Stochastic Simulation

Autumn Semester 2024

Prof. Fabio Nobile *Assistant: Matteo Raviola*

Lab 12 – 5 December 2024

Markov Chain Monte Carlo

Exercise 1

Consider the Random Walk Metropolis–Hastings (RWMH) algorithm with proposal density $q(x, y) = g_\sigma(y - x)$ and target density $f\colon \mathbb{R} \to \mathbb{R}^+$. Let g_σ denote the density of the $\mathcal{N}(0, \sigma^2)$ distribution and suppose that

$$f(y) = \frac{1}{Z} \exp\left[-\left(\frac{1}{4}y^4 - \frac{1}{2}y^2 + \frac{1}{4}\right)\right],$$

where Z is such that f is a PDF on $\mathcal{X} = \mathbb{R}$. Suppose we wish to estimate $\mu = \mathbb{E}_f(\phi)$ for a suitable function $\phi\colon \mathbb{R} \to \mathbb{R}$. Let $\hat{\mu}_n^{\mathrm{MH}}$ be the estimator for μ based on the Markov chain of length n generated by the RWMH algorithm. Derive asymptotic confidence intervals for μ at probability level α using the CLT for Metropolis–Hastings Markov Chains. Within your simulations, estimate[1] this confidence interval and stop the Markov chain once the half-length of the interval is smaller than a given tolerance $\tau > 0$. Implement the following heuristics to estimate the required *time average variance constant* and compare their performance for different functions ϕ; namely $\phi(x) = x^p, p \in \mathbb{N}$. The time average variance constant is the asymptotic variance introduced in Theorem 8.10 of the lecture notes.

1. *Initial positive sequence estimator:*

$$\tilde{\sigma}^2 \approx \hat{\sigma}^2_{\mathrm{pos},n} := -\hat{c}_n(0) + 2\sum_{k=1}^{K}\big(\hat{c}_n(2k) + \hat{c}_n(2k+1)\big),$$

   where K is the largest integer such that $\hat{c}_n(2k) + \hat{c}_n(2k+1) > 0$ for all $k = 1, \dots, K$. Here,

$$\hat{c}_n(j) := \frac{1}{n}\sum_{i=1}^{n-j}\Big(\phi(X_i) - \hat{\mu}_n^{\mathrm{MH}}\Big)\Big(\phi(X_{i+j}) - \hat{\mu}_n^{\mathrm{MH}}\Big)$$

   is an appropriate covariance estimator in this context.

2. *Initial monotone sequence estimator:*

$$\tilde{\sigma}^2 \approx \hat{\sigma}^2_{\mathrm{mon},n} := -\hat{c}_n(0) + 2\sum_{k=1}^{K}\min_{1\le j\le k}\big\{\hat{c}_n(2j) + \hat{c}_n(2j+1)\big\},$$

   where K and $\hat{c}_n$ are as for the initial positive sequence estimator above.

[1] The estimation has to be carried out on-the-fly, that is while the Markov chain evolves.

3. *Batch means estimator:* Suppose the Markov chain is $X_1, \ldots, X_n$ at iteration n. Divide these n values into $N_b \in \mathbb{N}$ batches, each of length $N_\ell = n/N_b$. A typical decomposition is $N_\ell = n^{1-a}$ and $N_b = n^a$ for $a \in [0,1]$, for example $a = 0.5$, modulo integer rounding. Let

$$\hat{\mu}_i = \frac{1}{N_\ell} \sum_{j=(i-1)N_\ell+1}^{iN_\ell} \phi(X_j) \ , \quad i = 1, \ldots, N_b \ ,$$

be the sample mean of the i-th batch. For N_ℓ sufficiently large, one can consider the batch means $\hat{\mu}_1, \hat{\mu}_2, \ldots, \hat{\mu}_{N_b}$ to be approximately mutually independent. Consequently, one can estimate the time average variance constant σ^2 by the sample variance estimator for independent realizations:

$$\tilde{\sigma}^2 \approx \hat{\sigma}^2_{\mathrm{BM},n} := \frac{n}{N_b} \frac{1}{N_b - 1} \sum_{i=1}^{N_b} \left(\hat{\mu}_i - \hat{\mu}_n^{\mathrm{MH}} \right)^2 .$$

In addition, instead of using all of the points in the Markov chain, one can as well discard a burn-in time B and replace $\sum_{k=1}$ with $\sum_{k=B}$ in the above estimators. Experiment with the effects of burn-in time on the above asymptotic variance estimators.

Solution

An asymptotic confidence interval can be derived in a straightforward manner from Theorem 8.17 of the lecture notes. Let $\{X_n\}_{n=1}^N$ denote the generated Markov chain and $\hat{\mu}_N^{mcmc}$ denote the standard MCMC estimator of $\mu = \mathbb{E}[\phi]$. We know that asymptotically as $N \to \infty$,

$$\sqrt{N}(\hat{\mu}_N^{mcmc} - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2_{mcmc}), \tag{1}$$

where σ^2_{mcmc} denotes the asymptotic variance or time average variance constant of the Markov chain. A $1-\alpha$ confidence interval then simply reads $I = \left[\mu \pm \frac{C_{1-\alpha/2}\sigma_{mcmc}}{\sqrt{N}}\right]$ where C_δ denotes the inverse of the CDF of the standard normal distribution evaluated at δ.

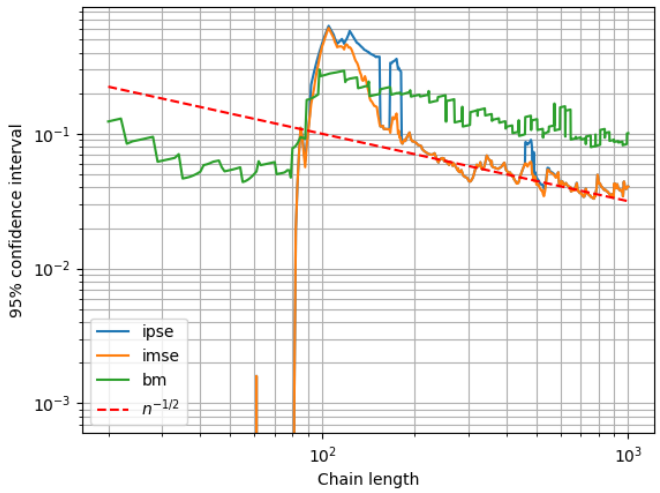

Figure 1: Convergence for different asymptotic variance estimators

Python code:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as st
import statsmodels.api as sm

def f(y):
    return np.exp(-(y**2-0.5)**2)

def get_K(auto_covars):
    K = None
    for k in range(int(len(auto_covars)/2)):
        if (auto_covars[2*k] + auto_covars[2*k+1]) <= 0:
            K = k
            break
    if K is None:
        K = int(len(auto_covars)/2-1)
    return K

def sigma_ipse(auto_covars):
    K = get_K(auto_covars)
    return -auto_covars[0] + 2*np.sum([auto_covars[2*k]+auto_covars[2*k+1] for k in range(1,K

def sigma_imse(auto_covars):
    K = get_K(auto_covars)
    return -auto_covars[0] + 2*np.sum([np.min([auto_covars[2*j]+auto_covars[2*j+1] for j in r

def sigma_bm(Xchain,a=0.5):
    n =  len(Xchain)
    Nb = int(np.floor(n**a))
    Nl = int(np.floor(n/Nb))
    batch_means = []
    mcmc_mean = np.mean(Xchain)
    for i in range(1,Nb+1):
        batch_means.append( np.mean( [Xchain[j] for j in range((i-1)*Nl,i*Nl)] ) )

    batch_means = np.array(batch_means)
    return np.sum((batch_means-mcmc_mean)**2)*n/(Nb-1)/Nb

def compute_auto_covars(Xchain):
    Xmean = np.mean(Xchain)
    N = len(Xchain)
    autocov = np.zeros(N)
    for k in range(N):
        for i in range(N-k):
            autocov[k] += ((Xchain[i+k])-Xmean)*(Xchain[i]-Xmean)
```

```
        autocov[k] = (1/(N-1))*autocov[k]
    return autocov

# Simulation parameters
alpha = 0.05
coeff = st.norm.ppf(1-alpha/2)
sigma = 4.0
p = 2

# Start the chain
Xchain = np.array([0])
n = 0
error1s = []
error2s = []
error3s = []

iteration = 0
iter_min = 20 # force a minimum number of iteration before checking stopping criterion
iter_stop = 1000 # max iteration limit

ns = []
while True:
    # Compute MH step
    Xn = Xchain[n]
    Xnew = st.norm.rvs()*sigma
    uniform = st.uniform.rvs()
    ratio = np.min([1, ( f(Xnew)*st.norm.pdf(Xn) )/( f(Xn)*st.norm.pdf(Xnew) )])
    if uniform <= ratio:
        Xchain = np.append(Xchain, Xnew)
    else:
        Xchain = np.append(Xchain, Xn)
    n = n+1

    if n >= iter_min:
        # Compute phi(X)
        Xchain_p = Xchain**p

        # Compute autocovariances
        # Autocovariances computed from scratch, not incrementally.
        # Can be reimplemented for speed
        auto_covars = compute_auto_covars(Xchain_p)

        # Compute variances
        sigma1 = sigma_ipse(auto_covars)
        sigma2 = sigma_imse(auto_covars)
        sigma3 = sigma_bm(Xchain_p)
```

```
        error1 = coeff*sigma1/np.sqrt(n)
        error2 = coeff*sigma2/np.sqrt(n)
        error3 = coeff*sigma3/np.sqrt(n)

        print(n, error1, error2, error3)
        error1s.append(error1)
        error2s.append(error2)
        error3s.append(error3)

        ns.append(n)
    if n >= iter_stop:
        break

plt.loglog(ns,error1s,label='ipse')
plt.loglog(ns,error2s,label='imse')
plt.loglog(ns,error3s,label='bm')
plt.loglog(ns,np.array(ns)**-0.5,'--',color='red',label=r'$n^{-1/2}$')
plt.grid(which='both')
plt.xlabel('Chain length')
plt.ylabel('95% confidence interval')
plt.legend()
plt.savefig('../figures/convergence.png')
plt.show()
```

Exercise 2

Consider the following probability density function in $\mathbb{R}^2$

$$f(\boldsymbol{x}) = \frac{1}{Z}\left[\exp\left\{-(1-x_1)^2 - (x_2 - x_1^2)^2\right\} + \exp\left\{-(x_1+1)^2 - (x_2+3+x_1^2)^2\right\}\right], \quad \boldsymbol{x} = (x_1, x_2)^T \in \mathbb{R}^2,$$

where Z is the (unknown) normalization constant. To generate samples from f, we consider Metropolis-Hastings (MH) type MCMC algorithms. Let us denote with $p(\cdot; \boldsymbol{\mu}, \Sigma)$ [2] the pdf of a $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ multivariate Gaussian random variable and by

- K_1 the Markov kernel associated to an *Independent Sampler* MH algorithm with proposal density $q_1(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2}p(\boldsymbol{y}; (1,1)^T, 0.5\, I_{2\times 2}) + \frac{1}{2}p(\boldsymbol{y}; (-1,-3)^T, 0.5\, I_{2\times 2})$
- K_2 the Markov kernel associated to a *Random Walk* MH algorithm with proposal density $q_2(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y}; \boldsymbol{x}, \sigma^2\, I_{2\times 2})$, where $\sigma = 0.15$.

1. Write the explicit expression of the densities corresponding to the Markov kernels K_1 and K_2.
2. Consider now the kernel $K(\omega) = \omega K_1 + (1-\omega)K_2$, with $\omega \in [0,1]$. Show that $K(\omega)$ is a reversible Markov kernel that has f as invariant distribution.

[2] $p(\boldsymbol{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{\det(2\pi\Sigma)}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$

3. Implement a MCMC algorithm that uses the Markov kernel $K(\omega)$ to estimate $\mathbb{E}_f[\|\mathbf{X}\|^2]$, with $\mathbf{X} = (X_1, X_2)^T \sim f(\cdot)$. Include the usual MCMC diagnostic plots in your experiment and describe how you would chose the sample size (length of the chain) to guarantee an error on the computation of $\mathbb{E}_f[\|\mathbf{X}\|^2]$ smaller than $\varepsilon = 1$ with confidence at least .95. Estimate also the expected acceptance rate $\chi(\omega)$ of the implemented algorithm. Try at least two different values for ω.

4. Write a closed-form formula for the expected acceptance rate $\chi(\omega)$ from the expression of the kernel $K(\omega)$. How could one use the results in the previous point to find the value of ω that leads to a target acceptance rate, say $\chi(\omega) = 0.4$?

Hint: You can use the following `Python` commands to plot your results:

```
import statsmodels.graphics.tsaplots as sm
import seaborn as sbn
.
.
.
sbn.kdeplot(x)   # To plot emperical density of samples x
sm.plot_acf(QoI)  # To plot autocorrelation plot of a quantity of interest QoI
```

Solution

1. Let $q(\boldsymbol{x}, \boldsymbol{y})$, $q_1(\boldsymbol{x}, \boldsymbol{y})$ and $q_2(\boldsymbol{x}, \boldsymbol{y})$ be the densities corresponding to the transition kernels K, K_1 and K_2 respectively. The MH density corresponding to K_1 reads:

$$p_1(\boldsymbol{x}, \boldsymbol{y}) = \alpha_1(\boldsymbol{x}, \boldsymbol{y}) q_1(\boldsymbol{x}, \boldsymbol{y}) + \left[1 - \int \alpha_1(\boldsymbol{x}, \boldsymbol{y}) q_1(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y}\right] \delta_{\boldsymbol{x}}(\boldsymbol{y}),$$

$$\text{where } \alpha_1(\boldsymbol{x}, \boldsymbol{y}) = \min\left\{1, \frac{q_1(\boldsymbol{x}, \boldsymbol{y}) f(\boldsymbol{y})}{q_1(\boldsymbol{y}, \boldsymbol{x}) f(\boldsymbol{x})}\right\}$$

   and similarly for K_2.

2. K_1 and K_2 are reversible. Convex combinations of reversible kernels are also reversible:

$$f(\boldsymbol{x}) K(\boldsymbol{x}, \boldsymbol{y}) = f(\boldsymbol{y}) K(\boldsymbol{y}, \boldsymbol{x})$$

$$\implies w f(\boldsymbol{x}) K_1(\boldsymbol{x}, \boldsymbol{y}) + (1 - w) f(\boldsymbol{x}) K_2(\boldsymbol{x}, \boldsymbol{y}) = w f(\boldsymbol{y}) K_1(\boldsymbol{y}, \boldsymbol{x}) + (1 - w) f(\boldsymbol{y}) K_2(\boldsymbol{y}, \boldsymbol{x}),$$

   where we have used the reversibility of K_1 and K_2.

3. A python implementation of this code is shown below.

4. We know that

$$\begin{aligned}
\mathbb{E}_f[\chi(w)] &= w \mathbb{E}_f[\alpha_1(\boldsymbol{x}, \boldsymbol{y})] + (1 - w) \mathbb{E}_f[\alpha_2(\boldsymbol{x}, \boldsymbol{y})] \\
&= w \int \alpha_1(\boldsymbol{x}, \boldsymbol{y}) q_1(\boldsymbol{x}, \boldsymbol{y}) f(x) d\boldsymbol{x} d\boldsymbol{y} \\
&\quad + (1 - w) \int \alpha_2(\boldsymbol{x}, \boldsymbol{y}) q_2(\boldsymbol{x}, \boldsymbol{y}) f(x) d\boldsymbol{x} d\boldsymbol{y}.
\end{aligned}$$

   One can then find $w \in [0, 1]$ that is a root of $g(w) = \mathbb{E}_f[\chi(w)] - \alpha_{opt}$ either by using an objective function in python, or doing a "sweep" of the MH algorithm for different values of $w \in [0, 1]$ (brute force).

Python code:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as st
import statsmodels.api as sm

def f(x, y):
    return np.exp(-(1-x)**2 - (y-x**2)**2) + np.exp(-(1+x)**2 - (y+3+x**2)**2)

def g(x, y, xm, ym, w, sigma):
    return w * K1(x,y) + (1-w) * K2(x,y,xm,ym,sigma)

def K1(x,y):
    return 0.5 * st.multivariate_normal.pdf([x,y],mean=(1,1),cov=0.5*np.eye(2)) \
        + 0.5 * st.multivariate_normal.pdf([x,y],mean=(-1,-3),cov=0.5*np.eye(2))

def K2(x,y,xm,ym,sigma):
    return st.multivariate_normal.pdf([x,y],mean=(xm,ym),cov=sigma*np.eye(2))

# Plot density
x = np.linspace(-3,3,101)
y = np.linspace(-10,7,101)
X,Y = np.meshgrid(x,y)
Z = f(X,Y)
plt.contourf(X,Y,Z)
plt.savefig('../figures/density.png')

# Parameters
alpha = 0.05
coeff = st.norm.ppf(1-alpha/2)
w = 0.7
sigma = 0.15

# Begin chain
Xn = [0,0]
Xchain = [Xn]
n = 1
n_acc = 0
while True:
    print(n)
    u1 = st.uniform.rvs()
    if u1 <= w:
        u2 = st.uniform.rvs()
        if u2<=0.5:
            Xnew = [1 + np.sqrt(0.5)*st.norm.rvs(), 1 + np.sqrt(0.5)*st.norm.rvs()]
        else:
```

```
            Xnew = [-1 + np.sqrt(0.5)*st.norm.rvs(), -3 + np.sqrt(0.5)*st.norm.rvs()]
    else:
        Xnew = [Xn[0]+sigma*st.norm.rvs(), Xn[1]+sigma*st.norm.rvs()]

    ratio = ( f(Xnew[0], Xnew[1]) * g(Xn[0], Xn[1], Xnew[0], Xnew[1], w, sigma) ) / ( f(Xn[0]
    if st.uniform.rvs() <= np.min([1,ratio]):
        Xn = Xnew
        n_acc +=1
    Xchain.append(Xn)
    n = n + 1
    if n==1000:
        break

print(n_acc/n)
plt.plot([X[0] for X in Xchain], [X[1] for X in Xchain],'+')
plt.show()
```