Projet Master HPC-IA Mines ParisTech

2022 - 2023

Miguel Munoz Zuniga

November 10, 2022

Problem 1 - Parameter estimation of the Gamma distribution.

The Gamma density function is given by

$$f_{(a,b)}(x) = \frac{b^a}{\Gamma(a)} x^{a-1} e^{-bx}, \quad x > 0$$

with parameters $a > 0$ et $b > 0$ and Γ the gamma function. Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a realisation of $\mathbf{X} = (X_1, \ldots, X_n)$ where the X_i are i.i.d. random variables following the Gamma distribution with distribution $f_{a,b}$. Your objectif is the estimation of the parameters a et b from the observations $\mathbf{x}$.

1. Write the log-likelihood of this model denoted $\ell(a, b)$

The maximum likelihood estimator will then be given by

$$(\hat{a}, \hat{b}) = \arg \max_{a>0, b>0} \ell(a, b).$$

2. With an already implemented python function generate one sample of $\mathbf{X}$ denoted $\mathbf{x}$ of size $n = 20$ for fixed values $a = 2$ and $b = 2$.

3. Write a function that evaluates the log-likelihood function $\ell(a, b)$. It takes as arguments values for the parameters a and b and the observations $\mathbf{x} = (x_1, \ldots, x_n)$. The

Gamma function Γ is for instance available as *math.gamma()* in python.

4. Write a function to visualise the log-likelihood as a function of a and b in three dimensions (for the fixed initial sample $\mathbf{x}$).

5. Make the previous graphical representation for different simulated data sets $\mathbf{x}$ of different sizes and with different parameter values a and b . What is the typical shape of the log-likelihood function? How many critical points does it have? Where is the maximum of the function apparently located ?

Calculation of the Maximum Likelihood Estimator (MLE) by the Newton-Raphson method.

It can be shown that the MLE does not have an explicit expression. Instead, the Newton-Raphson method can be used to approximate it. More precisely, it can be applied to find critical points of $\ell(a, b)$.

An update of Newton-Raphson.

Recall that the Newton-Raphson method is an iterative algorithm. Let us denote $\left(a^{(t)}, b^{(t)}\right)^T$ the current parameter values. An iteration of the algorithm consists of calculating the new parameter values $\left(a^{(t+1)}, b^{(t+1)}\right)^T$ by the formula

$$\left(a^{(t+1)}, b^{(t+1)}\right)^T = \left(a^{(t)}, b^{(t)}\right)^T - \left[H\left(a^{(t)}, b^{(t)}\right)\right]^{-1} \nabla \ell\left(a^{(t)}, b^{(t)}\right), \tag{1}$$

with H the Hessian matrix of ℓ.

6. For the gradient and the Hessian verify that you get

$$\nabla \ell(a, b) = \left(\frac{\partial}{\partial a}\ell(a, b), \frac{\partial}{\partial b}\ell(a, b)\right)^T = \left(n \log b - n(\log \Gamma(a))' + \sum_{i=1}^{n} \log x_i, \quad \frac{na}{b} - \sum_{i=1}^{n} x_i\right)$$

and

$$[H(a, b)]^{-1} = \begin{pmatrix} \frac{\partial^2}{\partial^2 a}\ell(a, b) & \frac{\partial^2}{\partial a \partial b}\ell(a, b) \\ \frac{\partial^2}{\partial a \partial b}\ell(a, b) & \frac{\partial^2}{\partial^2 b}\ell(a, b) \end{pmatrix}^{-1} = \frac{1}{n\left(1 - a(\log \Gamma(a))''\right)} \begin{pmatrix} a & b \\ b & b^2(\log \Gamma(a))'' \end{pmatrix}$$

7. Write a function python *UpdateNewton* which performs an update of the parameters. This function takes as arguments the current values of the parameters and the data. It

returns the new parameter values $a^{(t+1)}$ and $b^{(t+1)}$ obtained by (1). For the derivatives of $\log \Gamma(a)$ use the digamma and trigamma functions.

Iterate update.

The Newton-Raphson method consists in repeating the updates described above for $t = 1, 2, ldots$ by a while loop until convergence. We then need a stopping criterion that checks after each iteration whether the algorithm has converged. **Stopping criterion.**

Several stopping criteria are possible. Here we will consider the criterion which is based on the function to be maximized $\ell(a, b)$. Thus, we stop the calculations as soon as the value of the log-likelihood is stable. More precisely, we stop as soon as

$$\left| \frac{\ell\left(a^{(t+1)}, b^{(t+1)}\right) - \ell\left(a^{(t)}, b^{(t)}\right)}{\ell\left(a^{(t+1)}, b^{(t+1)}\right)} \right| < \varepsilon,$$

where $\varepsilon > 0$ is a fixed threshold.

Initialisation.

Newton's method requires an initialization point $\left(a^{(0)}, b^{(0)}\right)^T$. It is up to the user to choose it. We will see the importance of a good choice of the initial point.

8. Write a function, named *MLEGamma*, which calculates the MLE using the Newton-Raphson method. It takes as arguments the data and the initial values $a^{(0)}$ and $b^{(0)}$. Starting from these initial values we iterate parameter updates according to Newton's method until convergence. Use the above stopping criterion with the threshold $\varepsilon = 10^{-3}$. The function returns the estimators of a and b and the number of operations performed.

9. To avoid the algorithm running indefinitely, modify the function so that it stops when the algorithm has converged or after at most 100 iterations.

10. Test your function on simulated data. Test different parameter values a and b and different sample sizes. To begin with, always initialise with the true parameter values. In this case, the algorithm should converge very quickly. If it does not, your program that must be corrected.

11. Now try to initialise with initial points further and further away from the true parameter values. What do you observe?

12. We sometimes observe that this produces errors. In fact, it happens that the function *UpdateNewton* returns negative values for a and/or b (which is not admissible from the point of view of interpreting a and b as parameters of the Gamma law, but Newton's method does not respect the definition domain). In this case, the calculation of the log-likelihood function produces errors. Modify the function *MLEGamma* so that it stops the algorithm as soon as *UpdateNewton* returns negative values.

Initialisation with the moment method estimator.

Another estimator of the parameters a and b in this model is given by the moment method estimator (MME).

13. Show that the MME is given by

$$\tilde{a} = \frac{\left(\bar{X}_n\right)^2}{\hat{\sigma}_n^2}, \quad \tilde{b} = \frac{\bar{X}_n}{\hat{\sigma}_n^2},$$

with $\hat{\sigma}_n^2 = \frac{1}{n}\sum_{i=1}^n \left(X_i - \bar{X}_n\right)^2$ denotes the empirical variance and $\bar{X}_n$ the empirical mean.

14. Write a function named *MMEGamma* which takes as argument the observations and returns the estimator by the method of moments of a and b.

15. Modify your function *MLEGamma* so that by default it uses as initial values the method of moments estimator of a and b. Study again the convergence of the algorithm on simulated data.

Comparison of MLE and MME.

We want to compare the two estimators, MLE and MME, of the parameters of the Gamma distribution. We want to know which one has the smallest squared risk. Let us recall that the quadratic risk of an estimator $\hat{\theta}$ of a parameter θ is defined by $\mathcal{R}(\hat{\theta}, \theta) = \mathbb{E}\left[\|\hat{\theta} - \theta\|^2\right]$.

16. Write a program that

    (a) generates a large number of Gamma distribution data sets of fixed size n,

(b) evaluates the two estimators on these data

(c) calculates an approximation of the quadratic risk of the two estimators.

17. Run simulations to see which estimator performs better and plot the evolution of the risk with respect to n.

Problem 2 - Simulate an a posteriori distribution with the Monte Carlo Markov Chain algorithm

Consider that a random variable X follows a Poisson distribution $\mathcal{P}(\theta)$ where θ is an unknown parameter. On the other hand, consider that the a priori distribution law of θ is a Gamma distribution of known fixed parameters k and λ: $\mathcal{G}(k, \lambda)$ (See distribution in the course). The translation parameter *gamma* is considered null. Take $x = 10$. The a posteriori law of $\theta|X = x$ is known, its law is the follwoing Gamma distribution: $\mathcal{G}(k + x, \lambda + 1)$. We are now going to generate a sample of this law using the product of the likelihood and the prior introduced above with an MCMC algorithm.

1. In order to simulate a Markov chain following the Gamma objective posterior distribution, modify the true likelihood and the a priori of the "posterior" function of the following MCMC algorithm implemented in Python: https://moonbooks.org/Articles/Lalgorithme-de-Metropolis-Hastings-MCMC-avec-python/ (the code can also be found in annex of this document). Generate a long sequence and take only the second half of the sequence as a representative sample.

2. With the sample obtained in 1. make a kernel approximation of the associated density and compare this estimated density with the true objective density of $\mathcal{G}(k + x, \lambda + 1)$.

1 Annex

1.1 Bonus poblem

Bonus Problem - Calibration of the hyper-parameters of a gaussian process regressor by maximum likelihood

Nous souhaitons approcher un modèle numérique, ou encore simulateur, représenté mathématiquement par une fonction f, dont le temps de calcul est important limitant de fait le nombre d'évaluations de f réalisables en un temps de calcul limité. La modélisation par processus Gaussien se prête particulièrement bien à ce contexte.
Pour illustrer cette stratégie nous allons approximer/modéliser une fonction simple $x \rightarrow sin(4\pi x)$ par un processus Gaussien.

Un processus aléatoire sur $\mathbb{R}^d$ est une fonction de $Z : \Omega \times \mathbb{R}^d \rightarrow \mathbb{R}$ telle que pour tout $\omega \in \Omega$, $Z(\omega, .)$ est une réalisation (toute la fonction est une réalisation) du processus aléatoire Z et d'un autre côté $Z(., x)$ est une variable aléatoire (scalaire). Dans la suite on omettra la référence à l'ensemble Ω et noterons la variable aléatoire $Z(., x)$ par $Z(x)$.

Un processus aléatoire Gaussien est un processus aléatoire tel que pour tout n et tout $x_i \in \mathbb{R}^d$ $(i = 1, ..., n)$, le vecteur $(Z(x_1), ..., Z(x_n))$ est un vecteur Gaussien.
Un processus Gaussien est complètement déterminé par la connaissance de sa fonction moyenne i.e.

$$\forall x \in \mathbb{R}^d \quad m(x) = \mathbb{E}[Z(x)]$$

et de sa fonction de covariance :

$$\forall x, x' \in \mathbb{R}^d \quad k(x, x') = Cov(Z(x), Z(x')).$$

On note

$$Z \sim \mathcal{PG}(m, k).$$

Dans la suite nous supposerons le processus centré i.e. $\forall x \in \mathbb{R}^d$, $m(x) = 0$.
Ainsi pour tout n et tout $x_i \in \mathbb{R}^d$, $i = 1, ..., n$, le vecteur $(Z(x_1), ..., Z(x_n))$ est un vecteur Gaussien de moyenne nulle et de matrice de covariance Σ tel que

$$\Sigma_{i,j} = k(x_i, x_j) = Cov\Big(Z(x_i), Z(x_j)\Big)$$

et l'on note

$$\begin{pmatrix} Z(x_1) \\ \vdots \\ Z(x_n) \end{pmatrix} \sim \mathcal{N}(0, \Sigma)$$

Dorénavant $d = 1$ et la fonction de covariance est considérée paramétré par λ et s'exprime :

$$k_\lambda(x, x') = \Big(1 + \frac{x - x'}{\lambda} + \frac{(x - x')^2}{3\lambda^2}\Big) \exp(-\frac{x - x'}{\lambda}). \tag{2}$$

La matrice de covariance Σ a donc pour termes les $\Sigma_{i,j} = k_\lambda(x_i, x_j)$.

1. **Simuler plusieurs réalisations d'un processus Gaussien de moyenne nulle et de fonction de covariance (2).**

   (a) Choisissez N (grand) valeurs equi-distribuées $x_i \in [0, 1]$ tel que $x_1 < ... < x_N$. Fixez $\lambda = 1$ et calculez la matrice de covariance Σ correspondante. Soit L la décomposition de Cholesky de la matrice Σ. Soit $G = (G_1, ..., G_N)$ un vecteur dont les coordonnées sont des Gaussiennes centrées, réduites et indépendantes. En notant $\boldsymbol{m} = (m(x_1), ..., m(x_n))$ (que l'on considérera non nulle uniquement pour cette question), que vaut la moyenne et la matrice de covariance du vecteur aléatoire $\boldsymbol{m} + L^t G$?

   (b) Simulez $\boldsymbol{g} = (g_1, ..., g_N)$ une réalisation de G et calculez $(z_1, ..., z_N) = L^t g$

   (c) Tracez le graphe $(x_i, z_i)_{i=1,...,N}$ qui représente une réalisation du processus Gaussien Z pour une valeur de paramètre $\lambda = 1$

   (d) Recommencez (b) plusieurs fois et représentez sur un même graphe les différentes réalisations de Z.

   (e) Refaite (a) à (d) en changeant de valeur pour λ. Que constatez-vous ? Comment interpréter ce paramètre ?

2. **Estimation du paramètre λ par maximum de vraissemblance**

   (a) Divisez $[0, 1]$ en une partition de $n = 10$ intervalles et tirez un nombre aléatoire uniformément dans chacun d'entre eux. On notera $x_1, ...x_n$ ces n valeurs que l'on

stockera.

(b) Soit $\boldsymbol{z} = (z_1, ..., z_n)$ le vecteur de taille n définit pour tout i par

$$z_i = sin(4\pi x_i). \tag{3}$$

Ecrire, comme une fonction paramétrée par λ, la négative log-vraissemblance (i.e. moins le logarithme de la vraissemblance) associée au vecteur aléatoire Gaussien $(Z(x_1), ..., Z(x_n))$ dont z est considéré comme une réalisation. A l'aide d'un optimiseur, minimisez cette negative log-vraissemblance par rapport à lambda.

(c) Sur un même graphe représentez la fonction $sin(4\pi x)$ et quelques réalisations du processus Gaussien Z (en utilisant la technique proposez en 1.) en prenant pour λ la valeur optimale trouvez en 2.(b).

3. **Simuler plusieurs réalisations d'un processus Gaussien conditionné à un ensemble de données.**

Dans la suite le λ considéré sera celui obtenu dans 2. et nous reprenons également les mêmes n nombres $(x_1, ..., x_n)$ obtenus par simulation en 2.a. Soit N valeurs prises dans $[0, 1]$: $x'_1, ..., x'_N$, distinctes des x_i. Nous connaissons la distribution Gaussienne des deux vecteurs aléatoires $(Z(x_1), ..., Z(x_n))$ et $(Z(x'_1), ..., Z(x'N))$ qui sont de moyennes nulle et de matrices de covariance respectives Σ_n et Σ_N dont les termes sont calculés avec (2). Nous pouvons ensuite montrer que le vecteur aléatoire conditionné

$$Z(x'_1), ..., Z(x'_N)|Z(x_1) = z_1, ..., Z(x_n) = z_n$$

suit également une loi Gaussienne de vecteur moyenne

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{k}_n^t(x'_1)\Sigma_n^{-1}\boldsymbol{z} \\ \vdots \\ \boldsymbol{k}_n^t(x'_N)\Sigma_n^{-1}\boldsymbol{z}_n \end{pmatrix} \tag{4}$$

où $\boldsymbol{z}$ est le vecteur des données obtenu avec (3), et $\boldsymbol{k}_n(x'_i) = (k_\lambda(x_1, x'_i), ..., k_\lambda(x_n, x'_i))^t$.

La matrice de covariance du vecteur conditionné est elle donnée par

$$\Sigma_{N|n} = \Sigma_N - \Sigma_{n,N}^t \Sigma_n^{-1} \Sigma_{n,N} \tag{5}$$

où Σ_N est la matrice de taille $N \times N$ des covariances entres les N nouveaux points et $\Sigma_{n,N}$ la matrice de taille $n \times N$ dont la i-ième colonne correspond au vecteur de taille n des covariances du i-ième nouveau point avec les n points d'apprentissages.

(a) Pour un $x \in [0, 1]$ rappelez la moyenne et la variance de $Z(x)$ et donnez celles de $Z(x)$ sachant $Z(x_1) = z_1, ..., Z(x_n) = z_n$ que l'on notera respectivement $\mu_n(x)$ et

$$\sigma_n^2(x) := Var[Z(x)|Z(x_1) = z_1, ..., Z(x_n) = z_n].$$

On dit que le processus $Z(x)$ est le processus Gaussien a priori et $Z(x)$ sachant $Z(x_1) = z_1, ..., Z(x_n) = z_n$ le processus Gaussien a posteriori.

(b) Tracez sur un même graphe la fonction à approximer : $sin(4\pi x)$, la moyenne $\mu_n(x)$ ainsi que $\mu_n(x) + 2\sigma_n^2(x)$ et $\mu_n(x) - 2\sigma_n^2(x)$.

(c) En utilisant les formules de la moyenne et de la covariance a posteriori du processus Gaussien conditionné donnés par (4) et (5) ainsi que la technique introduite en 1., avec N grand, tracez sur un même graphe un ensemble de réalisations du processus a posteriori ainsi que la fonction $sin(4\pi x)$.

1.2 MCMC algorithm code

```
import matplotlib.pyplot as plt
import numpy as np
import math

#-----------------------------------------------------------------
# define posterior distribution
```

```
def posterior(x):
mu, sigma = 5, 2.0 # mean and standard deviation
num = math.exp( - ( x - mu )**2 / ( 2.0 * sigma **2 ) )
den = math.sqrt( 2 * math.pi * sigma **2)
return  num / den

#-----------------------------------------------------------------
# plot posterior

x_array = np.linspace(-5.0, 15.0, 100)
y_array = np.asarray( [posterior(x) for x in x_array] )

plt.plot(x_array,y_array)

plt.grid()
plt.title('Metropolis Hastings Posterior')
plt.savefig('posterior.png',bbox_inches='tight')
plt.show()
plt.close()

#-----------------------------------------------------------------
# Metropolis Hastings sampling from the posterior distribution

N = 100000
s = 10

x = 0
p = posterior(x)
```

```
samples = []

for i in xrange(N):
xn = x + np.random.normal(size=1)
pn = posterior(xn)
if pn >= p:
p = pn
x = xn
else:
u = np.random.rand()
if u < pn/p:
p = pn
x = xn
if i % s == 0:
samples.append(x)

samples = np.array(samples)

plt.scatter(samples, np.zeros_like(samples), s=10)

plt.plot(x_array,y_array)
plt.hist(samples, bins=50,normed=1)

plt.title('Metropolis Hastings sampling')
plt.grid()
plt.savefig('metropolis_hastings_1d.png',bbox_inches='tight')
plt.show()
plt.close()
```