Bayesian Statistics

Noric Couderc

Lund University
Department of Computer Science

7th December 2022

About Me

My name is Noric

- 5th year PhD student
- From: Digne les Bains, France
- Topics: Dynamic program analysis, performance engineering
- Supervisors: Christoph Reichenbach, Emma Söderberg

My job

My job

1. Run programs

My job

1. Run programs
2. Count stuff ⏱️

Performance Example

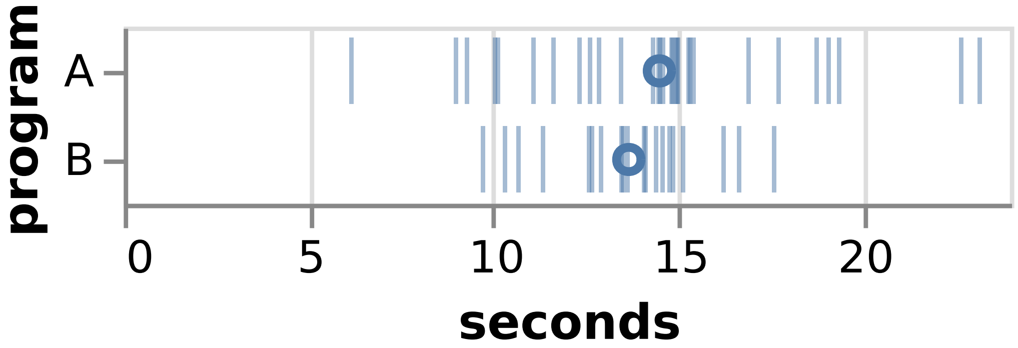
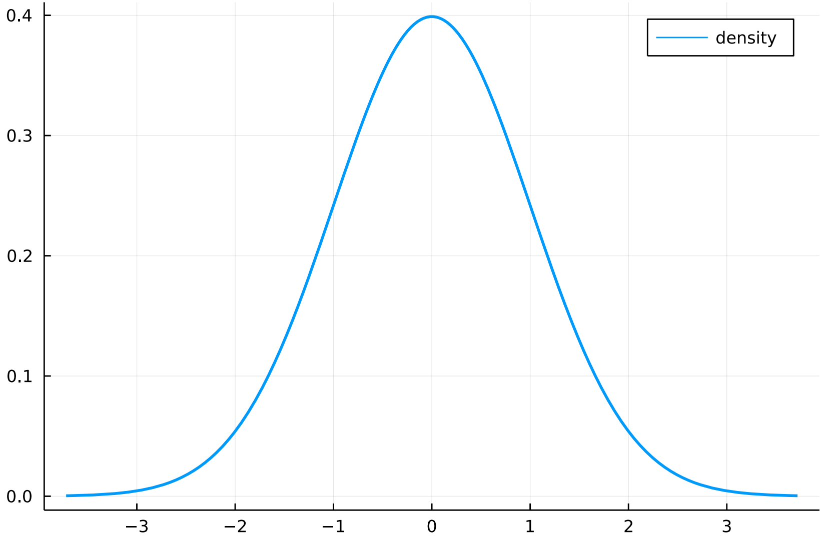


Figure: Which program is faster?

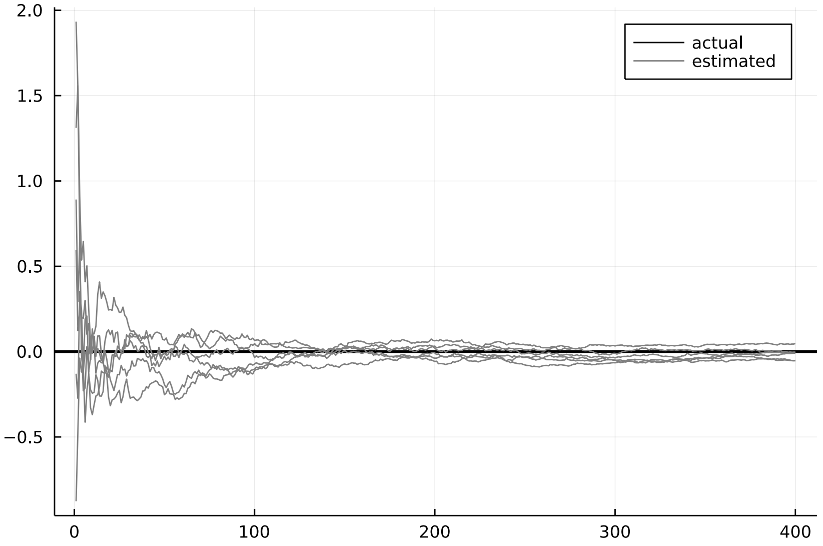
Can we trust the means we compute?

🧪 Simulate and test 🧪

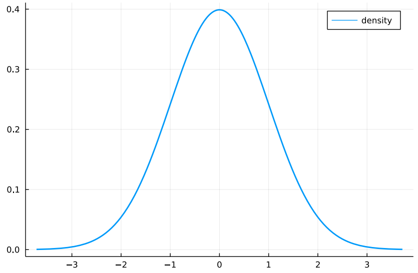
Normal Distribution

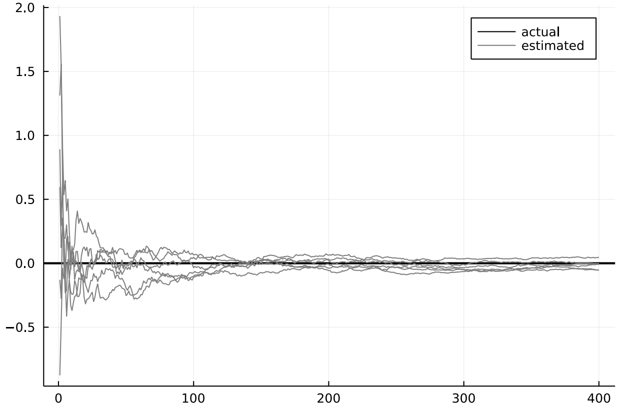


Sampling



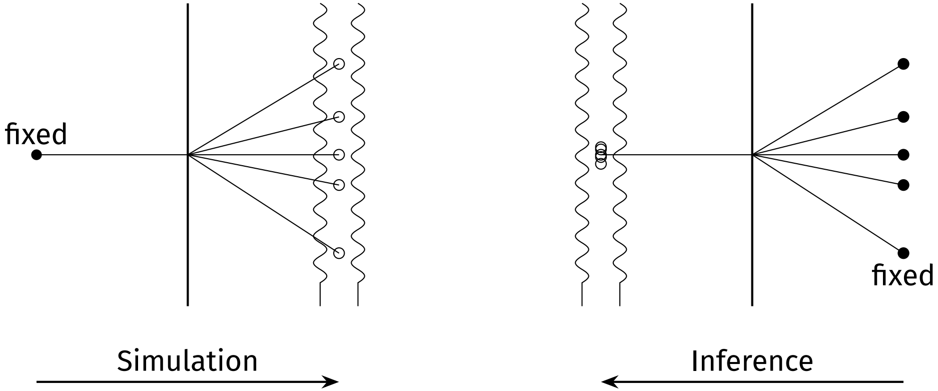
Sampling


density
0.0
0.1
0.2
0.3
0.4
−3
−2
−1
0
1
2
3


actual
estimated
2.0
1.5
1.0
0.5
0.0
−0.5
0
100
200
300
400

Statistics
Working with numbers you **can’t trust**

Figure: No Man’s Sky by Hello Games


fixed
Simulation
fixed
Inference

- **Prediction:** You know the inputs and the function, but not the outputs
- **Inference:** You know the outputs and the function, but not the inputs
- **Learning:** You know the inputs and the outputs, but not the function

Bayesian inference

Make fuzzy estimates of things

The Crush Problem

The Crush Problem

- You observed 💐.
- What is $P($❤️$|$💐$)$?

Maximum Likelihood Estimation

Likelihood

$$P(\text{observed data}|\text{hypothesis})$$

P(💐|❤️)

Maximum Likelihood

$$P(💐|❤️) = 0.3$$
$$P(💐|💔) = 0.02$$

Maximum Likelihood

$$P(\text{💐}|\text{❤️}) = 0.3$$
$$P(\text{💐}|\text{💔}) = 0.02$$

- P(💐|❤️) > P(💐|💔)

Maximum Likelihood

$$P(💐|❤️) = 0.3$$
$$P(💐|💔) = 0.02$$

- P(💐|❤️) > P(💐|💔)
- **Conclusion:** ❤️

Maximum Likelihood

$$P(\text{💐}|\text{❤️}) = 0.3$$
$$P(\text{💐}|\text{🧡}) = 0.15$$
$$P(\text{💐}|\text{💛}) = 0.075$$
$$P(\text{💐}|\text{🤍}) = 0.02$$

$$P(💐 | ❤️) \neq P(❤️ | 💐)$$

Maximum Likelihood Estimation

- Flexible!
- Returns *one single value*
- What if little data?

Bayesian Statistics

What does a probability **mean**?

Frequentist and Bayesian

Frequentist

If I flip a coin for an **infinite number of times**, half of the flips will turn heads.

Bayesian

I do not know which side the coin I will see, so I say they have **equal chances**.

Bayes' rule

$$P(B|A) \times P(A) \;=\; P(A|B) \times P(B)$$

Bayes' rule

$$\begin{tikzcd}[column sep=large]
P(A \cap B) \arrow[d, no head] & & P(B \cap A) \arrow[d, no head] \\
P(B|A) \times P(A) & = & P(A|B) \times P(B)
\end{tikzcd}$$

Bayes' Rule

$$P(❤️) \times P(💐|❤️) \;=\; P(💐) \times P(❤️|💐)$$

Bayes' Rule

$$\begin{tikzcd}
P(❤️ \cap 💐) \arrow[d, dash] & P(💐 \cap ❤️) \arrow[d, dash] \\
P(❤️) \times P(💐 | ❤️) \arrow[r, phantom, "="] & P(💐) \times P(❤️ | 💐)
\end{tikzcd}$$

Bayes' Rule

$$P(\text{❤️}|\text{💐}) = \frac{P(\text{❤️}) \times P(\text{💐}|\text{❤️})}{P(\text{💐})}$$

Bayes' Rule

$$P(\text{❤️}|\text{💐}) = \frac{P(\text{❤️}) \times P(\text{💐}|\text{❤️})}{P(\text{💐})}$$

Bayes' Rule

$$P(❤️|💐) = \frac{\overset{\text{Prior}\ \searrow}{P(❤️)} \times P(💐|❤️)}{P(💐)}$$

Bayes’ Rule

$$P(❤️|💐) = \frac{\overset{\text{Prior}}{P(❤️)} \times \overset{\text{Likelihood}}{P(💐|❤️)}}{P(💐)}$$

Bayes' Rule

$$\text{Posterior} \longrightarrow P(\text{❤️}|\text{💐}) = \frac{\overset{\text{Prior}}{\searrow} \; P(\text{❤️}) \times P(\text{💐}|\text{❤️}) \; \overset{\text{Likelihood}}{\swarrow}}{P(\text{💐})}$$

Bayes' Rule

$$\text{Posterior} \longrightarrow P(❤️|💐) = \frac{\overbrace{P(❤️)}^{\text{Prior}} \times \overbrace{P(💐|❤️)}^{\text{Likelihood}}}{\underbrace{P(💐)}_{\text{Normalization}}}$$

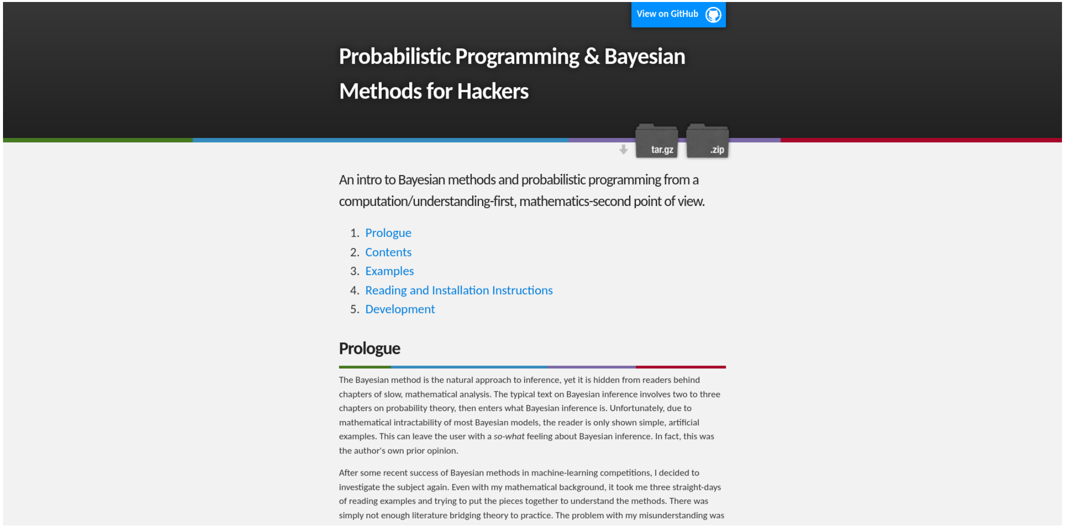
Bayes’ Rule

$$P(\text{❤️} \mid \text{💐}) \propto P(\text{❤️}) \times P(\text{💐} \mid \text{❤️})$$

Bayesian Statistics

1. Express your *beliefs* in math (priors)
2. Test them against the evidence (likelihood)
3. Update your beliefs (posteriors)

- Flexible!
- Returns distributions
- Works with little data



Probabilistic Programming & Bayesian Methods for Hackers

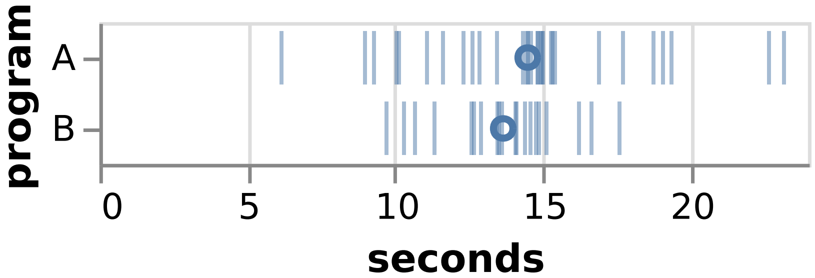
An intro to Bayesian methods and probabilistic programming from a computation/understanding-first, mathematics-second point of view.

Prologue

The Bayesian method is the natural approach to inference, yet it is hidden from readers behind chapters of slow, mathematical analysis. The typical text on Bayesian inference involves two to three chapters on probability theory, then enters what Bayesian inference is. Unfortunately, due to mathematical intractability of most Bayesian models, the reader is only shown simple, artificial examples. This can leave the user with a *so-what* feeling about Bayesian inference. In fact, this was the author's own prior opinion.

After some recent success of Bayesian methods in machine-learning competitions, I decided to investigate the subject again. Even with my mathematical background, it took me three straight-days of reading examples and trying to put the pieces together to understand the methods. There was simply not enough literature bridging theory to practice. The problem with my misunderstanding was

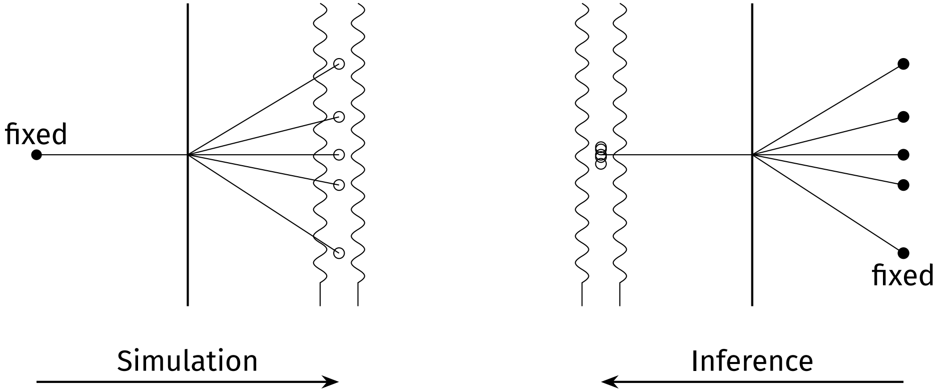
Comparing two means


program
A
B
0
5
10
15
20
seconds

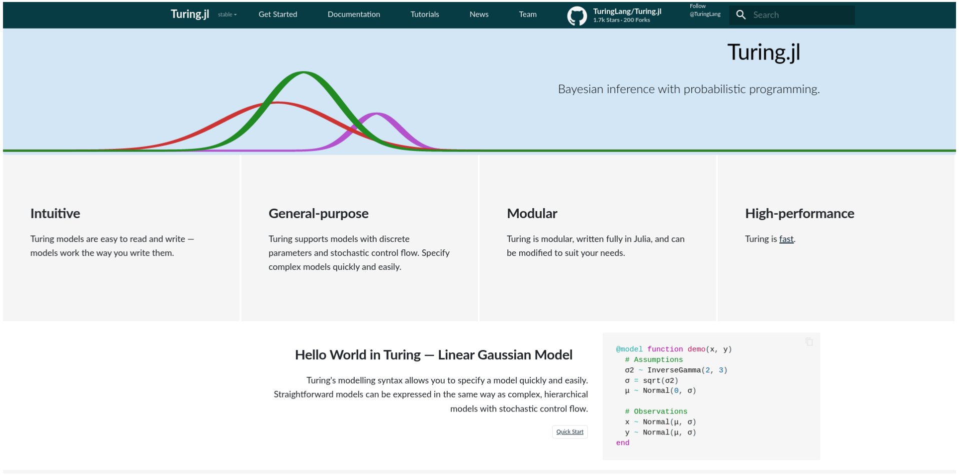
Mathematical Model

$$
\begin{aligned}
A_i &\sim \text{Normal}(\mu_A, \sigma_A) \\
B_i &\sim \text{Normal}(\mu_B, \sigma_B) \\
\mu_A &\sim \text{Normal}(10.0, 5.0) \\
\mu_B &\sim \text{Normal}(10.0, 5.0) \\
\sigma_A &\sim \text{Exponential}(1.0) \\
\sigma_B &\sim \text{Exponential}(1.0)
\end{aligned}
$$

- x Parameter
- x Observed


fixed
fixed
Simulation
Inference

```julia
function generate_samples(mean_a, mean_b, stdev_a, stdev_b)
    n_a = 30
    n_b = 20

    As = zeros(n_a)
    Bs = zeros(n_b)
    for i in 1:n_a
        As[i] = rand(Normal(mean_a, stdev_a))
    end

    for i in 1:n_b
        Bs[i] = rand(Normal(mean_b, stdev_b))
    end

    return (As,Bs)
end
```



Turing.jl

Bayesian inference with probabilistic programming.

Intuitive

Turing models are easy to read and write — models work the way you write them.

General-purpose

Turing supports models with discrete parameters and stochastic control flow. Specify complex models quickly and easily.

Modular

Turing is modular, written fully in Julia, and can be modified to suit your needs.

High-performance

Turing is fast.

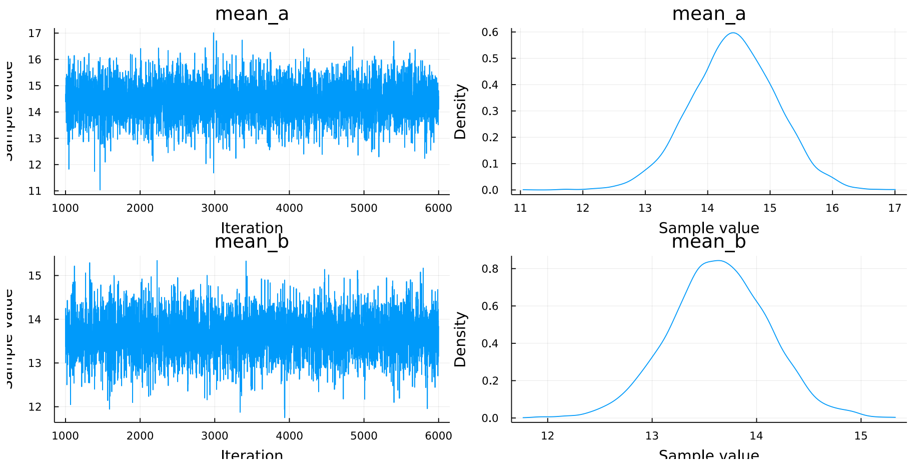
Hello World in Turing — Linear Gaussian Model

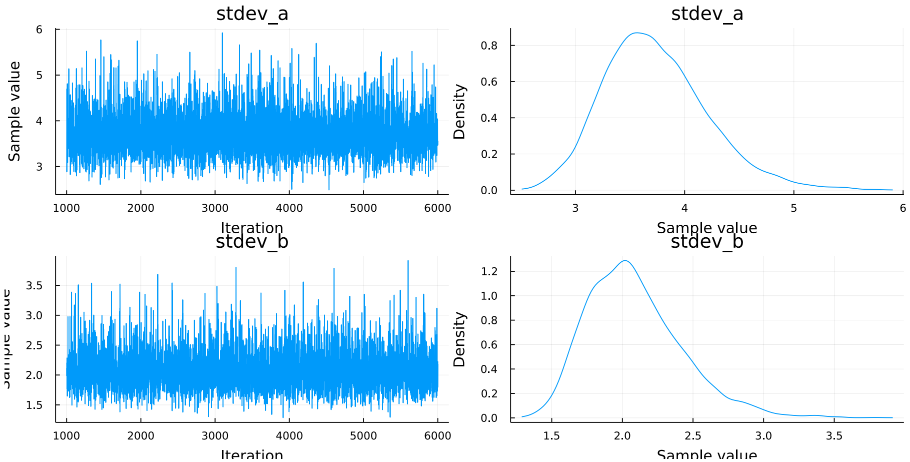
Turing's modelling syntax allows you to specify a model quickly and easily. Straightforward models can be expressed in the same way as complex, hierarchical models with stochastic control flow.

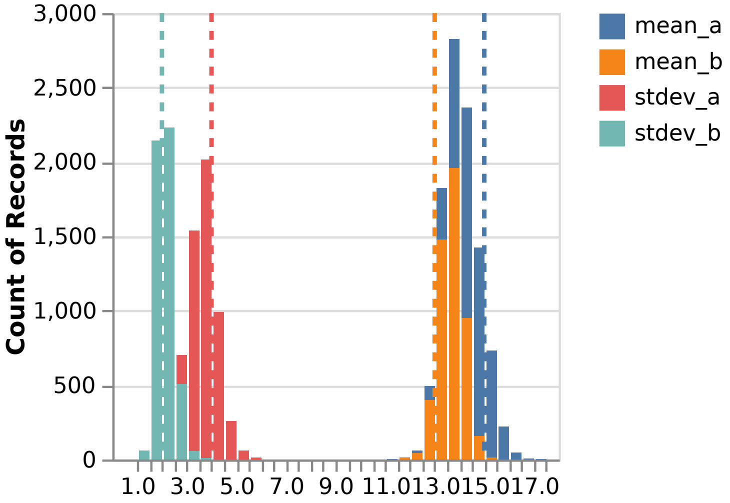
```
@model function demo(x, y)
  # Assumptions
  σ2 ~ InverseGamma(2, 3)
  σ = sqrt(σ2)
  μ ~ Normal(0, σ)

  # Observations
  x ~ Normal(μ, σ)
  y ~ Normal(μ, σ)
end
```

```julia
@model function means_model(As, Bs)
    # Priors
    mean_a ~ Normal(10.0, 5.0)
    mean_b ~ Normal(10.0, 5.0)

    stdev_a ~ Exponential(1.0)
    stdev_b ~ Exponential(1.0)

    # Likelihood
    for i in eachindex(As)
        As[i] ~ Normal(mean_a, stdev_a)
    end

    for i in eachindex(Bs)
        Bs[i] ~ Normal(mean_b, stdev_b)
    end
end
```

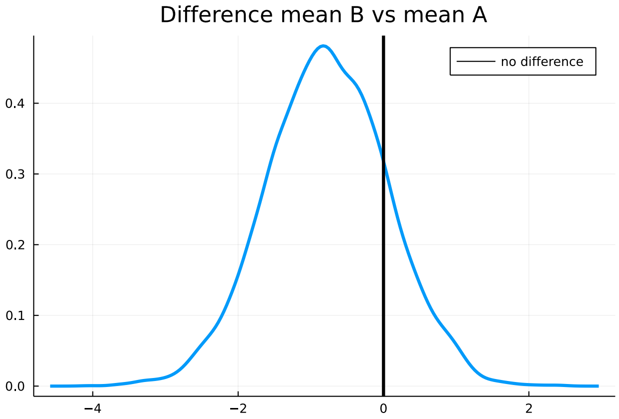
```
# Create model (pass the data to it)
model = means_model(samples_a, samples_b)

# Run bayesian inference
chain = sample(model, NUTS(0.65), 5000)

# Get posterior samples for a variable in model
chain[:mean_a]
```


mean_a
Sample value
Iteration
Density
mean_a
Sample value
mean_b
Sample value
Iteration
Density
mean_b
Sample value


stdev_a
Sample value
Iteration
Density
Sample value
stdev_b
Sample value
Iteration
Density
Sample value


3,000
2,500
2,000
1,500
1,000
500
0
Count of Records
1.0
3.0
5.0
7.0
9.0
11.0
13.0
15.0
17.0
mean_a
mean_b
stdev_a
stdev_b

```
diff = chain[:mean_b] - chain[:mean_a]
```



Posterior predictive simulation

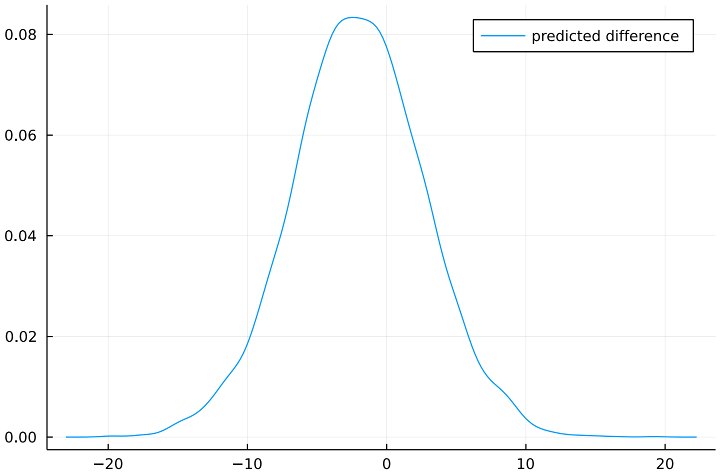
Predictions return *distributions* too!

- **Prediction:** You know the inputs and the function, but not the outputs
- **Inference:** You know the outputs and the function, but not the inputs
- **Learning:** You know the inputs and the outputs, but not the function

Posterior predictive simulation

Make **predictions** with some uncertain inputs

```julia
function generate_samples(mean_a, mean_b, stdev_a, stdev_b)
    n_a = 30
    n_b = 20

    As = zeros(n_a)
    Bs = zeros(n_b)
    for i in 1:n_a
        As[i] = rand(Normal(mean_a, stdev_a))
    end

    for i in 1:n_b
        Bs[i] = rand(Normal(mean_b, stdev_b))
    end

    return (As,Bs)
end
```


predicted difference
0.08
0.06
0.04
0.02
0.00
−20
−10
0
10
20

Linear regression

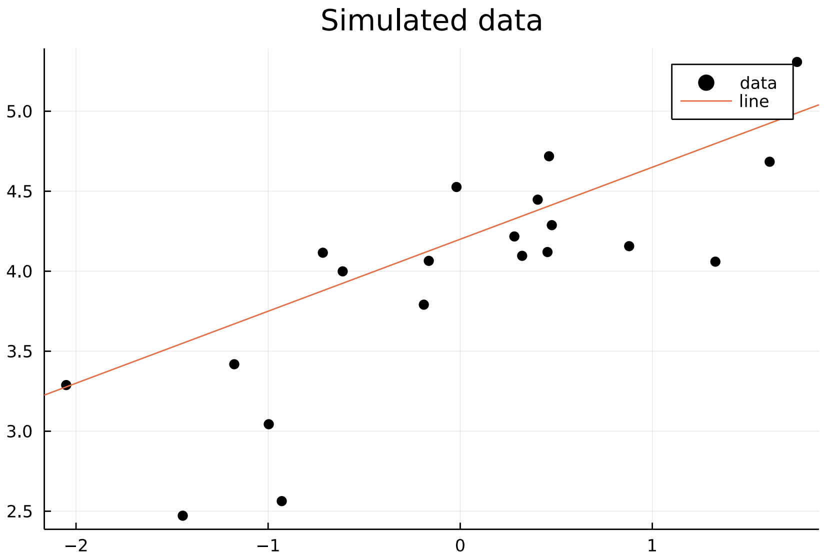


Figure: Simulated data

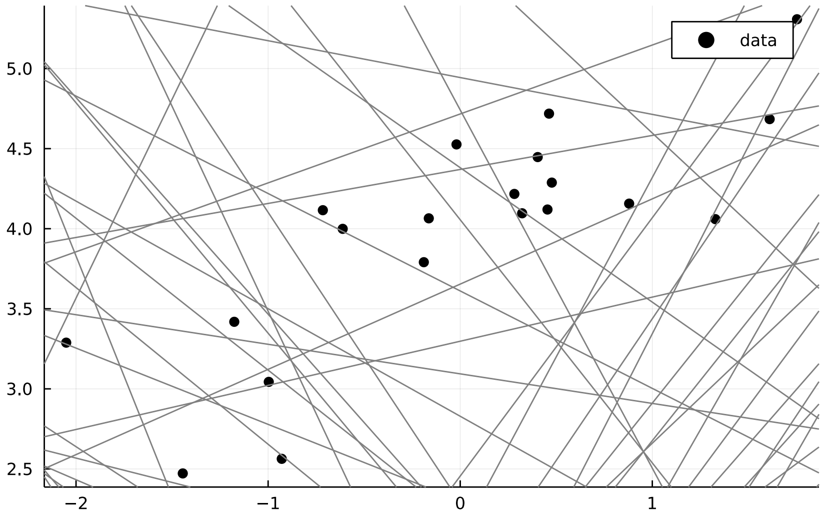
Why is linear regression useful?

- simple model, but easy to extend
- shows positive/negative associations
- interpretable

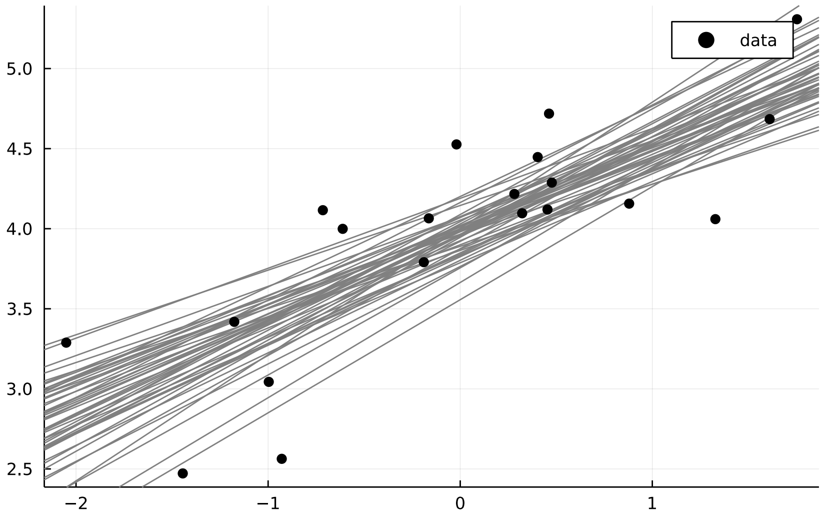
$$
\begin{aligned}
y_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= \alpha x_i + \beta \\
\alpha &\sim \text{Normal}(0, 1) \\
\beta &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Exponential}(0.2)
\end{aligned}
$$

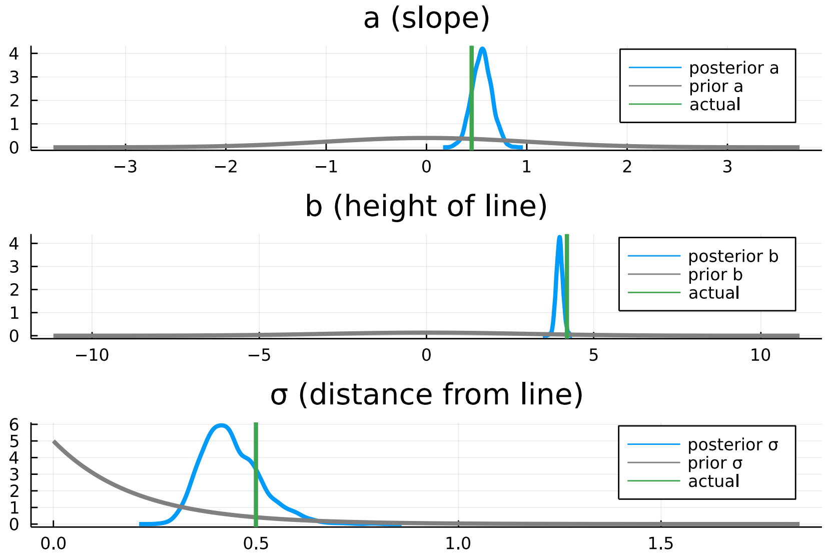
- x Parameter
- x Observed

Prior lines



Posterior lines

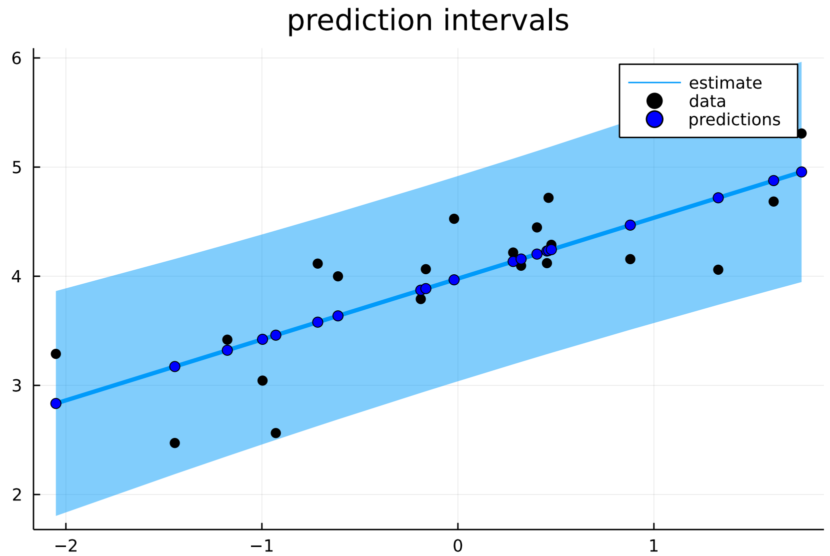



a (slope)
posterior a
prior a
actual
b (height of line)
posterior b
prior b
actual
σ (distance from line)
posterior σ
prior σ
actual

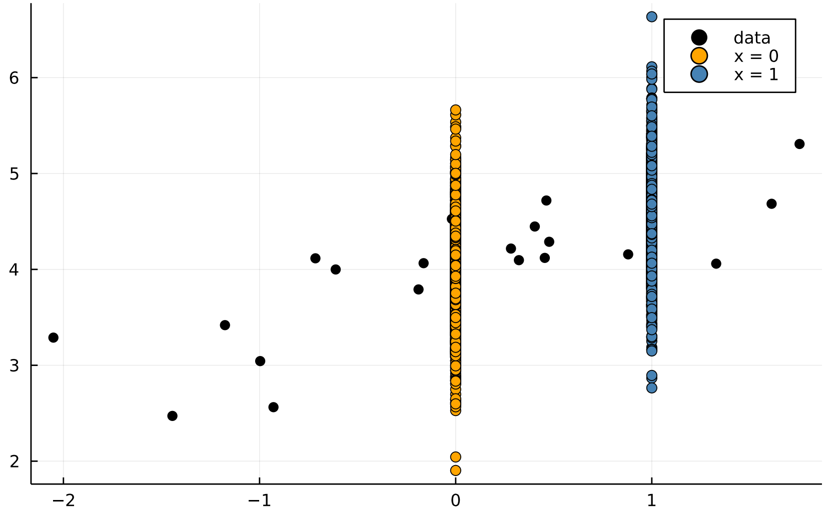
$$\begin{tikzcd}
 & \text{Observations} \arrow[dr] & & \text{Structure} \arrow[dl] \\
\text{Prior(s)} \arrow[dr] & & \text{Likelihood} \arrow[dl] & \\
 & \text{Posterior(s)} & &
\end{tikzcd}$$

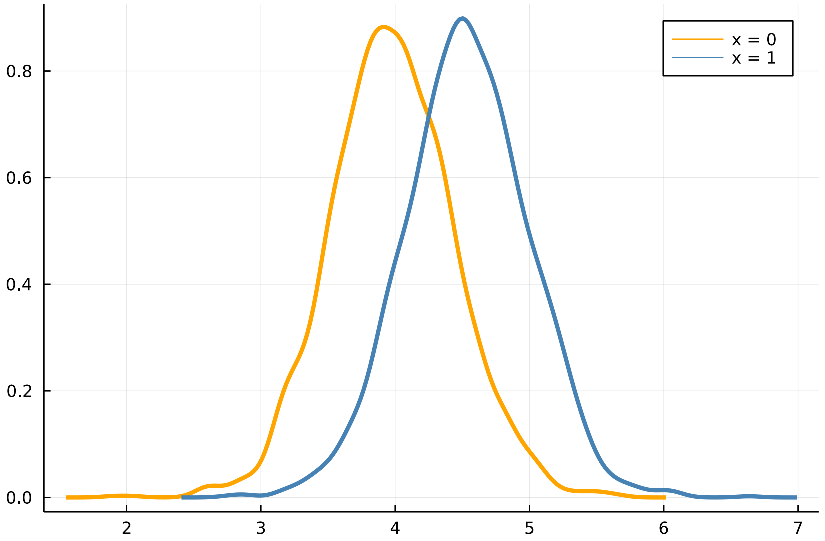
Posterior predictive simulation

Predictions return *distributions* too!

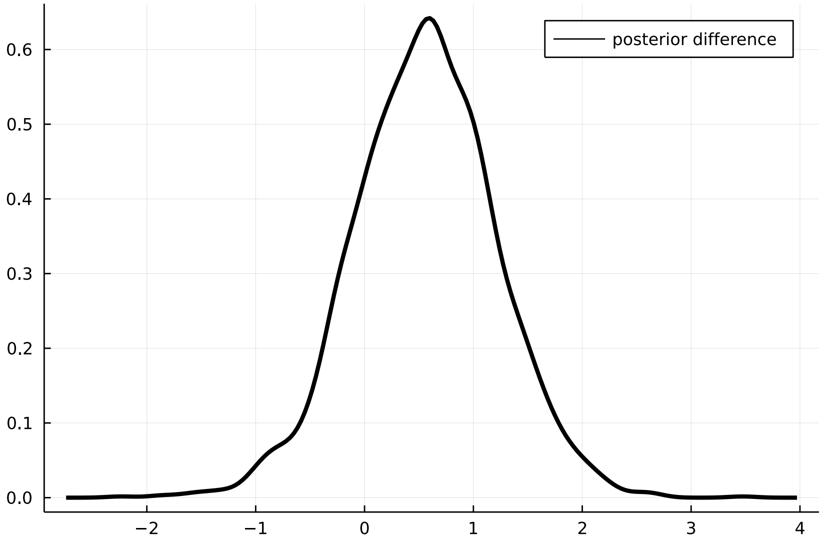

prediction intervals
estimate
data
predictions
6
5
4
3
2
−2
−1
0
1

Posterior prediction





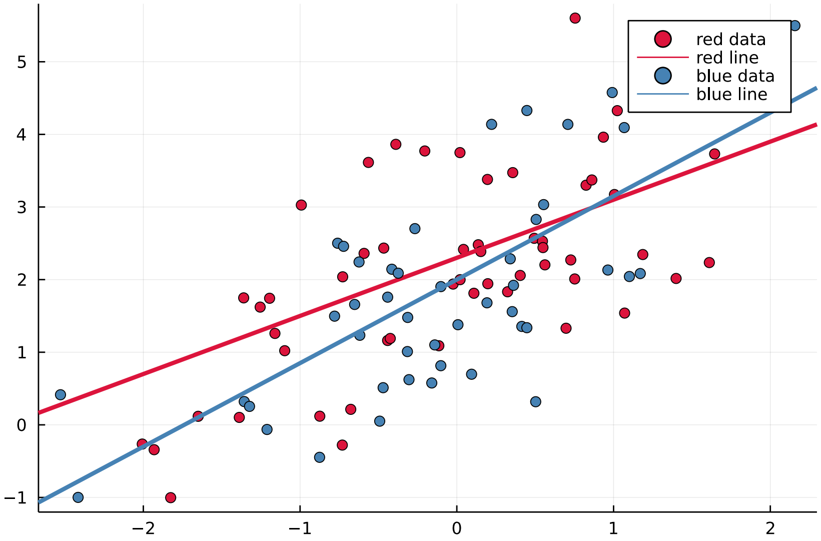
0.8
0.6
0.4
0.2
0.0
2
3
4
5
6
7
x = 0
x = 1


posterior difference
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−2
−1
0
1
2
3
4

Interactions

Examples of interactions

- Plants need both **light** and **water** to grow.
- To have sweet coffee you need to **add sugar** and **stir** the coffee.
- **Some optimizations** only work on **some CPUs**.


red data
red line
blue data
blue line
5
4
3
2
1
0
−1
−2
−1
0
1
2

```julia
@model function interaction_model(N, xs, ys, teams)
    a ~ filldist(Normal(0.0, 0.5), 2)

    b ~ filldist(Normal(0.0, 0.5), 2)

    sigma ~ Exponential(1.0)

    for i in 1:N
        if teams[i] == 1.0
            mu_1 = a[1] * xs[i] + b[1]
            ys[i] ~ Normal(mu_1, sigma)
        else
            mu_2 = a[2] * xs[i] + b[2]
            ys[i] ~ Normal(mu_2, sigma)
        end
    end
end
```

Analysing Experiments

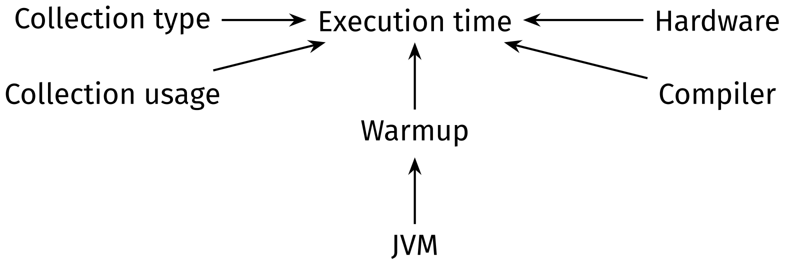
How to make my programs faster?

How to make my programs faster?
It **depends**

How to make my programs faster?
It **depends**
Depends on what?

$$\begin{tikzcd}
\text{Collection type} \arrow[r] & \text{Execution time} & \text{Hardware} \arrow[l] \\
\text{Collection usage} \arrow[ur] & & \text{Compiler} \arrow[ul] \\
& \text{Runtime} \arrow[uu] &
\end{tikzcd}$$

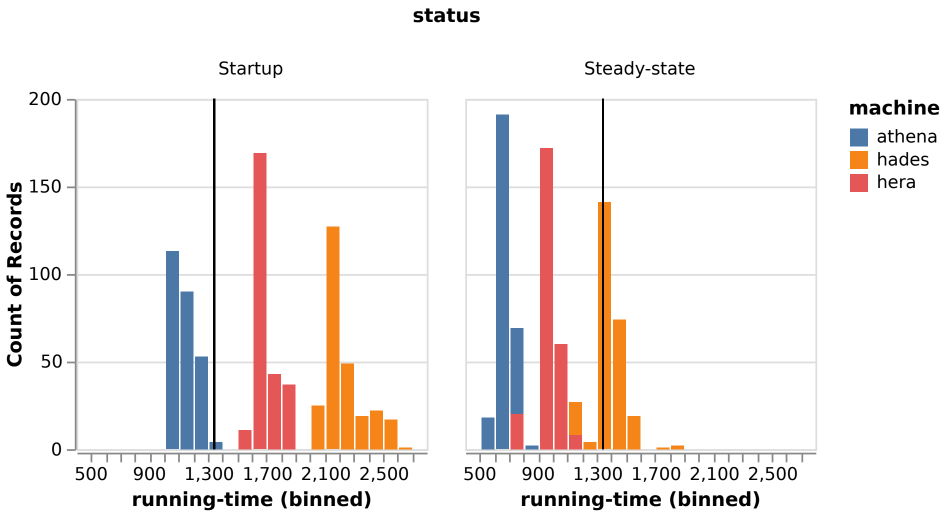
$$\begin{tikzcd}
\text{Collection type} \arrow[r] & \text{Execution time} & \text{Hardware} \arrow[l] \\
\text{Collection usage} \arrow[ur] & & \text{Compiler} \arrow[ul] \\
& \text{JIT-compilation} \arrow[uu] & \\
& \text{Runtime} \arrow[u] &
\end{tikzcd}$$


Collection type
Execution time
Hardware
Collection usage
Compiler
Warmup
JVM

$$\begin{tikzcd}
\text{Collection type} \arrow[r] & \text{Execution time} & \text{Hardware} \arrow[l] \\
 & \text{JVM Warmup} \arrow[u] &
\end{tikzcd}$$

$$\begin{tikzcd} \text{Collection type} \arrow[r] & \text{Execution time} & \text{Hardware} \arrow[l] \\ & \text{JVM Warmup} \arrow[u] & \end{tikzcd}$$

- **Machines:** Athena, Hades, Hera
- **Warmup:** Startup (cold JVM), Steady-state (warm JVM)
- **Treatments:** 0->arraylist, 1->arraymap, ...
  - Allocation site: 0, 1, 2, ...
  - Collection: hashmap, arraylist, ...


status
Startup
Steady-state
Count of Records
200
150
100
50
0
500
900
1,300
1,700
2,100
2,500
running-time (binned)
machine
athena
hades
hera

$$
\begin{aligned}
ET_j &\sim \text{Normal}(\mu_j, \sigma) \\
\mu_j &= \overline{ET} \times \alpha[M_j] \times \beta[W_j] \times \gamma[T_j] \\
\alpha_m &\sim \text{Normal}(0.0, 0.5) \\
\beta_w &\sim \text{Normal}(0.0, 0.2) \\
\gamma_t &\sim \text{Normal}(0.0, 0.2) \\
\sigma &\sim \text{Exponential}(30.0)
\end{aligned}
$$

- x Parameter
- x Observed

$$
\begin{aligned}
ET_j &\sim \text{Normal}(\mu_j, \sigma) \\
\mu_j &= \overline{ET} \times \alpha[M_j] \times \beta[W_j] \times \gamma[T_j] \longleftarrow \text{Array access} \\
\alpha_m &\sim \text{Normal}(0.0, 0.5) \\
\beta_w &\sim \text{Normal}(0.0, 0.2) \\
\gamma_t &\sim \text{Normal}(0.0, 0.2) \\
\sigma &\sim \text{Exponential}(30.0)
\end{aligned}
$$

- x Parameter
- x Observed

$$ET_j \sim \text{Normal}(\mu_j, \sigma)$$

$$\mu_j = \overline{ET} \times \alpha[M_j] \times \beta[W_j] \times \gamma[T_j] \longleftarrow \text{Array access}$$

$$\text{Arrays} \longrightarrow \alpha_m \sim \text{Normal}(0.0, 0.5)$$

$$\text{Arrays} \longrightarrow \beta_w \sim \text{Normal}(0.0, 0.2)$$

$$\gamma_t \sim \text{Normal}(0.0, 0.2)$$

$$\sigma \sim \text{Exponential}(30.0)$$

- x Parameter
- x Observed

$$\log(ab) = \log(a) + \log(b)$$

$$
\begin{aligned}
\log(ET_j) &\sim \text{Normal}(\mu_j, \sigma) \\
\mu_j &= \log(\overline{ET}) + \alpha[M_j] + \beta[W_j] + \gamma[T_j] \\
\alpha_m &\sim \text{Normal}(0.0, 0.5) \\
\beta_w &\sim \text{Normal}(0.0, 0.2) \\
\gamma_t &\sim \text{Normal}(0.0, 0.2) \\
\sigma &\sim \text{Exponential}(1.0)
\end{aligned}
$$

- x Parameter
- x Observed

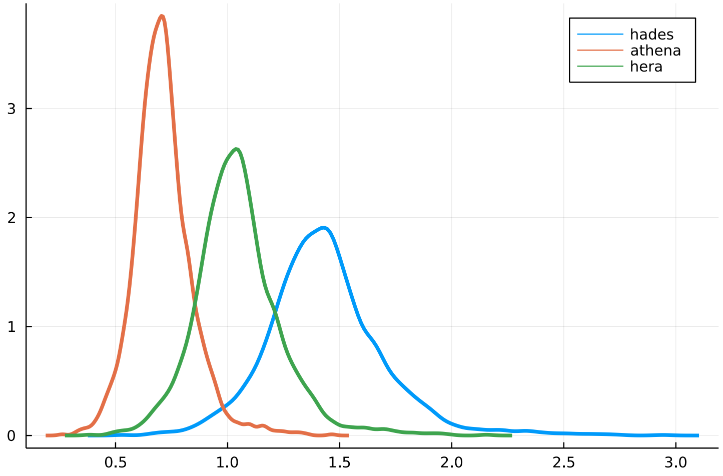


Figure: Effect of machine

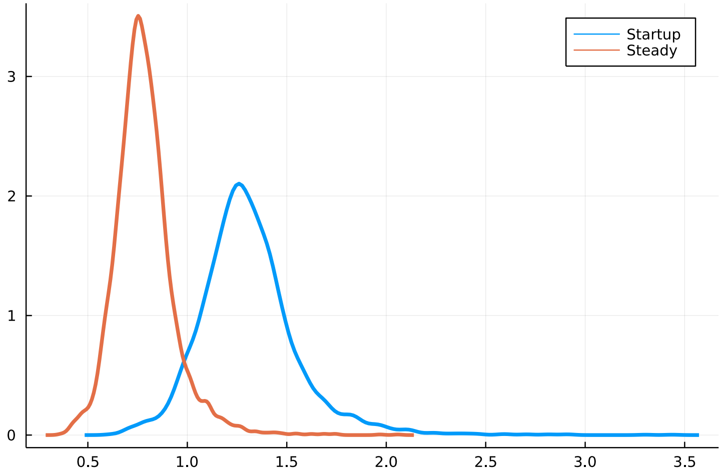


Figure: Effect of warmup

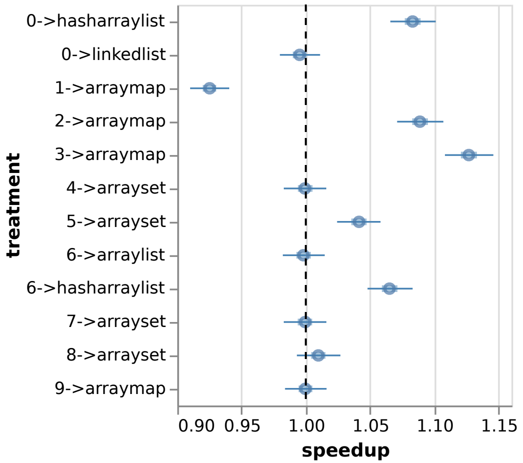


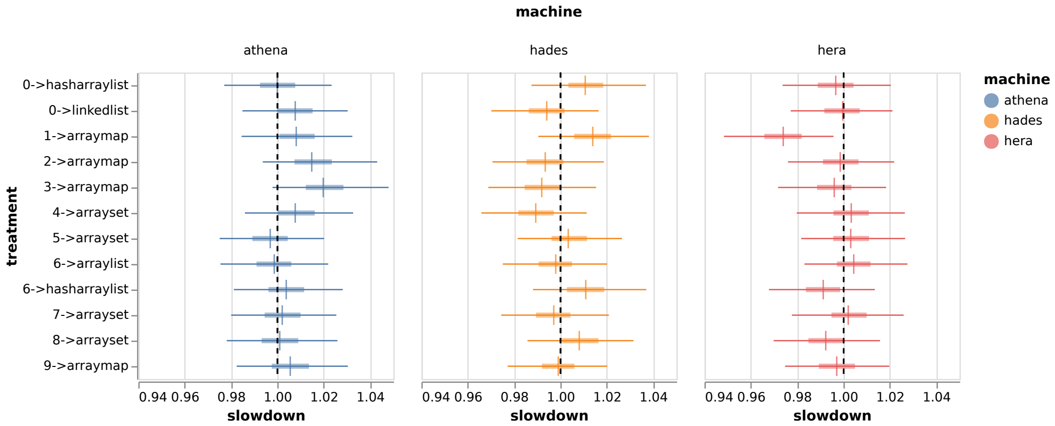
Figure: Effect of treatment

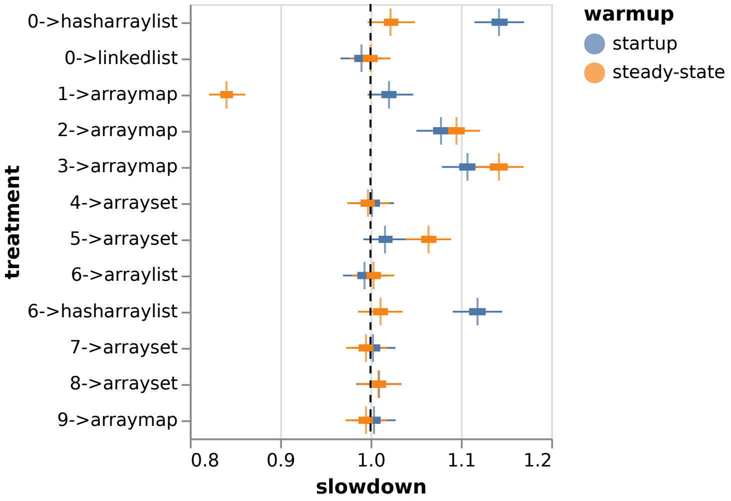
(Discrete) Interactions

Transform an **array** of coefficients into a **matrix**

$$
\begin{aligned}
\log(ET_j) &\sim \text{Normal}(\mu_j, \sigma) \\
\mu_j &= \log(\overline{ET}) + \alpha[M_j] + \beta[W_j] + \gamma^M[M_j, T_j] + \gamma^W[W_j, T_j] \\
\alpha_m &\sim \text{Normal}(0.0, 0.5) \\
\beta_w &\sim \text{Normal}(0.0, 0.2) \\
\gamma^W_{w,t} &\sim \text{Normal}(0.0, 0.2) \\
\gamma^M_{m,t} &\sim \text{Normal}(0.0, 0.2) \\
\sigma &\sim \text{Exponential}(1.0)
\end{aligned}
$$

- x Parameter
- x Observed


machine
athena
hades
hera
treatment
0->hasharraylist
0->linkedlist
1->arraymap
2->arraymap
3->arraymap
4->arrayset
5->arrayset
6->arraylist
6->hasharraylist
7->arrayset
8->arrayset
9->arraymap
0.94 0.96 0.98 1.00 1.02 1.04
slowdown
machine
● athena
● hades
● hera


warmup
startup
steady-state
treatment
0->hasharraylist
0->linkedlist
1->arraymap
2->arraymap
3->arraymap
4->arrayset
5->arrayset
6->arraylist
6->hasharraylist
7->arrayset
8->arrayset
9->arraymap
0.8
0.9
1.0
1.1
1.2
slowdown

Hierarchical Models

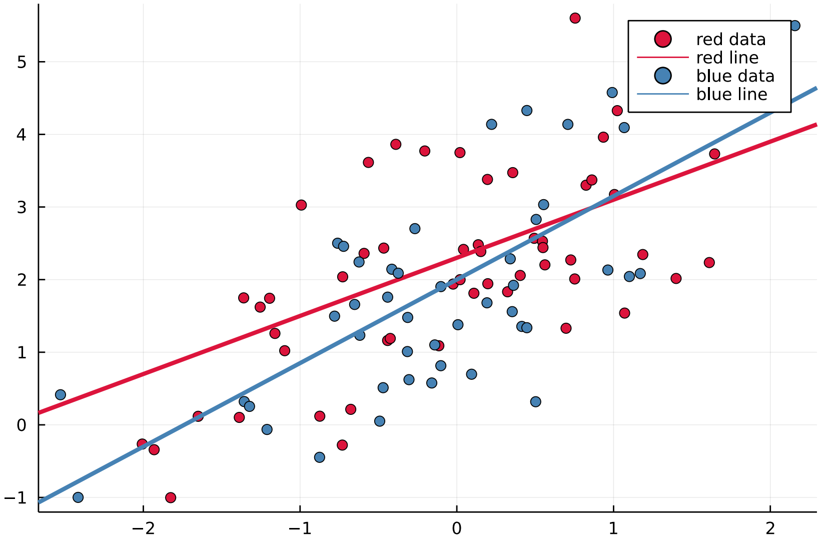
Hyperpriors

Priors can have parameters too

Hyperpriors

Priors can have parameters too

🤯


red data
red line
blue data
blue line
5
4
3
2
1
0
−1
−2
−1
0
1
2

```julia
@model function interaction_model(N, xs, ys, teams)
    a ~ filldist(Normal(0.0, 0.5), 2)

    b ~ filldist(Normal(0.0, 0.5), 2)

    sigma ~ Exponential(1.0)

    for i in 1:N
        if teams[i] == 1.0
            mu_1 = a[1] * xs[i] + b[1]
            ys[i] ~ Normal(mu_1, sigma)
        else
            mu_2 = a[2] * xs[i] + b[2]
            ys[i] ~ Normal(mu_2, sigma)
        end
    end
end
```

```julia
@model function interaction_model(N, xs, ys, teams)
    sigma_slopes ~ Exponential(0.1)
    a ~ filldist(Normal(0.0, sigma_slopes), 2)

    b ~ filldist(Normal(0.0, 0.5), 2)

    sigma ~ Exponential(1.0)

    for i in 1:N
        if teams[i] == 1.0
            mu_1 = a[1] * xs[i] + b[1]
            ys[i] ~ Normal(mu_1, sigma)
        else
            mu_2 = a[2] * xs[i] + b[2]
            ys[i] ~ Normal(mu_2, sigma)
        end
    end
end
```

Why hyperpriors

- Prevent overfitting
- Use *all* the data

Conclusion

Bayesian inference is:

- **intuitive:** Use probability distributions, no need to learn specific vocabulary
- **flexible:**
    - Use the distributions you want
    - Use the code you want
- **honest:** If not enough data, you get the priors back
- **explicit:** All your assumptions are visible

Bayesian inference is:

- **intuitive:** Use probability distributions, no need to learn specific vocabulary
- **flexible:**
  - Use the distributions you want
  - Use the code you want
- **honest:** If not enough data, you get the priors back
- **explicit:** All your assumptions are visible
- Sometimes **slow**

My Stats Workflow

1. Generate some fake data (makes you think about what causes what)
2. Make a statistical model
3. Test if the model can recover parameters
4. Use it on experimental data

Toys to Play With

- Stan (Standalone DSL)
- Turing (Julia)
- Pyro (Python, supported by Uber)
- Bean Machine (Python, supported by Meta)
- Tensorflow Probability (Python, supported by Google)
- PyMC3 (Python)

Books to Read

- Bayesian Methods for Hackers
- Statistical Rethinking (McElreath)
- Causal Inference In Statistics, a primer (Pearl, Glymour, Jewell)

Thanks!

Extra Slides

Prior Sensitivity Analysis

The Idea

Try out different priors, and check how predictions change!

Causal Inference

Correlation isn’t causation

(But then what is causation?)

Causation

X causes Y if Y relies on X for its value

How can correlation appear?

- X causes Y
- Y causes X
- X and Y share common cause Z (Confounding)
- Selection bias (Colliders)

Direct causes

$$\begin{tikzcd} X \arrow[r] & Y \end{tikzcd}$$

$$\begin{tikzcd} X & Y \arrow[l] \end{tikzcd}$$

Confounding

$$\begin{tikzcd}\text{Tobacco} \arrow[r, "?"] & \text{Cancer}\end{tikzcd}$$

Confounding

$$\begin{tikzcd}
 & \text{Genetic Factors} \arrow[dl] \arrow[dr] & \\
\text{Tobacco} \arrow[rr, "?"] & & \text{Cancer}
\end{tikzcd}$$

Confounder (or Fork)

$$\begin{tikzcd} & Z \arrow[dl] \arrow[dr] & \\ X & & Y \end{tikzcd}$$

Collider

$$\begin{tikzcd}X \arrow[dr] & & Y \arrow[dl] \\ & Z & \end{tikzcd}$$