Probabilités avec CAML

INFO1 - Semaine 22

Guillaume CONNAN

IUT de Nantes - Dpt d'informatique

Dernière mise à jour : 28 mai 2013

Sommaire

1. Outils de simulation
2. Loi binomiale
   - Simulation
   - Calcul des valeurs théoriques
3. Loi géométrique ?...
4. Loi hypergéométrique ?...
5. Loi de Poisson

Sommaire

1. Outils de simulation
2. Loi binomiale
   - Simulation
   - Calcul des valeurs théoriques
3. Loi géométrique ?...
4. Loi hypergéométrique ?...
5. Loi de Poisson

```
val int : int -> int
```

Random.int bound returns a random integer between 0 (inclusive) and bound (exclusive). bound must be greater than 0 and less than 2^30.

```
val float : float -> float
```

Random.float bound returns a random floating-point number between 0 and bound (inclusive). If bound is negative, the result is negative or zero. If bound is 0, the result is 0.

```
val int : int -> int
```

Random.int bound returns a random integer between 0 (inclusive) and bound (exclusive). bound must be greater than 0 and less than 2^30.

```
val float : float -> float
```

Random.float bound returns a random floating-point number between 0 and bound (inclusive). If bound is negative, the result is negative or zero. If bound is 0, the result is 0.

```
(* vecteur aléatoire de n entiers entre min et max *)
let vec_alea (mini: int) (maxi: int) (taille: int): int list =
 let rec aux_alea (size: int) (acc: int list): int list =
   if size = 0 then acc
   else aux_alea (size - 1) ((mini + Random.int (maxi - mini + 1)) :: acc)
 in aux_alea taille [];;
```

```
# vec_alea 1 6 15;;
- : int list = [5; 3; 2; 4; 1; 2; 6; 6; 3; 5; 4; 1; 6; 1; 6]
```

```
(* vecteur aléatoire de n entiers entre min et max *)
let vec_alea (mini: int) (maxi: int) (taille: int): int list =
 let rec aux_alea (size: int) (acc: int list): int list =
   if size = 0 then acc
   else aux_alea (size - 1) ((mini + Random.int (maxi - mini + 1)) :: acc)
 in aux_alea taille [];;
```

```
# vec_alea 1 6 15;;
- : int list = [5; 3; 2; 4; 1; 2; 6; 6; 3; 5; 4; 1; 6; 1; 6]
```

```
(* sommes des éléments d'une liste d'entiers *)
let som (liste : int list): int =
  List.fold_left (+) 0 liste;;
```

```
(* simulations du lancer de trois dés *)
let toscane ((): unit): int =
  som (vec_alea 1 6 3);;
```

```
# toscane;;
- : unit -> int = <fun>
```

```
# toscane ();;
- : int = 10
```

```
(* sommes des éléments d'une liste d'entiers *)
let som (liste : int list): int =
  List.fold_left (+) 0 liste;;
```

```
(* simulations du lancer de trois dés *)
let toscane ((): unit): int =
  som (vec_alea 1 6 3);;
```

```
# toscane;;
- : unit -> int = <fun>
```

```
# toscane ();;
- : int = 10
```

```
(* sommes des éléments d'une liste d'entiers *)
let som (liste : int list): int =
  List.fold_left (+) 0 liste;;
```

```
(* simulations du lancer de trois dés *)
let toscane ((): unit): int =
  som (vec_alea 1 6 3);;
```

```
# toscane;;
- : unit -> int = <fun>
```

```
# toscane ();;
- : int = 10
```

```
(* sommes des éléments d'une liste d'entiers *)
let som (liste : int list): int =
  List.fold_left (+) 0 liste;;
```

```
(* simulations du lancer de trois dés *)
let toscane ((): unit): int =
  som (vec_alea 1 6 3);;
```

```
# toscane;;
- : unit -> int = <fun>
```

```
# toscane ();;
- : int = 10
```

```
(* une liste d'expériences aléatoires *)
let simul_exp (exp: unit -> 'a) (occ: int): 'a list =
  let rec aux_exp = fun n acc ->
    if n = occ then acc
    else aux_exp (n + 1) ((exp ()) :: acc)
  in aux_exp 0 [];;
```

```
# simul_exp toscane 15;;
- : int list = [12; 6; 14; 12; 8; 15; 9; 11; 12; 9; 8; 6; 15; 11; 6]
```

```
(* une liste d’expériences aléatoires *)
let simul_exp (exp: unit -> ’a) (occ: int): ’a list =
  let rec aux_exp = fun n acc ->
    if n = occ then acc
    else aux_exp (n + 1) ((exp ()) :: acc)
  in aux_exp 0 [];;
```

```
# simul_exp toscane 15;;
- : int list = [12; 6; 14; 12; 8; 15; 9; 11; 12; 9; 8; 6; 15; 11; 6]
```

```
(* renvoie la liste des couples (élément, nombre d'occurrences dans l'
    ensemble)
  avant réduction *)
let compte (liste: 'a list) : ('a * float) list =
 let occ = float_of_int (List.length liste) in
 let rec count = fun l accu ->
   match l with
   | []    -> accu
   |t :: q ->
     let c = List.partition (fun x -> x = t) l in
     count (snd c) ((t,(float_of_int (List.length (fst c))) /. occ)::accu)
 in count liste [];;
```

```
# compte  [12; 6; 14; 12; 8; 15; 9; 11; 12; 9; 8; 6; 15; 11; 6];;
- : (int * float) list =
[(11, 0.133333333333333331); (9, 0.133333333333333331);
 (15, 0.133333333333333331); (8, 0.133333333333333331);
 (14, 0.0666666666666666657); (6, 0.2); (12, 0.2)]
```

```
# Printf.sprintf "%.3f" 0.123456789;;
- : string = "0.123"
```

```
# compte  [12; 6; 14; 12; 8; 15; 9; 11; 12; 9; 8; 6; 15; 11; 6];;
- : (int * float) list =
[(11, 0.133333333333333331); (9, 0.133333333333333331);
 (15, 0.133333333333333331); (8, 0.133333333333333331);
 (14, 0.0666666666666666657); (6, 0.2); (12, 0.2)]
```

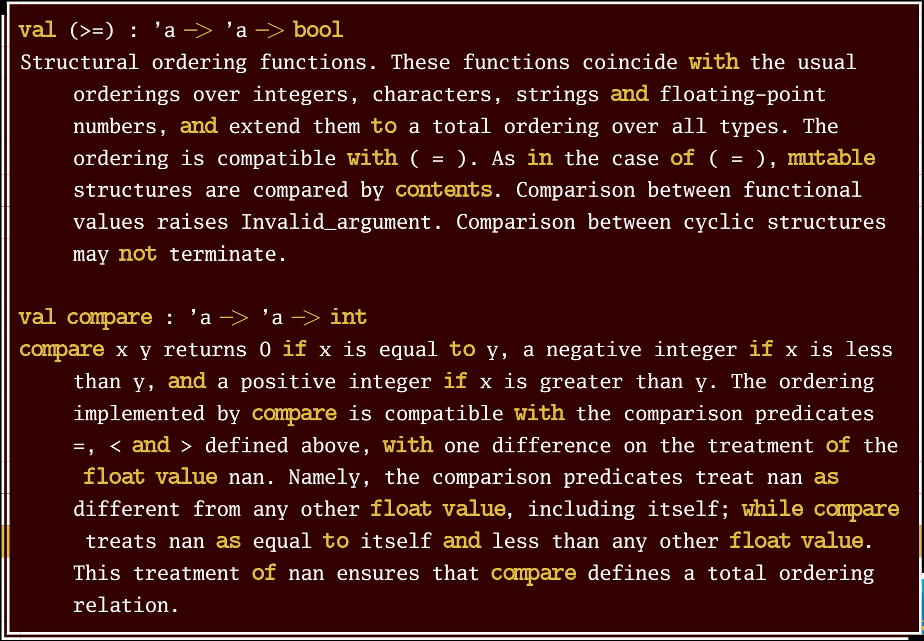
```
# Printf.sprintf "%.3f" 0.123456789;;
- : string = "0.123"
```

```
let pretty_dic (ft: ('i -> 's, unit, string) format) (dic: ('a * float)
    list): ('a * string) list =
  List.map (fun cpl -> (fst cpl, Printf.sprintf ft (snd cpl))) dic;;
```

```
# pretty_dic "%.3f" (compte  [12; 6; 14; 12; 8; 15; 9; 11; 12; 9; 8; 6;
    15; 11; 6]);;
- : (int * string) list =
[(11, "0.133"); (9, "0.133"); (15, "0.133"); (8, "0.133"); (14, "0.067");
 (6, "0.200"); (12, "0.200")]
```

```
let pretty_dic (ft: ('i -> 's, unit, string) format) (dic: ('a * float)
    list): ('a * string) list =
  List.map (fun cpl -> (fst cpl, Printf.sprintf ft (snd cpl))) dic;;
```

```
# pretty_dic "%.3f" (compte  [12; 6; 14; 12; 8; 15; 9; 11; 12; 9; 8; 6;
    15; 11; 6]);;
- : (int * string) list =
[(11, "0.133"); (9, "0.133"); (15, "0.133"); (8, "0.133"); (14, "0.067");
 (6, "0.200"); (12, "0.200")]
```



```
val (>=) : 'a -> 'a -> bool
Structural ordering functions. These functions coincide with the usual
    orderings over integers, characters, strings and floating-point
    numbers, and extend them to a total ordering over all types. The
    ordering is compatible with ( = ). As in the case of ( = ), mutable
    structures are compared by contents. Comparison between functional
    values raises Invalid_argument. Comparison between cyclic structures
    may not terminate.

val compare : 'a -> 'a -> int
compare x y returns 0 if x is equal to y, a negative integer if x is less
    than y, and a positive integer if x is greater than y. The ordering
    implemented by compare is compatible with the comparison predicates
    =, < and > defined above, with one difference on the treatment of the
     float value nan. Namely, the comparison predicates treat nan as
    different from any other float value, including itself; while compare
     treats nan as equal to itself and less than any other float value.
    This treatment of nan ensures that compare defines a total ordering
    relation.
```

```
# type truc = Lego | Playmobil | Schtroumpf | Hobbit;;
type truc = Lego | Playmobil | Schtroumpf | Hobbit
# Lego < Playmobil;;
- : bool = true
# compare Lego Hobbit;;
- : int = -1
# compare Hobbit Lego;;
- : int = 1
# compare Lego Lego;;
- : int = 0
```

```
let simul_dic (exp: unit -> 'a) (occ: int): ('a * float) list =
  List.sort compare (compte (simul_exp exp occ));;
```

```
# pretty_dic "%.3f" (simul_dic toscane 100_000);;
- : (int * string) list =
[(3, "0.005"); (4, "0.015"); (5, "0.028"); (6, "0.045"); (7, "0.071"); (8,
     "0.097"); (9, "0.114"); (10, "0.125"); (11, "0.126"); (12, "0.115");
     (13, "0.098"); (14, "0.070"); (15, "0.047"); (16, "0.027"); (17, "
    0.014"); (18, "0.005")]
```

```
let simul_dic (exp: unit -> 'a) (occ: int): ('a * float) list =
  List.sort compare (compte (simul_exp exp occ));;
```

```
# pretty_dic "%.3f" (simul_dic toscane 100_000);;
- : (int * string) list =
[(3, "0.005"); (4, "0.015"); (5, "0.028"); (6, "0.045"); (7, "0.071"); (8,
     "0.097"); (9, "0.114"); (10, "0.125"); (11, "0.126"); (12, "0.115");
     (13, "0.098"); (14, "0.070"); (15, "0.047"); (16, "0.027"); (17, "
    0.014"); (18, "0.005")]
```

Sommaire

1 Outils de simulation

2 Loi binomiale

- Simulation
- Calcul des valeurs théoriques

3 Loi géométrique ?...

4 Loi hypergéométrique ?...

5 Loi de Poisson

Sommaire

1 Outils de simulation

2 Loi binomiale

- Simulation
- Calcul des valeurs théoriques

3 Loi géométrique ?...

4 Loi hypergéométrique ?...

5 Loi de Poisson

```
(* renvoie 1 avec une probabilité p et 0 sinon *)
let bernoulli (p: float) : unit -> int  =
  fun () -> if (Random.float 1. < p) then 1 else 0;;
```

```
# bernoulli 0.2;;
- : unit -> int = <fun>
# bernoulli 0.2 ();;
- : int = 1

#  simul_dic (bernoulli 0.2) 100_000;;
- : (int * float) list = [(0, 0.80121); (1, 0.19879)]
```

```
(* renvoie 1 avec une probabilité p et 0 sinon *)
let bernoulli (p: float) : unit -> int  =
  fun () -> if (Random.float 1. < p) then 1 else 0;;
```

```
# bernoulli 0.2;;
- : unit -> int = <fun>
# bernoulli 0.2 ();;
- : int = 1

#  simul_dic (bernoulli 0.2) 100_000;;
- : (int * float) list = [(0, 0.80121); (1, 0.19879)]
```

```
(* X ~ B(n,p) comme somme de var de Bernoulli B(1,p) *)
let binomial  (n: int) (p: float) :  unit -> int  =
  fun () -> som (simul_exp (bernoulli p) n);;
```

5 jetons tirés successivement dans une urne avec 5 blancs et 9 noirs. X : nombre de blancs tirés.

```
# pretty_dic "%.3f" (simul_dic (binomial 5 (5. /. 14.)) 100_000);;
- : (int * string) list =
[(0, "0.110"); (1, "0.304"); (2, "0.339"); (3, "0.188"); (4, "0.052"); (5,
      "0.006")]
```

```
(* X ~ B(n,p) comme somme de var de Bernoulli B(1,p) *)
let binomial  (n: int) (p: float) :  unit -> int  =
  fun () -> som (simul_exp (bernoulli p) n);;
```

5 jetons tirés successivement dans une urne avec 5 blancs et 9 noirs. X : nombre de blancs tirés.

```
# pretty_dic "%.3f" (simul_dic (binomial 5 (5. /. 14.)) 100_000);;
- : (int * string) list =
[(0, "0.110"); (1, "0.304"); (2, "0.339"); (3, "0.188"); (4, "0.052"); (5,
     "0.006")]
```

```
(* X ~ B(n,p) comme somme de var de Bernoulli B(1,p) *)
let binomial  (n: int) (p: float) :  unit -> int  =
  fun () -> som (simul_exp (bernoulli p) n);;
```

5 jetons tirés successivement dans une urne avec 5 blancs et 9 noirs. X : nombre de blancs tirés.

```
# pretty_dic "%.3f" (simul_dic (binomial 5 (5. /. 14.)) 100_000);;
- : (int * string) list =
[(0, "0.110"); (1, "0.304"); (2, "0.339"); (3, "0.188"); (4, "0.052"); (5,
    "0.006")]
```

Sommaire

1 Outils de simulation

2 Loi binomiale

- Simulation
- Calcul des valeurs théoriques

3 Loi géométrique ?…

4 Loi hypergéométrique ?…

5 Loi de Poisson

```
(* on utilise C(n,p) = (n-p+1)/p * C(n,p-1) *)
let binom = fun n p ->
   if p > n || n < 0 || p < 0 then
     0
   else
     let rec aux = fun acc num den ->
       if den <= p then
         aux ((acc * num) / den) (num - 1) (den + 1)
       else
         acc
     in aux 1 n 1;;
```

```
# ( ** );;
- : float -> float -> float = <fun>
# 2. ** 10. ;;
- : float = 1024.
```

```
(* calcul de Px(k) quand X ~ B(n,p) *)
let proba_bin (n: int) (p: float) (k: int) : float =
  (float_of_int (binom n k)) *. (p ** (float_of_int k)) *. ((1. -. p)**(
      float_of_int (n - k)));;
```

```
# ( ** );;
- : float -> float -> float = <fun>
# 2. ** 10. ;;
- : float = 1024.
```

```
(* calcul de Px(k) quand X ~ B(n,p) *)
let proba_bin (n: int) (p: float) (k: int) : float =
  (float_of_int (binom n k)) *. (p ** (float_of_int k)) *. ((1. -. p)**(
      float_of_int (n - k)));;
```

```
(* renvoie le dictionnaire des (k, Px(k)) quand X ~ B(n,p) *)
let dic_bin (n: int) (p: float) : (int * float) list =
  let rec aux = fun i acc ->
    if i > n then
      List.sort compare acc
    else
      let pr = proba_bin n p i in
      aux (i + 1) ((i,pr) :: acc)
  in aux 0 [];;
```

```
# pretty_dic "%.3f" (simul_dic (binomial 5 (5. /. 14.)) 100_000);;
- : (int * string) list =
[(0, "0.108"); (1, "0.305"); (2, "0.340"); (3, "0.187"); (4, "0.054"); (5,
    "0.006")]
```

```
# pretty_dic "%.3f" (dic_bin 5 (5. /. 14.));;
- : (int * string) list =
[(0, "0.110"); (1, "0.305"); (2, "0.339"); (3, "0.188"); (4, "0.052"); (5,
    "0.006")]
```

```
# pretty_dic "%.3f" (simul_dic (binomial 5 (5. /. 14.)) 100_000);;
- : (int * string) list =
[(0, "0.108"); (1, "0.305"); (2, "0.340"); (3, "0.187"); (4, "0.054"); (5,
     "0.006")]
```

```
# pretty_dic "%.3f" (dic_bin 5 (5. /. 14.));;
- : (int * string) list =
[(0, "0.110"); (1, "0.305"); (2, "0.339"); (3, "0.188"); (4, "0.052"); (5,
     "0.006")]
```

Sommaire

1 Outils de simulation

2 Loi binomiale
- Simulation
- Calcul des valeurs théoriques

3 Loi géométrique ?...

4 Loi hypergéométrique ?...

5 Loi de Poisson

Sommaire

1 Outils de simulation

2 Loi binomiale

- Simulation
- Calcul des valeurs théoriques

3 Loi géométrique ?...

4 Loi hypergéométrique ?...

5 Loi de Poisson

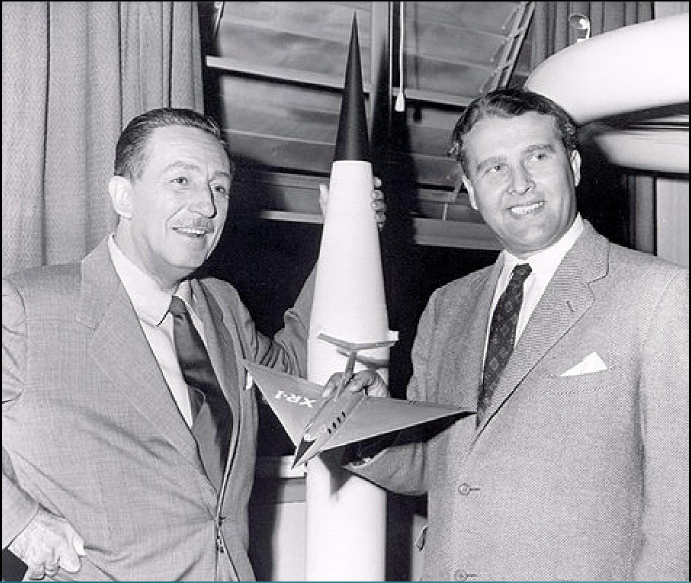
Sommaire

1. Outils de simulation
2. Loi binomiale
   - Simulation
   - Calcul des valeurs théoriques
3. Loi géométrique ?...
4. Loi hypergéométrique ?...
5. Loi de Poisson

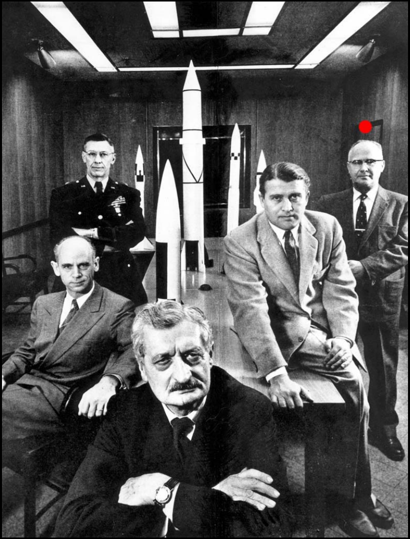
WWII

Von Braun et ses vieux amis

Von Braun et son nouvel ami



Lusser, père du V1 et Von Braun aux États-Unis...



Le V1

Le V1 de passage à Londres

No. of flying bombs per square	0	1	2	3	4	5 and over
No. of squares	229	211	93	35	7	1

Source :
http ://www.actuaries.org.uk/sites/all/files/documents/pdf/0481.pdf

576 carrés de $0,25$ km^2.

537 impacts de bombe.

$\lambda = 537/576$

No. of flying bombs per square	0	1	2	3	4	5 and over
No. of squares	229	211	93	35	7	1

Source :
http ://www.actuaries.org.uk/sites/all/files/documents/pdf/0481.pdf
576 carrés de $0,25\ km^2$.
537 impacts de bombe.
$\lambda = 537/576$

No. of flying bombs per square	0	1	2	3	4	5 and over
No. of squares	229	211	93	35	7	1

Source :
http ://www.actuaries.org.uk/sites/all/files/documents/pdf/0481.pdf
576 carrés de $0,25\ km^2$.
537 impacts de bombe.
$\lambda = 537/576$

$$\mathbb{P}(\{X = k+1\}) = \frac{\lambda}{k+1}\mathbb{P}(\{X = k\})$$

```
let poisson (lambda: float) (n: int) : float =
  let rec aux = fun k acc ->
    if k = n then acc
    else aux (k + 1) ((lambda /. (float_of_int (k + 1))) *. acc)
  in aux 0 (exp (-. lambda));;
```

$$\mathbb{P}(\{X = k+1\}) = \frac{\lambda}{k+1}\mathbb{P}(\{X = k\})$$

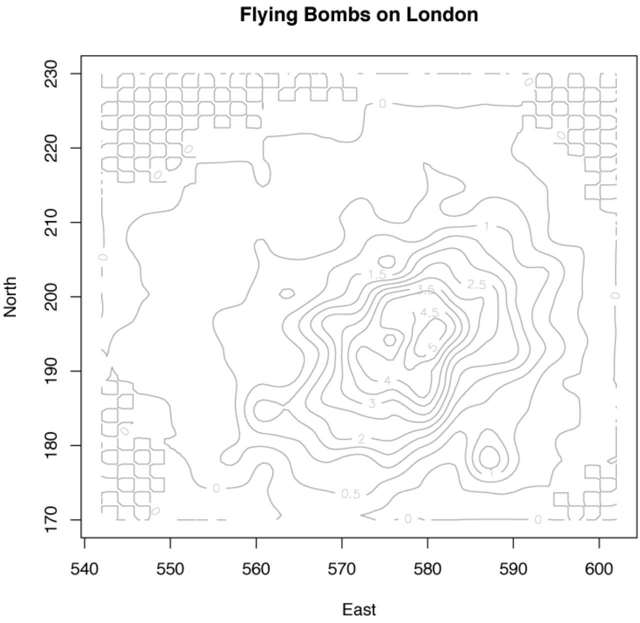
```
let poisson (lambda: float) (n: int) : float =
  let rec aux = fun k acc ->
    if k = n then acc
    else aux (k + 1) ((lambda /. (float_of_int (k + 1))) *. acc)
  in aux 0 (exp (-. lambda));;
```

```
let lambda = 537. /. 576. ;;

# List.map (fun k -> 576. *. (poisson lambda k)) [0;1;2;3;4];;
- : float list =
[226.742722583239527; 211.390350741666; 98.5387312050995092;
 30.6222793154736301; 7.13722395503877571]
```

No. of flying bombs per square	0	1	2	3	4	5 and over
No. of squares	229	211	93	35	7	1

```
let lambda = 537. /. 576. ;;

# List.map (fun k -> 576. *. (poisson lambda k)) [0;1;2;3;4];;
- : float list =
[226.742722583239527; 211.390350741666; 98.5387312050995092;
 30.6222793154736301; 7.13722395503877571]
```

No. of flying bombs per square	0	1	2	3	4	5 and over
No. of squares	229	211	93	35	7	1


Flying Bombs on London
North
230
220
210
200
190
180
170
East
540
550
560
570
580
590
600
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5