Automne 2009

IFT–3245
Examen INTRA

Rappel : le règlement sur le plagiat sera de stricte application.
L'ordre des réponses n'est pas pris en compte, aussi répondez en priorité aux questions dont vous connaissez la réponse.

1. (10 pts) Nous considérons les variables $X_1, \ldots, X_n$, que nous supposons i.i.d., mais de fonction de répartition inconnue F. Comment, à partir des observations $x_1, \ldots, x_n$ seules, pourrions-nous estimer $P[X \leq x]$, où X est une variable aléatoire suivant la même distribution que X_i $(i = 1, \ldots, n)$, sans faire aucune hypothèse sur X (en particulier on ne veut pas utiliser d'approches paramétriques) ?

   Il suffit de construire la fonction de répartition empirique, définie comme suit :

   $$\hat{F}_n(x) = \frac{1}{n} \sum_{i=1}^{n} I[x_i \leq x].$$

   En effet, nous avons que $\hat{F}_n(x)$ converge presque sûrement vers $F(x) = P[X \leq n]$ lorsque n tend vers l'infini. En pratique, cela revient à compter le nombre d'observations inférieures à x et à diviser le résultat par n.

   Il est possible d'affiner la construction comme décrit dans le cours, en supposant pas la fonction constante par morceaux, mais en interpolant par exemple linéairement entre les points.

2. (15 pts) Est-il utile d'avoir un générateur de nombres pseudo-aléatoires dont la période excède le nombre de réels compris entre 0 et 1 qui peuvent être représentés de manière exacte en virgule flottante [1] ? Justifiez votre réponse. Quel avantage offre l'utilisation de sous-suites pour un générateur de nombres aléatoires ?

   Une des propriétés recherchées avec un générateur de nombres aléatoires est de disposer d'une longue période, aussi il est effectivement utile de pouvoir générer plus de nombres que ceux représentables en précision machine. C'est en particulier utile pour la production de sous-suites avec un même générateur de nombres aléatoires. Ces différentes sous-suites doivent elle-mêmes être suffisamment grandes pour permettres d'être considérée individuellement lors d'une simulation, et leur intérêt premier est de pouvoir générer en parallèle plusieurs séquences de nombres aléatoires qui sont en apparence indépendantes.

1. La représentation en virgule flottante est une manière de stocker numériquement un nombre réel ; le nombre est représenté sur un nombre fixe de bits, habituellement 32 ou 64, de sorte que seul un ensemble fini de réel peuvent être codés de manière exacte)

3. (10 pts) Quelles sont les deux distributions de probabilité possédant la propriété sans mémoire ? Pour les systèmes de file d'attente, en supposant une politique de file de type FIFO et un seul serveur, à quel type de modèle (suivant les notations usuelles dans le domaine) est-on confronté en utilisant de telles lois ?

   Les deux seules distributions de probabilité possédant la propriété sans mémoire sont l'exponentielle (cas continu) et la géométrique (cas discret). Dans le cadre des files d'attente, l'utilisation de la loi exponentielle donne lieu à un processus markovien pour les interarrivées et les temps de services. La notation utilisée dans ce cas est M/M/1.

---

4. (10 pts) Dans le cadre de modèles logit, montrez que le rapport des probabilités de deux alternatives i et j ne dépend que de la différence entre les composantes déterministes de leurs utilités respectives, i.e. $V_i - V_j$, indépendamment des autres alternatives. Ceci donne lieu à la propriété IIA. Comment pourrions-nous tester les prédictions de choix (en termes de proportions au sein de la population) d'un modèle logit, et partant de là, tenter de vérifier empiriquement si la propriété IIA est respectée ?

   Nous savons que la probabilité de choix correspondant à l'alternative i est

$$P_i = \frac{e^{V_i}}{\sum_k e^{V_k}},$$

   aussi

$$\frac{P_i}{P_j} = \frac{e^{V_i}}{e^{V_j}} = e^{V_i - V_j}.$$

   Une fois le modèle calibré, connaître les utilités des différentes alternatives permet de connaître leurs probabilités respectives P_k. Un moyen simple de procéder est de diviser le jeu données recueilli pour la calibration du modèle en deux parties sensiblement de même taille. La première partie servira à la calibration proprement dite, tandis que la seconde partie permettra de tester les prédictions du modèles. Pour ce faire, chaque observation de choix est enregistré dans la seconde partie, et les proportions globales sont calculées en fonction de l'ensemble de choix ainsi obtenus. Puisque pour chaque choix, nous disposons des données permettant de calculer les utilités respectives, nous pouvons calculer la probabilité de chaque alternative, et partant de là, en supposons que nous avons n observations, calculer la proportions espérée comme suit :

$$\frac{1}{n} \sum_{l=1}^{n} P_i^{(l)}.$$

   Si les proportions prédites sont proches des proportions observèes, on peut supposer que le modèle est adapté. On peut de plus vérifier la propriété IIA de manière plus spécifiquement en en considérant des sous-ensembles d'alternatives. En se restreignant par exemple à 2 alternatives, et en ne retenant que les observations où une de ces 2 alternatives est sélectionnées,

on peut tester leur probabilités respectives. On peut vérifier si les rapports sont cohérents deux à deux en observant que

$$\frac{P_i}{P_k} = \frac{P_i}{P_j}\frac{P_j}{P_k}.$$

5. (15 pts) On se propose d'utiliser un générateur dont le code C est donné ci-dessous. Pouvez-vous caractériser ce générateur en définissant les composantes principales de celui-ci et la/les famille(s) auquel il appartient ? Considérant les générateurs vus au cours, en termes de performances statistiques (et non de temps d'exécution), semble-t-il intéressant (on demande ici uniquement l'intuition, la réponse précise exigeant bien entendu des tests numériques) ?

```
#define m1   2147483563
#define a1   40014
#define q1   53668
#define r1   12211
#define m2   2147483399
#define a2   40692
#define q2   52774
#define r2   3791
#define mm   2147483562
#define imm  4.656613e-10   /* 1/mm */

double random_get_val(long s1, s2)
{
  long k, z;

  k = s1/q1;
  s1 = a1*(s1-k*q1)-k*r1;
  if (s1 < 0) s1 += m1;

  k = s2/q2;
  s2 = a2*(s2-k*q2)-k*r2;
  if (s2 < 0) s2 += m2;

  z = s1-s2;
  if (z < 1) z += mm;

  return z*imm;
}
```

Il s'agit d'un générateur combiné reposant sur deux LCG's, proposé par L'Ecuyer en 1988. Les LCG's sont implantés en reprenant la technique de factorisation approximative. Ce générateur est repris dans le livre Numerical Recipes in C avec en plus l'addition d'une technique de sortie différée.

Le fait de combiner deux générateurs permet d'obtenir de bien meilleures propriétés statistiques que l'utilisation des LCG's sous-jacent seuls, et ce générateur passe pratiquement tous les tests statistiques de TestU01.

Il souffre néanmoins d'une période de l'ordre de 2^{61}, ce qui pour une application isolée est suffisant, mais est moins favorable à l'établissement d'un grand nombre de streams partageant cette séquence, lesquels streams devant eux-même garder une période aussi longue que possible.

L'utilisation de MRG combinés comme le MRG32K3a permet de conservant les bonnes propriétés liées à la combinaison de générateurs simples, pour un temps de calcul légèrement plus important, mais une période nettemant plus importante, favorisant la décomposition en streams.

---

6. (15 pts) Expliquez (en détaillant) comment générer, via le principe d'inversion,
   (a) la distribution de Cauchy, pour laquelle $F(x) = 1/2 + \arctan(x)/\pi$, $x \in \mathcal{R}$;
   (b) la distribution logistique, pour laquelle $F(x) = 1/(1 + e^{-x})$, $x \in \mathcal{R}$;
   (c) la distribution de Rayleigh de paramètre β, pour laquelle $F(x) = 1 - e^{-x^2/\beta}$, $x > 0$.

   Note : pour éviter les problèmes avec les distributions non bornées, on suppose que u est tiré d'un générateur $U(0, 1)$ (autrement dit, on ne génère jamais 0 ou 1).

   (a) Cauchy

$$\begin{aligned} u = F(x) &= 1/2 + \arctan(x)/\pi; \\ \Leftrightarrow F^{-1}(u) = x &= \tan(\pi(u - 1/2)). \end{aligned}$$

   (b) Logistique

$$\begin{aligned} u = F(x) &= 1/(1 + e^{-x}) \\ \Leftrightarrow F^{-1}(u) = x &= -\ln(1/u - 1) \\ \Leftrightarrow F^{-1}(u) &= \ln \frac{u}{1 - u} \end{aligned}$$

   (c) Rayleigh

$$\begin{aligned} u = F(x) &= 1 - e^{-x^2/\beta} \\ \Leftrightarrow F^{-1}(u) = x &= \sqrt{-\beta \ln(1 - u)} \\ \Leftrightarrow F^{-1}(u) &= \sqrt{-\beta \ln(1 - u)} \end{aligned}$$

   De manière équivalent, on peut générer X ave la procédure : tirer u de U(0,1) et retourner

$$\sqrt{-\beta \ln u}.$$

---

7. (25 pts) On considère le problème suivant. Lors d'un jeu-concours, on offre un prix aux 150 premiers appels qui fournissent une bonne réponse à une question donnée, la même pour tous les appelants. La bonne réponse est donnée avec une probabilité de 0.8 lorsqu'un appel est pris. Toutefois, pour augmenter l'aspect aléatoire du jeu, seul un appel sur cinq est accepté, et ce de manière aléatoire (uniforme).

   On sait que les appels arrivent suivant un processus de Poisson, avec une arrivée en moyenne toutes les 10 secondes. Si l'appel entrant est accepté, le temps accordé à la personne suit une loi uniforme entre une et deux minutes. Les appels entrants et acceptés sont mis en attente si la ligne est déjà occupée. On suppose qu'aucun joueur n'abandonne quand il est mis en attente.

   On souhaite connaître la durée moyenne que durera le jeu, i.e. après combien de temps les 150 prix auront été distribués, ainsi que le nombre moyen d'appels reçus.

   Ecrivez un pseudo-code de simulation par événements discrets permettant de simuler le jeu et de connaître ce temps moyen nécessaire pour obtenir les 150 gagnants. Explicitez en particulier chaque type d'événement à considérer, et comment planifier l'occurence des événements, sachant que les détails l'horloge de simulation sont supposés pris en charge en interne par le langage de simulation. Par pseudo-code, on attend que les propriétés de l'algorithme soient clairement et précisément présentées, mais il n'est pas nécessaire d'utiliser la syntaxe d'un langage de programmation réel.

Supposons que toutes les grandeurs temporelles sont exprimées en minutes. Nous avons besoin des variables aléatoires suivantes :

1. Soit $B \sim$ Bernouilli(0.8). B servira à déterminer si la personne dont l'appel est pris donne la bonne réponse.
2. $A \sim$ exponentielle(1/6) représente les temps d'arrivée, avec un taux de vu que $E[A] = 10s = 1/6$ minute.
3. $X \sim$ Bernouilli(0.2) la variable aléatoire d'acceptation d'un appel.
4. $T \sim U(1, 2)$: temps consacré au traitement d'un appel accepté.

Nous avons besoin des structures de données suivantes :

1. FIFO waiting list (initialisé à vide) ;
2. server ;
3. number of good answers (initialisé à 0) ;
4. time of game ;
5. number of calls (initialisé à 0).

Le premier événement que nous considérons est l'arrivée, qui se déroule comme suit :

```
Arrival extends event

number of calls++.
```

```
a = draw from A.
schedule next_arrival in a minutes.
X = draw from X
if (X == 0) reject the call; return.
(or draw from U(0,1) and reject if u < 0.2.)
If server is busy join waiting list.
else schedule departure from server, after time t, drawn from T.
check_answer()
```

La fonction `check_answer` est simple :

```
b = draw from B.
if (b == 1) number of good answers++
```

L'événement de depart est relativement simple

```
server = free;
if (number of good answers == 150) {
  time of game = sim.clock();
  sim.stop();
}
if (waiting list is non empty) {
 remove first from waiting list;
  schedule departure from server, after time t, drawn from T;
  server = busy;
  check_answer();
}
```

Une simulation consiste alors à

```
Create A, B, X;
sim.init();
Schedule next (first) arrival;
sim.start()
```

Finalement, on répète la simulation n fois et on affiche le temps et le nombre d'appels reçus moyens.