Simulation d'une distribution gaussienne tronquée sur un intervalle fini

Vincent Mazet

Simulation d'une distribution gaussienne tronquée sur un intervalle fini

Vincent Mazet

LSIIT, UMR CNRS/UDS 7005,
vincent.mazet@unistra.fr

5 juillet 2012

Résumé Cet article détaille l'algorithme du script Matlab `rtnorm` qui permet de générer un nombre pseudo-aléatoire (cas 1D) distribué suivant une loi gaussienne tronquée sur un intervalle fini $[a, b]$. Nous utilisons l'approche proposée par Chopin [2].

1 Introduction

1.1 Loi gaussienne tronquée

Une loi gaussienne de moyenne μ et d'écart-type σ que l'on tronque sur un intervalle fini $[a, b]$ a pour distribution :

$$p(x) = \frac{1}{Z} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \qquad \text{où} \qquad Z = \sqrt{\frac{\pi}{2}}\sigma \left[\operatorname{erf}\left(\frac{b-\mu}{\sqrt{2}\sigma}\right) - \operatorname{erf}\left(\frac{a-\mu}{\sqrt{2}\sigma}\right)\right]$$

et erf est la fonction erreur :

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{t^2}\, dt$$

On notera cette distribution $\mathcal{N}_{[a,b]}(x; \mu, \sigma^2)$. À cause de la troncation, la moyenne et l'écart-type de cette gaussienne tronquée ne sont donc plus μ et σ.

Sans perdre en généralité, on peut considérer uniquement le cas où $\mu = 0$ et $\sigma = 1$. Dans ce cas :

$$p(x) = \frac{1}{Z} \exp\left(-\frac{x^2}{2}\right) \qquad \text{où} \qquad Z = \sqrt{\frac{\pi}{2}} \left[\operatorname{erf}\left(\frac{b}{\sqrt{2}}\right) - \operatorname{erf}\left(\frac{a}{\sqrt{2}}\right)\right]$$

1.2 Simulation d'une gaussienne tronquée

L'algorithme proposé par Chopin [2] consiste à utiliser une table de valeurs pré-calculées pour simuler la loi. Cela permet de réduire le nombre de calculs et donc d'obtenir une méthode rapide. Cet algorithme est inspiré de l'algorithme Ziggurat de Marsaglia et Tsang [3, 4] (qui est généralement considéré comme la méthode la plus rapide) et de l'algorithme de Ahrens [1].
Le principe de la méthode est résumé figure 1. La loi candidate est constituée de $2N + 2$ régions : $2N$ rectangles verticaux, et deux queues gaussiennes. Ces $2N + 2$ régions ont toutes la même aire. L'idée est donc de choisir une de ces $2N + 2$ régions de manière uniforme, de générer un candidat distribué uniformément selon ce rectangle (loi candidate uniforme), et enfin de l'accepter selon la procédure d'acceptation-rejet. Pour ce qui est des deux régions extrêmes (les queues de la gaussienne), on utilise des lois candidates plus adaptées.

Le calcul des coordonnées des points des régions est effectué au préalable, grâce au script Python gentabForMatlab.py. Ce script est une adaptation du script original de Chopin (gentab.py) pour prendre en compte les spécificités de Matlab (notamment le fait que les indices des tableaux commencent à 1).

La section suivante détaille l'algorithme `rtnorm`

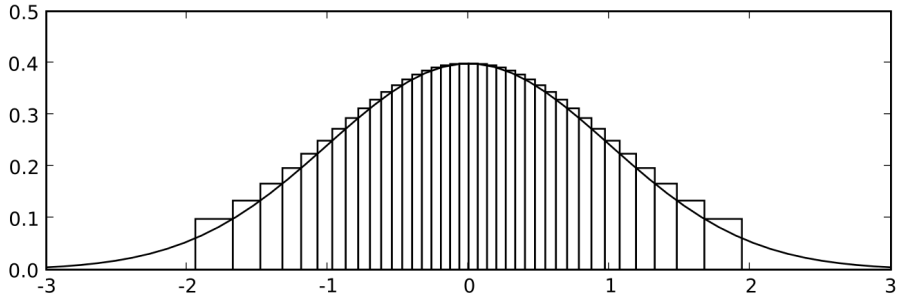

FIGURE 1 – Tracé de $\mathcal{N}_{]-\infty;+\infty[}(0,1)$ et de $2N$ rectangles pour $N = 20$ (figure tirée de [2]).

2 Description de l'algorithme

La méthode `rtnorm` dépend de quatre paramètres : les limites a et b de l'intervalle et les paramètres μ et σ. Plusieurs cas sont à distinguer en fonction de leur valeur.

2.1 Cas où $\mu \neq 0$ et $\sigma \neq 1$

Le cas où $\mu \neq 0$ et $\sigma \neq 1$ est traité en se ramenant au cas $\mu = 0$ et $\sigma = 1$ par changement de variable. Ainsi, pour générer $x \sim \mathcal{N}_{[a,b]}(x; \mu, \sigma^2)$, l'algorithme est le suivant :

1. Calculer $a' = (a - \mu)/\sigma$ et $b' = (b - \mu)/\sigma$
2. Générer $x' \sim \mathcal{N}_{[a',b']}(x; 0, 1)$
3. Calculer $x = x'\sigma + \mu$.

Dans la suite, on peut donc considérer sans perdre en généralité que $\mu = 0$ et $\sigma = 1$.

2.2 Cas où $a \geq b$

Ce cas est considéré impossible car on suppose toujours que a est la limite gauche et b la limite droite et que ces deux limites sont distinctes.

2.3 Cas où $|a| > |b|$

En inversant l'axe des abscisses, on peut se ramener au cas $|a| < |b|$ (que l'on traite ci-après) à l'aide d'un changement de variable :

1. Générer $x' \sim \mathcal{N}_{[-b,-a]}(x; 0, 1)$
2. Calculer $x = -x$.

2.4 Cas où $a > x_{\max}$

On définit une borne supérieure $a_{\max} \approx 2,61$ comme dans [2]. Dans cas, puisque $b > a$, alors l'intervalle de troncature est inclus dans la queue droite de la gaussienne. Pour cette raison, comme le préconise [2], on utilise un algorithme d'acceptation-rejet dont la loi candidate est une exponentielle tronquée :

$$q(x) = \frac{\lambda \exp(-\lambda x)}{\exp(-\lambda a) - \exp(-\lambda b)} I_{a \leq x \leq b}$$

où I est la fonction indicatrice.
Cela est fait avec l'algorithme suivant :

1. Faire
   - Générer $u, v \sim \mathcal{U}_{[0,1]}$
   - Calculer $z = \log\left(1 + u(e^{-a(b-a)} - 1)\right)$
   - Calculer $e = -\log(v)$ (e est donc tiré suivant une loi exponentielle)
2. Tant que $2a^2 e \leq z^2$
3. Calculer $x = a - z/a$

2.5 Cas où $a < x_{\min}$

Dans le cas où $a < x_{\min} \approx -2.00$, alors comme $b > 2$ (car $|a| < |b|$), on utilise comme dans [2] un algorithme d'acceptation-rejet dont la loi candidate est une gaussienne centrée réduite et non tronquée. L'algorithme est donc :

1. Faire
   - Générer $x, \sim \mathcal{N}(0, 1)$
2. Tant que $x < a$ ou $x > b$

2.6 Tous les autres cas ($a \in [x_{\min}, x_{\max}]$)

Dans les autres cas, on utilise les tables précalculées, à savoir les bornes x_i des $2N$ rectangles (`x`), leur hauteur y_i (`yu`), et le tableau j (`ncell`). Chopin utilise également d'autres tables ($\underline{y_i}$, d_i et δ_i), mais puisqu'elles peuvent être déterminées en fonction de x_i et y_i, cela ne nous a pas paru nécessaire de les créer.
On notera par ailleurs $N_{\max} = j^* + 2$ l'indice de la région à l'extrême droite : à l'indice j^* (`ncell(end)`) du dernier rectangle, on ajoute 1 pour avoir la région suivante puis une nouvelle fois 1 pour prendre en compte le décalage des indices en Matlab (les tableaux commencent à 1).

On calcule ensuite

$$i_a = j_{\lfloor a/h \rfloor},$$
$$i_b = \begin{cases} j_{\lfloor b/h \rfloor} & \text{si } b < x^* \\ N_{\max} & \text{sinon} \end{cases}$$

où $\lfloor \cdot \rfloor$ représente la partie entière (*floor function*) et x^* est la bordure droite du dernier rectangle. Le code Matlab correspondant est (`I0` $= -\lfloor x_1/h \rfloor$ permet de décaler les indices) :

```
i = I0 + floor(a/h) + 1;
ka = ncell(i) + 1;
if b>=x(end),
    kb = Nmax;
else
    i = I0 + floor(b/h) + 1;
    kb = ncell(i) + 1;
end;
```

2.6.1 Cas où $|b - a|$ est petit

Dans le cas où la distance entre a et b est très faible (on choisit $|b - a| < k_{\min} = 5$ comme proposé dans [2]), on utilise à nouveau un algorithme d'acceptation-rejet avec comme loi candidate une exponentielle tronquée :

1. Faire
   - Générer $u, v \sim \mathcal{U}_{[0,1]}$
   - Calculer $z = \log\left(1 + u(e^{-a(b-a)} - 1)\right)$
   - Calculer $e = -\log(v)$ (e est donc tiré suivant une loi exponentielle)
2. Tant que $2a^2 e \leq z^2$
3. Calculer $x = a - z/a$

2.6.2 Autres cas

Dans tous les autres cas, on effectue un algorithme d'acceptation-rejet où la loi candidat est choisie en fonction de la région où on tire l'échantillon. Le principe est donc de tirer au hasard un échantillon $k \sim \mathcal{U}_{\{k_a,\ldots,k_b\}}$ pour déterminer dans quelle région sera tiré le candidat. Si le candidat généré est refusé, on recommence la procédure. L'algorithme est donc le suivant.

1. Faire
2. Générer $k \sim \mathcal{U}_{\{k_a,\dots,k_b\}}$
3. Si $k = N_{\max}$ (queue droite de la gaussienne), on tire un candidat suivant une loi exponentielle. Attention, la loi candidate n'est pas tronquée à droite, afin que son aire soit égale à l'aire d'un rectangle.
   - Générer $u, v \sim \mathcal{U}_{[0,1]}$
   - Calculer $z = -\log(u)/x^*$ et $e = -\log(v)$
   - Si $z^2 \leq 2e$ (condition pour tirer suivant une loi exponentielle) et $z < b - x^*$ (condition pour que le candidat soit inférieur à b), alors on accepte la proposition et donc $x = x^* + z$ (le candidat généré z étant dans $[0, b - x^*]$, il faut effectuer un décalage pour avoir $x \in [x^*, b]$)
4. Sinon si $k \leq k_a + 1$ ou si $\geq k_b - 1$ avec $b < x^*$ (les deux régions à gauche ou les deux régions à droite doivent être traitées séparément des autres, voir [2, section 2.3]).
   - Générer $u \sim \mathcal{U}_{[0,1]}$
   - Calculer $x = x_i + d_i u$
   - Si $a \leq x \leq b$
     - Calculer $y = y_i * u$
     - Si $y < \underline{y_i}$ ou $x^2 + 2 * \log(y) + \log(2\pi) < 0$, on accepte la proposition x
5. Sinon (toutes les autres régions)
   - Générer $u \sim \mathcal{U}_{[0,1]}$
   - Calculer $y = y_i u$
   - Si $y < \underline{y_i}$ (cas le plus probable), alors accepter le candidat $x = x_i + u\delta_i$ et terminer.
   - Sinon, générer $v \sim \mathcal{U}_{[0,1]}$
   - Calculer $x = x_i + d_i v$
   - Si $x^2 + 2\log(y) + \log(2\pi) < 0$ (vérifier qu'on est sous la loi), alors accepter le candidat x.
6. Tant que le candidat proposé n'a pas été accepté.

Références

[1] J. Ahrens, « A one-table method for sampling from continuous and discrete distributions ». *Computing*, vol. 54, numéro 2, p. 127–146 (1995).

[2] N. Chopin, « Fast simulation of truncated Gaussian distributions ». *Statistics and Computing*, vol. 21, p. 275–288 (2011).

[3] G. Marsaglia et W. Tsang, « A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions ». *SIAM J. Sci. Stat. Comput.*, vol. 5, p. 349–359 (1984).

[4] G. Marsaglia et W. Tsang. « The Ziggurat method for generating random variables ». *J. Stat. Soft.*, vol. 5, numéro 8 (2000).