EPFL

Bayesian Inverse Problems in Large Dimensions

Stochastic Simulation Miniproject

Matthias Zeller

January 19, 2022

1 Introduction

We are interested in the problem of inferring a high-dimensional parameter $\boldsymbol{\theta}$ from noisy low-dimensional measurements $\mathbf{y}$, following the measurement model

$$\mathbf{y} = G(\boldsymbol{\theta}) + \boldsymbol{\epsilon}\,, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\, \Sigma_{\boldsymbol{\epsilon}})\,, \tag{1}$$

with $G : \mathbb{R}^D \to \mathbb{R}^d$, $D \gg d$. Under a Bayesian framework, we treat the $\boldsymbol{\theta}$ parameter as a random variable, so that we can inject *a priori* knowledge on the parameter and express the *posterior* distribution

$$\pi_{\text{post}}(\boldsymbol{\theta}) := \pi(\boldsymbol{\theta} \mid \mathbf{y}) = \pi(\mathbf{y} \mid \boldsymbol{\theta})\, \pi_0(\boldsymbol{\theta})\, \frac{1}{\pi(\mathbf{y})}\,. \tag{2}$$

From our measurement model (1), the likelihood of the data $\mathbf{y}$ reads $\pi(\cdot \mid \boldsymbol{\theta}) = \mathcal{N}(G(\boldsymbol{\theta}), \Sigma_{\boldsymbol{\epsilon}})$. In order to sample from the *posterior* without knowing the normalization constant $1/\pi(\mathbf{y})$, we will rely on Markov Chain Monte Carlo (MCMC). We will try different values for the number of dimensions D and different proposal distributions to study performance of MCMC when $D \to \infty$.

1.1 Specifications

In this project, we set $d = 4$ and use $G(\boldsymbol{\theta}) = (p(0.2;\boldsymbol{\theta}),\, p(0.4;\boldsymbol{\theta}),\, p(0.6;\boldsymbol{\theta}),\, p(0.8;\boldsymbol{\theta}))$, with p the solution of the PDE

$$\frac{d}{dx}\left(e^{u(x;\boldsymbol{\theta})}\, \frac{d}{dx} p(x;\boldsymbol{\theta})\right) = 0, \quad p(0;\boldsymbol{\theta}) = 0,\; p(1;\boldsymbol{\theta}) = 2,\; x \in [0,1]\,, \tag{3}$$

$$u(x;\boldsymbol{\theta}) = \frac{\sqrt{2}}{\pi} \sum_{k=1}^{D} \theta_k \sin(k\pi x)\,. \tag{4}$$

In the context of modeling subsurface flows, one can interpret p as the pressure and u as the log-permeability. The closed-form solution for the pressure is

$$p(x;\, \boldsymbol{\theta}) = 2\frac{S_x\{e^{-u(\cdot;\boldsymbol{\theta})}\}}{S_1\{e^{-u(\cdot;\boldsymbol{\theta})}\}}\,, \quad S_x\{f\} = \int_0^x f(y)dy\,. \tag{5}$$

The noise components are iid, i.e. $\Sigma_{\boldsymbol{\epsilon}} = \sigma^2\, \mathbb{I}_d$ with $\sigma = 0.04$, and the prior on the parameter is $\pi_0(\cdot) = \mathcal{N}(\mathbf{0}, \Sigma_0)$ with $\Sigma_0 = \text{diag}(1, 2^{-2}, \ldots, D^{-2})$, i.e. the magnitude of Fourier coefficients $\{\theta_k\}_{k=1}^D$ concentrate towards zero at a rate $\mathcal{O}(k^{-2})$. Under those specifications, we can write the target (un-normalized) distribution

x	0.2000	0.4000	0.6000	0.8000
$y(x;\,\boldsymbol{\theta})$	0.5041	0.8505	1.2257	1.4113

Table 1: Provided measured data

$$\tilde{\pi}_{\text{post}}(\boldsymbol{\theta}) := \exp\left(-\frac{1}{2\sigma^2}\|G(\boldsymbol{\theta}) - \mathbf{y}\|_2^2 - \frac{1}{2}\sum_{k=1}^{D}\theta_k^2 k^2\right) \propto \pi(\mathbf{y} \mid \boldsymbol{\theta})\,\pi_0(\boldsymbol{\theta})\;, \tag{6}$$

with $\mathbf{y}$ taken from measured data in Table 1.

2 Methods

We show three different methods used to tackle this inverse problem: Random Walk Metropolis (section 2.2), Preconditionned Cranck-Nicolson (section 2.3) and Laplace Approximation (section 2.4). Those are all based on MCMC Metropolis-Hastings but differ in their proposal distribution. We will use the starting point of the chains $\boldsymbol{\theta}_0 = \mathbf{0}$ in all experiments. Alternatively, one could easily start at the maximum a posteriori (MAP) $\boldsymbol{\theta}_0 = \boldsymbol{\theta}_{\text{MAP}} := \arg\min -\pi_{\text{post}}$ found with e.g. gradient descent. However, if the MAP turns out to be far from $\mathbf{0}$, it may help to assess quality of chain mixing, and the initial non-stationnary regime can easily be handled with a *burn in*.

2.1 Effective Sample Size

We will use the effective sample size in order to monitor the sampling quality. Let us briefly mention a few remarks. We will only monitor the ESS of the first component θ_1. Given a simulated MCMC chain $(\boldsymbol{\theta}^{(0)}, \dots, \boldsymbol{\theta}^{(N)})$, let us define the estimate ESS as

$$\widehat{\text{ESS}}_N\,\{\theta_1\} = \frac{N}{1 + 2\sum_{k=1}^{N}\widehat{R}_{\theta_1}[l]}\;,$$

where $\widehat{R}_{\theta_1}[l]$ is the estimated autocorrelation of θ_1 at lag l, computed with the function `acf` from `statsmodels.tsa.stattools`. Since $\widehat{R}_{\theta_1}[l]$ is computed on a single chain and since there are fewer samples for $l \to N$, the autocorrelation signal is quite noisy. Therefore, we need to truncate the sum. We expect autocorrelation to be zero for $l \to \infty$, as the chain will forget about very old values. Hence, we keep only the first L terms of the sum, until the $(L+1)^{\text{th}}$ lag is negative, i.e.

$$L := \min\left\{l \in \mathbb{N} \mid \widehat{R}_{\theta_1}[l] < 0\right\} - 1\;.$$

2.2 Random Walk Metropolis

As a first attempt we use a Gaussian proposal distribution centered at current state $q(\boldsymbol{\theta}_n, \cdot) = \mathcal{N}(\boldsymbol{\theta}_n, s^2\Sigma_0)$, $s < 1$, where Σ_0 is the same covariance matrix as for the prior π_0. We will monitor the effective sample size and the mixing of the chains for the the first component θ_1, for different values of D and s.

The code of the Random Walk Metropolis algorithm is shown in Listing 4 and summarized in Algorithm 1. There would be two potential bottlenecks in a naive version of this algorithm. First, generating high-dimensional random variables on the fly inside the for loop with `numpy` is very inefficient. Pre-computing all random variables once with the argument `size=N`, although slightly increasing memory usage, induces a great speedup (see benchmark shown in Listing 7). Since the proposal $\mathbf{Y}_n$ at step n is centered at the last iterate $\mathbf{X}_{n-1}$, we can simply shift the pre-computed proposal accordingly. Second, we can divide the number of calls to the density function by a factor two by storing the result of the evaluations $f(\cdot)$, so that $f(\mathbf{X}_{n-1})$ is never computed. If evaluating the density turns out to be the bottleneck of the algorithm, as it will likely be in a high-dimensional setting, we can effectively divide the runtime by two.

Algorithm 1: Random Walk Metropolis Hastings

Data: density $f : \mathbb{R}^D \to \mathbb{R}_+$, covariance matrix $\Sigma_0 \in \mathbb{R}^{D\times D}$, starting point $\mathbf{X}_0 \in \mathbb{R}^D$, chain length N

Precompute $U_1, \dots, U_N \overset{\text{iid}}{\sim} \mathcal{U}([0,1])$

Precompute $\tilde{\mathbf{Y}}_1, \dots, \tilde{\mathbf{Y}}_N \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \Sigma_0)$

for $n = 1, \dots, N$ **do**

  $\mathbf{Y}_n = \tilde{\mathbf{Y}}_n + \mathbf{X}_{n-1}$ `// so that` $\mathbf{Y}_n \sim \mathcal{N}(\mathbf{X}_{n-1}, \Sigma_0)$

  $\alpha = \min\left\{1, \frac{f(\mathbf{Y}_n)}{f(\mathbf{X}_{n-1})}\right\}$

  if $U_n < \alpha$ **then**

    $\mathbf{X}_n = \mathbf{Y}_n$

  else

    $\mathbf{X}_n = \mathbf{X}_{n-1}$

  end

end

2.3 Preconditionned Cranck-Nicolson

The preconditionned Cranck-Nicolson method uses the proposal distribution $q(\boldsymbol{\theta}_n, \cdot) = \mathcal{N}(\boldsymbol{\theta}_n\sqrt{1-s^2}, s^2\Sigma_0)$. Let us show that this proposal q is in *detailed balance* with the prior distribution π_0, i.e.

$$q(\mathbf{x},\mathbf{y})\pi_0(\mathbf{x}) = q(\mathbf{y},\mathbf{x})\pi_0(\mathbf{y}) \quad \forall\, \mathbf{x},\mathbf{y} \in \mathbb{R}^D\ . \tag{7}$$

Since $\pi_0(\mathbf{x}), q(\mathbf{x},\mathbf{y}) > 0\ \forall\, \mathbf{x},\mathbf{y} \in \mathbb{R}^D$, let us compute the two ratios

$$\frac{\pi_0(\mathbf{y})}{\pi_0(\mathbf{x})} = \exp\left(-\frac{1}{2}\mathbf{y}^\top\Sigma_0^{-1}\mathbf{y}\right)\exp\left(\frac{1}{2}\mathbf{x}^\top\Sigma_0^{-1}\mathbf{x}\right)$$

and

$$\begin{aligned}
\frac{q(\mathbf{x},\mathbf{y})}{q(\mathbf{y},\mathbf{x})} &= \exp\left(-\frac{1}{2}\left(\mathbf{y}-\mathbf{x}\sqrt{1-s^2}\right)^\top\left(s^2\Sigma_0\right)^{-1}\left(\mathbf{y}-\mathbf{x}\sqrt{1-s^2}\right)\right)\\
&\quad\times\exp\left(\frac{1}{2}\left(\mathbf{x}-\mathbf{y}\sqrt{1-s^2}\right)^\top\left(s^2\Sigma_0\right)^{-1}\left(\mathbf{x}-\mathbf{y}\sqrt{1-s^2}\right)\right)\\
&= \exp\left(-\frac{1}{2s^2}\mathbf{y}^\top\Sigma_0^{-1}\mathbf{y} - \frac{1-s^2}{2s^2}\mathbf{x}^\top\Sigma_0^{-1}\mathbf{x} - \frac{1}{2s^2}\mathbf{x}^\top\Sigma_0^{-1}\mathbf{x} + \frac{1-s^2}{2s^2}\mathbf{y}^\top\Sigma_0^{-1}\mathbf{y}\right)\\
&= \exp\left(-\frac{1}{2}\mathbf{y}^\top\Sigma_0^{-1}\mathbf{y} + \frac{1}{2}\mathbf{x}^\top\Sigma_0^{-1}\mathbf{x}\right)
\end{aligned}$$

where we did not explicitly show that cross terms involving both $\mathbf{x}$ and $\mathbf{y}$ cancel each other since Σ_0^{-1} is symmetric. Equation (7) then trivially follows. In turn, (π_0, q) being in detailed balance implies that we can rewrite the acceptance rante α as

$$\begin{aligned}
\alpha(\boldsymbol{\theta}_{n-1},\boldsymbol{\theta}_n) &:= \min\left\{1, \frac{\tilde{\pi}_{\text{post}}(\boldsymbol{\theta}_n)}{\tilde{\pi}_{\text{post}}(\boldsymbol{\theta}_{n-1})}\frac{q(\boldsymbol{\theta}_{n-1},\boldsymbol{\theta}_n)}{q(\boldsymbol{\theta}_n,\boldsymbol{\theta}_{n-1})}\right\}\\
&= \min\left\{1, \frac{\pi(\mathbf{y}\mid\boldsymbol{\theta}_n)}{\pi(\mathbf{y}\mid\boldsymbol{\theta}_{n-1})}\frac{\pi_0(\boldsymbol{\theta}_n)\,q(\boldsymbol{\theta}_{n-1},\boldsymbol{\theta}_n)}{\pi_0(\boldsymbol{\theta}_{n-1})\,q(\boldsymbol{\theta}_n,\boldsymbol{\theta}_{n-1})}\right\}\\
&= \min\left\{1, \frac{\pi(\mathbf{y}\mid\boldsymbol{\theta}_n)}{\pi(\mathbf{y}\mid\boldsymbol{\theta}_{n-1})}\right\}\ ,
\end{aligned}$$

i.e., the acceptance rate depends only on the ratio of likelihoods. Let us show that one can lower bound the acceptance rate $\alpha \geq \beta > 0$ independently of the dimension D. We have:

$$\begin{aligned}\frac{\pi(\mathbf{y} \mid \boldsymbol{\theta}_n)}{\pi(\mathbf{y} \mid \boldsymbol{\theta}_{n-1})} &= \exp\left(-\frac{1}{2\sigma^2}\|G(\boldsymbol{\theta}_n) - \mathbf{y}\|_2^2 + \frac{1}{2\sigma^2}\|G(\boldsymbol{\theta}_{n-1}) - \mathbf{y}\|_2^2\right) \\ &\geq \exp\left(-\frac{1}{2\sigma^2}\|G(\boldsymbol{\theta}_n) - \mathbf{y}\|_2^2\right) \\ &= \beta\end{aligned}$$

Let us now investigate more closely the possible values of $G(\cdot)$. First, note that for any non-negative function $f : \mathbb{R} \to \mathbb{R}_+$, and two real numbers $a \leq b$, we have $\int_0^a f(x)dx \leq \int_0^b f(x)dx$. Since the pressure in equation (5) involves integration of the exponential (a positive function), and since G evaluates the pressure at positions $x < 1$, we have that $S_x\{e^{-u(\cdot;\boldsymbol{\theta})}\} \leq S_1\{e^{-u(\cdot;\boldsymbol{\theta})}\}$, which implies $p(x;\boldsymbol{\theta}) \in [0,2]\,\forall\,\boldsymbol{\theta} \in \mathbb{R}^D$, and thus $G(\boldsymbol{\theta}) \in [0,2]^4$, independently of the dimension D. Thus, the distance $\|G(\boldsymbol{\theta}_n) - \mathbf{y}\|_2^2$ is bounded away from infinity, so that the likelihood ratio is lower bounded by $\beta > 0$ independently of D.

The preconditionned Cranck-Nicolson algorithm is shown in Listing 5 and Algorithm 2. Implementation is very similar to random walk, the only differences are the acceptance rate and the fact that we now shift precomputed proposal by $\mathbf{X}_{n-1}\sqrt{1-s^2}$ instead of $\mathbf{X}_{n-1}$.

Algorithm 2: Preconditionned Cranck-Nicolson MCMC

Data: likelihood $\pi(\mathbf{y} \mid \cdot) : \mathbb{R}^D \to \mathbb{R}_+$, covariance matrix $\Sigma_0 \in \mathbb{R}^{D\times D}$, starting point $\mathbf{X}_0 \in \mathbb{R}^D$, chain length N, factor $\sqrt{1-s^2}$

Precompute $U_1, \ldots, U_N \overset{\text{iid}}{\sim} \mathcal{U}([0,1])$

Precompute $\tilde{\mathbf{Y}}_1, \ldots, \tilde{\mathbf{Y}}_N \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \Sigma_0)$

for $n = 1, \ldots, N$ **do**

  $\mathbf{Y}_n = \tilde{\mathbf{Y}}_n + \mathbf{X}_{n-1}\sqrt{1-s^2}$ `// so that` $\mathbf{Y}_n \sim \mathcal{N}(\mathbf{X}_{n-1}\sqrt{1-s^2}, \Sigma_0)$

  $\alpha = \min\left\{1, \frac{\pi(\mathbf{y}|\mathbf{Y}_n)}{\pi(\mathbf{y}|\mathbf{X}_{n-1})}\right\}$

  if $U_n < \alpha$ **then**

    $\mathbf{X}_n = \mathbf{Y}_n$

  else

    $\mathbf{X}_n = \mathbf{X}_{n-1}$

  end

end

2.4 Laplace's Approximation

We now run an independence sampler with a proposal distribution that approximates the posterior distribution around the maximum a posteriori estimator, with a covariance matrix $\tilde{H}$ being a low-rank approximation of the Hessian $\nabla^2 - \log\pi(\boldsymbol{\theta} \mid \mathbf{y})$, computed with the BFGS algorithm. That is, we use the proposal $q(\boldsymbol{\theta}, \cdot) = q(\cdot) = \mathcal{N}(\boldsymbol{\theta}_{\text{MAP}}, \tilde{H} + \alpha^2\mathbb{I}_D)$. We have that

$$\begin{aligned}\boldsymbol{\theta}_{\text{MAP}} &:= \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} -\log\pi(\boldsymbol{\theta} \mid \mathbf{y}) \\ &= \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{2\sigma^2}\|G(\boldsymbol{\theta}) - \mathbf{y}\|_2^2 + \frac{1}{2}\sum_{k=1}^{D}\theta_k^2 k^2\ .\end{aligned}$$

We will use the BFGS algorithm implemented by `scipy.optimize.minimize` in order to compute the MAP and an approximation of the Hessian. The independent sampler is shown in Listing 6. The algorithm is omitted due to its similarity with the previous two algorithms.

3 Results

For Random Walk Metropolis and preconditionned Cranck-Nicolson algorithms, we show the acceptance rate and ESS in function of D and s (see Figure 1 and Figure 2) as well as diagnosis plots displaying histograms,

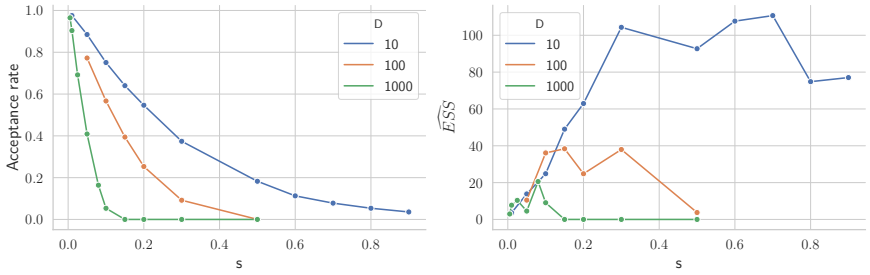

Figure 1: Overview of RWMH sampling quality with respect to acceptance rate (left panel) and effective sample size (right panel) in function of D, s.

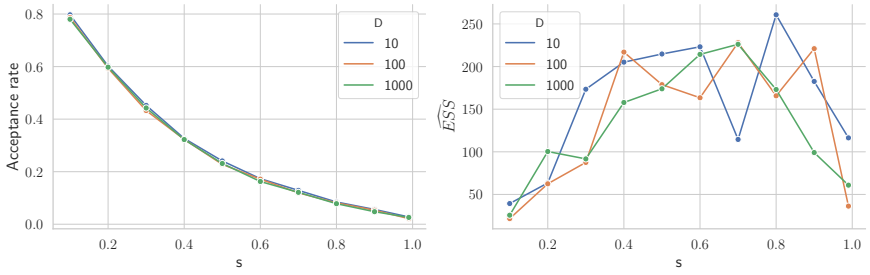

Figure 2: Overview of pCN sampling quality with respect to acceptance rate (left panel) and effective sample size (right panel) in function of D, s.

autocorrelation plots and trace plots (see Figure 3 and Figure 4). We chose $D = 10, 100, 1000$ and different values of s spanning the interval $[0; 1]$.

One can clearly see that the RWM algorithm is quite inefficient: as the dimension D increases, one must decrease the step size s in order to have non-zero acceptance ratio, and the maximum ESS achieved by tuning s remains quite low. In contrast, pCN acceptance rate and ESS is independent of D. The ESS in function of the step size exhibits a (noisy) inversed "u shape": low step sizes yield samples that are often accepted but fail to explore the whole sample space, thus samples are highly correlated and ESS is low. On the other hand, big step sizes characterize a more aggressive approach whose proposed samples are likely to end up in low-probability regions, so that samples are often rejected and thus also have high correlation. Hence, the optimal step size has to be tuned in order to obtain good chain quality.

The diagnoses plots also show how pCN is clearly superior over RWM: sampling quality is much less sensitive to the step size, and chain mixing look similar no matter the value of D by visual inspection of trace plots.

For the independent sampler with Laplace Approximation, we limit ourselves to $D = 10, 100$, as the optimizing function does not run in an acceptable amount of time for $D = 1000$. Figure 5 and Figure 6 present the results. The acceptance rate and ESS is constant independently of dimension D under the threshold $\alpha \leq 10^{-3}$. This method remains somewhat dependent on D: above the thresold, acceptance rate and ESS fall to zero, at a rate that depends on D. Let us now interpret those results. First, we indeed expect very good chain mixing, as the proposed samples are independent of the current chain state. Second, the α parameter should not be too big. Note that $\tilde{H}$ being a low-rank approximation of the Hessian, it is nearly-singular. The α parameter effectively brings $\tilde{H}$ away from singularity. However, the whole purpose of Laplace method is to approximate the target distribution by a Gaussian centered at the MAP estimation. If α is too big, the variance of components θ_i are too big and the proposal fails to correctly approximate the posterior.

Dimension D	$\hat{\mu}_{\text{MCMC}}$
10	1.621066
100	1.613834

Table 2: Values of the MCMC estimators of Equation (8) using independent sampler for $N = 10^5$ samples.

3.0.1 Expected value of scalar quantity

Lastly, we use the independent sampler to compute an MCMC approximator $\hat{\mu}_{\text{MCMC}}$ of the following expected value:

$$\mu := \mathbb{E}_{\boldsymbol{\theta}\sim\pi_{\text{post}}}[I(\boldsymbol{\theta})], \quad I(\boldsymbol{\theta}) := \int_0^1 e^{u(x;\boldsymbol{\theta})}dx, \quad \hat{\mu}_{\text{MCMC}} = \frac{1}{N}\sum_{i=1}^{N} I(\boldsymbol{\theta}^{(i)}) \tag{8}$$

If we assume the chain to be stationnary and ergodic, we can assume that the estimator converges to the true value for infinitely many samples. The true value is unknown, but we will use a surrogate for the error. For $D = 10, 100$, we compute three chain replicas of length $N = 10^1, 10^2, 10^3, 10^4, 10^5$, average over the replicas, and use the longer chain as a surrogate for the exact value, so that we have an approximation of the error. Convergence is shown in Figure 7, the estimators values for $N = 10^5$ are given in Table 2.

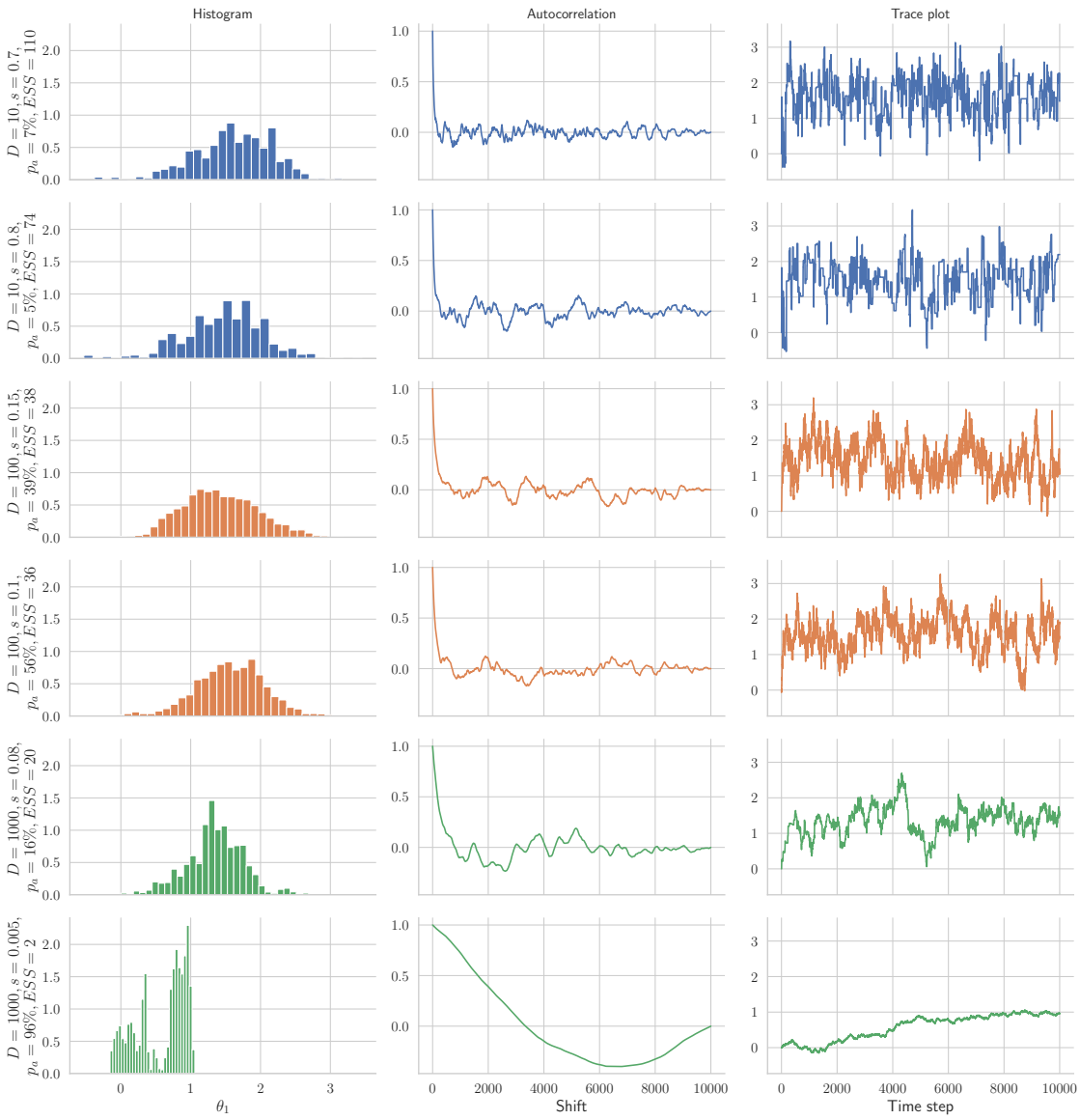

Figure 3: Diagnostics plots of RWMH with histograms of the first component θ_1 (left column), empirical autocorrelation function (middle column), traceplots (right column). Two chains are shown for each value of D, the first one being the best chain obtained in terms of $\widehat{\text{ESS}}$, the second one being of average quality.

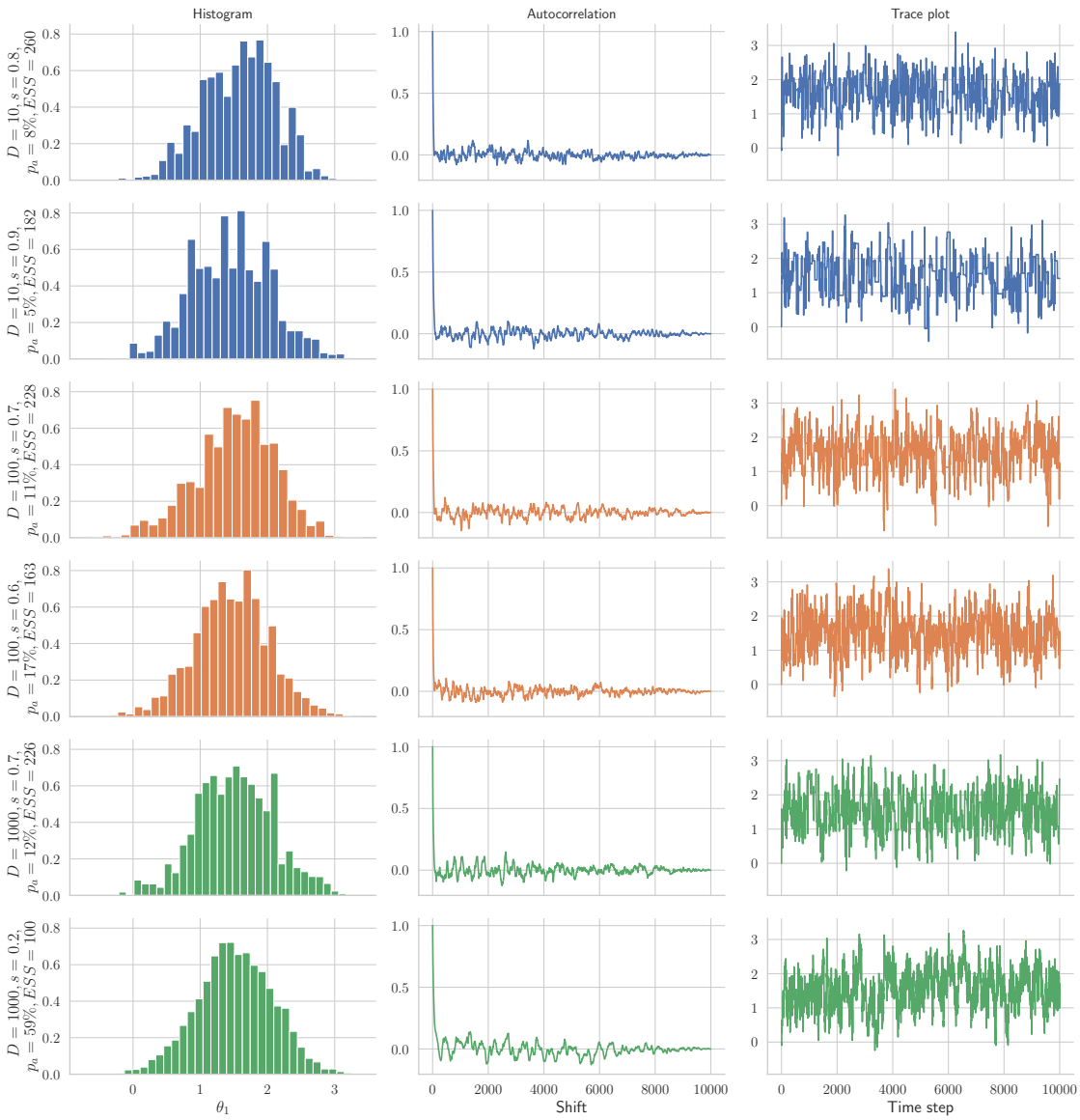

Figure 4: Diagnostics plots of pCN with histograms of the first component θ_1 (left column), empirical autocorrelation function (middle column), traceplots (right column). Two chains are shown for each value of D, the first one being the best chain obtained in terms of $\widehat{\text{ESS}}$, the second one being of average quality.

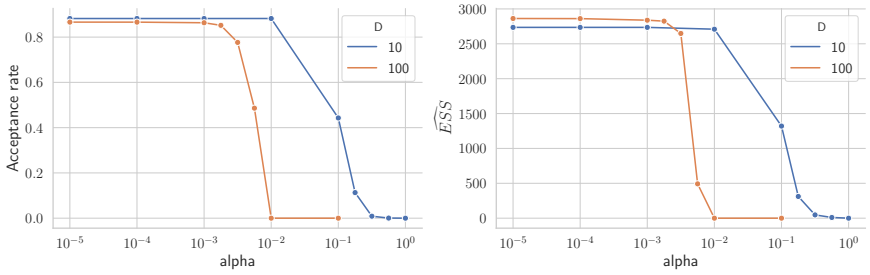

Figure 5: Overview of sampling quality of independent sampler with respect to acceptance rate (left panel) and effective sample size (right panel) in function of D, s.

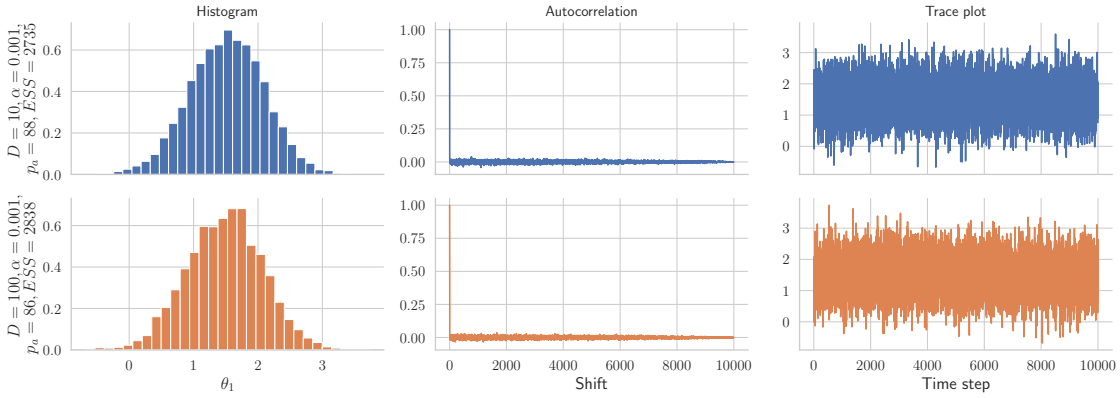

Figure 6: Diagnostics plots of independent sampler with histograms of the first component θ_1 (left column), empirical autocorrelation function (middle column), traceplots (right column). The two chosen chains have the same parameter $\alpha = 10^{-3}$.

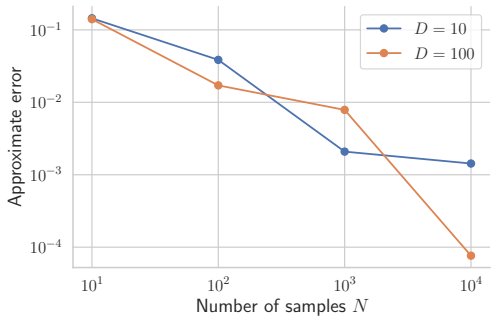

Figure 7: Convergence of MCMC estimator of Equation (8) with independent sampler.

A Appendix: Code

```python
def u(x: np.ndarray, theta: np.ndarray) -> np.ndarray:
    """
    Truncated sine-series expansion of log-permeability
    :param x: position at which to evaluate log permeability, 1D array
    :param theta: Fourier coefficients, 1D array
    """
    res = np.zeros_like(x)
    for k in range(1, theta.size+1):
        res += theta[k-1] * np.sin(np.pi * k * x)
    res *= np.sqrt(2) / np.pi
    return res
```

Listing 1: Log-permeability function $u(x; \boldsymbol{\theta})$ implementing Equation (4).

```python
from scipy import integrate

def G(theta: np.ndarray) -> np.ndarray:
    """
    Observation operator, returns the pressure (solution of PDE) evaluated at position x = [0.2, 0.4
    The closed-form solution of the pressure is an integral, approximated by discretization with
    a number of intervals twice as big as the dimension of theta.
    :param theta: Fourier coefficients, 1D array
    :return: 1D array of length 4, pressure evaluated at positions [0.2, 0.4, 0.6, 0.8]
    """
    # Discretization step
    h = 1 / (2 * theta.size)
    # Grid at which to evaluate the integrand
    x = np.arange(0, 1+h, h)
    # Integrand array
    y = np.exp(-u(x, theta))
    # Integrate with trapezoidal rule
    cumtrpz = integrate.cumtrapz(y, dx=h, initial=0) # initial is completely useless, just simpler
    # Evaluate primitive at x = 0.2, 0.4, 0.6, 0.8. In order for those points to be exactly at a gr
    # the number of discretization intervals must be a multiple of 5.
    k, remainder = divmod(2*theta.size, 5) # n_interval = 5 * k + remainder
    if remainder != 0:
        raise ValueError('Number of subintervals must be a multiple of 5')
    # The values of interest are at index k, 2k, 3k, 4k
    p = 2 * cumtrpz[[k, 2*k, 3*k, 4*k]]
    # Normalize
    p /= cumtrpz[-1]
    return p
```

Listing 2: Observation operator function implementing $G(\boldsymbol{\theta}) = (p(0.2; \boldsymbol{\theta}), p(0.4; \boldsymbol{\theta}), p(0.6; \boldsymbol{\theta}), p(0.8; \boldsymbol{\theta}))$ by approximating the PDE solution of Equation (5) with numerical integration.

```python
def target_density(theta: np.ndarray, sigma_noise: float, y_data: np.ndarray) -> float:
    """
    Un-normalized density of target distribution, that is the invariant distribution of the Markov C
    :param theta: Fourier coefficients of log-permeability, passed to G function
    :param sigma_noise: noise standard deviation
    :param y_data: measured data
    :return: approximate density evaluated at theta for the given measured data
    """
    # Likelihood term
    log_likelihood = - 0.5 / sigma_noise**2 * ((y_data - G(theta)) ** 2).sum()
    # Prior term
    log_prior = - 0.5 * ( (theta * np.arange(1, theta.size+1))**2 ).sum()
    # Return posterior density
    return np.exp(log_likelihood + log_prior)
```

Listing 3: Un-normalized density function $\tilde{\pi}(\boldsymbol{\theta}|\mathbf{y}) \propto \pi(\boldsymbol{\theta}|\mathbf{y})$ implementing Equation (6).

```python
def rwmh(ftilde: Callable, variances: np.ndarray, X0: np.ndarray, N: int, verbose: bool = True):
    """
    Random Walk Metropolis Hastings, i.e. type of Markov Chain Monte Carlo in continuous state space wi
    density centered at current state.
    For a multidimensional state space, the components of the proposal samples are independent, i.e. th
    matrix is diagonal.
    :param ftilde: un-normalized target density being the Markov Chain invariant distribution (after no
    :param variances: 1D array of variances for each component of the proposal distribution
    :param X0: starting point of the chain
    :param N: chain length
    :return: Markov Chain of length N
    """
    start = time()
    X = [X0.copy()]
    # Generate uniform variables used for acceptance/rejection
    Us = np.random.random(N-1)
    # Generate the proposal samples (centered at zero) in advance, much more efficient -> shift them la
    Ys = np.random.multivariate_normal(np.zeros_like(X0), np.diag(variances), size=N-1)
    # Initialize loop
    accepted = 0
    ftilde_previous = ftilde(X[-1])
    for U, Y in zip(Us, Ys):
        # Proposal sample properly shifted so that the Gaussian is centered around the current state
        Y += X[-1]
        # Compute acceptance probability
        ftilde_proposal = ftilde(Y)
        alpha = min(1., ftilde_proposal / ftilde_previous)
        # Accept or reject
        if U < alpha:
            X.append(Y)
            accepted += 1
            ftilde_previous = ftilde_proposal
        else:
            X.append(X[-1].copy())
            # No need to update ftilde_previous

    p_accept = accepted / (N-1)
    if verbose:
        print(f'took {f"{time() - start:.3} s":<8} acceptance rate {p_accept:.3}')

    return np.array(X), p_accept
```

Listing 4: Random Walk Metropolis Hastings algorithm.

```python
def pcn(ftilde: Callable, variances: np.ndarray, X0: np.ndarray, factor: float, N: int, verbose: bool =
    """
    Preconditionned Cranck-Nicolson.
    :param ftilde: un-normalized target density being the Markov Chain invariant distribution (after no
    :param variances: 1D array of variances for each component of the proposal distribution
    :param X0: starting point of the chain
    :param N: chain length
    :return: Markov Chain of length N
    """
    start = time()
    X = [X0.copy()]
    # Generate uniform variables used for acceptance/rejection
    Us = np.random.random(N-1)
    # Generate the proposal samples (centered at zero) in advance, much more efficient -> shift them la
    Ys = np.random.multivariate_normal(np.zeros_like(X0), np.diag(variances), size=N-1)
    # Initialize loop
    accepted = 0
    ftilde_previous = ftilde(X[-1])
    for U, Y in zip(Us, Ys):
        # Proposal sample properly shifted so that the Gaussian is centered around the current state
        Y += X[-1] * factor
        # Compute acceptance probability
        ftilde_proposal = ftilde(Y)
        alpha = min(1., ftilde_proposal / ftilde_previous)
        # Accept or reject
        if U < alpha:
            X.append(Y)
            accepted += 1
            ftilde_previous = ftilde_proposal
        else:
            X.append(X[-1].copy())
            # No need to update ftilde_previous

    p_accept = accepted / (N-1)
    if verbose:
        print(f'took {f"{time() - start:.3} s":<8} acceptance rate {p_accept:.3}')

    return np.array(X), p_accept
```

Listing 5: Preconditionned Cranck-Nicolson Algorithm.

```python
def indep_sampler(ftilde: Callable, proposal_density: Callable, proposal_sampler: Callable,
                  X0: np.ndarray, N: int, verbose: bool = True):
    """
    MCMC independent sampling.
    :param ftilde: target distribution
    :param proposal_density: function taking 1 input and returning density of the proposal
    :param proposal_sampler: function taking 1 input (number of samples) and returns N iid samples from
    :param X0: starting point
    :param N: chain length
    :param verbose: whether to print debugging informations
    :return:
    """
    start = time()
    X = [X0.copy()]
    # Generate uniform variables used for acceptance/rejection
    Us = np.random.random(N-1)
    # Generate the proposal samples
    Ys = proposal_sampler(N-1)
    # Initialize loop
    accepted = 0
    ftilde_previous = ftilde(X[-1])
    for U, Y in zip(Us, Ys):
        # Compute acceptance probability
        ftilde_proposal = ftilde(Y)
        alpha = min(1., ftilde_proposal / ftilde_previous * proposal_density(X[-1]) / proposal_density(
        # Accept or reject
        if U < alpha:
            X.append(Y)
            accepted += 1
            ftilde_previous = ftilde_proposal
        else:
            X.append(X[-1].copy())
            # No need to update ftilde_previous

    p_accept = accepted / (N-1)
    if verbose:
        print(f'took {f"{time() - start:.3} s":<8} acceptance rate {p_accept:.3}')

    return np.array(X), p_accept
```

Listing 6: Independent Sampler Algorithm.

B Appendix - Benchmarks

```
>>> def dummy(x):
>>>     return x

>>> %%timeit
>>> for _ in range(int(1e4)):
>>>    Z = np.random.multivariate_normal(np.zeros(100), np.eye(100))
>>>    res = dummy(Z)
6.96 s +- 640 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)

>>> %%timeit
>>> Z = np.random.multivariate_normal(np.zeros(100), np.eye(100), size=int(1e4))
>>> for i in range(int(1e4)):
>>>    res = dummy(Z[i])
27.4 ms +- 3.35 ms per loop (mean +- std. dev. of 7 runs, 10 loops each) each)
```

Listing 7: Benefits of pre-sampling for high-dimensional distributions.