Algorithmique et complexité M1

Université de Bourgogne, UFR Sciences et Techniques, Département I.E.M.

Projet « Algorithmique et complexité »

Groupe
Boris DINH

Sujet n°8 : Calcul de π

2022-2023

Table des matières

1.1 Introduction

"A la poursuite de PI" de Christoph Haenel et [Jörg Arndt, 2006] est un ouvrage consacré au nombre Pi, une constante mathématique qui fascine les mathématiciens et les scientifiques depuis longtemps. On nous présente 9 algorithmes différents pour calculer les décimales de Pi. Les auteurs décrivent chaque algorithme en détail avec des explications claires et accessibles à tous.
Le projet consiste à découvrir les algorithmes du livre et en implémenter 3 pour approximer π de plusieurs façons et plus ou moins rapidement. On va ensuite expliquer ces algorithmes et en faire l'analyse rapide de leur complexité.

1.2 Algorithmes utilisés

1.2.1 Algorithme MAG de Gauss

L'algorithme MAG de gauss est un algorithme se reposant sur le MAG : la moyenne arithméco-géométrique.
A chaque itération, on fait la moyenne géométrique et la moyenne arithmétique puis on compare leur chiffre après la virgule. C'est-à-dire que deux suites où chaque itération est une moyenne se rapprochent de plus en plus pour donner pratiquement le même résultat. Ce qui peut nous permettre de faire une approximation de π avec cette algorithme et cette formule venant de Gauss :

$$\pi = \frac{2MAG^2(1, \frac{1}{\sqrt{2}})}{\frac{1}{2} - \sum_{k=1}^{\infty}(2^k c_k^2)} \tag{1.1}$$

Tout d'abord on initialise 5 variables a, b, s, c et t qui sont nécessaires à l'algorithme :
Le a correspond à la moyenne arithmétique et b à la moyenne géométrique ce qui forment le MAG dans la formule que l'on met au carré.
Le terme c est obtenu par la mise en IJuivre de l'algorithme MAG, à partir des coefficients a et b calculés lors des étapes successives : selon le livre on appelle c la demi-différence de a et b ce qui donne :

$$c_{k+1} = \frac{a_k - b_k}{2} \tag{1.2}$$

Si on prend l'expression c_{k+1} en fonction de a_{k+1} et a_k

$$c_{k+1} = a_{k+1}^2 - b_{k+1}^2 = (a_{k+1} - a_k)^2 \tag{1.3}$$

La variable auxiliaire appelée t est nécessaire car on va réutiliser plusieurs fois a dans les calculs après avoir été remplacé par a_{k+1}.
Le s correspond au dénominateur dans la formule, c'est-à-dire qu'on initialise s à $\frac{1}{2}$ puis on soustrait par la somme allant de k à n-1 à chaque itération de k.

Et pour finir après la boucle, on fait un seul calcul de pi et on le retourne.
Voici un pseudo-code :

```
fonction gauss(n)
        a = 1.0
        b = 1.0 / sqrt(2.0)
        s = 0.5
        c = 0.0
        t = 0.0
        pour i allant de 0 a n-1 faire
                t = a
                a = (a + b) / 2.0
                b = sqrt(t * b)
                c = (a - t) ^ 2.0
                s = s - ((2.0 ^ (i+1)) * c)
        fin pour
        pi = ((a + b) ^ 2.0) / (2.0 * s)
        retourner pi
fin fonction
```

Et voici un algorigramme :

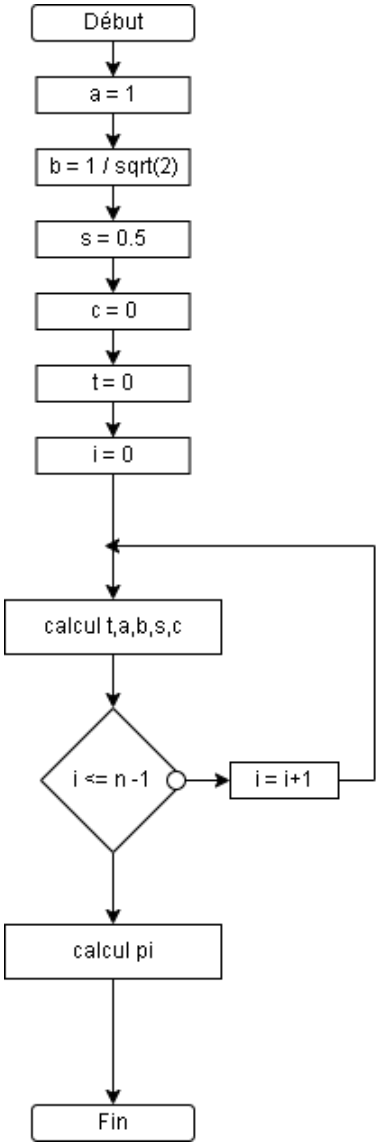


FIGURE 1.1 – Algorigramme de l'algorithme MAG de Gauss

1.2.2 Algorithme quartique de Borwein

L'algorithme quartique de Borwein est un dérivé de l'algorithme de Gauss, Brent et Salamin et qui a été utilisé depuis sa publication dans tous les records du monde de calcul de décimales de π.
L'algorithme utilise une formule qui converge vers la valeur de $\frac{1}{\pi}$ avec une vitesse quartique c'est-à-dire que le nombre de chiffre exacts après la décimale double à chaque itération.
Contrairement à l'algorithme de Gauss, on n'utilise seulement que 2 variables et pour obtenir la valeur de π il suffit de faire l'inverse car a converge vers $\frac{1}{\pi}$.
En voici son pseudo-code :

```
fonction borwein(n)
	y = sqrt(2.0) - 1.0
	a = 6.0 - 4.0 * sqrt(2.0)
	pour i de 0 a n-1 faire
		y = (1.0 - (1.0 - y^4.0)^(1.0 / 4.0)) / (1.0 + (1.0 - y^4.0)^(1.0 / 4.0))
		a = a * (1.0 + y)^4.0 - 2.0^((2.0 * i) + 3.0) * y * (1.0 + y + y^2.0)
	fin pour
	pi = 1.0 / a
	retourner pi
fin fonction
```

Ainsi que son algorigramme :

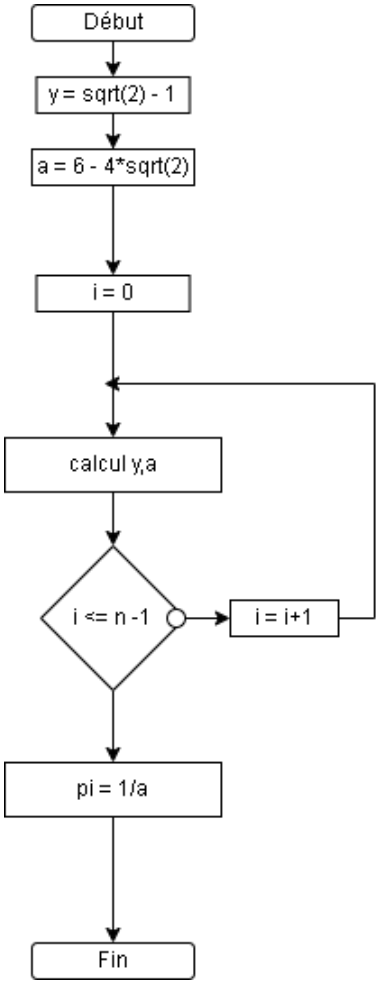


FIGURE 1.2 – Algorigramme de l'algorithme des Borwein

1.2.3 Formule de John Machin

Ici pour la formule de John Machin, il utilise une fonction arctangente afin d'avoir un résultat de $\frac{\pi}{4} = \arctan 1$.
En effet en partant de ce résultat, grâce à James Gregory qui a trouvé une façon d'exprimer la fonction arc tangente grâce à une série, il trouva comme résultat : $\arctan x = \int_0^x \frac{1}{1+t^2} dt$ grâce à cela on en déduit donc que $\arctan x = x - \frac{x^3}{5} + \frac{x^5}{5} - \frac{x^7}{7} + ...$
En remplaçant rapidement tout cela, cela nous donne la formule :
$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + ... = \sum_{n=0}^{\infty} (-1)^n \frac{1}{2n+1}$
Cependant d'après le livre, cette série ne convient pas au calcul numérique car son terme ne décroît vers 0 que très lentement car on a une alternance d'addition et de soustraction dans la somme. Même si on calcule 2 milliards de termes, on obtiendrait seulement 9 décimales exactes de π.
Il existe des formules beaucoup plus optimisés pour le calcul informatique en décomposant la formule en plusieurs arc. Mais cette formule a été simple à implémenter contrairement aux séries utilisant la suite de Fibonacci qui est moins simple.
L'algorithme commence simplement par un initialisation de pi à 0 puis on entre dans une boucle allant de 0 à n-1 ou bien de 1 à n, cela revient à la même chose. Vu qu'on fait une somme il suffit juste d'additionner à chaque itération la même expression en fonction de i.
Et comme la série converge vers $\frac{\pi}{4}$, il suffit donc de multiplier par 4 à la fin pour obtenir une approximation de π.
Voici un pseudo-code :

```
fonction jm(n):
        pi = 0
        pour i allant de 0 a n-1:
                pi = pi + (-1)^i / (2 * i + 1)
        fin pour
        pi = pi * 4
        retourner pi
fin fonction
```

Et voici un algorigramme :

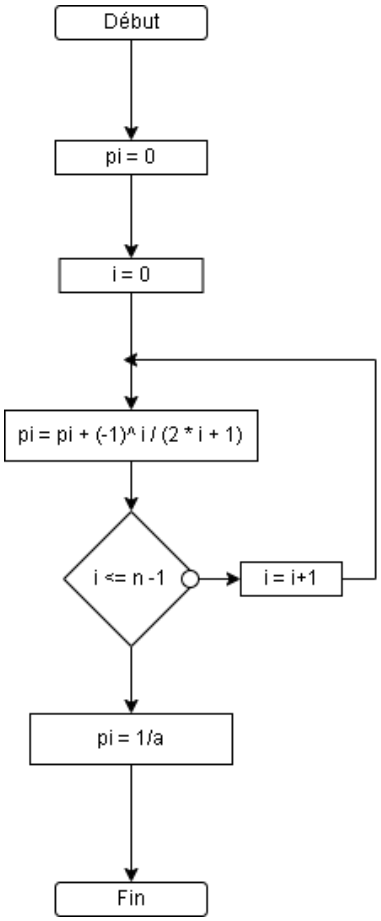


FIGURE 1.3 – Algorigramme de l'algorithme se basant sur la formule de John Machin

1.3 Analyse rapide de la complexité

1.3.1 Complexité de l'algorithme MAG de Gauss

Commençons pour l'analyse de la complexité de l'algorithme MAG de Gauss :

```
fonction gauss(n)
	a = 1.0
	b = 1.0 / sqrt(2.0)
	s = 0.5
	c = 0.0
	t = 0.0
	pour i allant de 0 a n-1 faire
		t = a
		a = (a + b) / 2.0
		b = sqrt(t * b)
		c = (a - t) ^ 2.0
		s = s - ((2.0 ^ (i+1)) * c)
	fin pour
	pi = ((a + b) ^ 2.0) / (2.0 * s)
	retourner pi
fin fonction
```

Si on prend que la complexité temporelle, on a 5 affectations plus une division ce qui donne 6 unités avant la boucle for, la complexité des opérations sont supposés être constantes donc on a bien 6 unités. Pour continuer avec la boucle for, on fait n itérations et à l'intérieur de cette boucle for on a que 5 affectations avec 10 opérations de calcul ce qui nous donne O(15n + 6) et enfin une affectation a pi donc au final on a une complexité de : O(15n+10) = O(n)
Ce qui revient à dire que la complexité est de O(n)

1.3.2 Complexité de l'algorithme quartique de Borwein

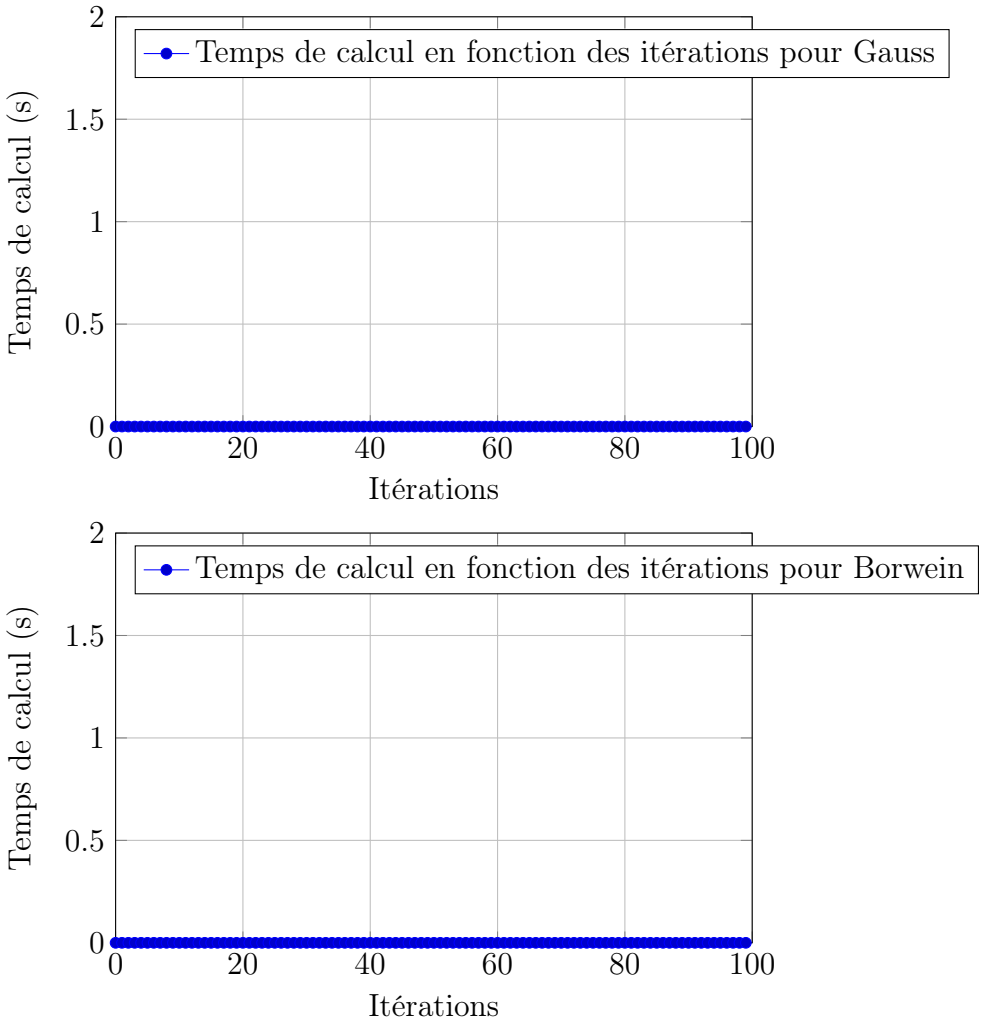
Pour continuer, on va analyser la complexité de l'algorithme quartique de Borwein :

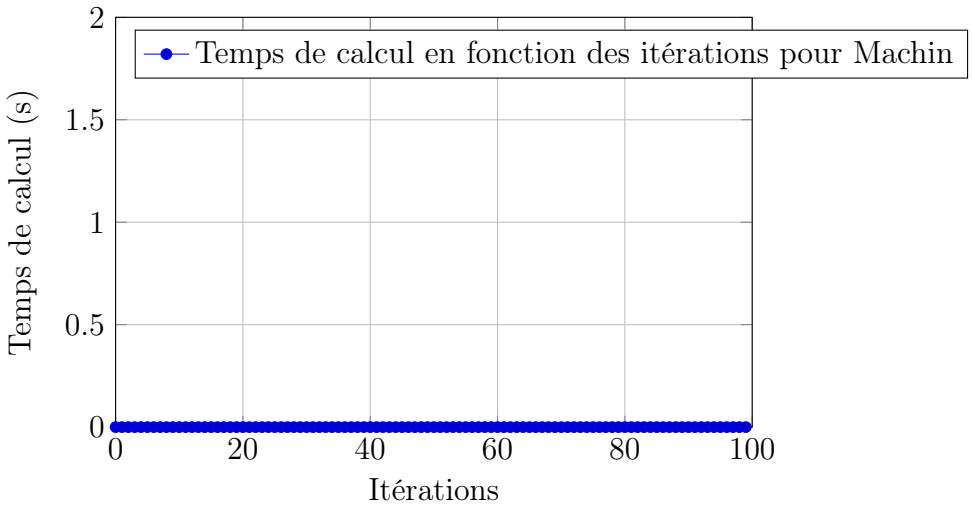
```
	fonction borwein(n)
	y = sqrt(2.0) - 1.0
	a = 6.0 - 4.0 * sqrt(2.0)
	pour i de 0 a n-1 faire
	y = (1.0 - (1.0 - y^4.0)^(1.0 / 4.0)) / (1.0 + (1.0 - y^4.0)^(1.0 / 4.0))
	a = a * (1.0 + y)^4.0 - 2.0^((2.0 * i) + 3.0) * y * (1.0 + y + y^2.0)
	fin pour
	pi = 1.0 / a
	retourner pi
	fin fonction
```

1.3.3 Complexité de la formule de John Machin

1.4 Résultats

Pour les résultats, la consigne est de dessiner une courbe de temps/itérations mais le problème pour les algorithmes que j'ai implémenté, le temps de calcul est tellement rapide que le temps est très proche de 0 s pour chaque itération, je suppose que le temps de calcul grandit en fonction du nombre d'itération mais je n'ai pas de résultats concrets car comme je l'ai expliqué dans le fichier README.txt fourni avec le projet, je ne pouvais utiliser le module Num pour travailler avec des grands nombres.
En effet, on peut supposer que le temps s'accroît au bout de sûrement d'un million d'itération car à chaque fois on recalcule π avec les itérations précédentes gardées en mémoire, du coup les nombres deviennent plus grands et plus on fait de calcul avec des nombres grands, plus cela prend du temps logiquement.
Par acquis de conscience, voici les résultats des fichiers csv :





1.5 Conclusion

Pour conclure,
Au niveau personnel, ce projet était enrichissant au niveau mathématiques, j'en ai appris plus sur les mathématiques et surtout sur les calculs de π où je n'avais aucune idée de comment cela marchait. Les formules et algorithmes étaient très bien pensés malgré leur divergence de rapidité ou simplicité.

1.6 Annexes

Listing 1.1 – Algorithme MAG de Gauss (sources/gauss.ml)

```
(*ALGO*)
(*Algorithme MAG de gauss*)
(*MAG = moyenne arithméco-géométrique*)
(*3 itérations -> 19 décimales exactes*)

let gauss_nonum n =
  let a = ref 1.0 in
  let b = ref (1.0 /. sqrt 2.0) in
  let s = ref 0.5 in
  let c = ref 0.0 in
  let t = ref 0.0 in
  for i = 0 to n - 1 do
    t := !a;
    a := (!a +. !b) /. 2.0;
    b := sqrt (!t *. !b);
    c := (!a -. !t) ** 2.0;
    s := !s -. ((2.0 ** float_of_int (i+1)) *. !c)
  done;

  let pi = ((!a +. !b) ** 2.0) /. (2.0 *. !s) in
  print_string "Gauss : ";
  Printf.printf "%.30f\n" pi;
  print_string "\n";
;;
gauss_nonum 3;;
```

Listing 1.2 – Algorithme quartique de Borwein (sources/borwein.ml)

```
(*algorithme quartique de borwein*)

let borwein n =
  let y = ref ((sqrt 2.0) -. 1.0) in
  let a = ref (6.0 -. 4.0 *. (sqrt 2.0)) in
  for i = 0 to n - 1 do
    y := (1.0 -. (1.0 -. (!y ** 4.0)) ** (1.0 /. 4.0)) /. (1.0 +. (1.0 -. (!y ** 4.0)) **
        (1.0 /. 4.0));
    a := !a *. ((1.0 +. !y) ** 4.0) -. ((2.0 ** ((2.0 *. float_of_int i) +. 3.0)) *. !y *.
        (1.0 +. !y +. !y ** 2.0))
  done;
  let pi = 1.0 /. !a in
  print_string "Borwein quartique : ";
  Printf.printf "%.30f\n" pi;
  print_string "\n";
;;


borwein 100;;
```

Listing 1.3 – Formule de John Machin (sources/machin.ml)

```
(*Formule de John Machin*)

let jm n =
    let pi = ref 0.0 in
    for i = 0 to n - 1 do
      pi := !pi +. ((-1.0) ** float_of_int i) *. (1.0/.((2.0 *. float_of_int i) +. 1.0))
    done;
    pi := !pi *. 4.0;
    print_string "John Machin : ";
    Printf.printf "%.30f\n" !pi;
    print_string "\n";
;;

jm 1000000;;
```

Bibliographie

[Jörg Arndt, 2006] Jörg Arndt, C. H. (2006). *A la poursuite de PI*. Vuibert.