Projet de Programmation Probabiliste
Inférence Semi-symbolique

Maxence Perion

15 Janvier 2024

Question 1

On veut montrer que $[\![infer\ beta\_bernoulli\ v]\!] = Beta(a+v, b-v+1)$.
On a:
$\{\![beta\_bernoulli\ v]\!\}_\emptyset(U) = \int_0^1 \{\![sample(beta \sim a \sim b)]\!\}(dp)\{\![let() = observe(bernoulli \sim p)v\ in\ p]\!\}$
$= \int_0^1 Beta(a,b)(dp) \int_{()} \{\![observe(bernoulli \sim p)v]\!\}\{\![p]\!\}(U)$
$= \int_0^1 Beta(a,b)(dp) pdf(Bernoulli(p))(v)\delta_p(U)$
$= \int_U Beta(a,b)(dp) pdf(Bernoulli(p))(v)$
$= \int_U \frac{p^{a-1}(1-p)^{b-1}}{B(a,b)} p^v(1-p)^{1-v}(dp)$
$= \int_U \frac{p^{a-1+v}(1-p)^{b-v}}{B(a,b)}(dp).$
Ainsi, $\{\![infer\ beta\_bernoulli\ v]\!\} = \frac{\int_U \frac{p^{a-1+v}(1-p)^{b-v}}{B(a,b)}(dp)}{\int_0^1 \frac{p^{a-1+v}(1-p)^{b-v}}{B(a,b)}(dp)}$
$= \frac{\frac{1}{B(a,b)} \int_U p^{a-1+v}(1-p)^{b-v}(dp)}{\frac{1}{B(a,b)} \int_0^1 p^{a-1+v}(1-p)^{b-v}(dp)}$
$= \frac{\int_U p^{a-1+v}(1-p)^{b-v}(dp)}{\int_0^1 p^{a-1+v}(1-p)^{b-v}(dp)}$
$= \frac{\int_U p^{a-1+v}(1-p)^{b-v}(dp)}{B(a+v,b-v+1)} = Beta(a+v, b-v+1)(U).$

On souhaite maintenant montrer que $[\![infer\ gauss\_gauss\ v]\!] = Gauss(m,s)$
avec $m = \frac{v}{2}$ et $s^2 = \frac{1}{2}$.
On a:
$\{\![gauss\_gauss\ v]\!\} = \int_{\mathbf{R}} \{\![sample\ gaussian(0,1)]\!\}(dx) \int_{()} observe(gaussian(x,1))(dy)\{\![x]\!\}(U)$
$= \int_{\mathbf{R}} gaussian(0,1)(dx) pdf(gaussian(x,1))(v)\delta_x(U)$
$= \int_U gaussian(0,1)(dx) pdf(gaussian(x,1))(v)$
$= \int_U \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}(dx) \frac{1}{\sqrt{2\pi}} e^{\frac{-(v-x)^2}{2}}$
$= \int_U \frac{1}{\sqrt{2\pi^2}} e^{-\frac{1}{2}(2x^2+v^2-2vx)}(dx)$

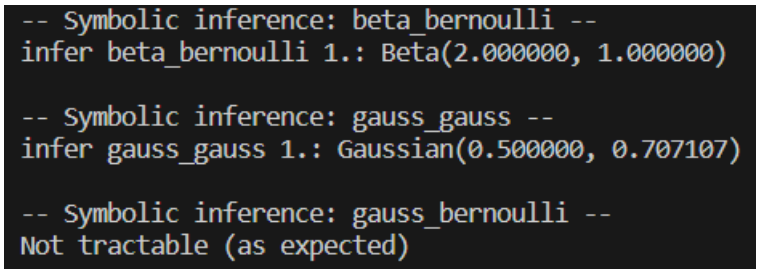

Figure 1: Résultat des exemples de la section A

Question 3

Nous avons testé les exemples de la section A et obtenu les résultats attendues: $infer\ beta\_bernoulli\ 1. = Beta(2,1)$ et $infer\ gauss\_gauss\ 1. = N(1/2, \sqrt{1/2})$. Le résultat est visible dans la figure 1, ainsi que celui de l'exemple d'un modèle pour lequel l'inférence échoue (Question 4).

Question 4

On propose le modèle gauss_bernoulli (implémenter dans le code par le modèle "fail" figure 2) qui utilise un paramètre suivant une loi normale dans une distribution de Bernoulli. Il s'agit de la modélisation d'une loi de Bernoulli dont le paramètre de succès suit une loi normale centrée sur 0.5.

```
float list -> node
let fail data =
  let m = sample (Gaussian (ref (V 0.5), ref (V 0.1))) in
  let () = List.iter (observe (Bernoulli(m))) data in
  m
```

Figure 2: Modèle "gauss_bernoulli" représentant une loi de Bernoulli dont le paramètre de succès dépend d'une loi de Gauss

Question 9

Le premier modèle d'example pour illustrer les points forts et faibles de cet algorithme d'inférence est le modèle "beta_bernoulli", un modèle qui consiste en l'observation d'une valeur v selon une loi de Bernoulli dont le paramètre exprimant la probabilité de succès suit une loi Beta. Son implémentation est visible en figure 3. L'intuition derrière le choix de ce modèle est qu'il est composé

uniquement de distributions conjuguées, et il devrait donc bien s'adapter à l'inférence symbolique.

```
prob -> float -> node
let example_1 prob v =  (* beta_bernoulli *)
  let p = sample prob (Beta (ref (V 1.), ref (V 1.))) in
  observe prob (Bernoulli(p)) v;
  p
```

Figure 3: Implémentation du modèle d'example 1: beta_bernoulli

En deuxième modèle, nous avons choisis le modèle "gaussian_bernoulli", un modèle comprenant deux distributions mais qui ne sont pas conjuguées, ce qui n'était pas tractable avec l'inférence symbolique simple. Son implémentation est décrite en figure 4.

```
prob -> float list -> node
let example_2 prob data = (* failed before *)
  let m = sample prob (Gaussian (ref (V 0.5), ref (V 0.1))) in
  let () = List.iter (observe prob (Bernoulli(m))) data in
  m
```

Figure 4: Implémentation du modèle d'example 2: gaussian_bernoulli

Avec la volonté d'exprimer un modèle "mi importance, mi symbolique", nous présentons désormais le modèle d'example 3 qui composé de deux distributions distribuées sur 3 tirages dépendant. Son implémentation est présente en figure 5.

```
prob -> float -> node
let example_3 prob v = (* half importance half symbolic *)
  let s = sample prob (Bernoulli (ref (V 0.5))) in
  let m = sample prob (Gaussian (s, ref (V 1.))) in
  observe prob (Gaussian(m, ref (V 1.))) v;
  m
```

Figure 5: Implémentation du modèle d'example 3: mi importance - mi symbolique

Pour finir, nous avons choisi de proposer un modèle que nous avons eu du mal à inférer jusqu'ici: le Hidden Markov Model. Sa complexité provient de son

nombre important de dimensions, mais il contient des distributions conjuguées alors il peut être intéressant d'observer ce que donne l'algorithme d'inférence semi-symbolique sur cet example. Son implémentation est présentée en figure 6.

```
prob -> float list -> node list
let example_4 prob data = (* hmm *)
  let rec gen states data =
    match (states, data) with
    | [], y :: data -> gen [ ref (V y) ] data
    | states, [] -> states
    | pre_x :: _, y :: data ->
        let x = sample prob (Gaussian(pre_x, ref (V 1.0))) in
        let () = observe prob (Gaussian(x, ref (V 1.0))) y in
        gen (x :: states) data
  in
  gen [] data
```

Figure 6: Implémentation du modèle d'example 3: Hidden Markov Model

Question 10

Pour évaluer les performances des différents algorithmes d'inférence (importance sampling, symbolique et semi-symbolique), nous réalisons un benchmark. Pour cela, nous allons lancer un grand nombre de fois (100000) les algorithmes d'inférence sur les différents examples de la Question 9 pour obtenir un temps d'exécution moyen. Le nombre de particules (lorsqu'il y en a) a été fixé à 1000. Les résultats sont transcrits dans le tableau 1.

Average execution time (milliseconds)	Beta_Bernoulli	Gaussian_Bernoulli	Example 3	HMM
Importance Sampling	0.132 ± 0.359	0.124 ± 0.446	0.114 ± 0.353	1.772 ± 1.155
Symbolique	0.012 ± 0.134	X	X	X
Semi-symbolique	0.179 ± 0.4	0.239 ± 0.801	0.247 ± 0.551	5.113 ± 1.934

Table 1: Benchmark du temps d'exécution moyen sur 100000 essais pour chaque algorithme d'inférence sur les examples

On peut constater que l'inférence symbolique simple ne permet de résoudre qu'un nombre réduit de cas, ici seulement un des quatre examples. En revanche, lorsque l'algorithme permet d'inférer la distribution le résultat est quasiment instantané et est le plus rapide de tous les algorithmes testés ici.

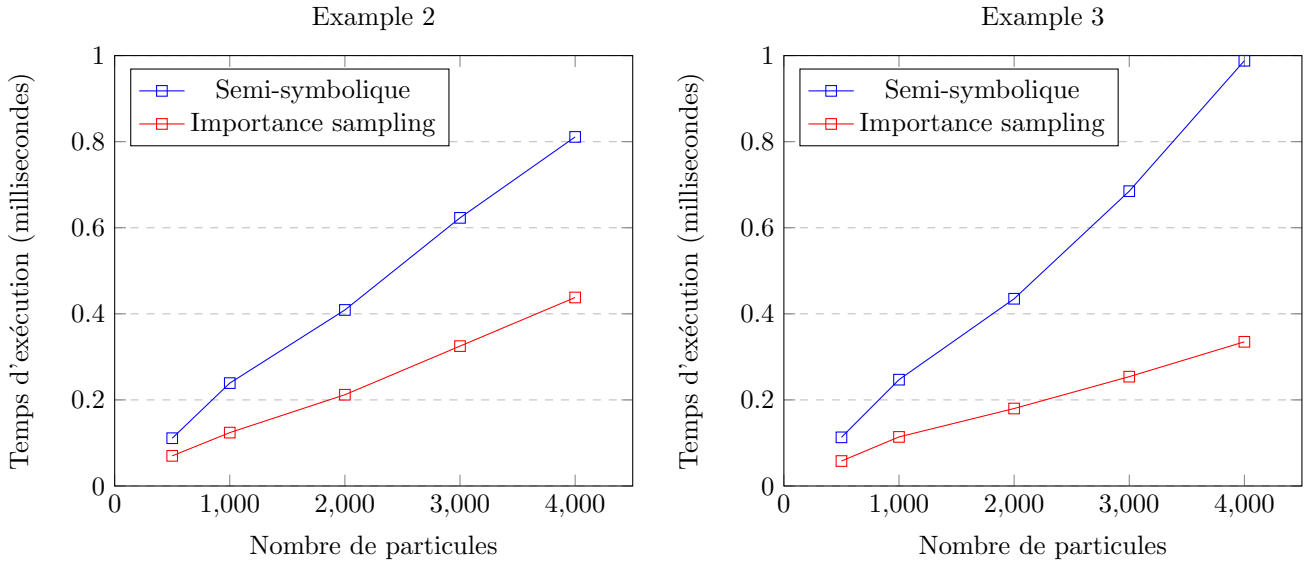

Figure 7: Evolution du temps d'exécution en fonction du nombre de particules sur les examples 2 et 3

Intéressons nous désormais à la comparaison entre l'algorithme d'inférence importance sampling et l'algorithme semi-symbolique. Un premier résultat est que sur l'example 1 (beta_bernoulli), l'algorithme d'importance sampling produit un résultat un peu plus rapide en moyenne (et similaire en prenant en compte l'écart-type). Cependant, ce résultat est approché puisque sa précision dépend du nombre de particules. En revanche, l'algorithme semi-symbolique produit, pour un temps approximativement similaire (avec une différence négligeable en prenant en compte l'écart-type) un résultat exact selon les formules mathématiques.

Sur les 3 autres examples, nous pouvons remarquer que l'exploration des relations potentielles de conjugaison entre les distributions induit un temps de calcul environ deux fois plus élevé, selon le nombre de distributions considérées. Intéressons nous d'avantage à cette relation avec de nouvelles expérience faisant varier le nombre de particules et en comparant les temps d'exécution des deux algorithmes sur les examples 2 et 3. Les résultats sont visibles sur la Figure 7 et leurs jeux de données respectifs dans les tableaux 2 et 3. Nous pouvons constater que l'algorithme d'inférence semi-symbolique induit bien un sur-coût par rapport à l'algorithme d'importance sampling et qu'il est d'un facteur 2 linéaire selon le nombre de particules sur nos expériences. De plus, nous pouvons remarquer que ces deux algorithmes ont l'air d'avoir un temps d'exécution linéaire en fonction du nombre de leurs particules, ce qui est tout à fait cohérent avec la théorie et ce que nous attendions.

Une autre observation intéressante du tableau 1 est que la différence de temps d'inférence est beaucoup plus marquée sur l'example 4. Nous avons donc

Average execution time (milliseconds)	500	1000	2000	3000	4000
Importance Sampling	0.070 ± 0.256	0.124 ± 0.446	0.212 ± 0.918	0.325 ± 0.724	0.438 ± 0.818
Semi-symbolique	0.111 ± 0.581	0.239 ± 0.801	0.409 ± 0.848	0.623 ± 0.849	0.811 ± 1.576

Table 2: Benchmark du temps d'exécution moyen sur l'example Gaussian_Bernoulli en fonction du nombre de particules.

Average execution time (milliseconds)	500	1000	2000	3000	4000
Importance Sampling	0.058 ± 0.472	0.114 ± 0.353	0.180 ± 0.357	0.254 ± 0.831	0.335 ± 0.595
Semi-symbolique	0.113 ± 0.774	0.247 ± 0.551	0.435 ± 0.440	0.685 ± 0.851	0.988 ± 0.918

Table 3: Benchmark du temps d'exécution moyen sur l'example 3 en fonction du nombre de particules

procéder à d'autres expériences sur ce modèle spécial, qui fait varier le nombre de distributions dépendantes, afin de déterminer l'importance du nombre de relations de distributions à explorer pour l'inférence symbolique pour le temps d'exécution. Nous avons fixé le nombre de particules à 1000 pour les deux algorithmes et faisons varier le nombre de distributions potentiellement conjuguées.

Average execution time (milliseconds)	2	4	8	14	20
Importance Sampling	0.119 ± 0.372	0.263 ± 0.453	0.548 ± 0.896	0.902 ± 0.923	1.772 ± 1.155
Semi-symbolique	0.245 ± 0.754	0.671 ± 0.840	1.820 ± 2.453	3.367 ± 1.789	5.113 ± 1.934

Table 4: Benchmark du temps d'exécution moyen en fonction du nombre de distributions potentiellement conjuguées

Les résultats de cette expérience peuvent consulter avec la Figure 8 et le tableau 4 contient le jeu de valeurs observées. On peut constater que nous avons un sur-coût lié au nombre de distribution à explorer, mais qui reste négligeable et linéaire par rapport à l'algorithme d'importance sampling.

Pour finir, on peut remarquer que l'algorithme d'inférence semi-symbolique ne produit pas un résultat purement symbolique sur l'example 4 du Hidden Markov Model alors que les distributions sont composées 2 à 2. Il s'agit d'un axe d'amélioration.

Question 12 (extension)

Lorsque l'on souhaite qu'une expression soit retournée par le modèle, un problème est que l'on peut essayer d'appliquer un opérateur qui attends une variable concrète et non une distribution. Dans l'example: $let\ x\ =\ sample\ d\ in\ x\ +\ 1$, on essaye d'appliquer l'opérateur "+" à une variable aléatoire (car potentiellement utilisée pour du calcul symbolique) à une variable entière, ce qui n'est pas correct. Il faudrait donc redéfinir les opérateurs usuels pour fonctionner avec

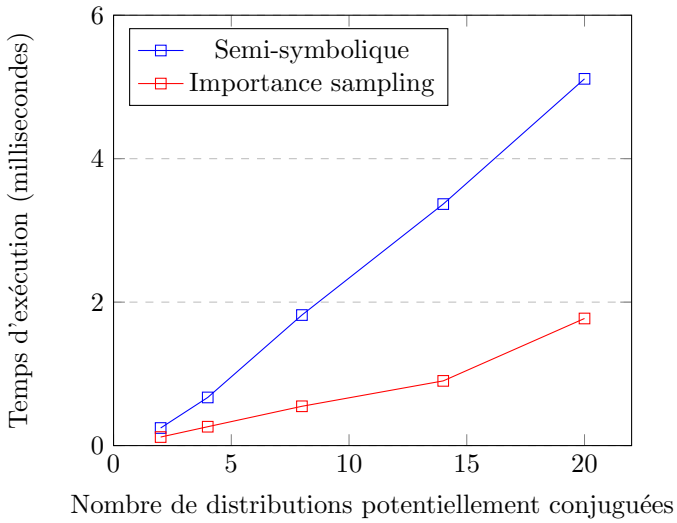

Figure 8: Evolution du temps d'exécution en fonction du nombre de distributions potentiellement conjuguées

des variables aléatoires sans avoir à instancier une valeur pour pouvoir garder les avantages du calcul symbolique.
Une solution pourrait ainsi être d'introduire des opérateurs de calcul formel (ou symbolique) pour garder le côté exact des distributions symboliques. Ils peuvent être inclu comme des distributions avec un nouveau type "Binop" (pour des opérateurs binaires par exemple), comme représenté dans la figure 9. On

```
type dist_op =
  | Beta of node_op * node_op
  | Bernoulli of node_op
  | Gaussian of node_op * node_op
  | V of float
  | Binop of node_op * node_op * (float -> float -> float)
and node_op = dist_op ref
```

Figure 9: Modélisation des distributions symboliques étendues avec des opérateurs symboliques

peut ensuite réaliser l'inférence comme précédemment.
Lors du tirage d'une valeur concrète dans ces nouvelles distributions (fonction value), il est désormais possible d'appliquer la fonction de l'opérateur, comme représenté dans la figure 10.

```
node_op -> float
let rec value d =
  match !d with
  | V v -> d := V v; v
  | Beta (a, b) -> let v = Distribution.draw (Distribution.beta ~a:(value a) ~b:(value b)) in d := V v; v
  | Bernoulli (x) -> let v = Distribution.draw (Distribution.bernoulli_float ~p:(value x)) in d := V v; v
  | Gaussian (m, s) -> let v = Distribution.draw (Distribution.gaussian ~mu:(value m) ~sigma:(value s)) in d := V v; v
  | Binop (a, b, f) -> let v = f (value a) (value b) in d := V v; v
```

Figure 10: Tirage d'une valeur concrète dans une distribution avec application d'opérateurs