Stochastic Simulation
Professor: Prof. Fabio Nobile
Assistant: Eliott Van Dieren

Week of 2025-12-08 to 2025-12-12

Lab 13 of Thursday 11th December 2025

Exercice 1.

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

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

where Z is the (unknown) normalization constant. To generate samples from f, we consider Metropolis-Hastings (MH) type MCMC algorithms. Let us denote with $p(\cdot; \mu, \Sigma)$ [1] the pdf of a $\mathcal{N}(\mu, \Sigma)$ multivariate Gaussian random variable and by

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

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
```

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

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

np.random.seed(42)

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)

# 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],
     ↪  Xn[1]) * g(Xnew[0], Xnew[1], Xn[0], Xn[1], w, sigma) )
    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()
```

Exercice 2.

Consider the problem of estimating

$$\frac{d}{d\theta} I(\theta) \tag{2.1}$$

where

$$I(\theta) = \mathbb{E}[\mathbb{1}_{\{\theta X > 1\}}] \tag{2.2}$$

and $X \sim \mathcal{N}(0,1)$. Since the indicator function is discontinuous, we may consider the smoothed version of the integral defined by

$$I_\varepsilon(\theta) = \mathbb{E}\left[\Phi\left(\frac{(\theta X - 1)}{\varepsilon}\right)\right], \tag{2.3}$$

where Φ denotes the CDF of a standard normal random variable. Address the following points.

1) Compute the analytic value of $\frac{d}{d\theta}I(\theta)$ and $\frac{d}{d\theta}I_\varepsilon(\theta)$.

2) Implement the IPA method to compute an approximation of $I_\varepsilon(\theta)$. Using the previously computed values, assess the Monte Carlo error and the error with respect to the smoothing parameter ε.

3) Implement the LR method on the non-regularized function $I(\theta)$.

Solution

1) The function $I(\theta)$ is given by

$$I(\theta) = \mathbb{E}[\mathbb{1}_{\{\theta X>1\}}] = \int_{1/\theta}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\, dx = 1 - \Phi\Big(\frac{1}{\theta}\Big),$$

where $\Phi(x)$ is the CDF of the standard normal distribution, hence differentiating $I(\theta)$ with respect to θ gives

$$\frac{d}{d\theta}I(\theta) = -\phi\Big(\frac{1}{\theta}\Big)\cdot\Big(-\frac{1}{\theta^2}\Big) = \frac{\phi\Big(\frac{1}{\theta}\Big)}{\theta^2},$$

where ϕ is the PDF of a standard normal distribution.

The derivative of $I_\varepsilon(\theta)$ with respect to θ is

$$\frac{d}{d\theta}I_\varepsilon(\theta) = \int_{-\infty}^{\infty} \phi\Big(\frac{\theta x - 1}{\varepsilon}\Big)\cdot\frac{x}{\varepsilon}\cdot\frac{1}{\sqrt{2\pi}}e^{-x^2/2}dx,$$

hence

$$\frac{d}{d\theta}I_\varepsilon(\theta) = \int_{-\infty}^{\infty} \frac{1}{2\pi\varepsilon} e^{-\frac{1}{2}\left[\left(\frac{\theta x-1}{\varepsilon}\right)^2 + x^2\right]} x\, dx.$$

Completing the square at the exponent and using the formula for the mean of a Gaussian variable gives

$$\frac{d}{d\theta}I_\varepsilon(\theta) = e^{-\frac{1}{2(\varepsilon^2+\theta^2)}}\cdot\frac{\theta}{\sqrt{2\pi}(\varepsilon^2+\theta^2)^{3/2}}.$$

2) The results are shown in Figure 1. It is interesting to note that, as $\varepsilon \to 0^+$, the constant of the MC approximation error increases. This is consistent with the fact that, in the limit, the IPA approximaiton is not applicable. Furthermore, this suggests that, if we are given a certain error tolerance, one should take the largest ε such that the analytical error is below a fraction of the tolerance in order to achieve a total error below the tolerance with the least number of samples.

3) Note that, defining $\tilde{X} = \theta X \sim \mathcal{N}(0,\theta^2)$ with p.d.f. f_θ, $I(\theta)'$ can be written as

$$\begin{aligned}
\frac{d}{d\theta}I(\theta) &= \frac{d}{d\theta}\mathbb{E}[\mathbb{1}_{\{\tilde{X}>1\}}] \\
&= \mathbb{E}\Big[\mathbb{1}_{\{\tilde{X}>1\}}\frac{d}{d\theta}\log(f_\theta(\tilde{X}))\Big] \\
&= \mathbb{E}\Big[\mathbb{1}_{\{\tilde{X}>1\}}\frac{d}{d\theta}\Big(-\log(\theta) - \frac{1}{2\theta^2}\tilde{X}^2\Big)\Big] \\
&= \mathbb{E}\Big[\mathbb{1}_{\{\tilde{X}>1\}}\Big(-\frac{1}{\theta} + \frac{1}{\theta^3}\tilde{X}^2\Big)\Big].
\end{aligned} \tag{2.4}$$

The results are shown in Figure 1.

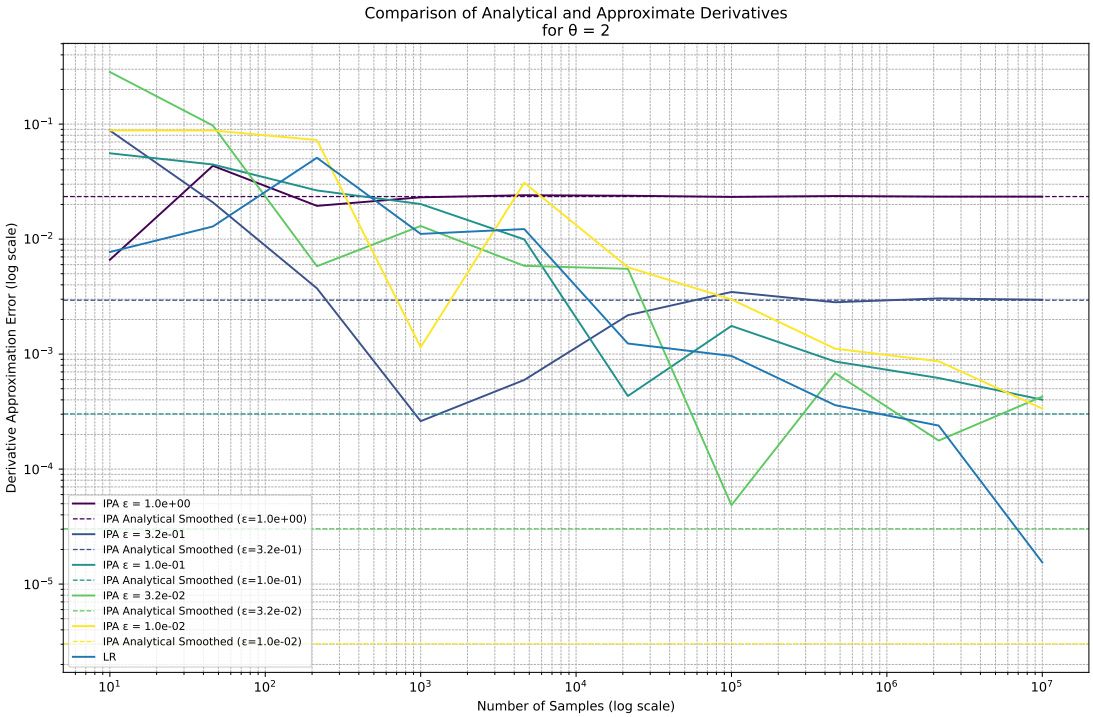

Figure 1: Convergence of the IPA approximation of $I'_\varepsilon(\theta)$ for different values of the smoothing parameter ε and the LR approximation of $I'(\theta)$, in $\theta = 2$.

Python code:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

np.random.seed(42)

# Define auxiliary functions
def phi(x):
    """Standard normal PDF."""
    return norm.pdf(x)

def I_theta_derivative(theta):
    """Analytical derivative of I(theta)."""
    return phi(1 / theta) / theta**2

def I_epsilon_analytical_derivative(theta, epsilon):
    """Analytical derivative of I_epsilon(theta)."""
    denominator = (
        np.exp(1 / (2 * (epsilon**2 + theta**2))) *
        epsilon *
        np.sqrt(2 * np.pi) *
        (epsilon**2 + theta**2) *
        np.sqrt(1 + theta**2 / epsilon**2)
    )
    return theta / denominator
```

```python
def I_epsilon_approximation(theta, epsilon, n_samples=10000):
    """Monte Carlo approximation of I_epsilon(theta)."""
    X = np.random.normal(0, 1, n_samples)
    smoothed_term = norm.cdf((theta * X - 1) / epsilon)
    return np.mean(smoothed_term)

def I_epsilon_derivative_approximation(theta, epsilon, n_samples=10000):
    """IPA approximaiton of the derivative of I_epsilon(theta)."""
    X = np.random.normal(0, 1, n_samples)
    phi_term = phi((theta * X - 1) / epsilon)
    return np.mean(phi_term * X / epsilon)

def likelihood_ratio_method(theta, n_samples=10000):
    """Likelihood Ratio (LR) method for dI/dtheta."""
    X = np.random.normal(0, 1, n_samples)
    indicator = (theta * X > 1)
    term1 = -1 / theta
    term2 = X**2 / theta
    return np.mean(indicator * (term1 + term2))

# Parameters
theta = 2
epsilons = np.logspace(0, -2, 5)  # Log-spaced grid for epsilon from 1 to 10^-4
n_samples_grid = np.logspace(1, 7, 10, dtype=int)  # Log-spaced grid for sample sizes

# Compute analytical values
analytical_value = I_theta_derivative(theta)
smoothed_analytical_values = [I_epsilon_analytical_derivative(theta, epsilon) for epsilon in
↪  epsilons]
analytical_errors = [abs(smoothed - analytical_value) for smoothed in smoothed_analytical_values]

plt.figure(figsize=(12, 8))
colors = plt.cm.viridis(np.linspace(0, 1, len(epsilons)))

# Loop over epsilon values
for i, epsilon in enumerate(epsilons):
    approximation_errors = []
    # Compute IPA approximations
    for n_samples in n_samples_grid:
        approx = I_epsilon_derivative_approximation(theta, epsilon, n_samples)
        approximation_errors.append(abs(approx-analytical_value))

    # Plot IPA approximation errors
    plt.plot(n_samples_grid, approximation_errors, label=f"IPA   = {epsilon:.1e}",
    ↪  color=colors[i])
    plt.axhline(abs(smoothed_analytical_values[i]-analytical_value), color=colors[i],
    ↪  linestyle='--', linewidth=1,
                label=f"IPA Analytical Smoothed ( ={epsilon:.1e})")

# Compute LR approximaitons
LR_approximation_errors = []
for n_samples in n_samples_grid:
    approx = likelihood_ratio_method(theta, n_samples)
    LR_approximation_errors.append(abs(approx-analytical_value))

# Plot LR approximaiton errors
plt.plot(n_samples_grid, LR_approximation_errors, label=f"LR")

# Customize the plot
plt.xscale('log')
plt.yscale('log')
plt.xlabel("Number of Samples (log scale)")
plt.ylabel("Derivative Approximation Error (log scale)")
```

```
plt.title(f"Comparison of Analytical and Approximate Derivatives\nfor   = {theta}")
plt.legend(loc="best", fontsize='small')
plt.grid(True, which="both", linestyle="--", linewidth=0.5)
plt.tight_layout()
plt.savefig('../figures/smoothed-ipa-approximaitons-theta{}.pdf'.format(theta))
# plt.show()
```