Chapter 4: Statistical uncertainty

UE Computational statistics

Pierre Pudlo

Aix-Marseille Université / Faculté des Sciences

About this chapter

Objectives:

- How the computer can be harness to obtain measures of statistical uncertainty such as standard errors, confidence intervals, etc.

Content:

- Fisher and the maximum likelihood estimator (MLE)
- The bootstrap
- MCMC methods to approximate the posterior distribution in Bayesian statistics

> ⚠ **Warning**
>
> This chapter is **much longer** than the first chapters.

1. Introduction

Statistical uncertainty

Assess the accuracy or the uncertainty of something computed **from data**: estimate of a parameter, predictions on new individuals, etc.

In linear regression, e.g., the response of individual i is

$$Y_i = \beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip} + \varepsilon_i$$

where ε_i is the error term, independent of the covariates X_{ij}'s, with $\mathbb{E}(\varepsilon_i) = 0$ and $\mathbb{V}(\varepsilon_i) = \sigma^2$.

- Estimation uncertainty: true β_j's $\neq$ estimates,…
- Prediction uncertainty: exact prediction impossible because $\sigma^2 > 0$

1.1 Statistics (revisited): the statistical model

- **All observed numbers are modeled by their own random variables** (rv's)
- In **linear regression**, e.g., we introduce many rv's:

$$D = \begin{pmatrix} Y_1 & X_{11} & \cdots & X_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ Y_n & X_{n1} & \cdots & X_{np} \end{pmatrix}$$

  - The individuals $D_i = (Y_i \; X_{i1} \; \cdots \; X_{ip})$ are independent and identically distributed (iid) random vectors
  - We might introduce a new individual $D_{n+1} = (Y_{n+1} \; X_{n+1,1} \; \cdots \; X_{n+1,p})$ and assume that Y_{n+1} is not observed

- In **logistic regression**, the dataset is modeled likewise, but the response Y_i is binary, i.e., $Y_i \in \{0, 1\}$ and we assume that

$$[Y_i|X_i] \sim \text{Bernoulli}\Big(p(X_i)\Big), \quad \text{with } p(X_i) = \frac{1}{1 + \exp\left(-\beta_0 - \sum_j \beta_j X_{ij}\right)}.$$

- In a **Poisson regression**, the response Y_i is a count, i.e., $Y_i \in \{0, 1, 2, \ldots\}$ and we assume that

$$[Y_i|X_i] \sim \text{Poisson}\Big(\lambda(X_i)\Big), \quad \text{with } \lambda(X_i) = \exp\left(\beta_0 + \sum_j \beta_j X_{ij}\right).$$

- In a **Gaussian mixture model**, the model $Y_1, \ldots, Y_n$ is composed of n iid random vectors and we assume that

$$Y_i \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k), \quad \text{with } \sum_{k=1}^{K} \pi_k = 1.$$

1.2 Frequentist statistics

- In the frequentist paradigm, the parameters θ of the statistical model are fixed but **unknown**
  - E.g., $\beta_0, \ldots, \beta_p$ in regression problems, $\sigma^2 > 0$ in linear regression,
  - π_k's, μ_k's and Σ_k's in the Gaussian mixture model, etc.
- The observed dataset d is a realization of the random dataset D with distribution P_θ
- Sometimes, rv's are **not observed**, e.g., Y_{n+1} in linear regression
- Two type of problems:
  - **Estimation**: approximate the unknown parameters θ or a function of the unknown parameters $\psi = g(\theta)$ (E.g., $\psi = \beta_0$ in linear regression, or $\psi = \beta_0 + \beta_1 x_{n+1,1} + \cdots + \beta_p x_{n+1,p}$ when $x_{n+1,j}$'s are known)
  - **Prediction**: approximate the unobserved value of a rv (E.g., Y_{n+1} in linear regression) or its distribution

- The **uncertainty** arises from the fact that the observed dataset d is a single realization of the random dataset D
- **Estimation procedures** are functions of the observed dataset d, and maybe a tuning parameters λ.
  - E.g., the LASSO estimate of $\beta = (\beta_0, \ldots, \beta_p)$ in linear regression is

$$\widehat{\beta}(\lambda) = \underset{b \in \mathbb{R}^{p+1}}{\operatorname{argmin}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(y_i - b_0 - \sum_{j=1}^{p} b_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |b_j| \right]$$

  - Such procedures can be seen as a **deterministic function** of the observed dataset d and the tuning parameter λ:

$$\widehat{\beta}(\lambda) = s(d, \lambda)$$

    where s is the function that implements the LASSO algorithm in Python, R, etc.

Uncertainty measures of a parameter estimate

- **Standard errors**: an estimate of the standard deviation of $s(D, \lambda)$
- **Confidence intervals**: range of values that are likely to contain the true value of a parameter

- To get formula for uncertainty measures, we need to resort to **probability calculus** to understand the distribution of $s(D, \lambda)$ when $D \sim P_\theta$.
- Moreover, we need **the distribution whatever the unknown value of the parameter** θ is, i.e., we need to consider the distribution of $s(D, \lambda)$ when $D \sim P_\theta$ for all θ.
- Often in the frequentist paradigm,
  - the **estimator** $s(D, \lambda)$ is denoted $\widehat{\Theta}$ and
  - its observed value $\hat{\theta} = s(d, \lambda)$ is the **estimate**.

- How to **compute the distribution** of $s(D, \lambda)$?
  - Very specific cases: the distribution of $s(D, \lambda)$ can be **computed exactly** (Student, Fisher, Gaussian linear model,…)
  - Asymptotic results: the distribution of $s(D, \lambda)$ can be **approximated** by a known distribution when n is large…
  - Monte Carlo methods: **the Bootstrap**, see next Section

Consider now that we want to estimate $\psi = g(\theta)$, and a statistic $s(D, \lambda)$ that estimates ψ:

- ψ is a given coordinate of θ
- or $\psi = \beta_0 + \sum_j \beta_j x_{n+1,j}$ in (generalized) linear models
- or …

Confidence intervals

- A confidence interval of level $(1 - \alpha)$ is an interval $[a(D), b(D)]$ whose bounds are two statistics such that

$$\forall \theta, \quad \mathbb{P}_\theta\Big(a(D) \leq g(\theta) \leq b(D)\Big) \geq 1 - \alpha.$$

- For a large proportion of datasets D drawn from the model, the true value of the parameter is in the confidence interval
- The confidence interval is a **random interval** that depends on the dataset D

- Once D is fixed to d, $a(d) \leq g(\theta) \leq b(d)$ **is either true or false**, but is no longer an event with a probability $\in (0, 1)$
- We **have to pray** that it is true for the observed dataset d! (We do not even know whether the data have been generated by the model or not)

1.3 Bayesian statistics

- In the Bayesian paradigm, **the parameters** of the statistical model **are also random variables**
  - E.g., $\beta_0, \ldots, \beta_p$ in regression problems, and additionally $\sigma^2 > 0$ in linear regression,
  - π_k's, μ_k's and Σ_k's in the Gaussian mixture model, etc.
- The **randomness of the parameters** is a way of **modelling** the uncertainty about the values of the parameters, in other word **our belief about their values**
- To stress that the parameters are now rv's, we denote them Θ instead of θ
- Likewise, the observed dataset d is a realization of the random dataset D
- The **statistical model** is now the conditional distribution of the data given the parameter: $[D|\Theta = \theta] \sim P_\theta$

- The **prior distribution** of the parameter:
  - is the marginal distribution of $\left[\Theta\right]$:
  - models our **belief** about the values of the parameters **before observing the data**
- The **posterior distribution** of the parameter is the conditional distribution of the parameter given the data:
  - it is the distribution of $\left[\Theta \middle| D = d\right]$
  - it takes into account the **actual data** d we have observed **to update our belief** about the values of the parameters
  - models our **belief** about the values of the parameters **after observing the data**

In the continuous case,

- the **prior density**: $\pi(\theta)$
- the **likelihood**: $f(d|\theta)$
- the **posterior density**:

$$\pi(\theta|d) = \frac{\pi(\theta) f(d|\theta)}{f_\pi(d)} \propto \pi(\theta) f(d|\theta), \quad \text{where } f_\pi(d) = \int \pi(\theta') f(d|\theta') \, \mathrm{d}\theta'$$

- the **marginal likelihood**, or the **evidence** $f_\pi(d)$
  - is the normalizing constant of the posterior distribution
  - is difficult to compute in general
  - is the probability of observing d under the Bayesian model (prior + likelihood)
  - can be used for Bayesian model selection, or for averaging over models

- The posterior distribution (our belief after the data) can be summarized:
  - the **posterior mean**: $\mathbb{E}\left[\Theta \middle| D = d\right]$
  - the **posterior variance**: $\mathbb{V}\left[\Theta \middle| D = d\right]$ and standard deviation
  - the **posterior median**, and other quantiles
- The posterior mean and median can serve as **estimates** of the true parameter
- The posterior variance, standard deviation and quantiles can serve as **measures of uncertainty** of the estimate
- The **main difficulties in Bayesian** statistics are:
  - **set the prior** to reflect our prior belief
  - **compute the posterior** distribution

Basic example

The Beta distribution $B(\alpha, \beta)$

- The Beta distribution is a continuous distribution on the interval $(0, 1)$
- Has density function

$$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)} \quad \text{where } B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

- If $X \sim B(\alpha, \beta)$, then

$$\mathbb{E}[X] = \frac{\alpha}{\alpha+\beta}, \quad \mathbb{V}[X] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

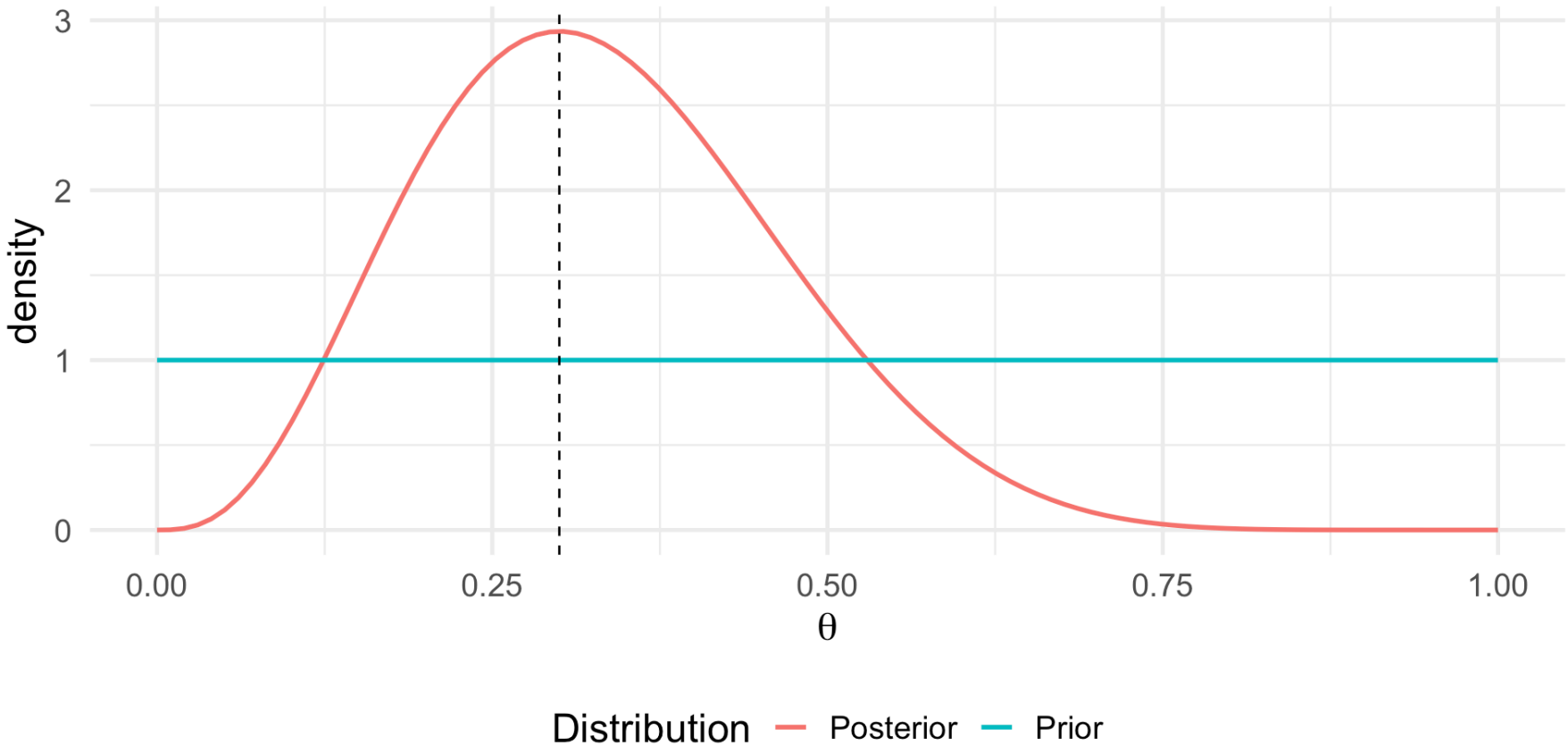
- $[D|\Theta = \theta] \sim \mathbf{Binomial}(n, \theta)$ with unknown $\theta \in (0, 1)$
- The likelihood of θ is

$$f(d|\theta) = \binom{n}{y} \theta^d (1-\theta)^{n-d}$$

- The **prior** distribution of Θ is $B(\alpha, \beta)$
- The **posterior** distribution of Θ is $B(\alpha + d, \beta + n - d)$:

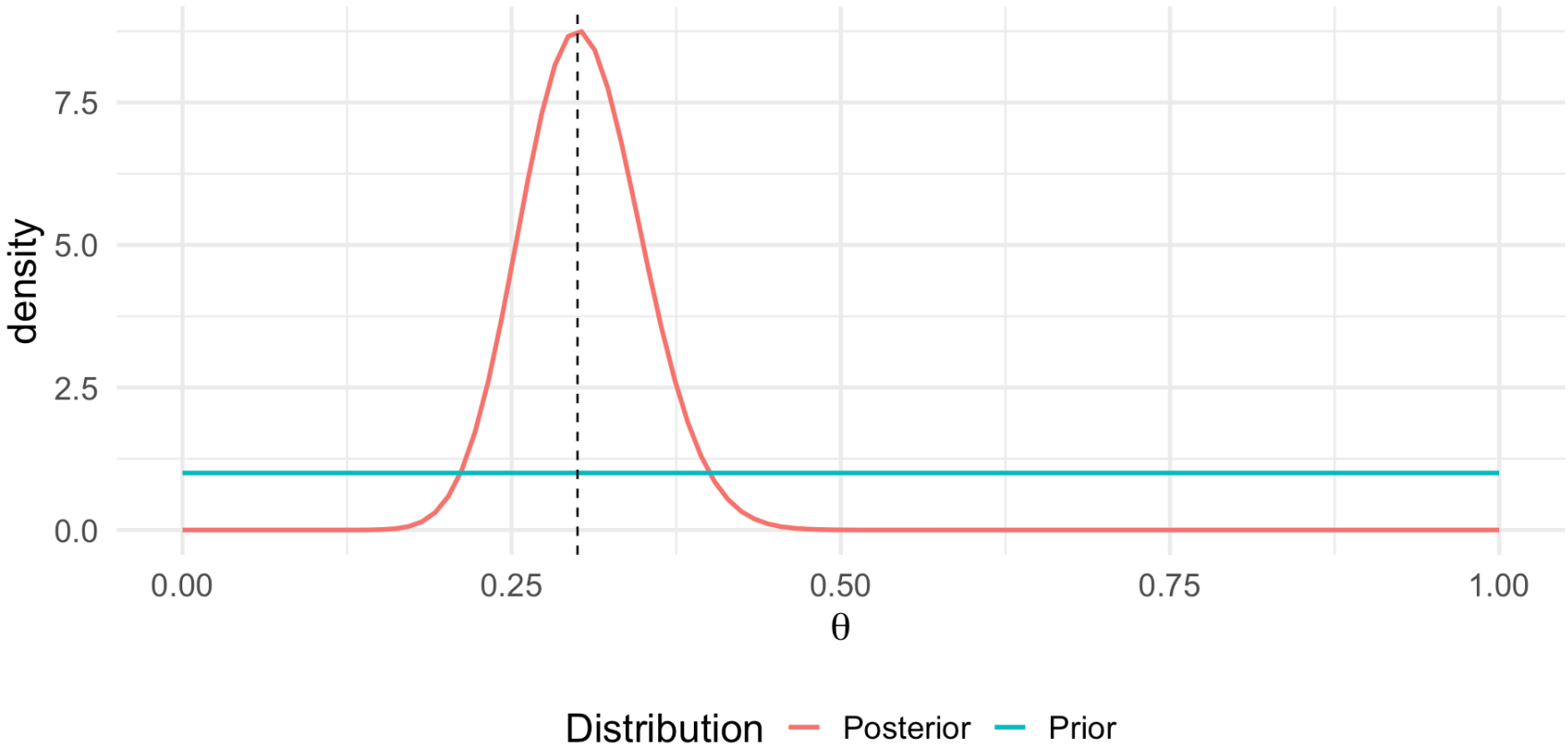
$$\begin{aligned} \pi(\theta|d) &\propto \pi(\theta) f(d|\theta) \\ &\propto \theta^{\alpha-1}(1-\theta)^{\beta-1}\theta^d(1-\theta)^{n-d} \\ &\propto \theta^{\alpha+d-1}(1-\theta)^{\beta+n-d-1} \end{aligned}$$

Numerical example: $\alpha = \beta = 1, n = 10, d = 3$



$$\mathbb{E}(\Theta|D=d) = \frac{4}{12} = \frac{1}{3}, \quad \mathrm{SD}(\Theta|D=d) = \sqrt{\frac{4 \times 8}{12^2 \times 13}} \approx 0.1307$$

2nd numerical example: $\alpha = \beta = 1, n = 100, d = 30$



$$\mathbb{E}(\Theta|D=d) = \frac{31}{102} \approx 0.3039, \quad \mathrm{SD}(\Theta|D=d) = \sqrt{\frac{31 \times 71}{102^2 \times 103}} \approx 0.0453$$

Consider now that we want to estimate $\psi = g(\theta)$, a coordinate of θ or a determinist function of θ:

Credible intervals

- A credible interval of posterior probability $(1-\alpha)$ is an interval $[a, b]$ such that

$$\mathbb{P}\Big(a \leq g(\Theta) \leq b \Big| D = d\Big) = 1 - \alpha.$$

- Quantile credible intervals are computed as:
  - a is the $\alpha/2$ quantile of the posterior distribution $[g(\Theta)|D = d]$
  - b is the $1 - \alpha/2$ quantile of the posterior distribution $[g(\Theta)|D = d]$

- We have a strong belief that $\psi = g(\theta)$ lies in this interval.
- There is a real difference with the definition of a confidence interval in the frequentist paradigm

1.4 The problems

Given a statistical model $D \sim P_\theta$, we want

1. In the frequentist paradigm: to approximate the distribution of $s(D, \lambda)$ when $D \sim P_\theta$ for all θ

- asymptotic approximations for the MLE
- the bootstrap

2. In the Bayesian paradigm: to compute the posterior distribution of $[\Theta | D = d]$

- MCMC methods
- variational inference (not for this course)

2. The frequentist paradigm

In this part, we will assume that $\theta \in \mathbb{R}^d$

2.1 Fisher and the maximum likelihood estimator (MLE)

- The **likelihood** of θ is the density of $[D|\Theta = \theta]$ wrt. a dominating measure: $f(d|\theta)$
- The **log-likelihood** is the logarithm of the likelihood: $\ell(d|\theta) = \log f(d|\theta)$
- The **maximum likelihood estimator** (MLE) is

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmax}}\, \ell(d|\theta)$$

- The **score function** is the gradient of the log-likelihood:

$$\nabla_\theta \ell(d|\theta) = \frac{1}{f(d|\theta)} \nabla_\theta f(d|\theta)$$

Now, let us consider the random dataset $D \sim P_\theta$ and the rv $\widehat{\Theta}$ that models the MLE

- The MLE cancels the score function: $\nabla_\theta \ell(D|\widehat{\Theta}) = 0$
- Whatever the true value θ, we have

$$\mathbb{E}_\theta\Big[\nabla_\theta \ell(D|\widehat{\Theta})\Big] = \int \nabla_\theta \ell(d|\theta) f(d|\theta)\,\mathrm{d}d = \int \frac{1}{f(d|\theta)} \nabla_\theta f(d|\theta) f(d|\theta)\,\mathrm{d}d$$
$$= \int \nabla_\theta f(d|\theta)\,\mathrm{d}d = \nabla_\theta \left\{ \int f(d|\theta)\,\mathrm{d}d \right\} = \nabla_\theta 1 = 0$$

- I.e., at the true value θ, the score function has expectation zero. How far is the score function from zero?
- **Fisher information**: $\mathscr{I}_\theta$ is the variance matrix:

$$\mathscr{I}_\theta = \mathbb{V}_\theta\Big[\nabla_\theta \ell(D|\Theta)\Big] = -\mathbb{E}_\theta\Big[\nabla^2_{\theta\theta} \ell(D|\Theta)\Big]$$

- Hence, if we draw a random $D \sim P_\theta$, the random score function $\nabla_\theta \ell(D|\theta)$ at the true θ has
  - expectation zero and
  - variance $\mathscr{I}_\theta$
- Note that $\mathscr{I}_\theta$ does not depend on the data at hand!
- $\mathscr{I}_\theta$ measures
  - the **local curvature** of the log-likelihood at θ
  - the amount of information from any random dataset $D \sim P_\theta$ about θ

- With a Taylor approx fo the score function at θ, we have

$$0 = \nabla_\theta \ell(d|\hat{\theta}) \approx \nabla_\theta \ell(d|\theta) + \mathscr{J}(d|\theta)(\hat{\theta} - \theta), \quad \text{where } \mathscr{J}(d|\theta) = -\nabla^2_{\theta\theta}\ell(d|\theta)$$

- Hence, $\hat{\theta} \approx \theta + \mathscr{J}^{-1}(d|\theta)\nabla_\theta \ell(d|\theta) \approx \theta + \mathscr{I}_\theta^{-1}\nabla_\theta \ell(d|\theta)$
- The approx. error term $\mathscr{I}_\theta^{-1}\nabla_\theta \ell(d|\theta)$
  - has expectation zero and
  - variance $\mathscr{I}_\theta^{-1}\mathscr{I}_\theta\mathscr{I}_\theta^{-1} = \mathscr{I}_\theta^{-1}$
- Hence Fisher Gaussian approximation: $\widehat{\Theta} \sim \mathscr{N}(\theta, \mathscr{I}_\theta^{-1})$
- Since $\mathscr{I}_\theta = -\mathbb{E}_\theta\left[\nabla^2_{\theta\theta}\ell(D|\theta)\right]$, we can approximate $\mathscr{I}_\theta$ by the **observed information matrix** $\mathscr{J}(d|\theta) = -\nabla^2_{\theta\theta}\ell(d|\theta)$

Numerically

- The score function $\nabla_\theta \ell(d|\theta)$ can be computed by **automatic differentiation**, as well as the observed information matrix $\mathscr{J}(d|\theta) = -\nabla^2_{\theta\theta}\ell(d|\theta)$

- When the data $D = (D_1, \ldots, D_n)$ form an iid sample of size n, $P_\theta = \bigotimes_{i=1}^{n} p_\theta$ and

$$\mathscr{I}_\theta = n i_\theta, \quad \text{where } i_\theta = -\mathbb{E}_\theta\left[\nabla^2_{\theta\theta}\ell(D_1|\theta)\right]$$

- The last expected value can be approximated by **Monte Carlo methods**…
- In this case, a specific MC approximation is $\frac{1}{n}\mathscr{J}(d|\theta) = -\frac{1}{n}\sum_{i=1}^{n}\nabla^2_{\theta\theta}\ell(d_i|\theta)$.

  Thus $\mathscr{J}(d|\theta)$ is a **Monte Carlo** approximation of $\mathscr{I}_\theta$

The binomial case : $D \sim \text{Binomial}(n, \theta)$

```python
import jax.numpy as jnp
from jax import grad, hessian

# Define the log-likelihood of the binomial distribution
def log_likelihood(theta, data, n):
    return jnp.sum(data*jnp.log(theta) + (n-data)*jnp.log(1-theta))

# Define the score function and the Hessian
score = grad(log_likelihood, argnums=0)
hess_loglik = hessian(log_likelihood, argnums=0)

# The observed data: 30 successes out of 100 trials
data, n = 30, 100
mle = data/n
mle_score = score(mle, data, n)
observed_info = - hess_loglik(mle, data, n)
SErr = jnp.sqrt(1/observed_info)
print(f"The MLE is {mle:.4f}, its score is {mle_score:.4f} and the standard error is {S
```

```
The MLE is 0.3000, its score is -0.0000 and the standard error is 0.0458.
```

The Gaussian case: $D \sim \mathcal{N}^{\otimes 100}(\mu, \sigma^2), \theta = (\mu, \sigma^2)$

```
# Define the log-likelihood of the Gaussian distribution
def log_likelihood_gaussian(theta, data):
    mu, sigma2 = theta[0], theta[1]
    return -0.5*jnp.sum(jnp.log(2*jnp.pi*sigma2) + (data-mu)**2/sigma2)
# Define the score function and the Hessian
score_gaussian = grad(log_likelihood_gaussian, argnums=0)
hess_loglik_gaussian = hessian(log_likelihood_gaussian, argnums=0)
# The observed data: sample of size 100 from a N(0,1)
import numpy as np
np.random.seed(1234)
data = jnp.array(np.random.normal(0, 1, 100))
# MLE
mle = jnp.array([jnp.mean(data), jnp.var(data)])
mle_score = score_gaussian(mle, data)
observed_info = - hess_loglik_gaussian(mle, data)
SErr_t, SErr_s = jnp.sqrt(1/observed_info[0, 0]), jnp.sqrt(1/observed_info[1, 1])
print(f"The MLE is ({mle[0]:.4f}, {mle[1]:.4f}), its score is ({mle_score[0]:.4f}, {mle
```

```
The MLE is (0.0351, 0.9914), its score is (0.0000, 0.0000) and the standard errors are
0.0996, 0.1402.
```

```
print(f"The observed information matrix is\n{jnp.round(observed_info, decimals = 4)}.")
```

```
The observed information matrix is
[[100.8686   0.    ]
 [  0.      50.8724]].
```

2.2 The bootstrap

- The **bootstrap** is a **non-parametric** method to approximate the distribution of a statistic $s(D, \lambda)$
  - where $D \sim P_\theta$ is the random dataet $\sim P_\theta$
  - and λ is the tuning parameter
- It can be applied to any statistic $s(D, \lambda)$, not only the MLE
- The dataset D should be composed of $(D_1, \dots, D_n)$ iid rv's, with $D_i \sim p_\theta$, i.e.

$$P_\theta = \bigotimes_{i=1}^{n} p_\theta$$

- In the above notations:
  - p_θ is the marginal **distribution of one individual** D_i
  - P_θ is the **joint distribution of the dataset** D, which is a product (because of independence)

- The idea of bootstrap is to approximate the marginal p_θ by the empirical distribution of the data, as in Monte Carlo methods:

$$p^* = \frac{1}{n} \sum_{i=1}^{n} \delta_{d_i}$$

- Thus, $p_\theta \approx p^*$ and $P_\theta \approx P^* = \bigotimes_{j=1}^{n} p^*$ (the population is replaced by the sample)

The bootstrap principle

- The distribution of $s(D, \lambda)$ when $D \sim P_\theta$ can be approximated by the distribution of $s(D^*, \lambda)$ when $D^* \sim P^*$
- It is easy to draw $D^* \sim P^*$ by **drawing n times with replacement** from the data D
- A dataset d^* drawn from P^* is called a **bootstrap sample**
- It has the same size n as the original dataset d

Example with $\mathcal{N}^{\otimes 100}(\mu, \sigma^2)$ to compute the bootstrap standard errors

```
def estimate(data, lmbda=0):
    return np.array([np.mean(data), np.var(data, ddof=lmbda)])
data2 = np.array(data)
# Bootstrap
N = 10000
bootstrap_samples = \
    np.array([estimate(np.random.choice(data2, size=100, replace=True), 0) for _ in ran
bootstrap_var = np.var(bootstrap_samples, axis=0)
print(f"The bootstrap standard errors are {np.sqrt(bootstrap_var[0]):.4f} and {np.sqrt(
```

```
The bootstrap standard errors are 0.0994 and 0.1729.
```

```
print(f"The standard errors obtained with the normal approximation were {SErr_t:.4f} an
```

```
The standard errors obtained with the normal approximation were 0.0996 and 0.1402.
```

We can also get Standard errors with $\lambda = 1$ (unbiased estimator of the variance)

```
# Bootstrap
bootstrap_samples = \
    np.array([estimate(np.random.choice(data2, size=100, replace=True), 1) for _ in ran
bootstrap_var = np.var(bootstrap_samples, axis=0)
print(f"The bootstrap standard errors are now {np.sqrt(bootstrap_var[0]):.4f} and {np.s
```

```
The bootstrap standard errors are now 0.0994 and 0.1762.
```

- It can be applied to any statistic, not only the MLE
- The **bootstrap** is a **non-parametric** method, i.e., no assumption on the distribution of the data
- The **bootstrap** is a **resampling** method, i.e., it draws n times with replacement from the data
- The **bootstrap** is particularly useful to approximate the **standard errors** of a statistic $s(D, \lambda)$:

$$\mathbb{V}_\theta(\widehat{s(D, \lambda)}) = \mathbb{V}^*(s(D^*, \lambda)), \quad \text{where } D^* \sim P^*$$

The above variance being itself approximated with a basic Monte Carlo method

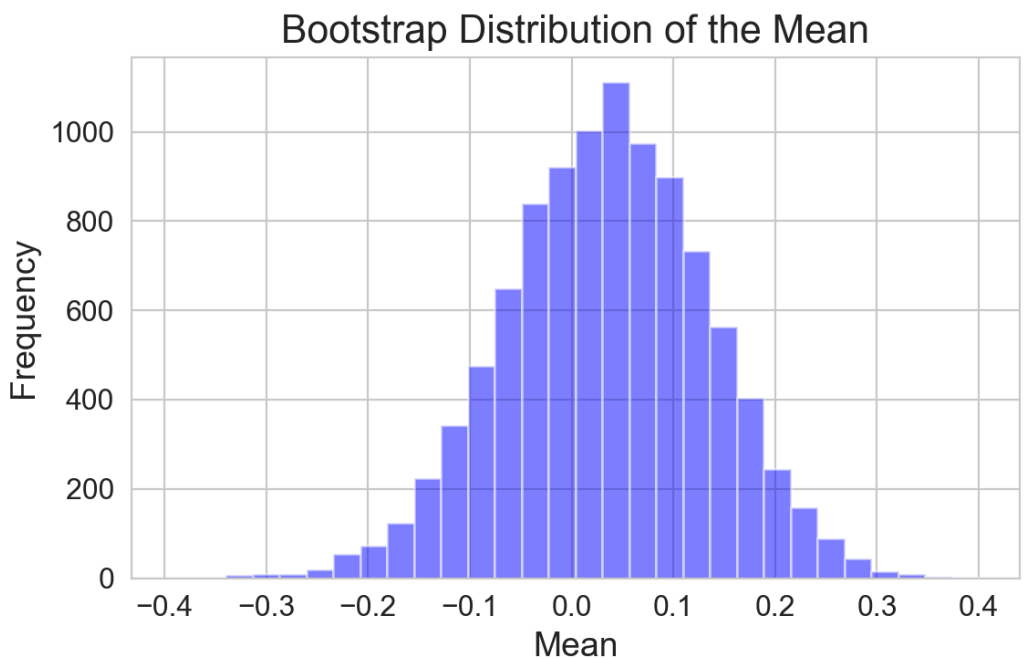
Confidence intervals with the bootstrap

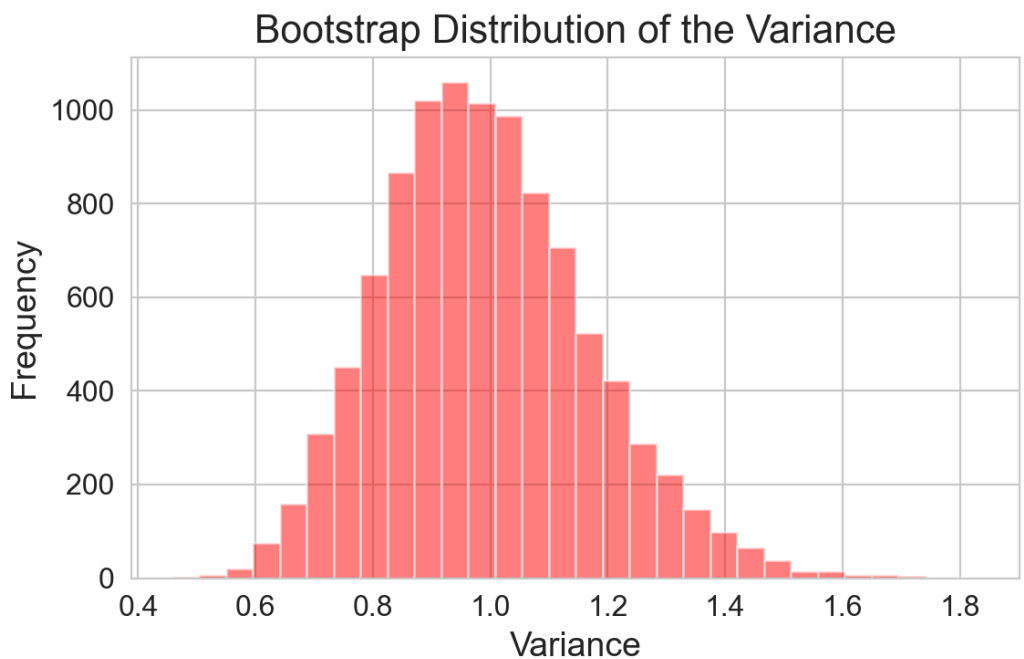
Consider now that we want to estimate ψ, and a statistic $s(D, \lambda)$ that estimates ψ:

- ψ is a given coordinate of θ
- or $\psi = \beta_0 + \sum_j \beta_j x_{n+1,j}$ in (generalized) linear models
- The **percentile method** to get a $1 - \alpha$ confidence interval for ψ is obtained as follows:
  - draw B bootstrap samples $d_1^*, \ldots, d_B^*$ from P^*
  - compute the B bootstrap estimates $\widehat{\psi}_1^*, \ldots, \widehat{\psi}_B^*$ of ψ
  - compute the quantiles of order $\alpha/2$ and $1 - \alpha/2$ of the bootstrap estimates
  - the percentile confidence interval of level $(1 - \alpha)$ is $[\widehat{\psi}^*_{(\alpha/2)}, \widehat{\psi}^*_{(1-\alpha/2)}]$

- To undestand the percentile method, we can
  - consider the bootstrap estimates as a sample drawn a Bayesian posterior distribution
  - approximate a credible interval of posterior probability $(1-\alpha)$ with the quantiles of order $\alpha/2$ and $1-\alpha/2$
  - consider that the credible interval is an approximation of a confidence interval
- The percentile method is a **non-parametric** method, i.e., no assumption on the distribution of the data
- The percentile method is a **first** approximation of the confidence interval, but it is easy to implement
- **Better approximations** exist, e.g., the **BCa method** (bias-corrected and accelerated), but will not be studied in this course

Example with the Gaussian case (same data as before)






```python
# Quantiles
alpha = 0.05
quantiles = np.quantile(bootstrap_samples, [alpha/2, 1-alpha/2], axis=0)
print(f"The percentile confidence interval is [{quantiles[0, 0]:.4f}, {quantiles[1, 0]:
```

```
The percentile confidence interval is [-0.1624, 0.2253] for the mean.
```

```python
print(f"The percentile confidence interval is [{quantiles[0, 1]:.4f}, {quantiles[1, 1]:
```

```
The percentile confidence interval is [0.6845, 1.3744] for the variance.
```

Is the bootstrap always working?

- The answer is definitely **no**!
- E.g., if the estimator is constant, $s(D, \lambda) = \check{\theta}$ for a given $\check{\theta}$ whatever the data D, the bootstrap will not work
  - the bootstrap samples will be all equal to $\check{\theta}$
  - the bootstrap standard errors will be zero
- Remember that, for an estimator $\widehat{\Theta} = s(D, \lambda)$, the **mean square error** is

$$\mathrm{MSE}(\widehat{\Theta}) = \mathbb{E}_\theta\left((\widehat{\Theta} - \theta)^2\right) = \mathbb{V}_\theta(\widehat{\Theta}) + \mathrm{Bias}_\theta^2(\widehat{\Theta})$$

- **Bootstrap methods cannot estimate the bias**
- Thus, the **bootstrap** requires that

$$\mathrm{Bias}_\theta^2(\widehat{\Theta}) \ll \mathbb{V}_\theta(\widehat{\Theta})$$

- The MLE of the parameter of an iid model $P_\theta = \otimes_{i=1}^n p_\theta$ satisfies this condition if the size n is large:
  - the bias is of order $1/n$ (if not zero)
  - the variance is of order $1/n$
- Any unbiased estimator satisfies this condition…

3. The Bayesian paradigm

Now, Θ is a rv, with marginal probability density function $\pi(\theta)$, which is the **prior density**

3.1 The posterior distribution

- In Bayesian statistics, the **likelihood** $f(d|\theta)$ is used only as a **weight** to update the prior $\pi(\theta)$ regarding its fit to the data d
- The **posterior density** of Θ given the data d has density

$$\pi(\theta|d) = \frac{f(d|\theta)\pi(\theta)}{f_\pi(d)} \propto f(d|\theta)\pi(\theta),$$

  where $f_\pi(d)$ is the normalizing constant.
- Appart from very specific cases, we cannot recognize a well known distribution for the posterior law
- Instead, we used **Monte Carlo methods to get a sample** $\theta_1, \ldots, \theta_N$ from the posterior

3.2 Metroplis-Hastings algorithm

- We have already met a distribution known up to a multiplicative constant: the Gibbs measures in simulated annealing
- The MCMC method that was introduced then was a **Metropolis-Hastings algorithm**
- The Metropolis-Hastings algorithm is a **Markov chain Monte Carlo (MCMC) method** that generates a sample from a distribution known up to a multiplicative constant
- Let us denote $\pi_u(\theta|d)$ an **unormalized version** of the posterior density. E.g. $\pi_u(\theta|d) = f(d|\theta)\pi(\theta)$
- (We can get rid of any multiplicative term that does not depend on θ in $\pi_u(\theta|d)$)

Metropolis-Hastings Algorithm

1. Initialize θ_0
2. For $t = 1, 2, \ldots, T$:
   - Generate $\zeta \sim g(\zeta|\theta_t)$, a proposal kernel given θ_t
   - Compute the acceptance ratio $\rho = \dfrac{\pi_u(\zeta|d)g(\theta_t|\zeta)}{\pi_u(\theta_t|d)g(\zeta|\theta_t)}$
   - Draw $U \sim \text{Unif}(0,1)$ and set $\theta_{t+1} = \zeta \mathbf{1}\{U < \rho\} + \theta_t \mathbf{1}\{U \geq \rho\}$

- The **proposal kernel** $g(\zeta|\theta)$ is a density that generates a candidate ζ given the current value θ

The same remarks as for the simulated annealing algorithm apply here:

- if g is symmetric, the acceptance ratio simplifies
- **choose the proposal kernel $g(\zeta|\theta)$ wisely**
- the average proportion of accepted moves along time should be around 0.234
- the **burn-in period** is the time needed to reach the stationary distribution
  - assume we start from a θ_0 from the prior
  - we can get rid of the first B iterations until we reach the part of the sample where the posterior distribution is not near 0.

- We can also start from the $\operatorname{argmax}_\theta \pi_u(\theta|d)$
  - this is the **maximum a posteriori (MAP)** estimator
  - it can often be computed numerically with an optimization algorithm…
- During the burn-in period, we can do anything we want, in particular change the proposal kernel
- At the end, the posterior distribution is approximated by

$$\pi(\theta|d)\mathrm{d}\theta \approx \frac{1}{T-B}\sum_{t=B+1}^{T}\delta_{\theta_t}$$

Numerical example

- Statistical model: $D = (D_1, \ldots, D_n) \sim \mathcal{N}^{\otimes n}(\mu, \sigma^2)$
- Prior: $\pi(\mu, \sigma^2) = \pi(\mu|\sigma^2)\pi(\sigma^2)$, where $[\mu|\sigma^2] \sim \mathcal{N}(0, \sigma^2)$ and $1/\sigma^2 \sim \mathrm{Exp}(1)$
- Thus, $\pi(\sigma^2)\mathrm{d}\sigma^2 = \exp(-\sigma^{-2})\sigma^{-4}\mathrm{d}\sigma^2$
- The likelihood is

$$f(d|\theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(d_i-\mu)^2}{2\sigma^2}\right) \propto \frac{1}{\sigma^n}\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(d_i-\mu)^2\right)$$

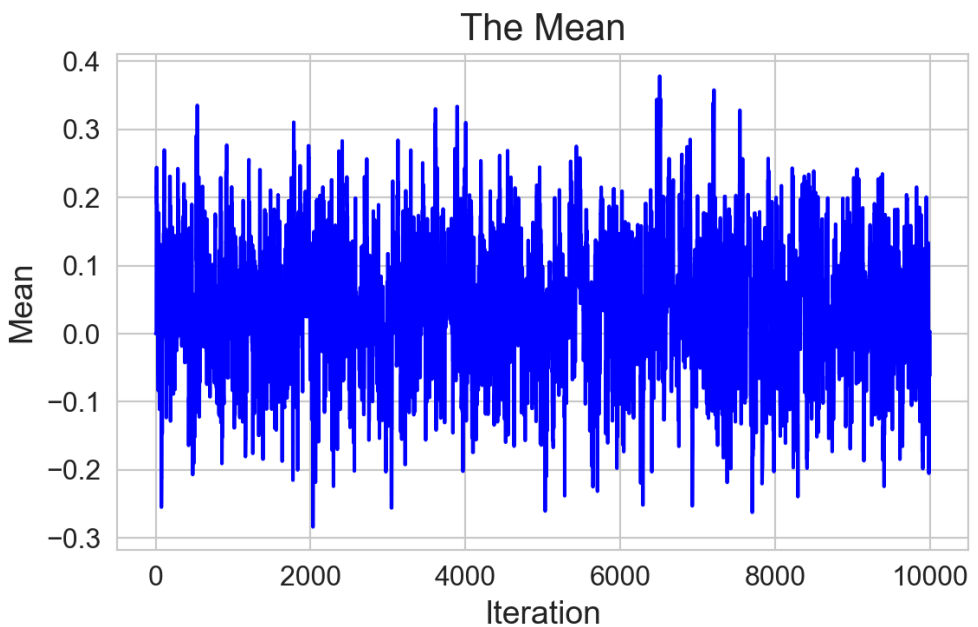
$$\propto \frac{1}{\sigma^n}\exp\left(-\frac{n\mu^2}{2\sigma^2} - \frac{1}{2\sigma^2}\underbrace{\sum_{i=1}^{n} d_i^2}_{B} + \frac{\mu}{\sigma^2}\underbrace{\sum_{i=1}^{n} d_i}_{A}\right)$$

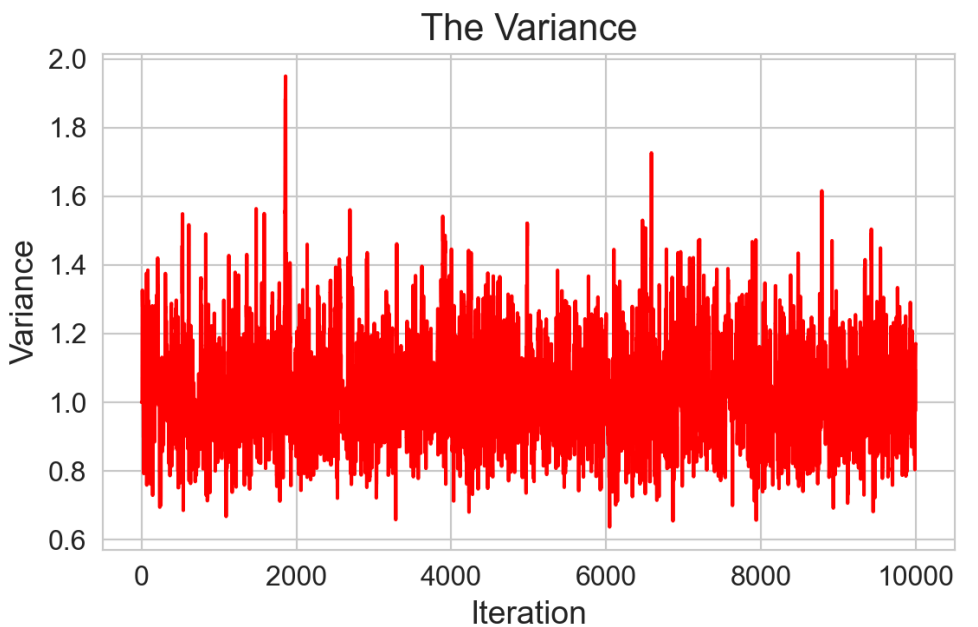
depends on the data only through A and B

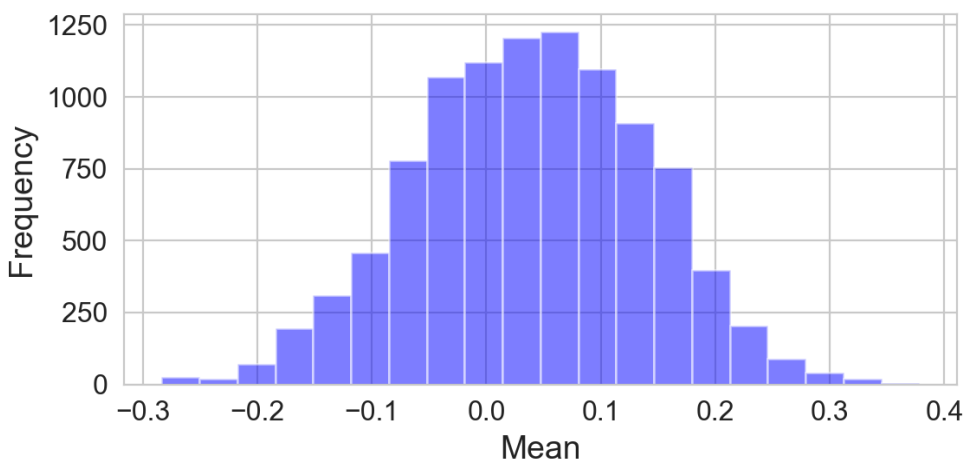
Always the same data of size 100, drawn from a $\mathcal{N}(0, 1)$

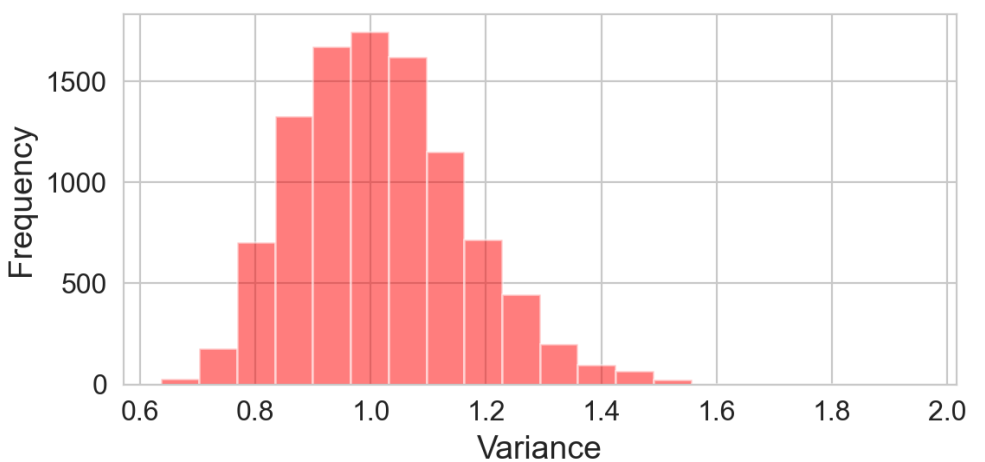
```python
import numpy as np
from scipy.stats import norm, expon, multivariate_normal
A, B = np.sum(data2), np.sum(data2**2)
def log_posterior(theta, n=100):
    if theta[1] <= 0:
        return -np.inf
    else:
      mu, sigma2 = theta[0], theta[1]
      return -1./sigma2 - 2.*np.log(sigma2) -0.5*n*np.log(sigma2) - n*mu**2/sigma2/2 +

# Metropolis-Hastings
def metropolis_hastings(log_posterior, T, theta0, g):
    theta = theta0
    samples = [theta]
    for t in range(T):
        zeta = theta + g.rvs(size = 1)
        rho = np.exp(log_posterior(zeta) - log_posterior(theta))
        U = np.random.uniform()
        theta = zeta if U < rho else theta
        samples.append(theta)
    return np.array(samples)
```

Paths of the algo. and posterior distributions

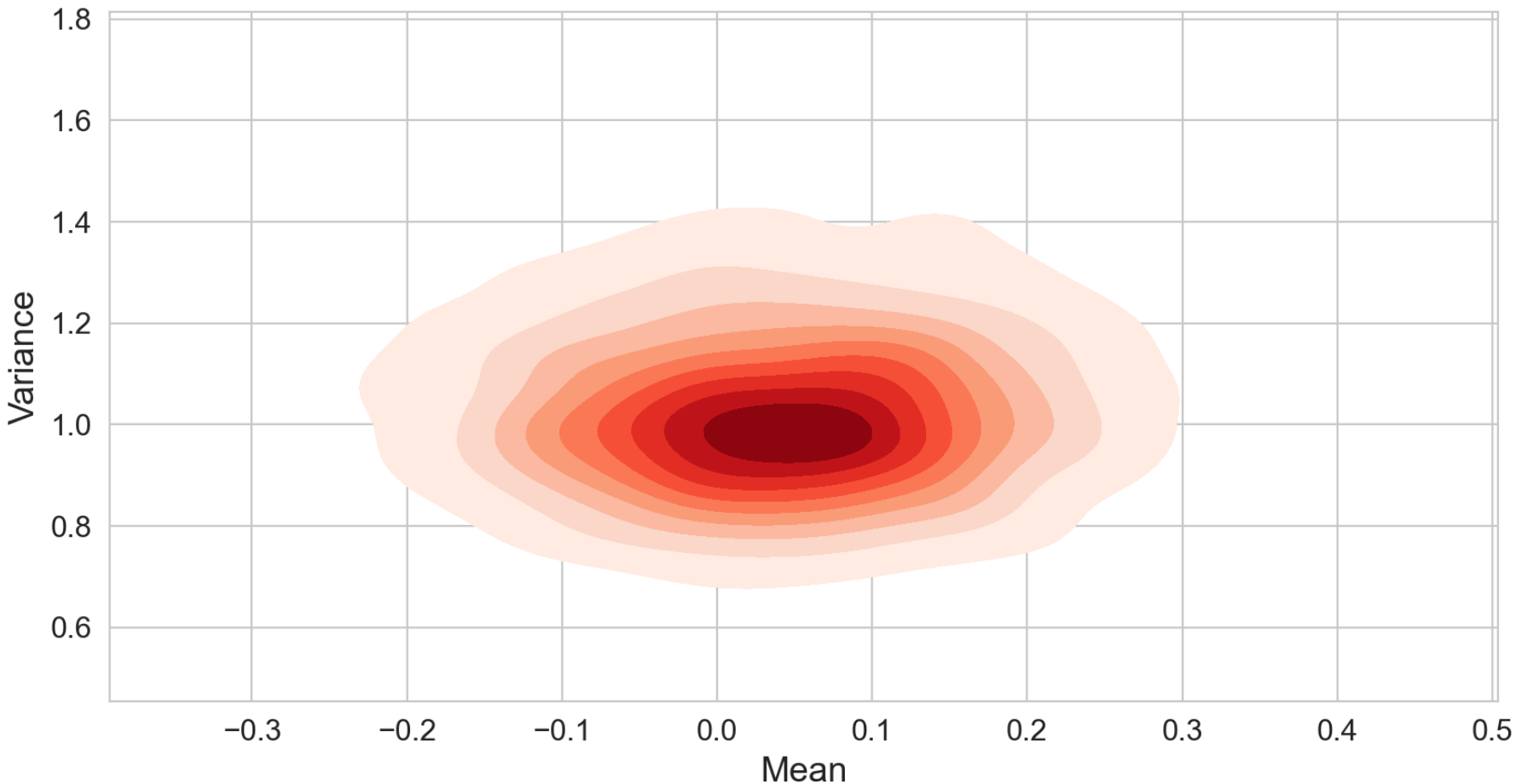








Bivariate heatmap of the posterior samples



Summaries of the posterior distribution

```
# Summaries of the posterior distribution
mean_posterior = np.mean(posterior_sample, axis=0)
median_posterior = np.median(posterior_sample, axis=0)
cov_posterior = np.round(np.cov(posterior_sample, rowvar=False), decimals=4)
print(f"The posterior mean is {mean_posterior[0]:.4f} for the mean and {mean_posterior[
```

```
The posterior mean is 0.0399 for the mean and 1.0215 for the variance.
```

```
print(f"The posterior covariance matrix is\n{cov_posterior}.")
```

```
The posterior covariance matrix is
[[0.0102 0.0002]
 [0.0002 0.0203]].
```

```
print(f"The posterior median is {median_posterior[0]:.4f} for the mean and {median_post
```

```
The posterior median is 0.0417 for the mean and 1.0046 for the variance.
```

3.3 The Gibbs algorithm

- The Gibbs algorithm relies on a **decomposition of the parameter** θ into K blocks

$$\theta = (\theta^{(1)}, \ldots, \theta^{(K)}).$$

- We assume that we are able to compute the **full conditional distributions** of each block $\theta^{(k)}$ given the other blocks $\theta^{(-k)}$ and the data d

$$\pi(\theta^{(k)}|d, \theta^{(-k)}) = \frac{\pi(\theta^{(k)}, \theta^{(-k)}|d)}{\pi(\theta^{(-k)}|d)} \propto \pi(\theta|d)$$

- The Gibbs algorithm updates one block given the other blocks and the data to obtain the new value θ_{t+1}.

Gibbs Algorithm with random block selection

1. Initialize θ_0
2. For $t = 1, 2, \ldots, T$:
   - Draw $k \sim \text{Unif}(1, K)$
   - Draw $\theta_{t+1}^{(k)} \sim \pi(\theta^{(k)} | d, \theta_t^{(-k)})$ and set $\theta_{t+1}^{(-k)} = \theta_t^{(-k)}$

Gibbs Algorithm with systematic block selection

1. Initialize θ_0
2. For $t = 1, 2, \ldots, T$:
   - Set $k = (t \mod K) + 1$
   - Draw $\theta_{t+1}^{(k)} \sim \pi(\theta^{(k)} | d, \theta_t^{(-k)})$ and set $\theta_{t+1}^{(-k)} = \theta_t^{(-k)}$

- Like the Metropolis-Hastings algorithm, the Gibbs algorithm generates a correlated sample from the posterior distribution
- The path $\theta_0, \ldots, \theta_T$ is a Markov chain
- As with Metropolis-Hastings, the Gibbs algorithm requires a **burn-in period**
- At the end, the posterior distribution is approximated by

$$\pi(\theta|d)\mathrm{d}\theta \approx \frac{1}{T-B}\sum_{t=B+1}^{T}\delta_{\theta_t}$$

where B is the number of burn-in iterations

Numerical example

- Statistical model: $D = (D_1, \ldots, D_n) \sim \mathcal{N}^{\otimes n}(\mu, \sigma^2)$, and we set $\tau = 1/\sigma^2$ and $\theta = (\mu, \tau)$
- Prior: $\pi(\mu, \tau) = \pi(\mu|\tau)\pi(\tau)$, where $\mu|\tau \sim \mathcal{N}(0, 1/\tau)$ and $\tau \sim \mathrm{Exp}(1)$
- Likelihood:

$$f(d|\theta) \propto \tau^{n/2} \exp\left(-n\mu^2\tau/2 - B\tau/2 + \mu\tau A\right)$$

  with $A = \sum_{i=1}^n d_i$ and $B = \sum_{i=1}^n d_i^2$

- Posterior:

$$\pi(\mu, \tau|d) \propto \tau^{n/2} \exp\left(-n\mu^2\tau/2 - B\tau/2 + \mu\tau A\right) e^{-\tau} \exp(-\tau\mu^2/2)$$

- The full conditional distributions are
  - For $[\mu|\tau, d]$:

$$\pi(\mu|\tau, d) \propto \exp(-n\mu^2\tau/2 + \mu\tau A - \mu^2/2)$$
$$\propto \exp\left\{-\frac{(n+1)\tau}{2}\left(\mu - \frac{A}{n+1}\right)^2\right\}$$

    i.e., $[\mu|\tau, d] \sim \mathcal{N}\left(\frac{A}{n+1}, \frac{1}{(n+1)\tau}\right)$

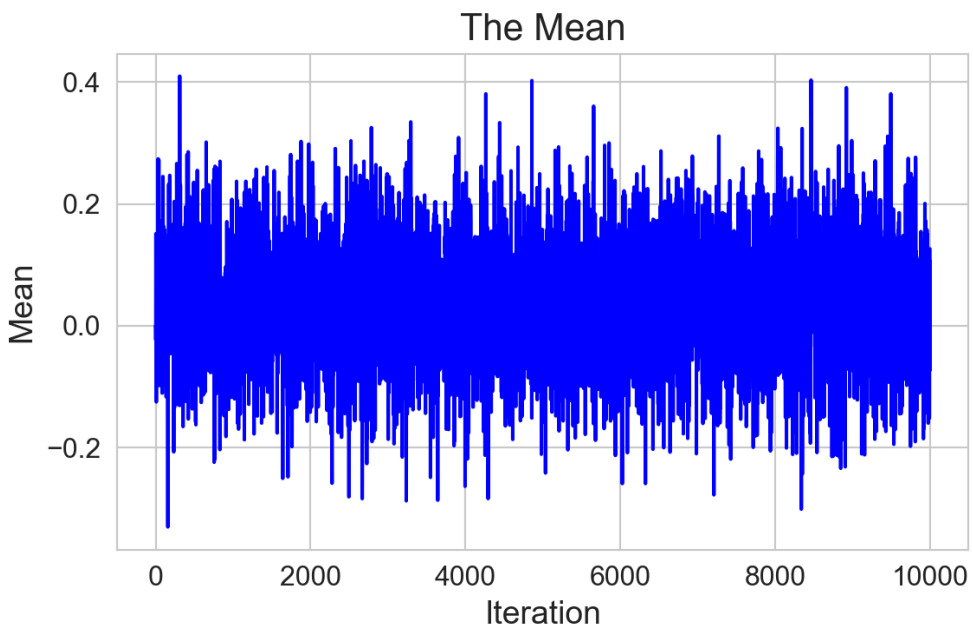
  - For $[\tau|\mu, d]$:

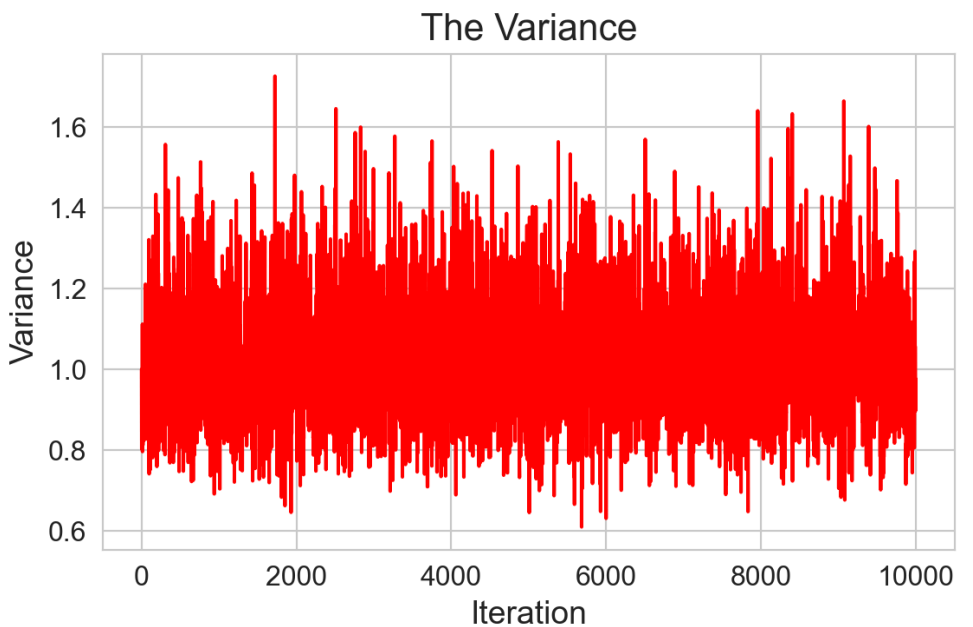
$$\pi(\tau|\mu, d) \propto \tau^{n/2} \exp\left(-n\mu^2\tau/2 - B\tau/2 + \mu\tau A\right) e^{-\tau} \exp(-\tau\mu^2/2)$$
$$\propto \tau^{n/2} \exp\left\{-\frac{\tau}{2}\left((n+1)\mu^2 + B - 2\mu A + 2\right)\right\}$$

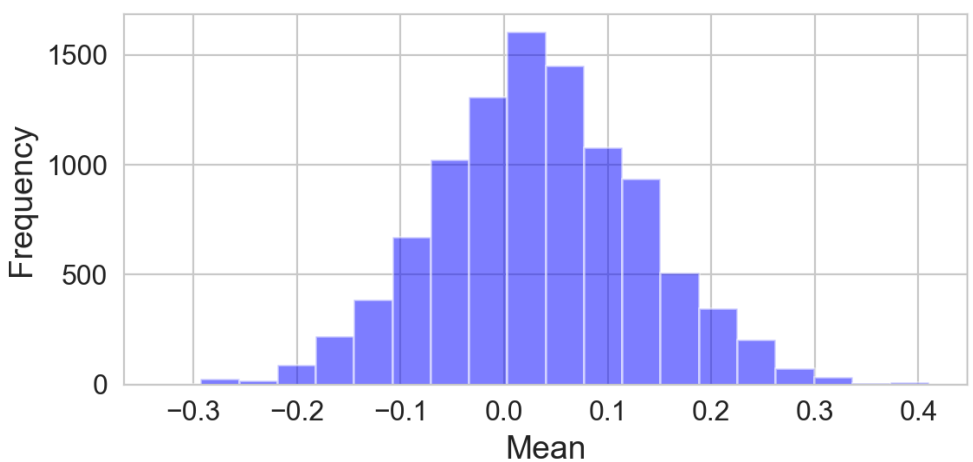
    i.e., $[\tau|\mu, d] \sim \text{Gamma}\left(\frac{n}{2} + 1, \frac{1}{2}(B - 2\mu A + (n+1)\mu^2) + 1\right)$

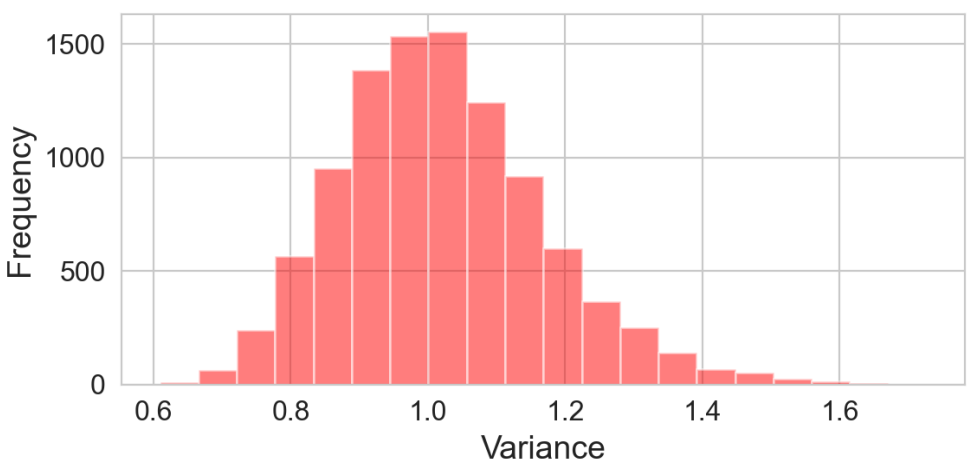
```python
from scipy.stats import norm, gamma
def gibbs_sampler(n, data, T, theta0, scan='random'):
    A, B = np.sum(data), np.sum(data**2)
    theta = theta0
    samples = [theta]
    for t in range(T):
        if scan == 'random':
            k = np.random.choice([0, 1])
        else:
            k = t % 2
        if k == 0:
          mu = np.random.normal(A/(n+1), np.sqrt(1/(n+1)/theta[1]))
          tau = theta[1]
        else:
          mu = theta[0]
          rate = 0.5*(B - 2*mu*A + (n+1)*mu**2) + 1
          tau = np.random.gamma(n/2+1, 1/rate)
        theta = np.array([mu, tau])
        samples.append(theta)
    return np.array(samples)
# Run
```

Paths of the algo. and posterior distributions

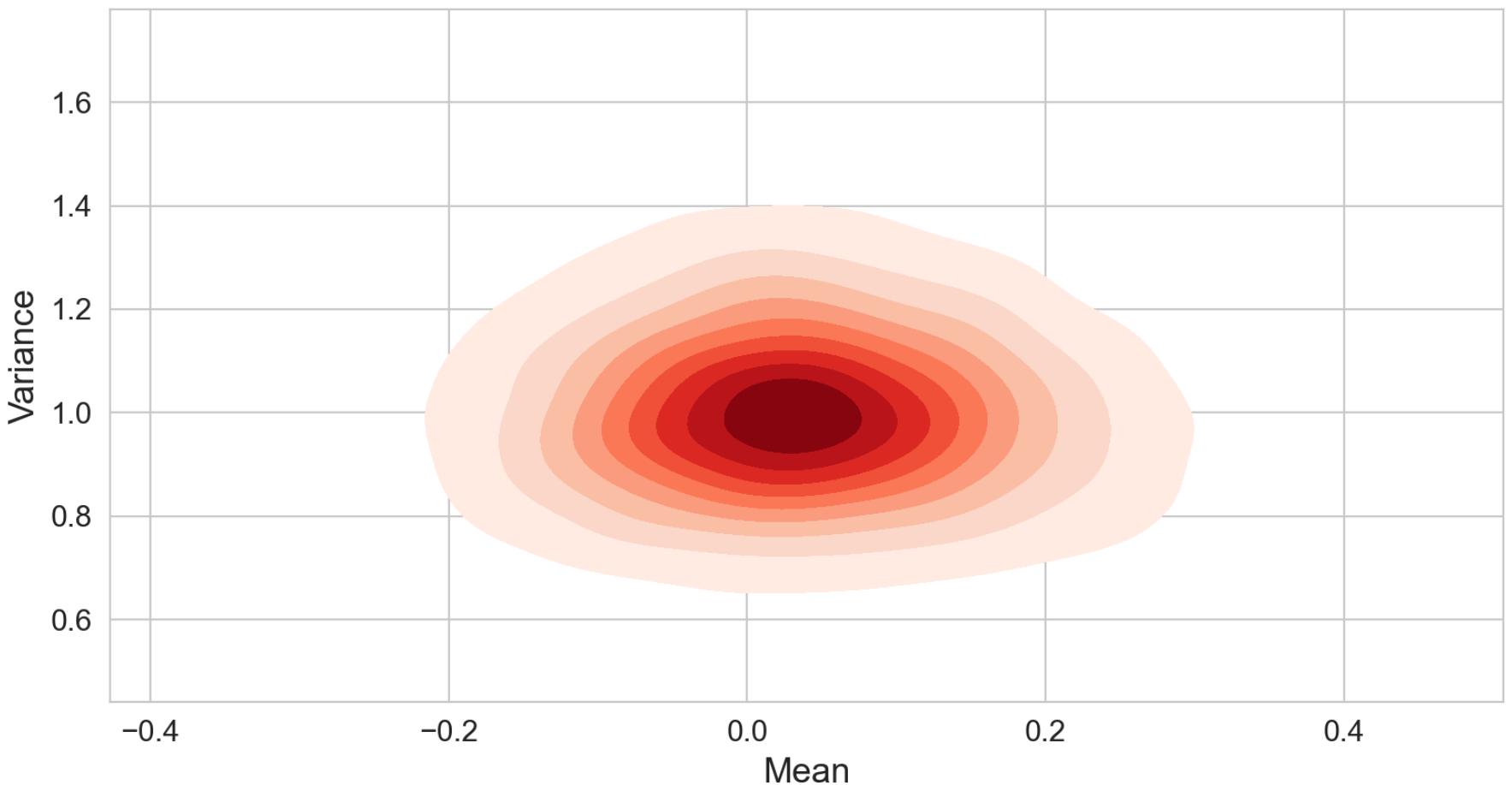








Bivariate heatmap of the Gibbs samples



Summaries of the Gibbs distribution

```
# Summaries of the Gibbs distribution
mean_gibbs = np.mean(gibbs_sample, axis=0)
median_gibbs = np.median(gibbs_sample, axis=0)
cov_gibbs = np.round(np.cov(gibbs_sample, rowvar=False), decimals=4)
print(f"The Gibbs mean is {mean_gibbs[0]:.4f} for the mean and {mean_gibbs[1]:.4f} for
```

```
The Gibbs mean is 0.0343 for the mean and 0.9996 for the variance.
```

```
print(f"The Gibbs covariance matrix is\n{cov_gibbs}.")
```

```
The Gibbs covariance matrix is
[[ 0.0099 -0.0002]
 [-0.0002  0.0205]].
```

```
print(f"The Gibbs median is {median_gibbs[0]:.4f} for the mean and {median_gibbs[1]:.4f
```

```
The Gibbs median is 0.0313 for the mean and 0.9917 for the variance.
```

Python libraries to perform Bayesian inference

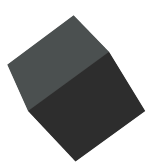
- PyMC: a Python library for probabilistic programming
- Stan: a probabilistic programming language
- Edward: a library for probabilistic modeling, inference, and criticism
- …

These packages provide high-level languages to define the model and perform the Monte Carlo approximation of the posterior

- MCMC: Metropolis-Hastings, Gibbs, Hamiltonian Monte Carlo, No-U-Turn Sampler, …
- Variational inference (=approximation of the posterior by e.g. a Gaussian, or another family of distribution): Variational Bayes, Black Box Variational Inference, …

PyMC

Stan



Edward

```
import numpy as np
import pymc as pm
```

```
WARNING (pytensor.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

```
import arviz as az
import matplotlib.pyplot as plt

# Modèle bayésien avec pymc
with pm.Model() as model:
    tau = pm.Exponential("tau", 1.0)  # tau = 1/sigma^2
    sigma2 = pm.Deterministic("sigma2", 1 / tau)
    mu = pm.Normal("mu", mu=0, sigma=pm.math.sqrt(sigma2))

    # Vraisemblance
    y_obs = pm.Normal("y_obs", mu=mu, sigma=pm.math.sqrt(sigma2), observed=data2)

    # Inférence
    trace = pm.sample(2000, cores=1, return_inferencedata=True, progressbar=False)
```

```
          mean     sd  hdi_3%  hdi_97%  ...  mcse_sd  ess_bulk  ess_tail  r_hat
mu       0.036  0.100  -0.148    0.227  ...    0.002    4029.0    2925.0    1.0
sigma2   1.010  0.142   0.746    1.266  ...    0.002    3755.0    2759.0    1.0

[2 rows x 9 columns]
```

```
array([<Axes: title={'center': 'mu'}>, <Axes: title={'center': 'sigma2'}>],
      dtype=object)
```