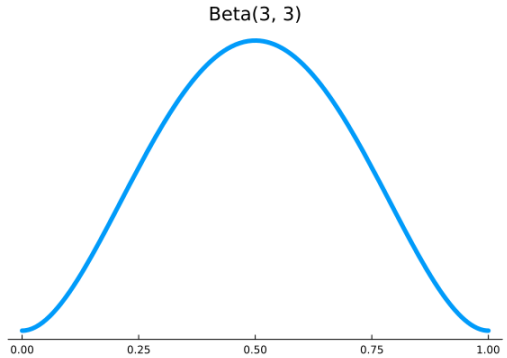
Initial belief: it's a fair coin

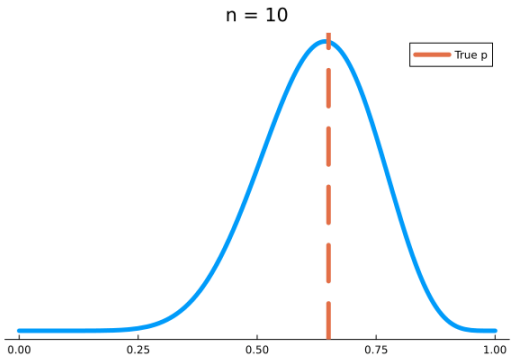
Update our belief using the observations

More data => more certain!

$$\theta \sim \mathrm{Beta}(3, 3)$$

$$x_i \sim \mathrm{Bernoulli}(\theta) \quad \text{for } i = 1, \ldots, n$$





Bayesian inference: biased coin

Data $\mathcal{D}$ of coin flips from a **biased** coin

$\mathcal{D} = (H, H, H, T, \ldots)$

Initial

Update our b

More

$\theta \sim \text{Beta}$

$x_i \sim \text{Bernoulli}(\theta) \quad \text{for } i = 1, \ldots, n$

WARNING!

Updating the posterior with incoming data is, generally, not that as easy as these animations make it seem!

Bayesian inference: linear regression

$$\mathcal{D} = \big((x_1, y_1), \ldots, (x_n, y_n)\big)$$

$$\alpha \sim \mathcal{N}(0, 1)$$

$$\beta \sim \mathcal{N}(0, I)$$

$$\sigma \sim \mathcal{N}_+(0, 1)$$

$$\mu_i = \beta x_i + \alpha$$

$$y_i \sim \mathcal{N}(\mu_i, \sigma^2)$$

Bayesian inference: linear regression

$$\mathcal{D} = \left((x_1, y_1), \ldots, (x_n, y_n)\right)$$

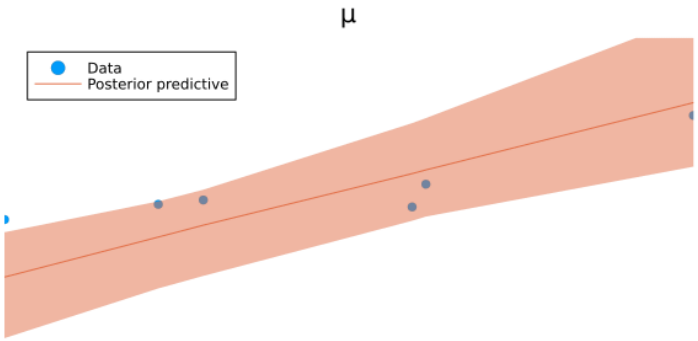
$$\alpha \sim \mathcal{N}(0, 1)$$

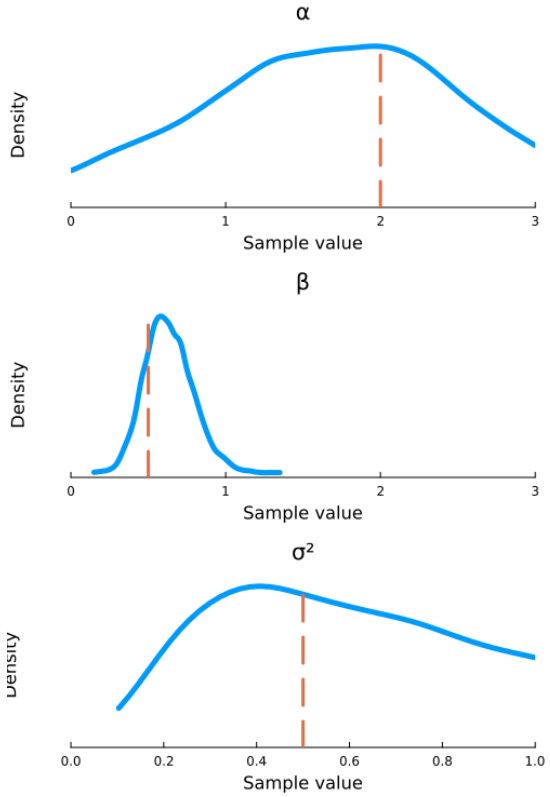
$$\beta \sim \mathcal{N}(0, I)$$

$$\sigma \sim \mathcal{N}_+(0, 1)$$

$$\mu_i = \beta x_i + \alpha$$

$$y_i \sim \mathcal{N}(\mu_i, \sigma^2)$$





Bayesian inference: in practice

Posterior $p(\theta \mid \mathcal{D})$ is generally **not available in closed-form**

Requires various approximate methods to inspect $p(\theta \mid \mathcal{D})$

E.g. Markov Chain Monte Carlo (MCMC), variational inference (VI)

Annoying & difficult to implement!

Bayesian inference: summary

✓ Principled way to incorporate prior knowledge through p(theta)

✓ Allows quantification of uncertainty

$$\theta \sim p(\theta)$$

$$\mathcal{D} \sim p(\mathcal{D} \mid \theta)$$

✗ Non-trivial for interesting problems

Both computationally expensive and requires knowledge of inference methods

Probabilistic programming

Aim: make Bayesian inference approachable.

User should only have to write down their model, and the rest should be “automatic”

Key ideas:

1. Decouple modelling and inference
2. General-purpose inference

And many more!

Probabilistic programming

Probabilistic programming language (PPL)

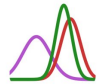
=

Suite of tools allowing easy specification of models + inference method

Many different approaches to PPLs

TuringLang/**Turing.jl**

Bayesian inference with probabilistic programming.



Bean Machine

Stan

PyMC

BUGS

Anglican

PYRO

And many more!

Probabilistic programming

Many different approaches to PPLs

Stan

Has its own DSL

Implemented in C++

Provides convenient interfaces in R, Python, Julia, etc.

Very well-established with great track record (gold standard)

BUGS

And many more!

Probabilistic programming

Many different approaches to PPLs

BUGS

Has its own DSL

Implemented in Component Pascal

Provides convenient interface in R

Old now, but was one of the very first PPLs

Bean Machine

PYRO

And many more!

Probabilistic programming

Many different approaches to PPLs

PyMC, Pyro and **Bean**

Embedded in Python

Relies on underlying “graph” computational frameworks, e.g. JAX, PyTorch

But have to work with a subset of Python compatible with framework

And many more!

Many different approaches to PPLs

Model and inference are closely coupled

↓

Model and inference are decoupled, but modelling language is closely coupled with inference

e.g. **BUGS**: Gibbs, **Stan**: HMC, **Infer.NET**: EP/VMP

↓

Modelling language is decoupled with inference details, but closely coupled with a computational backend

e.g. **Pyro**: JAX, **PyTorch**; PyMC: Theano, **BlackJAX**: JAX

TuringLang/**Turing.jl**

Bayesian inference with probabilistic programming.

Bean Machine

Stan

PyMC

BUGS

Anglican

PYRO

And many more!

Turing.jl: what is it?

⇒ **More general purpose**

Inference-agnostic: models are compatible with a wide range of MCMC and VI algorithms

Interoperability with other modelling libraries (GPs, DiffEqs): requires all modelling libraries using the same computational backend

Good tools for **profiling/debugging/interactively_run** models and inference; **High performance**

Modular / Compositional inference

among others

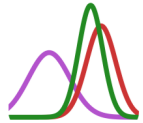
BUGS

And many more!

Turing.jl: what is it?

TuringLang/**Turing.jl**

Bayesian inference with probabilistic programming.



Fully implemented in the julia programming language

Approachable for users familiar with Matlab, Python and R

Support BUGS syntax via JuliaBUGS

Growing in popularity

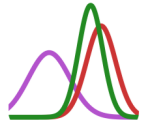
88 Contributors | 69 Issues | 23 Discussions | 2k Stars | 213 Forks

```julia
using Turing

# Define the model.
@model function demo()
    θ ~ Prior
    𝒟 ~ Likelihood(θ)
end

# Instantiate the model.
model = demo()
# Condition `model` on the data.
model_cond = model | (𝒟 = data,)
# Sample from the posterior.
chain = sample(model_cond, NUTS(), 1000)
```

Turing.jl: what is it?

TuringLang/**Turing.jl**

Bayesian inference with probabilistic programming.



Fully implemented in the julia programming language

Approachable for users familiar with Matlab, Python and R

Support BUGS syntax via JuliaBUGS

Growing in popularity

88 Contributors | 69 Issues | 23 Discussions | 2k Stars | 213 Forks

```julia
using Turing

# Define the model.
@model function demo()
    θ ~ Prior
    𝒟 ~ Likelihood(θ)
end

# Instantiate the model.
model = demo()
# Condition `model` on the data.
model_cond = model | (𝒟 = data,)
# Sample from the posterior.
chain = sample(model_cond, NUTS(), 1000)
```

Turing.jl: Bayesian linear regression

```
using Turing

# Define model.
@model function linear_regression(x)
    # Priors.
    α ~ Normal(0, 1)
    β ~ Normal(0, 1)
    σ² ~ InverseGamma(2, 3)

    # Likelihood.
    y ~ MvNormal(β * x .+ α, σ²)
end

# Condition model on some data.
model = linear_regression(x_data) | (y = y_data, )

# Perfurm inference!
sample(model, NUTS(), 1000)
```

$$\alpha \sim \mathcal{N}(0, 1)$$

$$\beta \sim \mathcal{N}(0, \mathrm{I})$$

$$\sigma \sim \mathcal{N}_+(0, 1)$$

$$\mu_i = \beta x_i + \alpha$$

$$y_i \sim \mathcal{N}(\mu_i, \sigma^2)$$

JuliaBUGS: Rats model

$$
\begin{aligned}
Y_{ij} &\sim \text{Normal}\left(\alpha_i + \beta_i \left(x_j - \bar{x}\right), \tau_c\right) \\
\alpha_i &\sim \text{Normal}\left(\alpha_c, \tau_\alpha\right) \\
\beta_i &\sim \text{Normal}\left(\beta_c, \tau_\beta\right)
\end{aligned}
$$

```
model_def = @bugs("""model{
    for( i in 1 : N ) {
        for( j in 1 : T ) {
        Y[i , j] ~ dnorm(mu[i , j],tau.c)
        mu[i , j] <- alpha[i] + beta[i] * (x[j] - xbar)
        }
        alpha[i] ~ dnorm(alpha.c,alpha.tau)
        beta[i] ~ dnorm(beta.c,beta.tau)
    }
    tau.c ~ dgamma(0.001,0.001)
    sigma <- 1 / sqrt(tau.c)
    alpha.c ~ dnorm(0.0,1.0E-6)
    alpha.tau ~ dgamma(0.001,0.001)
    beta.c ~ dnorm(0.0,1.0E-6)
    beta.tau ~ dgamma(0.001,0.001)
    alpha0 <- alpha.c - xbar * beta.c
}""", false) # `false` means R-style variable names are kept
```

Nothing is different from the original BUGS

Turing.jl: workflow

Bayesian workflow is important

Bayesian workflow*

Andrew Gelman† Aki Vehtari‡ Daniel Simpson§ Charles C. Margossian†
Bob Carpenter¶ Yuling Yao† Lauren Kennedy‖ Jonah Gabry†
Paul-Christian Bürkner** Martin Modrák††

2 Nov 2020

Should be “simple” to perform

Turing.jl: workflow

Standard Bayesian workflow

Define model “in maths”

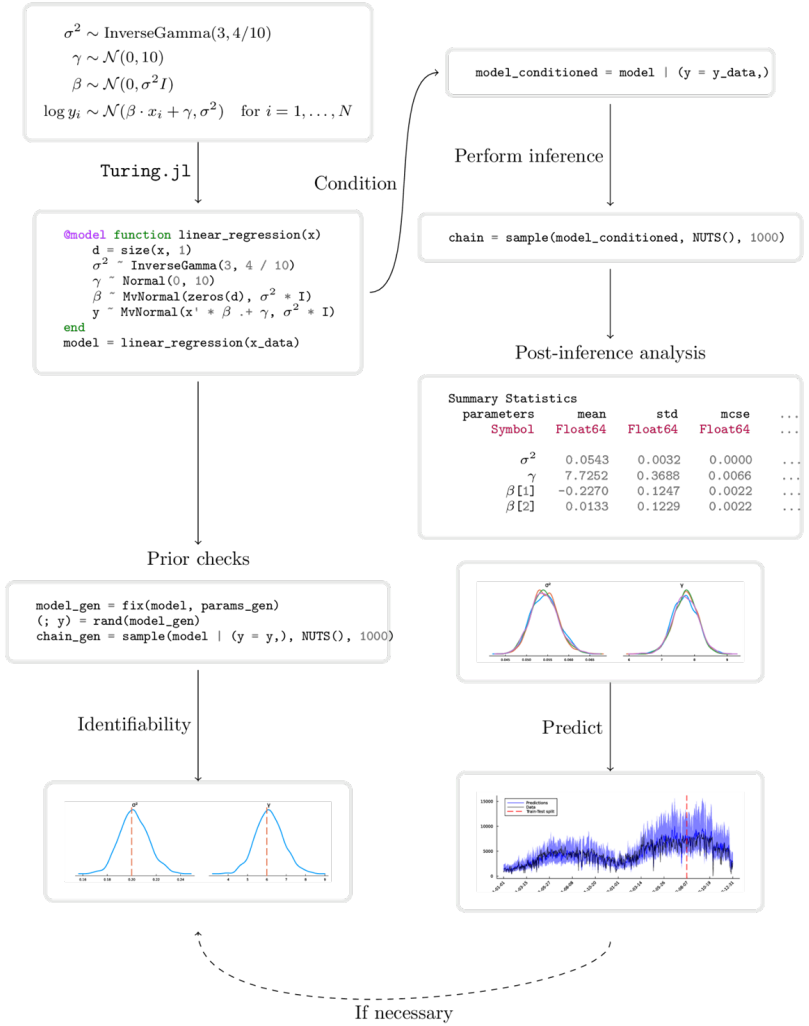
Define model in Turing.jl

Perform prior checks

Perform inference

Evaluate, e.g. predict

Rinse and repeat!



Turing.jl: workflow

Standard Bayesian workflow

Define model “in maths”

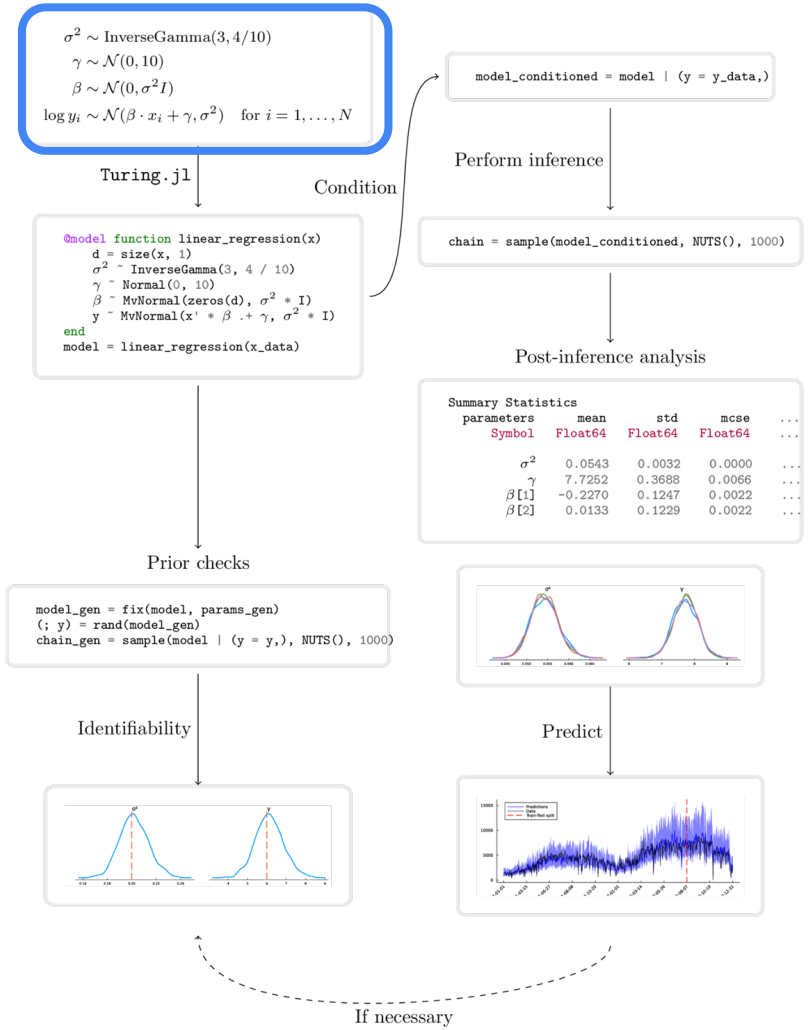
Define model in Turing.jl

Perform prior checks

Perform inference

Evaluate, e.g. predict

Rinse and repeat!

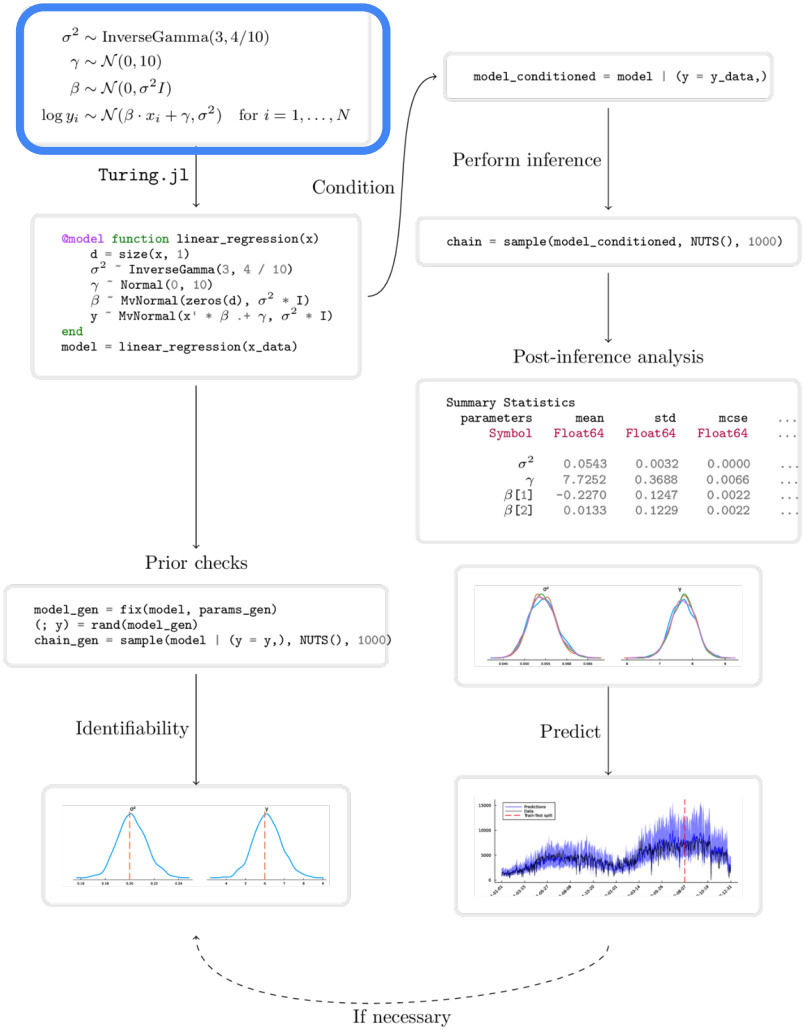


Turing.jl: workflow

Standard Bayesian workflow

Define model “in maths”

$$
\begin{aligned}
\sigma^2 &\sim \text{InverseGamma}(3, 4/10) \\
\gamma &\sim \mathcal{N}(0, 10) \\
\beta &\sim \mathcal{N}(0, \sigma^2 I) \\
\log y_i &\sim \mathcal{N}(\beta \cdot x_i + \gamma, \sigma^2) \quad \text{for } i = 1, \ldots, N
\end{aligned}
$$



Turing.jl: workflow

Standard Bayesian workflow

Define model “in maths”

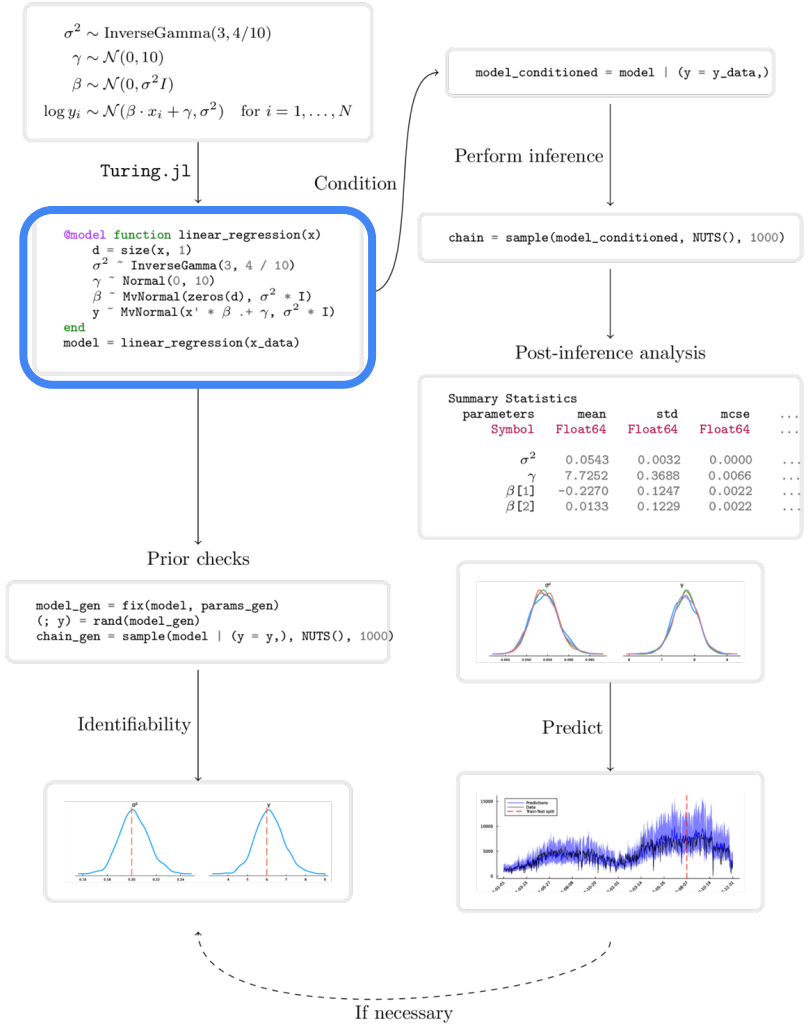
Define model in Turing.jl

Perform prior checks

Perform inference

Evaluate, e.g. predict

Rinse and repeat!

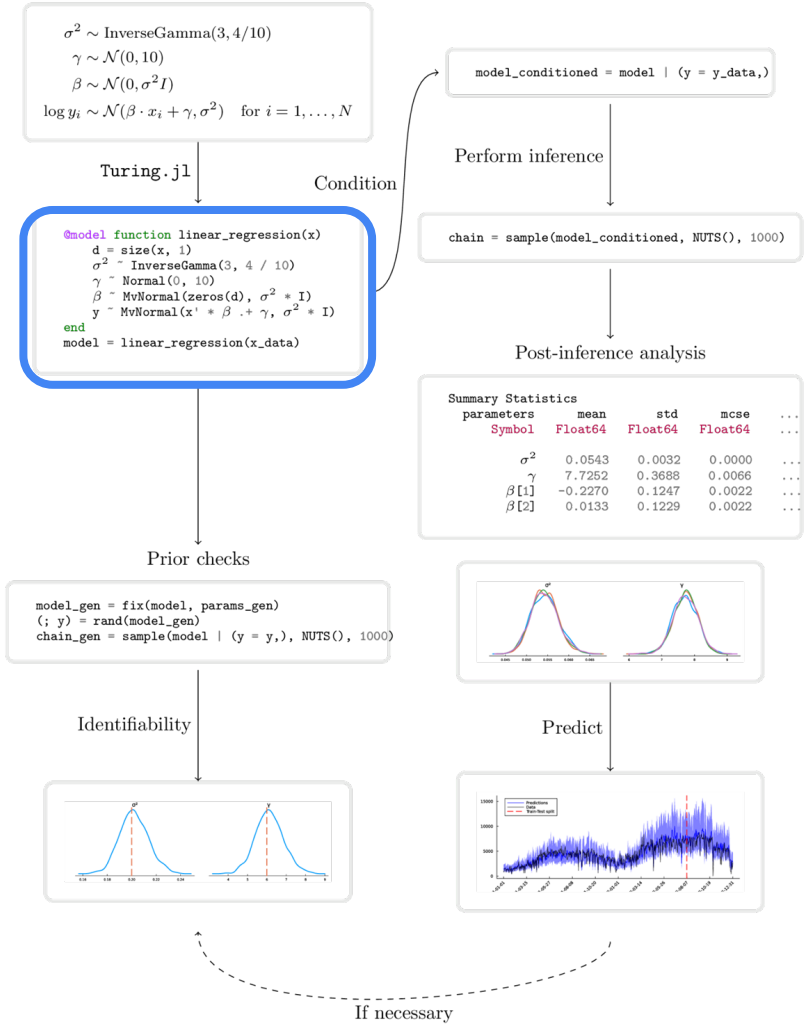


Turing.jl: workflow

Standard Bayesian workflow

Define model in Turing.jl

```
@model function linear_regression(x)
    d = size(x, 1)
    σ² ~ InverseGamma(3, 4 / 10)
    γ ~ Normal(0, 10)
    β ~ MvNormal(zeros(d), σ² * I)
    y ~ MvNormal(x' * β .+ γ, σ² * I)
end
model = linear_regression(x_data)
```



Turing.jl: workflow

Standard Bayesian workflow

Define model “in maths”

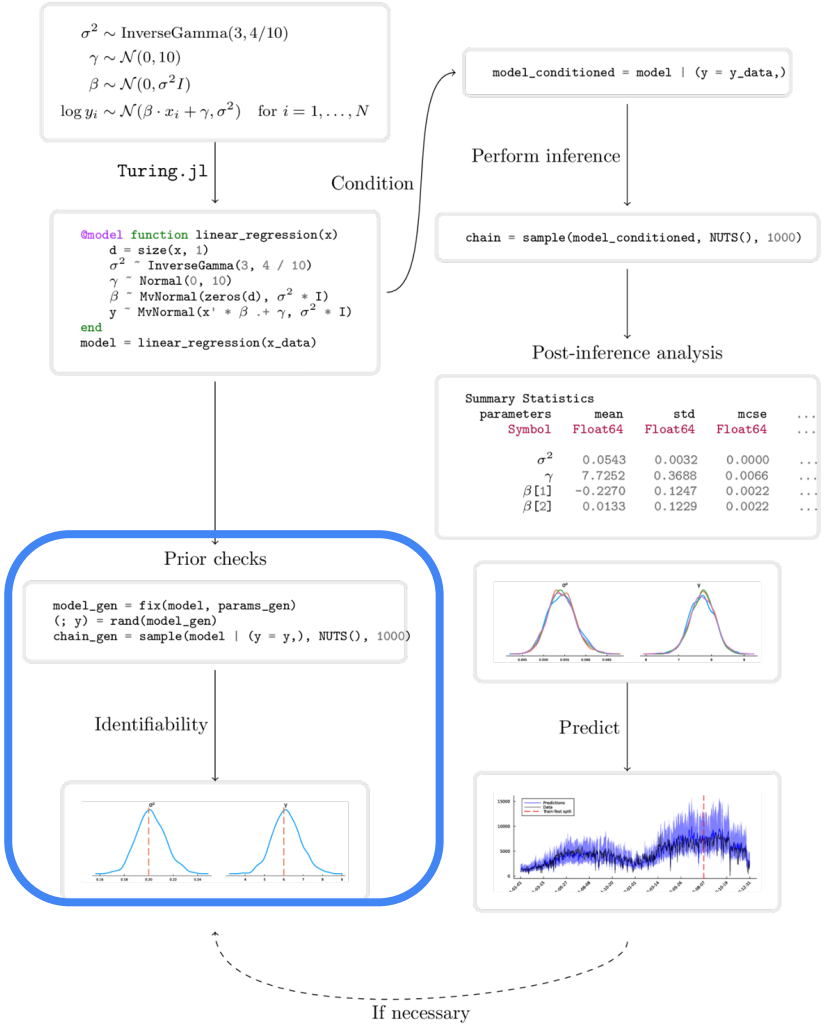
Define model in Turing.jl

Perform prior checks

Perform inference

Evaluate, e.g. predict

Rinse and repeat!



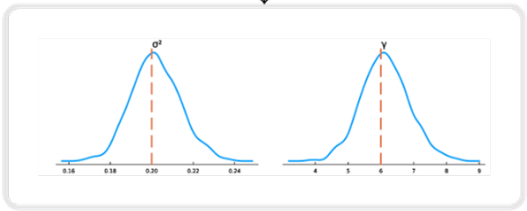
Turing.jl: workflow

Standard Bayesian workflow

Perform prior checks

```
model_gen = fix(model, params_gen)
(; y) = rand(model_gen)
chain_gen = sample(model | (y = y,), NUTS(), 1000)
```

Identifiability



$$\sigma^2 \sim \text{InverseGamma}(3, 4/10)$$
$$\gamma \sim \mathcal{N}(0, 10)$$
$$\beta \sim \mathcal{N}(0, \sigma^2 I)$$
$$\log y_i \sim \mathcal{N}(\beta \cdot x_i + \gamma, \sigma^2) \quad \text{for } i = 1, \ldots, N$$

Turing.jl

```
@model function linear_regression(x)
    d = size(x, 1)
    σ² ~ InverseGamma(3, 4 / 10)
    γ ~ Normal(0, 10)
    β ~ MvNormal(zeros(d), σ² * I)
    y ~ MvNormal(x' * β .+ γ, σ² * I)
end
model = linear_regression(x_data)
```

Condition

```
model_conditioned = model | (y = y_data,)
```

Perform inference

```
chain = sample(model_conditioned, NUTS(), 1000)
```

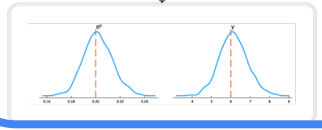
Post-inference analysis

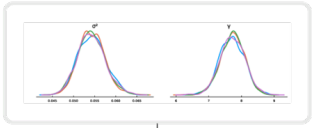
```
Summary Statistics
  parameters      mean       std      mcse   ...
      Symbol   Float64   Float64   Float64   ...

          σ²    0.0543    0.0032    0.0000   ...
           γ    7.7252    0.3688    0.0066   ...
        β[1]   -0.2270    0.1247    0.0022   ...
        β[2]    0.0133    0.1229    0.0022   ...
```

Prior checks

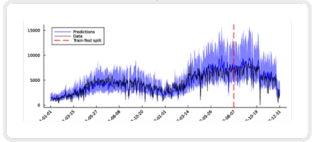
```
model_gen = fix(model, params_gen)
(; y) = rand(model_gen)
chain_gen = sample(model | (y = y,), NUTS(), 1000)
```

Identifiability





Predict



If necessary

Turing.jl: workflow

Standard Bayesian workflow

Define model “in maths”

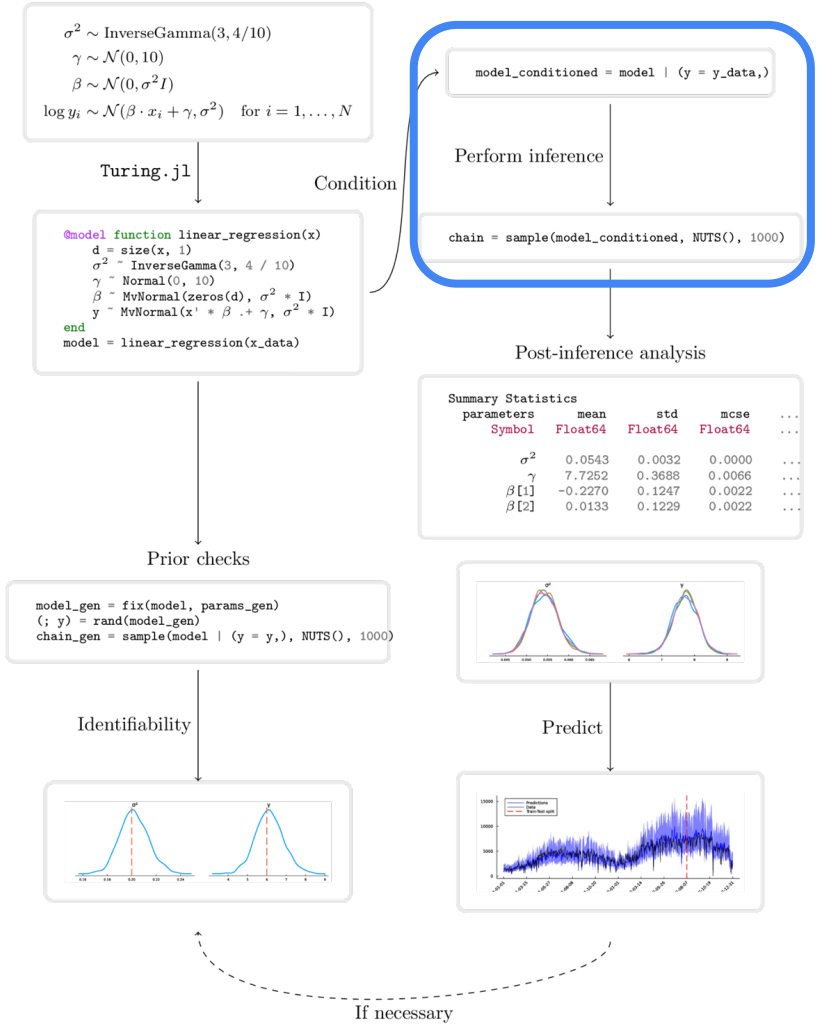
Define model in Turing.jl

Perform prior checks

Perform inference

Evaluate, e.g. predict

Rinse and repeat!



Turing.jl: workflow

Standard Bayesian workflow

Perform inference

```
model_conditioned = model | (y = y_data,)
```

Perform inference

```
chain = sample(model_conditioned, NUTS(), 1000)
```

$$\sigma^2 \sim \text{InverseGamma}(3, 4/10)$$
$$\gamma \sim \mathcal{N}(0, 10)$$
$$\beta \sim \mathcal{N}(0, \sigma^2 I)$$
$$\log y_i \sim \mathcal{N}(\beta \cdot x_i + \gamma, \sigma^2) \quad \text{for } i = 1, \ldots, N$$

Turing.jl

```
@model function linear_regression(x)
    d = size(x, 1)
    σ² ~ InverseGamma(3, 4 / 10)
    γ ~ Normal(0, 10)
    β ~ MvNormal(zeros(d), σ² * I)
    y ~ MvNormal(x' * β .+ γ, σ² * I)
end
model = linear_regression(x_data)
```

Condition

```
model_conditioned = model | (y = y_data,)
```

Perform inference

```
chain = sample(model_conditioned, NUTS(), 1000)
```

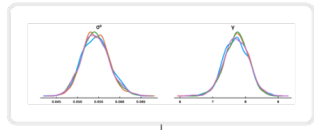
Post-inference analysis

Summary Statistics

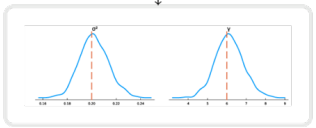
parameters	mean	std	mcse	...
Symbol	Float64	Float64	Float64	...
σ^2	0.0543	0.0032	0.0000	...
γ	7.7252	0.3688	0.0066	...
$\beta[1]$	-0.2270	0.1247	0.0022	...
$\beta[2]$	0.0133	0.1229	0.0022	...

Prior checks

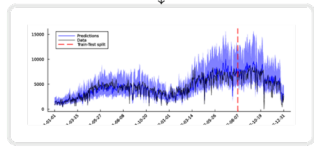
```
model_gen = fix(model, params_gen)
(; y) = rand(model_gen)
chain_gen = sample(model | (y = y,), NUTS(), 1000)
```



Identifiability



Predict



If necessary

Turing.jl: workflow

Standard Bayesian workflow

Define model “in maths”

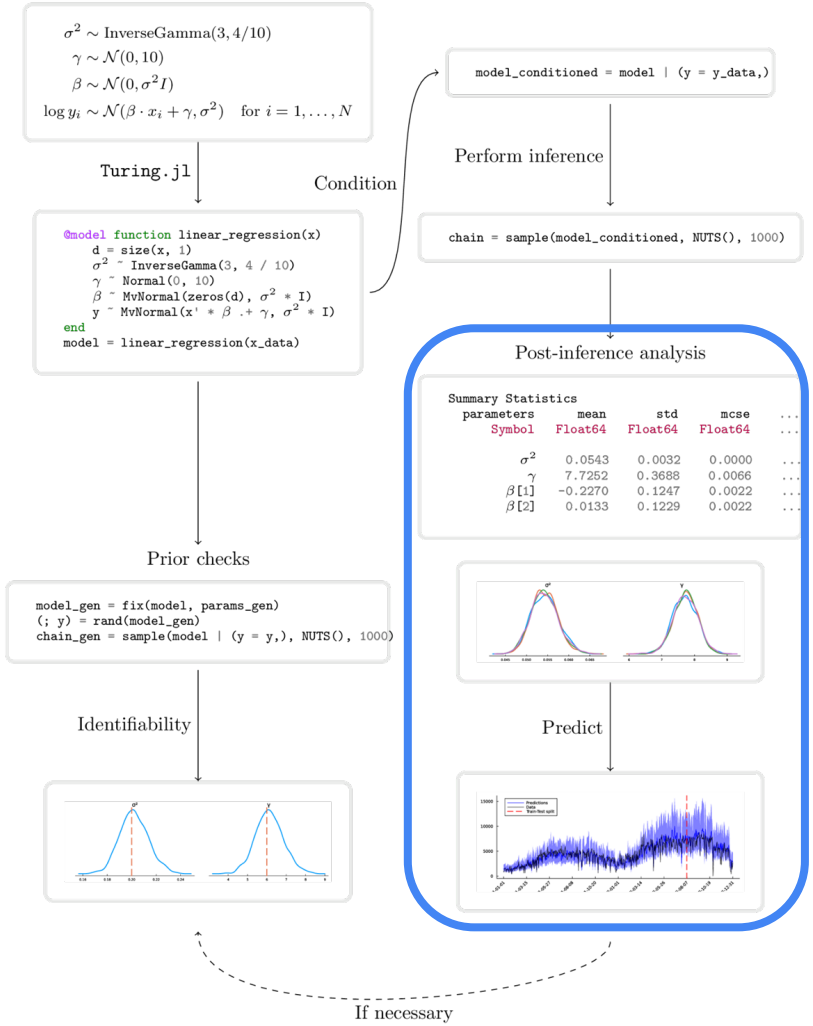
Define model in Turing.jl

Perform prior checks

Perform inference

Evaluate, e.g. predict

Rinse and repeat!



Turing.jl: workflow

Standard Bayesian workflow

Evaluate, e.g. predict

$$\sigma^2 \sim \text{InverseGamma}(3, 4/10)$$
$$\gamma \sim \mathcal{N}(0, 10)$$
$$\beta \sim \mathcal{N}(0, \sigma^2 I)$$
$$\log y_i \sim \mathcal{N}(\beta \cdot x_i + \gamma, \sigma^2) \quad \text{for } i = 1, \ldots, N$$

Turing.jl

```
@model function linear_regression(x)
    d = size(x, 1)
    σ² ~ InverseGamma(3, 4 / 10)
    γ ~ Normal(0, 10)
    β ~ MvNormal(zeros(d), σ² * I)
    y ~ MvNormal(x' * β .+ γ, σ² * I)
end
model = linear_regression(x_data)
```

Condition

```
model_conditioned = model | (y = y_data,)
```

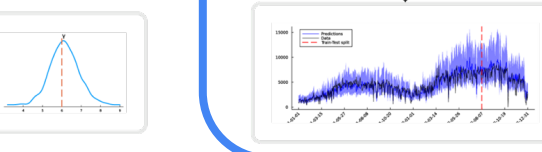
Perform inference

```
chain = sample(model_conditioned, NUTS(), 1000)
```

Post-inference analysis

```
Summary Statistics
  parameters      mean       std      mcse    ...
      Symbol   Float64   Float64   Float64    ...

          σ²    0.0543    0.0032    0.0000    ...
           γ    7.7252    0.3688    0.0066    ...
        β[1]   -0.2270    0.1247    0.0022    ...
        β[2]    0.0133    0.1229    0.0022    ...
```

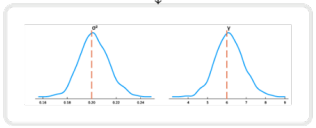
Predict



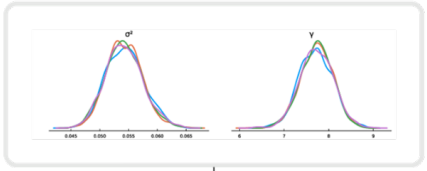
Prior checks

```
model_gen = fix(model, params_gen)
(; y) = rand(model_gen)
chain_gen = sample(model | (y = y,), NUTS(), 1000)
```

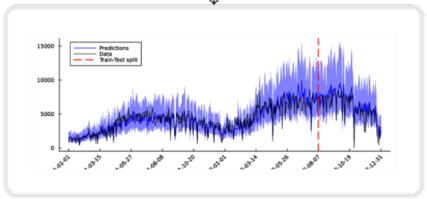
Identifiability



If necessary



Predict



```
Summary Statistics
  parameters      mean       std      mcse    ...
      Symbol   Float64   Float64   Float64    ...

          σ²    0.0543    0.0032    0.0000    ...
           γ    7.7252    0.3688    0.0066    ...
        β[1]   -0.2270    0.1247    0.0022    ...
        β[2]    0.0133    0.1229    0.0022    ...
```

Turing.jl: workflow

Standard Bayesian workflow

Define model “in maths”

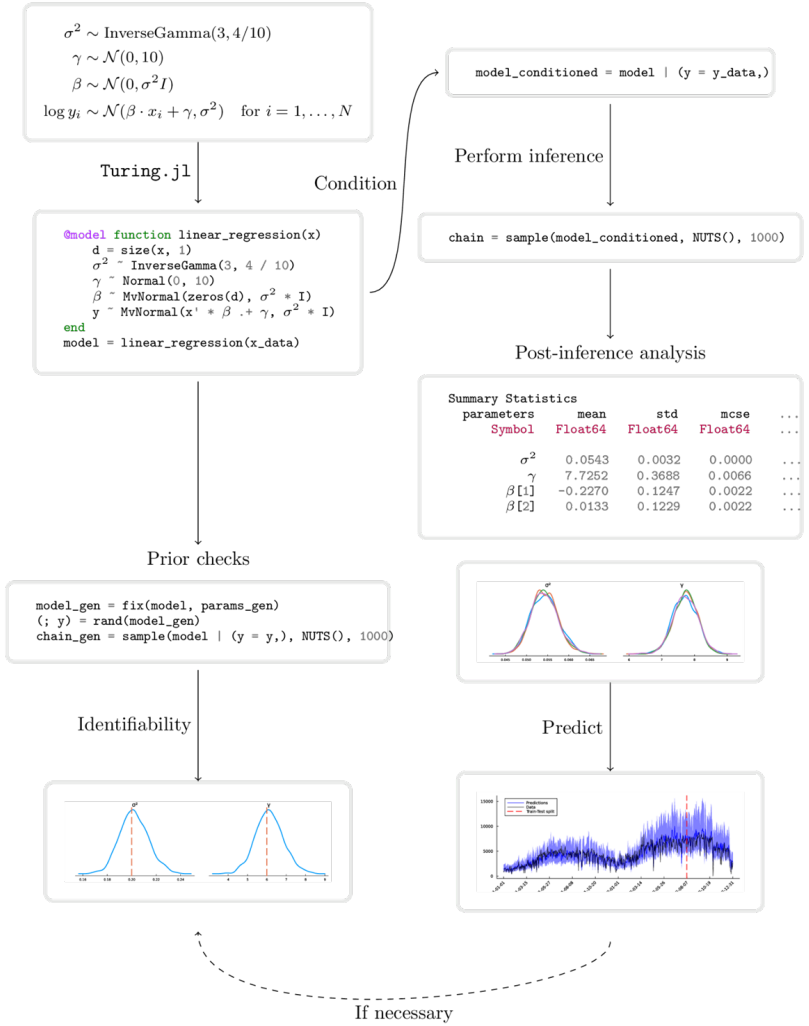
Define model in Turing.jl

Perform prior checks

Perform inference

Evaluate, e.g. predict

Rinse and repeat!



Turing.jl: overview

✓ **Convenience** and **performance** from Julia

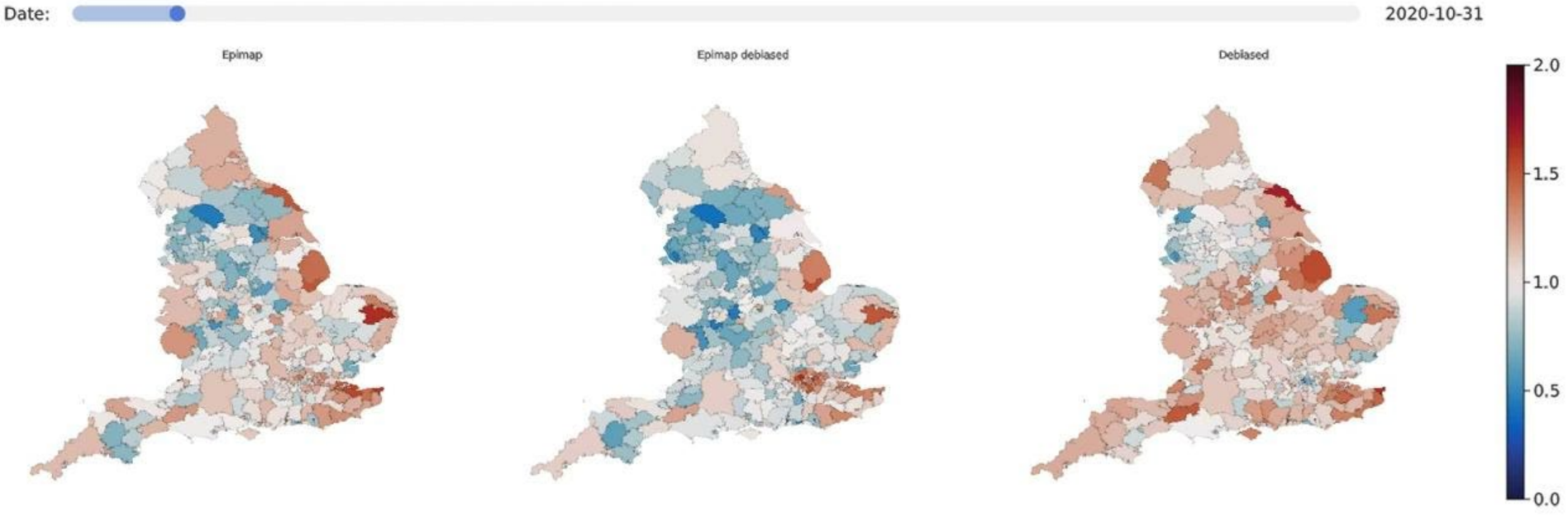
✓ Similar design philosophy as Julia: **accept everything** and **optimize when possible**

✓ Models are Julia functions, distributions are Julia, *everything* is Julia!

✗ Does mean that c**ertain operations are not possible**, e.g. automatic marginalization

✗ **Naive implementation** of model might **not** be the **most performant**

The Epimap model for COVID-19



[Nicholson et al, 2022]

Turing.jl: in the wild

Mixed IgG Fc immune complexes exhibit blended binding profiles and refine FcR affinity estimates

Zhixin Cyrillus Tan,[1] Anja Lux,[2] Markus Biburger,[2] Prabha Varghese,[2] Stephen Lees,[3] Falk Nimmerjahn,[2] and Aaron S. Meyer[1,3,4,5,*]

APPLIED RESEARCH

Bayesian Inference for Thermal Model of Synchronous Generator—Part I: Parameter Estimation

MADHUSUDHAN PANDEY AND BERNT LIE, (Member, IEEE)
Telemark Modeling and Control Center (TMCC), 3901 Porsgrunn, Norway
Corresponding author: Bernt Lie (bernt.lie@usn.no)

A Framework for Inference and Selection of Cell Signaling Pathway Dynamic Models

Marcelo Batista, Fabio Montoni, Cristiano Campos, Ronaldo Nogueira, Hugo A. Armelin & Marcelo S. Reis

Correlating tau pathology to brain atrophy using a physics-based Bayesian model

Amelie Schäfer[1] · Pavanjit Chaggar[2] · Alain Goriely[2] · Ellen Kuhl[1] · the Alzheimer's Disease Neuroimaging Initiative

Received: 1 December 2021 / Accepted: 4 April 2022 / Published online: 7 June 2022

Statistical Science
2022, Vol. 37, No. 2, 183–206
https://doi.org/10.1214/22-STS854
This research was funded, in whole or in part, by UKRI and/or Plan S sponsored funding (details on grants are at the end of the paper).

Interoperability of Statistical Models in Pandemic Preparedness: Principles and Reality

George Nicholson[+], Marta Blangiardo[+], Mark Briers[+], Peter J. Diggle[+], Tor Erlend Fjelde[+], Hong Ge[+], Robert J. B. Goudie, Radka Jersakova[+], Ruairidh E. King[+], Brieuc C. L. Lehmann[+], Ann-Marie Mallon[+], Tullia Padellini[+], Yee Whye Teh[+], Chris Holmes[+] and Sylvia Richardson[+]

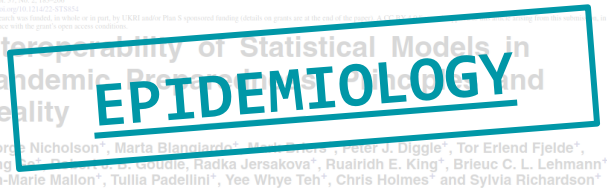
Hicks-Arrow Prices for US Federal Debt 1791-1930

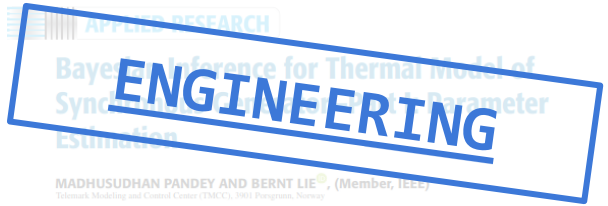
George Hall, Jonathan Payne, Thomas J. Sargent, Bálint Szőke[§]

Digital twinning of cardiac electrophysiology for congenital heart disease

Matteo Salvador [1 2 3], Fanwei Kong [2 3], Mathias Peirlinck [4], David W Parker [5], Henry Chubb [3], Anne M Dubin [3], Alison Lesley Marsden [6 1 2 3]

Turing.jl: in the wild





David Widmann

PhD (Uppsala University)
Now at Pumas AI

Hong Ge

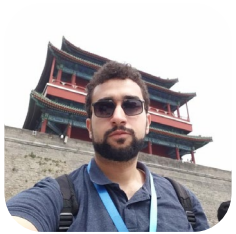
Senior Research Fellow
(University of Cambridge)

Cameron Pfiffer

PhD (University of Oregon)
Now postdoc at Stanford

Kai Xu

PhD (University of Edinburgh)
Now at MiT-IBM Watson AI Lab



Mohamed Tarek

PhD (University of Sidney)
RA (University of Sideny) and Pumas AI

Tor Erlend Fjelde

PhD student
(University of Cambridge)

Martin Trapp

Postdoc
(Aalto University)

Qingliang Zhuo

Beijing Paoding Technology Co., LTD.

Seth Axen

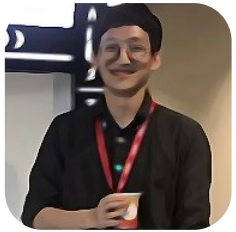
ML Research Engineer
(University of Tübingen)

Will Tebbutt

Postdoc
(University of Cambridge & ATI)

Markus Hauru

Research Engineer
(Alan Turing Institute)



Kyurae Kim

PhD student
(University of Pennsylvania)

And many other contributors to **Turing.jl** and the **Julia** ecosystem!

Thank you for listening!