Computer Practical: Metropolis-Hastings

In this computer practical, you can use either Python or R. There is a list of useful Python and R functions at the end of this handout.

Theoretical background: bivariate Normal distribution

We return to the problem of sampling from the bivariate Normal distribution, only this time using the Metropolis-Hastings algorithm. The density of $\mathbf{X} = (X_1, X_2)' \sim \mathsf{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ distribution with $\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$ and $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}$ is

$$f_{(\boldsymbol{\mu},\boldsymbol{\Sigma})}(x_1, x_2) = \frac{1}{2\pi|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

where $\mathbf{x} = (x_1, x_2)$.

Sampling from the bivariate Normal distribution using the Metropolis-Hastings algorithm

Suppose that we want to generate samples from $f(\mathbf{X})$, where $\mathbf{X} = (X_1, \ldots, X_p)$. We have seen in the lectures that the Metropolis-Hastings algorithm with symmetric proposal distribution g proceeds as follows.

Start with $\mathbf{X}^{(0)} := (X_1^{(0)}, \ldots, X_p^{(0)})$. Iterate for $t = 1, 2, \ldots$

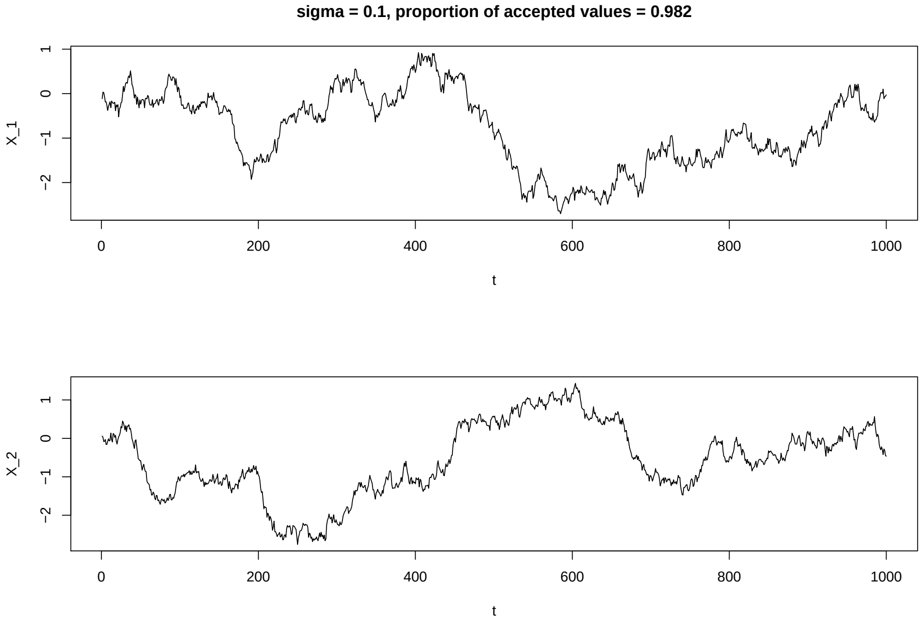
1. Draw $\boldsymbol{\epsilon} \sim g$ and set $\mathbf{X} = \mathbf{X}^{(t-1)} + \boldsymbol{\epsilon}$.
2. Compute
$$\alpha(\mathbf{X}, \mathbf{X}^{(t-1)}) = \min\left\{1, \frac{f(\mathbf{X})}{f(\mathbf{X}^{(t-1)})}\right\}.$$
3. With probability $\alpha(\mathbf{X}, \mathbf{X}^{(t-1)})$, set $\mathbf{X}^{(t)} = \mathbf{X}$, otherwise, set $\mathbf{X}^{(t)} = \mathbf{X}^{(t-1)}$.

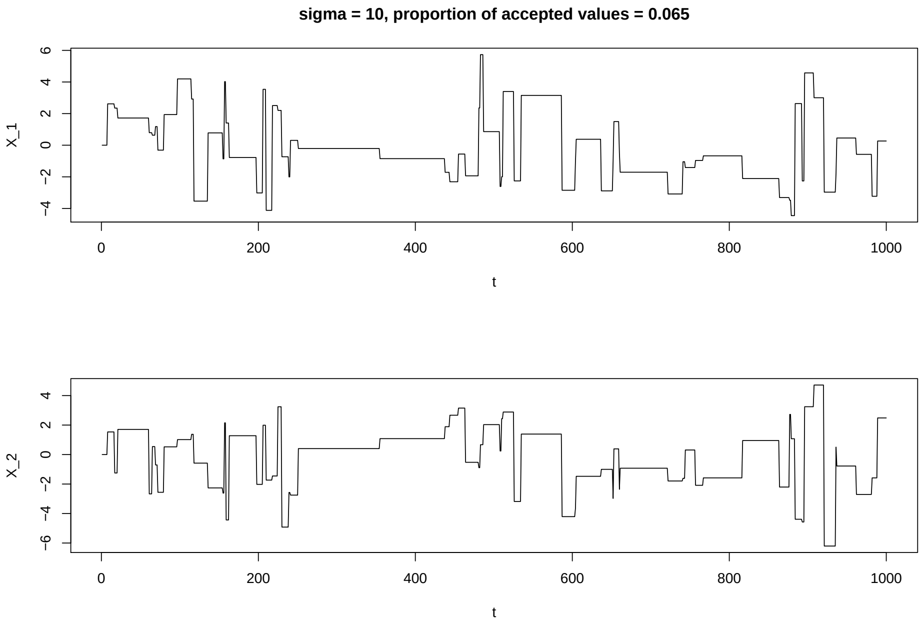
Task 1. Write a function that evaluates the density of a bivariate normal distribution at a single point. The function should take as input the point x, the mean parameter μ, and the covariance parameter Σ.

Task 2. Implement the Metropolis-Hastings algorithm to generate an approximate sample of size $n = 1000$ from the bivariate Normal distribution with mean $\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and covariance $\boldsymbol{\Sigma} = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$. Use the following symmetric proposal distribution: $\mathsf{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix}\right)$ with $\sigma = 2.5$. Report the proportion of accepted values.

Task 3. To assess convergence, look at sample plots for both variables, and plots of cumulative average estimates of $\mathbb{E}(X_1)$ and $\mathbb{E}(X_2)$. These will be called the diagnostic plots. Compute the autocorrelation for both variables, and the effective sample size. *Hint: If ρ is the autocorrelation of the chain, and n is the length, then the estimate of the effective sample size is given by $n(1-\rho)/(1+\rho)$.*

Task 4. Change the standard deviation of the proposal once to $\sigma = 0.1$, and once to $\sigma = 10$. Compare the diagnostic plots, the proportion of accepted values, the autocorrelations, and the effective sample sizes. Your plots should be similar to the following ones.





Task 5. Run the algorithm again to generate 1000 samples from the bivariate Normal distribution with $\boldsymbol{\mu} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and covariance $\boldsymbol{\Sigma} = \begin{pmatrix} 4 & 2.8 \\ 2.8 & 2 \end{pmatrix}$. What can you observe from the diagnostic plots?

Useful Python functions

The following Python functions might be of use:

`linalg.inv` The function `linalg.inv(`*mat*`)` returns the inverse of the matrix mat.

```
from scipy import linalg
mat = array( [[1,2],[3,4]] )
linalg.inv( mat )
```

`linalg.det` The function `linalg.det(`*mat*`)` returns the determinant of the matrix mat.

```
from scipy import linalg
linalg.det( mat )
```

`statistics.correlation` The function `statistics.correlation(`*x*`, `*y*`)` computes the correlation of x and y.

```
import sys
sys.path.append(”/home/ludger/lib/python2.6/site-packages/”)
import statistics
x = random.random( 10 )
y = random.random( 10 )
statistics.correlation( x, y )
```

Useful R functions

The following R functions might be of use:

`solve` The function `solve(`*a*`, `*b*`)` solves the equation $ax = b$ for x, where a is a square matrix, and b is a vector or matrix. If b is missing, the function returns the inverse of a.

```
a <- matrix(1:4, nrow=2)
b <- c(1, 2)
r <- solve(a,b)
a %*% as.matrix(r)
a_inv <- solve(a)
a_inv %*% a
```

`det` The function `det(`*x*`)` calculates the determinant of matrix x.

```
det(a)
```

`cor` The function `cor(`*x*`, `*y*`)` computes the correlation of x and y.

```
cor(1:10,2:11)
```