public2018_D2

May 21, 2019

1 Table of Contents

2 Texte d'oral de modélisation - Agrégation Option Informatique

2.1 Préparation à l'agrégation - ENS de Rennes, 2017-18

- *Date* : 6 décembre 2017
- *Auteur* : Lilian Besson
- *Texte*: Annale 2018, "Polynômes avec des nombres flottants" (public2018-D2)

2.2 À propos de ce document

- Ceci est une *proposition* de correction, partielle et probablement non-optimale, pour la partie implémentation d'un texte d'annale de l'agrégation de mathématiques, option informatique.
- Ce document est un notebook Jupyter, et est open-source sous Licence MIT sur GitHub, comme les autres solutions de textes de modélisation que j'ai écrite cette année.
- L'implémentation sera faite en OCaml, version 4+ :

```
In [1]: Sys.command "ocaml -version";;
        print_endline Sys.ocaml_version;;
```

```
The OCaml toplevel, version 4.04.2

Out[1]: - : int = 0

4.04.2

Out[1]: - : unit = ()

In [7]: let print f =
            let r = Printf.printf f in
            flush_all();
            r
        ;;

Out[7]: val print : ('a, out_channel, unit) format -> 'a = <fun>
```

2.3 Question de programmation

La question de programmation pour ce texte était donnée en page 6 :

Cet exercice consiste à observer le comportement des nombres en virgule flottante de la machine. Toutes les variables utilisées seront de type `double` ou `float` selon le langage choisi. On écrit un programme qui évalue

$$S = \sum_{i=0}^{\infty} \left(1 - \frac{1}{\rho}\right)^i$$

où ρ est une puissance de 2 suffisamment grande, par exemple $\rho = 2^{20}$.

Si on calcule naïvement $S_n = \sum_{i=0}^{n} (1 - \frac{1}{\rho})^i$, en ajoutant les termes un par un dans l'ordre des puissances croissantes, on remarque quand n est suffisamment grand, $S_{n+1} = S_n \oplus (1 - \frac{1}{\rho})^{n+1}$ lorsqu'on calcule en `double` (ou `float`).

En évaluant S sous la forme

$$S = \sum_{j=0}^{\infty} \sum_{k=0}^{K-1} \left(1 - \frac{1}{\rho}\right)^{jK+k}$$

ce phénomène se produit pour des rangs plus grands que précedemment, l'approximation de S calculée est donc meilleure. On pourra essayer plusieurs valeurs de K $(1, 10, 100, 1000, 10000)$ et comparer. $K = 1$ correspond au calcul naïf de S ci-dessus. Optionnellement, le programme pourra également calculer une borne sur l'erreur com- mise sur S.

2.3.1 Modélisation

On est libre de choisir l'approche, mais ici il n'y a pas tellement de choix, on va utiliser des `float` de OCaml.

2.4 Solution

On va essayer d'être rapide et de faire simple.

2.4.1 Types et représentations

Rien à faire ici, on va travailler avec des `float` de OCaml classiques.

2.4.2 Algorithmes

Première méthode de calcul. On remarque qu'on est déjà malin, on n'utilise aucune exponentiation mais simplement un accumulateur `terme` qui vaut $(1-1/\rho)^j$ pour les valeurs successives de j, et qu'on met à jour par une simple multiplication.

```
In [39]: let premiercalcul (rho : float) (n : int) =
             let somme = ref 0. in
             let unmoinsunsurrho = 1. -. (1. /. rho) in
             let terme = ref 1. in
             for _ = 0 to n do  (* terme = (1-1/rho)^j pour j = _ *)
                 somme := !somme +. !terme; (* somme = sum_k=0^j terme_k *)
                 terme := !terme *. unmoinsunsurrho; (* conserve l'invariant de boucle *)
             done;
             !somme
         ;;

Out[39]: val premiercalcul : float -> int -> float = <fun>
```

Voyons un exemple :

```
In [10]: let rho = 2. ** 20.;;

Out[10]: val rho : float = 1048576.
```

```
In [14]: 1. -. 1. /. rho;;

Out[14]: - : float = 0.999999046325683594
```

Cette valeur est proche de 1, mais strictement plus petite que 1, et donc $(1-1/\rho)^i \to 0$ géométriquement vite, ainsi S_n converge bien vers S pour $n \to \infty$.

```
In [40]: for n = 0 to 10 do
             print "\nS_%i \t= %g" n (premiercalcul rho n);
         done;

S_10000         = 1.04847e+06
S_100000         = 1.04847e+06
S_0         = 1
```

```
S_1         = 2
S_2         = 3
S_3         = 3.99999
S_4         = 4.99999
S_5         = 5.99999
S_6         = 6.99998
S_7         = 7.99997
S_8         = 8.99997


Out[40]: - : unit = ()


In [42]: for n = 1000 to 1010 do
             print "\nS_%i \t= %g" n (premiercalcul rho n);
         done;

S_1009          = 1009.51
S_1010          = 1010.51
S_1000          = 1000.52
S_1001          = 1001.52
S_1002          = 1002.52
S_1003          = 1003.52
S_1004          = 1004.52
S_1005          = 1005.52
S_1006          = 1006.52
S_1007          = 1007.52
S_1008          = 1008.52


Out[42]: - : unit = ()
```

Deuxième méthode de calcul :

```
In [21]: let deuxiemecalcul (rho : float) (k : int) (n : int) =
             let somme = ref 0. in
             let unmoinsunsurrho = 1. -. (1. /. rho) in
             let terme = ref 1. in
             for _ = 0 to n do
                 for _ = 0 to k-1 do
                     somme := !somme +. !terme;
                     terme := !terme *. unmoinsunsurrho;
                 done;
                 terme := !terme *. unmoinsunsurrho;
             done;
             !somme
         ;;
```

```
Out[21]: val deuxiemecalcul : float -> int -> int -> float = <fun>
```

Voyons le même exemple :

```
In [27]: let valeurs_k = [|1; 10; 100; 1000; 10000|] in

         for i = 0 to (Array.length valeurs_k) - 1 do
             let k = valeurs_k.(i) in
             print "\n\nFor K = %i ..." k;
             for n = 0 to 10 do
                 print "\nS_%i \t= %g" n (deuxiemecalcul rho k n);
             done;
         done;

S_9         = 95379.3
S_10         = 104426

For K = 1 ...
S_0         = 1
S_1         = 2
S_2         = 2.99999
S_3         = 3.99999
S_4         = 4.99998
S_5         = 5.99997
S_6         = 6.99996
S_7         = 7.99995
S_8         = 8.99993
S_9         = 9.99991
S_10         = 10.9999

For K = 10 ...
S_0         = 9.99996
S_1         = 19.9998
S_2         = 29.9996
S_3         = 39.9992
S_4         = 49.9987
S_5         = 59.9982
S_6         = 69.9975
S_7         = 79.9967
S_8         = 89.9958
S_9         = 99.9949
S_10         = 109.994

For K = 100 ...
S_0         = 99.9953
S_1         = 199.981
S_2         = 299.957
```

```
S_3         = 399.923
S_4         = 499.88
S_5         = 599.827
S_6         = 699.765
S_7         = 799.693
S_8         = 899.611
S_9         = 999.52
S_10         = 1099.42

For K = 1000 ...
S_0         = 999.524
S_1         = 1998.09
S_2         = 2995.71
S_3         = 3992.38
S_4         = 4988.09
S_5         = 5982.86
S_6         = 6976.67
S_7         = 7969.54
S_8         = 8961.46
S_9         = 9952.43
S_10         = 10942.5

For K = 10000 ...
S_0         = 9952.47
S_1         = 19810.5
S_2         = 29574.9
S_3         = 39246.6
S_4         = 48826.6
S_5         = 58315.6
S_6         = 67714.5
S_7         = 77024.2


Out[27]: - : unit = ()


S_8         = 86245.5
```

Avec de grandes valeurs de n, on semble converger vers la limite S, d'autant plus vite que K est grand.

```
In [38]: let valeurs_n = [|100; 1000; 10000; 100000|] in
         let valeurs_k = [|1; 10; 100; 1000; 10000|] in

         for i = 0 to (Array.length valeurs_k) - 1 do
             let k = valeurs_k.(i) in
             print "\n\nFor K = %i ..." k;
             for j = 0 to (Array.length valeurs_n) - 1 do
```

```
                let n = valeurs_n.(j) in
                print "\nS_%i \t= %g" n (deuxiemecalcul rho k n);
            done;
        done;
```

```
S_10000          = 1.04847e+06
S_100000          = 1.04847e+06

For K = 1 ...
S_100          = 100.99
S_1000          = 1000.05
S_10000          = 9906.23
S_100000          = 91042.7

For K = 10 ...
S_100          = 1009.47
S_1000          = 9957.64
S_10000          = 94942.6
S_100000          = 619357

For K = 100 ...
S_100          = 10051
S_1000          = 95425.8
S_10000          = 641990
S_100000          = 1.03813e+06

For K = 1000 ...
S_100          = 96283.8
S_1000          = 644662
S_10000          = 1.04745e+06
S_100000          = 1.04753e+06

For K = 10000 ...
S_100          = 648345
S_1000          = 1.0484e+06


Out[38]: - : unit = ()
```

2.5 Complexités

2.5.1 En espace

Ces calculs coûtent un espace constant en mémoire.
(ou $\log n$ si on considère que stocker des entiers bornés par n coûtent un espace $\log n$)

2.5.2 En temps

Ces calculs coûtent un temps linéaire en n.

2.6 Conclusion

Voilà pour la question obligatoire de programmation.

2.6.1 Qualités

- On a fait des exemples et *on les garde* dans ce qu'on présente au jury,
- On a testé la fonction exigée sur de petits exemples et sur un exemple de taille réelle (venant du texte).

2.6.2 Défauts

- Par contre, on est un peu pauvre en visualisation et explication,
- Et on a implémenté aucun autre développement. A suivre, j'en ajouterai quand je peux.

  Bien-sûr, ce petit notebook ne se prétend pas être une solution optimale, ni exhaustive.

  Vous auriez pu choisir de modéliser le problème avec une autre approche, mais je ne vois pas ce qu'on aurait pu faire différement.

  C'est tout pour aujourd'hui les amis, allez voir ici pour d'autres corrections, et que la force soit avec vous !