Rapport de stage

MIM1

1er juin - 9 juillet 1999

CALCUL RAPIDE ET EXACT DE FONCTIONS ÉLÉMENTAIRES EN PRÉCISION ARBITRAIRE PAR LA MOYENNE ARITHMÉTICO-GÉOMETRIQUE

stage effectué par Sylvie Boldo
sous la direction de Paul Zimmermann

Remerciements

Je tiens à exprimer mes remerciements au LORIA qui a accepté de m'accueillir dans ses murs.

Ma gratitude va aussi à toute l'équipe PolKA et au personnel administratif pour son accueil chaleureux.

Que tous ceux qui ont participé à MPFR trouvent ici mes remerciements pour avoir répondu sans défaillance à mes nombreuses sollicitations.

Merci à Jean-Michel Muller pour m'avoir faite découvrir l'arithmétique des ordinateurs.

Un grand merci enfin à Paul Zimmermann pour avoir proposé ce passionnant sujet, pour avoir été disponible et attentif à mes problèmes et pour avoir relu ce rapport.

Chapitre 1

Introduction

1.1 Les problèmes de précision et d'exactitude

Les méthodes et les instruments de mesure ont évolué rapidement jusqu'à des précisions qui paraissaient jusque-là idylliques. Ces valeurs ont ensuite souvent besoin d'un traitement numérique pour fournir le résultat voulu.

Or, les bibliothèques mathématiques contiennent des formats fixes pour stocker les réels (simple ou double précision...) qui sont parfois trop limités pour certaines applications.

Il est donc logique qu'apparaissent des bibliothèques permettant le calcul en précision arbitraire.

Une autre réalité des bibliothèques mathématiques est leur non-exactitude : le dernier bit, voire l'avant-dernier est parfois faux pour certaines fonctions. Cela peut sembler négligeable, mais il est souvent malconnu et même méconnu que les bibliothèques usuelles ont des problèmes d'exactitude. Ainsi, lors de la guerre du Golfe, un missile Patriot a échoué dans son interception d'un missile irakien à cause d'une accumulation d'erreurs de calcul.

L'idée toute naturelle est donc de créer une bibliothèque de fonctions mathématiques permettant de calculer en précision arbitraire et de façon exacte. La bibliothèque MPFR répond à cette attente.

Mon stage avait pour but de contribuer à cette bibliothèque en y ajoutant la moyenne arithmético-géométrique et le logarithme, en vérifiant les critères d'exactitude et en tenant compte également d'un critère de rapidité.

1.2 Les intérêts du stage

Il y a plusieurs aspects de ce stage qui m'ont séduite et m'ont poussée à le choisir.

- un mélange de théorie et de pratique : ce n'est ni uniquement de l'informatique théorique, ni seulement de la programmation : c'est un calcul d'erreur théorique qui permet l'écriture d'un algorithme fiable qui est ensuite implanté et testé.
- la découverte d'une discipline sur laquelle nous n'avons pas d'enseignement en licence ou en maîtrise : l'arithmétique des ordinateurs ;
- le fait de ne pas rester à Lyon qui permet de découvrir un autre laboratoire de recherche en informatique ;
- la participation, même minime, à un projet de "grande envergure" : la construction de la bibliothèque MPFR ;
- un travail en équipe où les résultats et performances de chacun dépendent du travail de tous.

Chapitre 2

Les problèmes du calcul exact

2.1 Représentation des flottants

Pour l'utilisateur, la représentation usuelle des flottants est une représentation décimale mais la représentation et le stockage interne sont le plus souvent en binaire : un flottant est une suite de bits dont un bit de signe, des bits pour la mantisse et d'autres pour l'exposant ou certains cas particuliers (Nan ...) :

$$\underbrace{+}_{\text{signe}}\ \underbrace{1.001101001}_{\text{mantisse}}\quad \mathrm{E}\quad \underbrace{2}_{\text{exposant}}$$
$$= 2^2 + 2^{-1} + 2^{-2} + 2^{-4} + 2^{-7} = 4.7578125$$

Par exemple, les *doubles* du langage C possèdent 53 bits de mantisse et un exposant appartenant à [-1022 ;1023].

$ulp(x)$ est la distance entre les deux nombres machine les plus proches de x.

2.2 Arrondis

La mémoire allouée au stockage de ces flottants est bien entendue finie, on ne peut donc représenter qu'un nombre fini de rationnels, appelés "nombres machine". Pour décrire les autres réels, il faut donc les **arrondir**.

La norme IEEE-754 conseille les 4 modes d'arrondi suivants :
- arrondi vers $-\infty$: $\nabla(x)$ est le plus grand nombre machine inférieur ou égal à x ;
- arrondi vers $+\infty$: $\Delta(x)$ est le plus petit nombre machine supérieur ou égal à x ;

- arrondi vers 0 : $\mathcal{Z}(x)$ vaut $\nabla(x)$ si $x \geq 0$ et $\Delta(x)$ sinon ;
- arrondi au plus proche : $\mathcal{N}(x)$ est le nombre machine le plus proche de x. Si 2 nombres machine sont à égale distance de x, c'est celui qui est pair (dont la mantisse finit par un 0) qui est choisi.

L'arrondi exact est celui préconisé par la norme IEEE-754 : le résultat obtenu doit être le même que si l'on avait calculé à chaque étape avec une précision infinie et que l'on avait ensuite arrondi le résultat final.

2.3 Dilemme du fabricant de tables

Cet arrondi est effectivement ce que l'on peut espérer de mieux, mais il pose un problème simple : les étapes internes du calcul d'une fonction mathématique donnée ne peuvent être éxécutées avec une infinité de chiffres. Il faut donc trouver un moyen de contourner ce problème puis de justifier que la précision avec laquelle on a travaillé est suffisante pour démontrer le respect de la norme IEEE.

On a alors envie de dire qu'il suffit de calculer avec quelques chiffres de plus, mais calculer avec p chiffres n'implique pas que l'on pourra en déduire un résultat exact à $p-1$ chiffres, ou même à $p-2$ ou $p-5\ldots$

En effet on peut avoir un résultat avec une erreur de 2^{1-p-q} de la forme : $\underbrace{+1.b_1\ldots\ldots\ldots b_n0}_{q \text{ bits}}\underbrace{11\ldots\ldots111}_{p \text{ bits}} = 1.b_1\ldots b_n01^p$ où 1^p représente p fois "1".

Supposons que l'on doive arrondir ce résultat vers zéro à la précision q. Cela n'est pas possible. En effet, on sait seulement que le résultat exact est compris entre $1.b_1\ldots b_n01^{p-1}\ 0$, que l'on doit arrondir à $1.b_1\ldots b_n0$ et $1.b_1\ldots b_n10^p$, que l'on doit arrondir à $1.b_1\ldots b_n1$.

Il est impossible, pour les fonctions non algébriques, de prévoir à l'avance le nombre minimal de bits significatifs dont on aura besoin et celui-ci peut être de l'ordre de deux fois la précision voulue. Ce phénomène est appelé dilemme du fabricant de tables [4].

Il existe néanmoins des majorations du nombre de chiffres maximal dont on peut avoir besoin, mais elles sont pour l'instant trop grossières : il ne sert à rien de savoir qu'on est sûr de pouvoir arrondir si l'on calcule avec quatre milliards de chiffres pour une précision de 53 bits.

Chapitre 3

Une solution : la bibliothèque MPFR

3.1 Qu'est-ce que MPFR ?

Le sigle MPFR signifie `Multiple Precision Floating-point Reliable library`.

GNU MPFR est une bibliothèque de fonctions écrites dans le langage C, basée sur GNU MP [5].

Elle est développée au LORIA à Nancy dans le cadre de l'équipe PolKA sous la direction de Paul Zimmermann.

Elle sera gratuite et accessible à tous.

3.2 L'objectif de MPFR

Le but de cette bibliothèque est de fournir un certain nombre de primitives de calcul sur les flottants permettant le calcul exact en précision arbitraire.

Ceci signifie que si l'on fournit

- des opérandes de précisions quelconques,
- la précision voulue pour le résultat,
- l'arrondi souhaité,

on peut programmer de façon simple un calcul et obtenir le résultat à la précision demandée avec la garantie de l'exactitude de tous les bits.

3.3 Les avantages de MPFR

D'une part, MPFR permet le calcul en précision quelconque : l'utilisateur peut imposer la précision qu'il souhaite, obtenir un résultat composé de 50000 bits s'il le désire.

D'autre part, MPFR est conforme à la norme IEEE-754 : le résultat obtenu est le même que si l'on avait travaillé avec une infinité de chiffres et arrondi ensuite.

La garantie de l'exactitude de tous les bits fournis est l'idée novatrice à la base du projet MPFR.

3.4 Le contenu de MPFR

À mon arrivée au LORIA, MPFR contenait déjà les quatre fonctions élémentaires (addition, soustraction, multiplication, division) et la racine carrée. À mon départ, elle contient en plus la moyenne arithmético-géométrique, le logarithme et l'exponentielle, une grande partie des autres fonctions ayant été notablement améliorées.

MPFR contient aussi un grand nombre de fonctions diverses d'initialisation, d'arrondi, de conversion, de comparaison, de calcul avec des entiers machine (multiplication par un entier machine, division par une puissance de 2…), d'affichage en base 2 ou 10 ou autre…

À terme, MPFR contiendra toutes les fonctions mathématiques usuelles : sinus, cosinus, arctan, tanh, argsinh…et même la fonction zêta.

Chapitre 4

Comment calculer de façon exacte la moyenne arithmético-géométrique

4.1 Définitions

Le log désignera toujours le logarithme en base 2 et ln le logarithme népérien en base e.

On supposera vouloir calculer la moyenne arithmético-géométrique de a et b, notée $AG(a,b)$, avec $b > a > 0$, qui est la limite commune des deux suites (u_n) et (v_n) définies ci-dessous.

L'ensemble des calculs se font avec une précision p, c'est-à-dire pour un calcul sur x, l'erreur δx est telle que $|\frac{\delta x}{x}| \leq 2^{1-p}$.

Les suites exactes (u_n) et (v_n) sont définies par :

$$u_0 = a$$

$$v_0 = b$$

$$u_{n+1} = \sqrt{u_n\ v_n}$$

$$v_{n+1} = \frac{u_n + v_n}{2}$$

Quelques résultats simples sur ces 2 suites :

- (v_n) est décroissante et (u_n) est croissante ;
- pour tout n, $u_n \leq AG(u_0, v_0) \leq v_n$;
- $v_n - u_n \leq \frac{v_0 - u_0}{2^n}$.

Le dernier point est loin d'être une majoration optimale, mais il sera utilisé par la suite. On le démontre par récurrence :

$$\begin{aligned}
v_{n+1} - u_{n+1} &= \frac{1}{2}\,(u_n + v_n - 2\,\sqrt{u_n\,v_n}) \\
&\leq \frac{1}{2}\left(2\,u_n + \frac{v_0 - u_0}{2^n} - 2\,u_n\,\sqrt{1 + \frac{v_0 - u_0}{u_n\,2^n}}\right) \\
&\leq \frac{1}{2}\,(2\,u_n + \frac{v_0 - u_0}{2^n} - 2\,u_n) \\
&= \frac{v_0 - u_0}{2^{n+1}}
\end{aligned}$$

Si tous les calculs étaient exacts, c'est-à-dire si l'on calculait réellement (u_n) et (v_n) à chaque itération, l'algorithme consisterait simplement à calculer ces deux suites tant que $\frac{v_n - u_n}{u_n} > O(2^{-p})$ selon l'arrondi demandé ; u_n ou v_n arrondi serait alors une approximation du résultat à la précision p. Mais en fait, à chaque calcul, on effectue une approximation du résultat exact de ce calcul. On peut donc avoir une dérive des calculs dont il faut tenir compte.

On va approcher (u_n) et (v_n) au plus proche. Soit $\mathcal{N}$ l'arrondi au plus proche à la précision relative p.

On pose donc en tant que valeurs approchées des suites précédentes :

$$\tilde{u}_0 = \mathcal{N}(a)$$

$$\tilde{v}_0 = \mathcal{N}(b)$$

$$\tilde{u}_{n+1} = \mathcal{N}\left(\sqrt{\mathcal{N}(\tilde{u}_n\ \tilde{v}_n)}\right)$$

$$\tilde{v}_{n+1} = \frac{\mathcal{N}(\tilde{u}_n + \tilde{v}_n)}{2}$$

En effet, la division par 2 est exacte. Le calcul de $\tilde{v}_{n+1}$ nécessite donc une unique approximation.

4.2 Premier théorème sur les suites approchées

Soit $\delta = 2^{-p}$.

Propriété 1 *On a par définition de* $\mathcal{N}$, $\dfrac{|x - \mathcal{N}(x)|}{x} \leq \delta$.
D'où $x - \mathcal{N}(x) \leq \delta\ x$ *et* $\mathcal{N}(x) - x \leq \delta\ x$. *Donc, on a*

$$x\ (1 - \delta) \leq \mathcal{N}(x) \leq x\ (1 + \delta).$$

Théorème 1

$$u_n\ (1-\delta)^{\frac{3}{2}n+1} \leq \tilde{u}_n \leq u_n\ (1+\delta)^{\frac{3}{2}n+1}$$

$$v_n\ (1-\delta)^{\frac{3}{2}n+1} \leq \tilde{v}_n \leq v_n\ (1+\delta)^{\frac{3}{2}n+1}$$

La démonstration se fait bien entendu par récurrence.

On a $\tilde{u}_0 = \mathcal{N}(a)$ donc $u_0\ (1-\delta) \leq \tilde{u}_0 \leq u_0\ (1+\delta)$ par la propriété 1 et de même $\tilde{v}_0 = \mathcal{N}(b)$ donc $v_0\ (1-\delta) \leq \tilde{v}_0 \leq v_0\ (1+\delta)$.

D'après la propriété 1, on a aussi :

$$\begin{array}{rcccl} \tilde{u}_n\ \tilde{v}_n\ (1-\delta) & \leq & \mathcal{N}(\tilde{u}_n\ \tilde{v}_n) & \leq & \tilde{u}_n\ \tilde{v}_n\ (1+\delta) \\ \sqrt{\tilde{u}_n\ \tilde{v}_n\ (1-\delta)} & \leq & \sqrt{\mathcal{N}(\tilde{u}_n\ \tilde{v}_n)} & \leq & \sqrt{\tilde{u}_n\ \tilde{v}_n\ (1+\delta)} \\ \sqrt{\tilde{u}_n\ \tilde{v}_n\ (1-\delta)}\ (1-\delta) & \leq & \tilde{u}_{n+1} & \leq & \sqrt{\tilde{u}_n\ \tilde{v}_n\ (1+\delta)}\ (1+\delta) \end{array}$$

Donc, grâce à l'hypothèse de récurrence :

$$\sqrt{u_n\ v_n\ \left((1-\delta)^{\frac{3}{2}n+1}\right)^2\ (1-\delta)}\ (1-\delta) \quad \leq \tilde{u}_{n+1} \leq \quad \sqrt{\tilde{u}_n\ \tilde{v}_n\ (1+\delta)}\ (1+\delta)$$

$$\begin{array}{rcl} \sqrt{u_n\ v_n}\ (1-\delta)^{\frac{3}{2}n+\frac{3}{2}+1} & \leq \tilde{u}_{n+1} \leq & \sqrt{\tilde{u}_n\ \tilde{v}_n\ \left((1+\delta)^{\frac{3}{2}n+1}\right)^2\ (1+\delta)}\ (1+\delta) \\ u_{n+1}\ (1-\delta)^{\frac{3}{2}(n+1)+1} & \leq \tilde{u}_{n+1} \leq & u_{n+1}\ (1+\delta)^{\frac{3}{2}(n+1)+1} \end{array}$$

D'autre part , la propriété 1 donne :

$$(\tilde{u}_n + \tilde{v}_n)\ (1-\delta) \leq \mathcal{N}(\tilde{u}_n + \tilde{v}_n) \leq (\tilde{u}_n + \tilde{v}_n)\ (1+\delta)$$

Donc,

$$\begin{array}{rcl} \dfrac{\tilde{u}_n + \tilde{v}_n}{2}\ (1-\delta) & \leq \tilde{v}_{n+1} \leq & \dfrac{\tilde{u}_n + \tilde{v}_n}{2}\ (1+\delta) \\ \dfrac{u_n + v_n}{2}\ (1-\delta)^{\frac{3}{2}n+2} & \leq \tilde{v}_{n+1} \leq & \dfrac{u_n + v_n}{2}\ (1+\delta)^{\frac{3}{2}n+2} \\ v_{n+1}\ (1-\delta)^{\frac{3}{2}(n+1)+1} & \leq \tilde{v}_{n+1} \leq & v_{n+1}\ (1+\delta)^{\frac{3}{2}(n+1)+1} \end{array}$$

Le lemme est donc entièrement démontré.

On a donc un majorant de l'erreur commise en calculant u_n et v_n de façon approchée, c'est-à-dire de la dérive du résultat due aux approximations successives.

On a montré que $(\tilde{u}_n)$ et $(\tilde{v}_n)$ sont proches de (u_n) et (v_n). Mais on perd un certain nombre de propriétés des suites exactes : on ne sait absolument pas comment sont rangés u_n, $\tilde{u}_n$, $\mathrm{AG}(u_0,v_0)$, $\tilde{v}_n$ et v_n.

C'est-à-dire que bien que l'on sache que $u_n \leq AG(u_0, v_0) \leq v_n$, il est possible que $u_n \leq AG(u_0, v_0) \leq \tilde{u}_n \leq v_n \leq \tilde{v}_n$, voire même que $\tilde{v}_n \leq \tilde{u}_n$, ce qui est tout à fait contraire à l'intuition.

Par exemple, si l'on a une précision de quatre bits, et que $u_0 = 9$ et $v_0 = 12$, soit $u_0 = 1001$ et $v_0 = 1100$, on a alors :

$u_1 = \mathcal{N}\left(\sqrt{\mathcal{N}(u_0\ v_0)}\ \right) = \mathcal{N}\left(\sqrt{\mathcal{N}(1001 * 1100)}\ \right) = \mathcal{N}\left(\sqrt{\mathcal{N}(1101100)}\ \right)$

$= \mathcal{N}\left(\sqrt{1110\mathrm{E}3}\ \right) = \mathcal{N}(1010.1001) = 1011.$

Soit $u_1 = 11$.

$v_1 = \mathcal{N}(u_0 + v_0)\ 2^{-1} = \mathcal{N}(10101)\ 2^{-1} = 1010\mathrm{E}1\ 2^{-1} = 1010.$

Soit $v_1 = 10$.

On a bien interversion de l'ordre entre u_n et v_n.

4.3 Convergence des $(\tilde{u}_n)$ et $(\tilde{v}_n)$

Soit $\epsilon_n = \frac{v_n - u_n}{u_n}$.

Un calcul montre que $f(x) = \sqrt{1+x} - 1 - \frac{x}{2} + \frac{x^2}{8}$ est positive dès que $x \geq 0$. Or ϵ_n est toujours positif donc $\forall n, \sqrt{1+\epsilon_n}\ \geq 1 + \frac{\epsilon_n}{2} - \frac{{\epsilon_n}^2}{8}$.

Comme (ϵ_n) tend vers 0, soit ϵ_{n_0} tel que $\epsilon_{n_0} \leq 2$.

Il suffit que :

$$\begin{aligned} \epsilon_{n_0} = \frac{v_{n_0} - u_{n_0}}{u_{n_0}} &\leq 2 \\ \frac{v_0 - u_0}{2^{n_0}\ u_{n_0}} &\leq 2 \\ \frac{v_0 - u_0}{2\ u_0} &\leq 2^{n_0} \\ n_0 &\geq \log\left(\frac{v_0 - u_0}{2\ u_0}\right) \end{aligned}$$

On pose $n_0 = \lceil \log\left(\frac{v_0 - u_0}{2\ u_0}\right) \rceil$

On a donc :

$$\begin{aligned} \epsilon_{n+1} &= \frac{1}{u_{n+1}}\left(\frac{v_n + u_n}{2} - \sqrt{u_n\ v_n}\right) \\ &= \frac{1}{u_{n+1}}\left(u_n\ \left(1 + \frac{\epsilon_n}{2}\right) - u_n\ \sqrt{1+\epsilon_n}\right) \\ &\leq 1 + \frac{\epsilon_n}{2} - \sqrt{1+\epsilon_n} \leq \frac{{\epsilon_n}^2}{8} \end{aligned}$$

On peut alors montrer par récurrence que, si $n \geq n_0$:

$$\epsilon_n \leq (\epsilon_{n_0})^{2^{n-n_0}} \; \frac{8}{8^{2^{n-n_0}}}$$

Comme $\epsilon_{n_0} \leq 2$, il vient :

$$\begin{aligned} \epsilon_n &\leq 2^{2^{n-n_0}} \; \frac{8}{8^{2^{n-n_0}}} \\ &\leq 8 \cdot 4^{-2^{n-n_0}} \end{aligned}$$

Cette évaluation de la vitesse de convergence de la suite exacte permet le calcul suivant :

$$\begin{aligned} |\tilde{v}_n - AG(u_0, v_0)| &\leq |\tilde{v}_n - v_n| + |v_n - AG(u_0, v_0)| \\ &\leq |\tilde{v}_n - v_n| + |v_n - u_n| \qquad \text{car } AG(u_0, v_0) \in [u_n; v_n] \\ &\leq v_n \; \max((1+\delta)^{\frac{3}{2}n+1} - 1 \; , \; 1 - (1-\delta)^{\frac{3}{2}n+1}) + \epsilon_n \; u_n \end{aligned}$$

Si $f(\delta) = (1+\delta)^{\frac{3}{2}n+1} - 1 - n \; 2^{\frac{3}{2}n+2} \; \delta$ et si $n \geq 1$, la dérivée de f est négative pour $\delta \leq 1$. Or $f(0) = 0$ donc $(1+\delta)^{\frac{3}{2}n+1} - 1 \leq n \; 2^{\frac{3}{2}n+2} \; \delta$.

Si $g(\delta) = 1 - (1-\delta)^{\frac{3}{2}n+1} - n \; 2^{\frac{3}{2}n+2} \; \delta$ et si $n \geq 1$, la dérivée de g est négative pour $\delta \geq 0$. Or $g(0) = 0$ donc $1 - (1-\delta)^{\frac{3}{2}n+1} \leq n \; 2^{\frac{3}{2}n+2} \; \delta$.

Donc pour $n \geq n_0$,

$$\begin{aligned} |\tilde{v}_n - AG(u_0, v_0)| &\leq v_0 \; n \; 2^{\frac{3}{2}n+2} \; \delta + v_0 \; \epsilon_n \\ &\leq v_0 \; n \; 2^{\frac{3}{2}n+2} \; \delta + v_0 \; 8 \; 4^{-2^{n-n_0}} \\ &\leq v_0 \; n \; 2^{\frac{3}{2}n+2} \; 2^{-p} + v_0 \; 8 \; 4^{-2^n \; \frac{u_0}{v_0-u_0}} \\ &\leq v_0 \; \left(n \; 2^{\frac{3}{2}n+2} \; 2^{-p} + 8 \; 4^{-\frac{2^n \; u_0}{v_0-u_0}} \right) \end{aligned}$$

On a alors démontré le

Théorème 2 *Si $n \geq n_0$,*

$$\frac{|\tilde{v}_n - AG(u_0, v_0)|}{v_0} \leq n \; 2^{\frac{3}{2}n+2-p} + 8 \; 4^{-\frac{2^n \; u_0}{v_0-u_0}}$$

On peut considérer la signification des deux termes de droite de l'inégalité :

- $n\ 2^{\frac{3}{2}n+2}\ 2^{-p}$ est un terme qui augmente avec n et diminue avec p. Ce terme correspond à la dérive des calculs : plus on fait d'itérations, plus on fait de calculs, donc plus le résultat est susceptible d'être inexact. Cet effet peut être contrebalancé par la précision : on calcule avec plus de chiffres pour compenser cette perte.

- $8\ 4^{-\frac{2^n\ u_0}{v_0-u_0}}$ diminue avec n et ne dépend pas de p : plus on fait d'itérations, plus $\tilde{u}_n$ et $\tilde{v}_n$ sont proches et proches du résultat, cette convergence étant quadratique. Ce terme dépend aussi des données initiales : si u_0 et v_0 sont éloignés (relativement à la valeur du plus petit : u_0), il faudra plus d'itérations pour rapprocher $\tilde{u}_n$ et $\tilde{v}_n$.

4.4 Précision nécessaire

On aura donc un n optimal quand les deux termes d'erreur sont égaux :

$$\begin{aligned}
n\ 2^{\frac{3}{2}n+2-p} &= 8\ 4^{-\frac{2^n\ u_0}{v_0-u_0}} \\
\log n + \frac{3}{2}n + 2 - p &= 3 - \frac{2^{n+1}\ u_0}{v_0 - u_0} \\
2^n &= \frac{v_0 - u_0}{2\ u_0}\left(p - \log n - \frac{3}{2}n + 1\right) \\
n &= \log\frac{v_0 - u_0}{2\ u_0} + \log(p - \log n - \frac{3}{2}n + 1) \\
n &\leq \log\frac{v_0 - u_0}{2\ u_0} + \log p
\end{aligned}$$

On pose

$$n_{opt} = \log\left(\frac{v_0 - u_0}{2\ u_0}\right) + \log p$$

On a bien $n_{opt} \geq n_0$. De plus, n_{opt} est supérieur au nombre d'itérations optimal.

On a alors :

$$\begin{aligned}
\frac{|\tilde{v}_{n_{opt}} - AG(u_0, v_0)|}{v_0} &\leq 2\ n_{opt}\ 2^{\frac{3}{2}n_{opt}+2-p} \\
&= 8\ \log\left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right)\ 2^{\frac{3}{2}\log\left(\frac{p\ (v_0-u_0)}{2\ u_0}\right)}\ 2^{-p} \\
&= 8\ \log\left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right)\ \left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right)^{\frac{3}{2}}\ 2^{-p}
\end{aligned}$$

Théorème 3

$$\frac{|\tilde{v}_{n_{opt}} - AG(u_0, v_0)|}{v_0} \leq 8 \ \log\left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right)\ \left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right)^{\frac{3}{2}}\ 2^{-p}$$

Si on veut une précision relative q sur le résultat final, il suffit donc :

$$\begin{array}{rcl} 2^{-q} & \geq & 8\ \log\left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right)\ \left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right)^{\frac{3}{2}}\ 2^{-p} \\ -q & \geq & 3 + \log\left(\log\left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right)\right) + \frac{3}{2}\log\left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right) - p \\ p & \geq & q + 3 + \frac{3}{2}\log\left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right) + \log\left(\log\left(\frac{p\ (v_0 - u_0)}{2\ u_0}\right)\right) \end{array}$$

Ce n'est bien sûr qu'une borne inférieure pour p à cause du dilemme du fabricant de tables.

4.5 Complexité

La section suivante s'appuie largement sur les travaux de Richard P. Brent [2].

On pose $M(n)$ la complexité de la multiplication de deux nombres de taille n. On considérera que $n = o(M(n))$ et que $M(\alpha\ n) \approx \alpha\ M(n)$. En effet, le meilleur algorithme connu donne $M(n) = O(n\ \log n\ \log\log n)$.

4.5.1 Complexité de l'inverse de la racine carrée

Pour calculer $a^{-\frac{1}{2}}$, on utilise une méthode de Newton du troisième ordre :

$$x_{i+1} = x_i - \frac{1}{2}\ x_i\ (\epsilon_i - \frac{3}{4}\ \epsilon_i^2) \qquad \text{où } \epsilon_i = a\ x_i^2 - 1$$

Selon [2], pour obtenir une précision n, il suffit, à la dernière itération, de calculer $a\ x_i^2$ à la précision n, ϵ_i^2 à la précision $n/3$ et $x_i\ (\epsilon_i - \frac{3}{4}\ \epsilon_i^2)$ à la précision $2n/3$.

La dernière itération coûte donc $2\ M(n) + M(n/3) + M(2n/3) + O(n)$ soit $M(n)\ (2 + 1/3 + 2/3) = 3\ M(n)$.

Le nombre de chiffres exacts triple à chaque itération : il suffit donc d'obtenir une précision $n/3$ à l'itération précédente, et $n/9$ à la précédente...

On a donc $Q(n) \approx 3\ M(n)\ (1 + 1/3 + 1/9 + \ldots) \approx 9/2\ M(n)$.

Pour obtenir $\sqrt{a}$ pour a non nul, il suffit de calculer $a^{-\frac{1}{2}}$ et de le multiplier par a. Donc $S(n) = 11/2\ M(n)$.

4.5.2 Complexité de la moyenne arithmético-géométrique

Une itération correspond à une multiplication, une racine carrée, une addition et une division par 2. Le coût d'une itération est donc de 6.5 $M(n)$. Le coût des itérations est donc environ de 6.5 $n_{opt}\ M(n)$.

Si on calcule avec $\log n$ bits en plus, la probabilité qu'on ne puisse pas arrondir est d'environ 1/n. De plus, on a $n_{opt}(n) \approx n_{opt}(n + \log n)$ et de même $M(n + \log n) \approx M(n)$.

La complexité est donc d'environ :
$AGM(n) \approx 6.5\ n_{opt}\ M(n)\ (1 + 1/n + 1/n^2 + \ldots) \approx \frac{6.5}{1-1/n}\ n_{opt}\ M(n)$
$AGM(n) \leq 7\ n_{opt}\ M(n) = 7\ \left(\log\left(\frac{v_0-u_0}{2\ u_0}\right) + \log n\right)\ M(n)$

4.5.3 Complexité du logarithme

Pour calculer le logarithme, il faut calculer π et $\log 2$ en précision n, calculer une moyenne arithmético-géométrique et faire une division (les autres opérations sont négligeables). La division elle-même est négligeable devant la moyenne arithmético-géométrique. Donc un calcul a une complexité de $Pi(n) + Log2(n) + AGM(n)$.

Si l'on ne peut pas arrondir à la fin du premier calcul, on rajoute trois bits, la probabilité qu'on ne puisse toujours pas arrondir après ce rajout est d'environ 1/8, elle est d'environ 1/64 si on rajoute encore trois bits...

Ajouter 3 bits de précision ne modifie pas les complexités des calculs de π, $\log 2$ ou de la moyenne arithmético-géométrique.

On a donc :
$Log(n) \approx (Pi(n) + Log2(n) + AGM(n))\ (1 + 1/8 + 1/64 + \ldots)$
$Log(n) \approx 8/7\ (Pi(n) + Log2(n) + AGM(n))$

$$Log(n) \approx 8/7(Pi(n) + Log2(n)) + 8\ \left(\log\left(\frac{v_0 - u_0}{2\ u_0}\right) + \log n\right)\ M(n)$$

En fait, l'algorithme utilisé dans MPFR pour calculer la racine carrée n'est pas celui-ci mais la méthode de Newton usuelle : $x_{i+1} = \frac{1}{2}\ (x_i + \frac{a}{x_i})$. Cela conduit simplement à une constante plus grande dans la complexité de la moyenne arithmético-géométrique.

Voir aussi le chapitre 6 page 18 ainsi que les tableaux C.1 et C.2 pour des informations sur la complexité réelle et la répartition du temps entre les fonctions.

Chapitre 5

Ma contribution à MPFR

5.1 Implantation de la moyenne arithmético-géométrique

Après l'étude théorique, on peut majorer l'erreur du résultat de façon à être sûr de pouvoir arrondir au résultat exact. Soit err, l'erreur maximale sur le résultat, fonction de p, calculée au cours du chapitre précédent :
$err = v_0\ 8\ \log\left(\frac{p\ (v_0-u_0)}{2\ u_0}\right)\ \left(\frac{p\ (v_0-u_0)}{2\ u_0}\right)^{\frac{3}{2}}\ 2^{-p}$.

On considère que "retourner" renvoie le résultat et sort définitivement de la fonction.

L'expression "si on peut arrondir" signifie que l'on regarde si, connaissant l'erreur maximale, on est capable d'arrondir, ou si on est dans un cas où, comme dans le dilemme du fabricant de tables, on ne peut pas arrondir.

L'algorithme finalement implanté est le suivant :

agm(a,b,rnd_mode,q**)**
  q est la précision du résultat
  si a ou b est négatif, renvoyer NaN
  si a ou b est nul, renvoyer 0
  si $a = b$, renvoyer a
  $p = q + 12$
  On calcule désormais en précision p
  $u = a$
  $v = b$
  tant que ($VRAI$)
  calculer err en fonction de p
    tant que $|u - v|$ varie

$tmp = \mathcal{N}\left(\sqrt{\mathcal{N}\,(u * v)}\right)$
$v = \mathcal{N}\left(\frac{u+v}{2}\right)$
$u = tmp$
si on peut arrondir v à la précision q,
suivant l'arrondi *rnd_mode*
et connaissant *err*
renvoyer cet arrondi
sinon
$p = p + 5$

Il reste à montrer que cet algorithme termine. Mais la moyenne arithmético-géométrique entre deux nombres non nuls et non égaux est un nombre transcendant, ce n'est donc pas un rationnel et a fortiori pas un nombre machine.

Il est donc exclus qu'un tel résultat termine son développement en base 2 par une infinité de 0. On finit donc toujours par pouvoir l'arrondir.

5.2 Implantation du logarithme

Il est en effet très facile de déduire le logarithme d'après la moyenne arithmético-géométrique.

On a la formule *approchée* [6] :

pour obtenir p bits du logarithme de x, on a :

$$\ln x \approx \frac{\pi}{2\ \mathrm{AG}(1{,}4/\mathrm{s})} - m \ln 2 + o(\ln x\ 2^{-p})$$

où $s = x\ 2^m > 2^{p/2}$.

L'algorithme de calcul étant fourni par cette formule, il ne reste qu'à évaluer l'erreur du résultat obtenu. On a en fait deux causes d'erreurs distinctes.

D'une part, on a la dérive du calcul puisque l'on fait plusieurs opérations enchaînées.

D'autre part, on perd des bits significatifs lors de la soustraction. Ce dernier point n'existait pas dans la moyenne arithmético-géométrique puisqu'il n'y avait pas de soustraction. Il peut sembler négligeable mais est en fait très important près de 1 : les deux termes peuvent alors être de l'ordre de 10 alors que le résultat est de l'ordre de 10^{-2}.

On peut perdre beaucoup de bits, cette perte doit donc être évaluée : si $x = 1 + \epsilon$, $m = p/2$ donc les termes sont proches de $p \ln 2/2$ et le résultat $\log x \geq \epsilon/2$, le nombre de bits perdus est donc inférieur à $\log\left(\frac{p\ln 2}{|x-1|}\right)$.

5.3 Contribution au déboguage de la bibliothèque

Mes fonctions sont exclusivement basées sur les fonctions de MPFR écrites avant mon arrivée : elles utilisent les fonctions de base dans des boucles. Un unique test du logarithme en précision p teste donc au moins $\log p$ fois la racine carrée, la multiplication, l'addition... Il y a de plus des divisions, des arrondis... Un test du logarithme ou de la moyenne arithmético-géométrique implique donc de nombreux tests d'un grand nombre de fonctions de base de MPFR.

Il était donc logique que mes propres tests de déboguage fournissent parfois des erreurs dans des fonctions autres que les miennes, que je transmettais donc aux personnes concernées qui les corrigeaient avec patience.

Chapitre 6

Tests

Il est intéressant de constater la fréquence et l'importance des erreurs de la bibliothèque mathématique usuelle `libm.a`. Le tableau A.1 montre que ces erreurs sont très rares surtout sur le PC Linux et qu'elles sont en général d'un ulp, sauf sur la vieille station SUN où elles peuvent atteindre 2 ulps.

La rapidité de mpfr_log est tout à fait correcte : d'après le tableau B.1 elle est comparable à la bibliothèque Pari voire nettement plus rapide au delà de 5000 bits de précision et elle est toujours nettement plus rapide que MAPLE.

On peut ensuite considérer la répartition du temps entre les différentes opérations. Le tableau C.1 montre que le nombre d'itérations n augmente bien conformément à $\log p$; on a aussi le terme dépendant de la variable initiale : on peut donc difficilement avoir une augmentation strictement parallèle à $\log p$.

On voit aussi que le temps se partage équitablement entre le calcul de la moyenne arithmético-géométrique, celui de $\log 2$ et celui de π.

En regardant le tableau C.2, on voit que près de 80% du temps est passé dans les fonctions commençant par __mpn qui sont celles de GMP, et donc sur lesquelles notre équipe n'a aucune influence.

On voit aussi que mpfr_agm est appelé en moyenne 1.059 fois : on a besoin de l'appeler une nouvelle fois car on n'a pas pu arrondir dans seulement 6% des cas.

Par contre, mpfr_round_raw est appelée en moyenne plus de 82 fois. Il faut en effet arrondir à chaque opération arithmétique : addition, multiplication, plusieurs fois par racine carrée... et ceci à chaque itération.

Chapitre 7

Perspectives

7.1 Mes réalisations

Au cours de ces six semaines de stage, j'ai écrit et prouvé la validité de l'algorithme de calcul de la moyenne arithmético-géométrique. Puis je l'ai implanté, débogué et testé. J'ai ensuite implanté le calcul du logarithme, plus utile et surtout plus facile à comparer aux calculs de la bibliothèque mathématique `libm.a`.

J'ai donc écrit environ 275 lignes de code pour le calcul proprement dit du logarithme et de la moyenne arithmético-géométrique, et à peu près 450 lignes de code de fonctions de tests, permettant de déboguer, de comparer à la bibliothèque mathématique, ou de tester la rapidité.

À l'heure actuelle, mes fonctions produisent le bon résultat ce qui n'est pas toujours le cas de la bibliothèque mathématique usuelle du C. De plus, ces calculs se font en un temps honorable par rapport aux bibliothèques existantes.

7.2 Les enseignements de ce stage

Ce stage m'a montré les problèmes d'exactitude, qui ne sont pas aussi simples que je le supposais. En particulier, le fait d'enchaîner des opérations nécessite forcément un calcul d'erreur pour garantir un résultat exact, et ceci même si chacune des opérations, prise individuellement, est exacte.

Ce stage m'a aussi fait découvrir un travail de recherche en équipe sur un projet commun. Cela comporte des inconvénients : il m'était impossible de trouver la raison d'une erreur que je venais de découvrir dans une fonction écrite par quelqu'un d'autre. Mais cela inclut aussi de nombreux avantages, notamment la possiblité d'obtenir facilement des réponses à mes questions sur MPFR, la présence continuelle de quelqu'un capable de corriger les bogues que je mettais en évidence et bien sûr la diversité des connaissances et des aptitudes.

7.3 Les perspectives de MPFR

MPFR va continuer à se développer, elle doit encore être améliorée au point de vue de la rapidité et doit aussi s'enrichir de nouvelles fonctions telles que sinus, arctan... Même la fonction zêta est en train d'être écrite.

MPFR devrait avoir une diffusion restreinte d'ici la fin de l'année pour lui permettre d'être bêta-testée.

MPFR sera ensuite diffusée et pourra être utilisée dans le monde entier pour calculer avec exactitude à une précision arbitraire.

Annexe A

Exactitude

Ces tests ont été effectué sur des `doubles` positifs tirés au hasard : on calcule leur logarithme par MPFR (en précision 53 bits) et par `libm.a` et on compare les résultats.

machine	description	OS	max	moyenne	nb de tests
Latour	SUN SPARC5	Solaris	2	0.0026	10000
Craffe	SUN sparc Ultra-4	Solaris	1	0.0019	50000
Mattaincourt	PC Pentium II 450 Mhz	Linux	1	0.00024	100000

FIG. A.1: Erreurs en ulps de `libm.a` sur différentes machines

Annexe B

Rapidité

Tous ces tests ont été effectué sur mattaincourt, un PC Pentium II 450 Mhz sous Linux. La compilation de MPFR a été faite avec gcc sans aucune option d'optimisation.

p	pari : agm	mpfr_agm	mpfr_log	pari : log	maple : log
53	0.11	0.16	0.18	0.15	0.65
100	0.16	0.19	0.26	0.27	0.96
200	0.22	0.31	0.39	0.47	2.01
500	0.63	0.85	1.05	1.24	4.27
1000	1.71	2.11	2.94	3.85	13.1
2000	4.32	6.56	11.4	13.8	28.3
5000	28.5	30.6	45.2	89.9	149
10000	114	105	217	443	561

FIG. B.1: Temps CPU (en ms) mis par diverses bibliothèques mathématiques en fonction de la précision en bits p demandée.

Annexe C

Analyse temporelle

Tous ces tests ont été effectués sur mattaincourt, un PC Pentium II 450 Mhz sous Linux. La compilation de MPFR a été faite avec gcc sans aucune option d'optimisation.

p	n	sqrt(total)	agm	pi	log2	total
10000	23	90	130	80	120	360
50000	27	2260	2810	2070	2800	7920
100000	29	9120	10910	9880	12220	34060
500000	34	294830	322300	360260	320510	1033920
1000000	36	1231670	1319380	1552300	1253020	4245990

FIG. C.1: Répartition des temps CPU (en ms) nécessaires aux diverses opérations ; n est le nombre d'itérations effectivement réalisées lors du calcul d'agm.

% temps CPU	temps	nb appels	nom de la fonction
36.97	1.05		__mpn_addmul_1
17.96	0.51		__mpn_submul_1
8.80	0.25		__mpn_divrem
4.23	0.12	82.505	mpfr_round_raw
4.23	0.12		__mpn_sqrtrem
3.52	0.10	16.960	mpfr_sqrt3
2.82	0.08	16.961	mpfr_add1
2.46	0.07		__mpn_mul_1
2.11	0.06	18.019	mpfr_mul
2.11	0.06		__mpn_divmod_1
1.76	0.05		__mpn_impn_mul_n
1.76	0.05		__mpn_sub_n
1.41	0.04	23.286	mpfr_can_round_raw
1.41	0.04		__mpn_add_n
1.06	0.03		__mpn_impn_sqr_n
1.06	0.03		__mpn_impn_sqr_n_basecase
0.70	0.02	18.405	mpfr_cmp3
0.70	0.02	18.017	mpfr_cmp2
0.35	0.01	125.836	mpfr_round_raw2
0.35	0.01	44.453	mpfr_set4
0.35	0.01	16.960	mpfr_add
0.35	0.01	3.177	mpfr_set_si
0.35	0.01	3.117	mpfr_get_d2
0.35	0.01	2.118	mpfr_div3
0.35	0.01	1.059	mpfr_agm
0.35	0.01	1.059	mpfr_log2
(...)	(...)	(...)	(...)
0.00	0.00	1.6960	mpfr_div_2exp
0.00	0.00	4.192	mpfr_can_round
0.00	0.00	3.117	mpfr_get_d
0.00	0.00	2.118	mpfr_mul_2exp
0.00	0.00	1.059	mpfr_pi
0.00	0.00	1.059	mpfr_sub
0.00	0.00	1.057	mpfr_sub1
0.00	0.00	1	mpfr_log

FIG. C.2: Nombre d'appels aux fonctions et répartition du temps pour un calcul en précision 1000 (le temps CPU est donné en millisecondes).

Bibliographie

[1] Batut, C., Bernardi, D., Cohen, H., and Olivier, M. *User's Guide to PARI-GP*, Jan. 1995. URL `ftp ://megrez.math.u-bordeaux.fr/pub/pari`.

[2] Brent, R. P. Multiple-precision zero-finding methods and the complexity of elementary function evaluations. In *Analytic Computational Complexity*. Academic Press, 1976, pp. 151–176.

[3] Char, B. W., Geddes, K. O., Gonnet, G. H., Leong, B. L., Monagan, M. B., and Watt, S. M. *Maple V : Language Reference Manual*. Springer-Verlag, 1991.

[4] Daumas, M., and Muller, J.-M. *Qualité des calculs sur ordinateur. Vers des arithmétiques plus fiables ?* Masson, 1997.

[5] Granlund, T. *GNU MP : The GNU Multiple Precision Arithmetic Library*, 2.0.2 ed., June 1996.

[6] Muller, J.-M. *Elementary Functions. Algorithms and Implementation.* Birkhauser, 1997. 232 pages.

Table des matières

1 Introduction **1**

1.1 Les problèmes de précision et d'exactitude 1

1.2 Les intérêts du stage 1

2 Les problèmes du calcul exact **3**

2.1 Représentation des flottants 3

2.2 Arrondis 3

2.3 Dilemme du fabricant de tables 4

3 Une solution : la bibliothèque MPFR **5**

3.1 Qu'est-ce que MPFR ? 5

3.2 L'objectif de MPFR 5

3.3 Les avantages de MPFR 6

3.4 Le contenu de MPFR 6

4 Comment calculer de façon exacte la moyenne arithmético-géométrique **7**

4.1 Définitions 7

4.2 Premier théorème sur les suites approchées 8

4.3 Convergence des $(\tilde{u}_n)$ et $(\tilde{v}_n)$ 10

4.4 Précision nécessaire 12

4.5 Complexité 13

4.5.1 Complexité de l'inverse de la racine carrée 13

4.5.2 Complexité de la moyenne arithmético-géométrique . . 14

4.5.3 Complexité du logarithme 14

5 Ma contribution à MPFR **15**

5.1 Implantation de la moyenne arithmético-géométrique 15

5.2 Implantation du logarithme 16

5.3 Contribution au déboguage de la bibliothèque 17

6 Tests **18**

7 Perspectives **19**
7.1 Mes réalisations 19
7.2 Les enseignements de ce stage 19
7.3 Les perspectives de MPFR 20

A Exactitude **21**

B Rapidité **22**

C Analyse temporelle **23**