Documentation de l'extension

Louis Jézéquel-Royer
Andriy Parkhomenko
Etienne Gacel
Bastien Fabre
Mohamed Ali Lagha

Janvier 2021

Table des matières

1 Introduction

Nous avons choisi comme extension celle qui porte sur les fonctions trigonométriques. Celle ci demande d'implémenter les fonctions dans le fichier *Math.decah* :

- *cos*, renvoyant une valeur approchée du cosinus du flottant entré en argument
- *sin*, renvoyant une valeur approchée du sinus du flottant entré en argument
- *atan*, renvoyant une valeur approchée du Arctan (ou tan^{-1}) du flottant entré en
- *asin*, renvoyant une valeur approchée du Arcsin (ou sin^{-1}) du flottant entré en argument
- *ulp*, renvoyant une valeur approchée du ulp du flottant entré en argument, c'est à dire la différence entre le dit flottant et le flottant le plus proche
- *pi*, renvoyant une valeur approchée de π

Les deux principales problématiques à l'implémentation de ces fonctions seront le temps d'exécution ainsi que la précision du retour.

2 Algorithmes possibles

2.1 Cosinus et Sinus

La première méthode à laquelle nous avons pensé afin de calculer le cosinus et le sinus est le calcul par **développement limité**, consistant à exploiter les formules :

$$sin(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!} + o(x^{2n+2}) \tag{1}$$

et

$$cos(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k}}{2k!} + o(x^{2n+1}) \tag{2}$$

Cependant, il existe également une configuration de la méthode de CORDIC afin de calculer Arctan.

Nous pouvons également ne calculer qu'une seule des deux fonctions en prenant en compte que

$$sin(x + \frac{\pi}{2}) = cos(x) \tag{3}$$

Ce qui permettrait de n'implémenter qu'une seule des deux fonctions.

Enfin, nous pouvons utiliser **CORDIC** (COordinate Rotation DIgital Computer). Cette méthode consiste à applique de très petites matrices de rotation à un vecteur de cooronnées (1,0), toute en se rapprochant de l'angle voulu. Au final nous nous retrouvons avec un vecteur de coordonnées (cos(x), sin(x)).

2.2 Arctan

Pour Arctan, comme pour sinus et cosinus, nous avons envisagé le développement limité avec la formule :

$$sin(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{2k+1} + o(x^{2n+2}) \tag{4}$$

2.3 Arcsin

Pour Arcsin, nous pouvons également utiliser un développement limité.

$$arcsin(x) = x + \frac{x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \cdots + \frac{1 \cdot 3 \cdot 5 \cdots (2n-1)x^{2n+1}}{2 \cdot 4 \cdot 6 \cdots (2n) \cdot (2n+1)} + o(x^{2n+2}) \tag{5}$$

Mais nous pouvons également utiliser la relation entre Arcsin et Arctan :

$$arcsin(x) = 2arctan(\frac{1}{1 + \sqrt{1 - x^2}}) \tag{6}$$

Mais encore une fois, nous pouvons également utiliser CORDIC.

Enfin, on peut également utiliser CORDIC pour calculer Arcsinus.

2.4 ULP

Nous n'avons pas trouvé de méthode particulière pour calculer ULP (Unit in the Last Place), mais en étudiant la fonction *Math.ulp(float f)*, nous avons remarqué que ULP suivait :

$$Pour |x| \in [8388608 * 2^i, 16777216 * 2^i[, ulp(x) = 2^i \tag{7}$$

Nous pouvons donc faire une recherche d'intervalle, et en déduire quelle est la valeur de ulp(x).

2.5 Pi

Pour pi, nous avons juste à enregistré sa valeur comme attribut de la classe Math, puis de la retourner quand besoin.

3 Choix

3.1 Cosinus et sinus

Pour cosinus et sinus, nous avons choisi d'utiliser CODRIC. Cette méthode étant assez efficace (23 tours pour arriver à une mesure précise) et que les calculs qu'elle nécessite sont soit des additions, soit des multiplications par 2, soit des opérations simples. De plus, sur l'intervalle $[-\frac{\pi}{2}; \frac{\pi}{2}]$, le calcul est en temps constant.

CODRIC renvoie en sortie le cosinus et le sinus de l'entrée, ce qui ne fait implémenter qu'une seule fonction. Ceci est mieux que la relation (3) de 2.1, car avec l'imprécision de π comme flottant, cela aurait rajouter une erreur.

3.2 Arctan et Arcsin

Afin de calculer Arctan et Arcsin, vu que CORDIC fonctionne également pour leur calcul, nous avons également choisi cette méthode, également en temps constants et composé d'opérations simples.

Nous n'avons pas utilisé la relation (6) ,non plus car elle aurait nécessité l'implémentation et l'utilisation d'une méthode retournant une racine carrée qui est assez coûteuse.

3.3 ULP et Pi

N'ayant qu'une seule méthode pour ULP, nous avons utilisé celle ci. Pi est également retourné comme décrit en 2.5.

4 Implementation

4.1 CODRIC

4.1.1 Sinus et cosinus

Les méthodes choisies pour les fonctions trigonométriques sont donc à peu de choses identiques, reposant toutes sur CORDIC. Leurs implémentations sont donc assez semblable.
Pour sinus et cosinus la rotation est assez simple, il suffit de trois vraiables que l'on modifie à chaque tour de boucle :

$$x_{k+1} = x_k - \sigma_k y_k 2^{-k} \tag{8}$$

$$y_{k+1} = y_k + \sigma_k x_k 2^{-k} \tag{9}$$

$$z_{k+1} = z_k - \sigma_k \varepsilon_k \tag{10}$$

avec $x_0 = 0.607252$, $y_0 = 0$, $z_0 = \theta$, θ étant l'argument.
On aura $\sigma_k = sign(z_k)$, et $\varepsilon_k = \arctan 2^{-1}$.
Après plusieurs itérations (23 au minimum) on aura $x_n \simeq cos(\theta)$ et $y_n \simeq sin(\theta)$.

4.1.2 Arctan

L'algorithme pour Arctan est similaire à celui de sinus et cosinus, cette fois, nous aurons pour $x_0 = 1$, $y_0 = -t$, $z_0 = 0$, t l'argument :

$$x_{k+1} = x_k - \sigma_k y_k 2^{-k} \tag{11}$$

$$y_{k+1} = y_k + \sigma_k x_k 2^{-k} \tag{12}$$

$$z_{k+1} = z_k - \sigma_k \varepsilon_k \tag{13}$$

avec $\sigma_k = -1$ si $y_k > 0$ et $\sigma_k = 1$ si $y_k \leq 0$ et toujours $\varepsilon_k = \arctan(2^{-1}$.
AU bout de suffisamment d'itérations (27), on aura $z_n \simeq \arctan(-t)$.

4.1.3 Arcsinus

Pour Arcsin diffère un peu, cette fois, nous appliquerons deux fois la matrice de rotation au vecteur, ce qui va donc nous donner pour $x_0 = 1$, $y_0 = 0$, $z_0 = 0$ et $w_0 = t$, t l'argument. A chaque itération nous ferons donc :

$$\begin{pmatrix} \mathrm{x}_{k+1} \\ \mathrm{y}_{k+1} \end{pmatrix} = \begin{pmatrix} 1 & -\sigma_k 2^{-k} \\ \sigma_k 2^{-k} & 1 \end{pmatrix} * \begin{pmatrix} x_k \\ y_k \end{pmatrix} \tag{14}$$

$$z_{k+1} = z_k + 2\varepsilon_k \tag{15}$$

$$w_{k+1} = w_k + 2^{-2k} w_k \tag{16}$$

Avec $\sigma_k = sign(x_k)$ si $y_k \leq w_k$ et $-sign(x_k)$ sinon.
Au bout de suffisament d'itération (27), on aura $z_n \simeq \arcsin(t)$.

4.2 ULP

L'algorithme de calcule de ULP est juste une recherche pour trouvé dans quel intervalle l'argument se trouve. On retourne ensuite le ULP correspondant.

4.3 Problèmes rencontrés

Deca a quelques contraintes supplémentaires comparé à Java, que ce soit le faait qu'on ne puisse déclarer de variable qu'au début, ou encore la non présence de listes. Pour utiliser CORDIC, comme on ne peut pas calculer les $\varepsilon_k = \arctan 2^{-1}$, il nous a donc fallu stocké ses valeurs. Or la meilleure manière de stocker une cinquantaine de valeurs est une liste, comme celles ci n'existent pas en deca, pour contourner le problème, il nous a fallu créer une classe *Cellule* en parallèle, correspondant à une liste chaînée.

Il a également fallu ne pas oublier que les boucles for n'existent pas en Deca, elles ont donc toutes été remplacées par des boucles while.

5 Précision

La question étant, quelle est la précision de notre algorithme ? La précision sera étudiée en comparaison avec le module Java.lang.Math, comportant les mêmes fonctions. Dans l'ensemble la précision de CORDIC devrait être constante, sauf pour cosinus et sinus à cause du fait que nous devons ramener l'argument à un équivalent sur $[-\frac{\pi}{2}; \frac{\pi}{2}]$ afin que CORDIC marche.

5.1 CORDIC pour sinus et cosinus

x	Erreur sur sinus	Erreur sur cosinus
-1.4	-1.1379547E-7	-1.720476E-7
-1.2	-1.1448691E-7	-1.3774957E-7
-1.0	-1.5123717E-8	1.08923686E-7
-0.8	-4.8116103E-8	-1.05004965E-7
-0.6	-1.4357498E-7	-1.7227593E-7
-0.4	-1.0578929E-7	2.392104E-8
-0.2	6.9560073E-9	-1.4883813E-8
0	-2.3297615E-8	0.0
0.2	9.974887E-9	-2.4357105E-8
0.4	9.010426E-8	5.352126E-9
0.6	1.8293005E-7	-1.9920019E-7
0.8	8.133767E-8	-1.3921117E-7
1.0	4.088734E-8	-5.0410065E-8
1.2	1.3176547E-7	-1.8219265E-7
1.4	1.2190014E-7	-2.4883983E-7

Ainsi sur l'intervalle $[-\frac{\pi}{2}; \frac{\pi}{2}]$, l'algorithm est assez précis, provoquant une erreur d'ordre maximum E-7, ce qui correspond presque à un ulp sur ces valeurs, la précision est donc satisfaisante.

x	Erreur sur sinus	Erreur sur cosinus
1.0	2.8005529E-8	-3.0347834E-8
10.0	-4.0885862E-8	1.0235513E-7
100.0	2.3999287E-6	1.4293937E-6
1000.0	1.4463513E-5	-2.1145534E-5
10000.0	-3.4964985E-5	1.1248658E-5
100000.0	-0.94941163	-1.170997
1000000.0	0.6351669	1.1083883

Cependant, sur des valeurs plus élévées, étant donné que pour calculer cosinus et sinus, on ramène l'argument sur l'intervalle $[-\frac{\pi}{2}; \frac{\pi}{2}]$, on soustrait ou ajoute un certain nombre de fois une valeur approchée π, multipliant donc l'imprécision que l'on a déjà. A final nous nous retrouvons avec une erreur beaucoup plus élévée sur de très grandes valeurs comme prévu.

5.2 CORDIC pour Arcsin

x	Erreur sur asin
-1.0	-3.364841E-4
-0.1	9.0234494E-8
-0.01	4.3804746E-8
-0.001	4.8262105E-8
-0.0001	4.845771E-8
-0.00001	4.8478796E-8
0.0	-s4.8481176E-8
0.00001	-4.8481525E-8
0.0001	-4.847954E-8
0.001	-4.8262105E-8
0.01	-4.3804746E-8
0.1	-9.772261E-8
1.0	-8.782872E-6

La précision de Arcsinus est assez satisfaisante, sauf sur les extrémités ou celle ci est bien moins grande. Ceci dit l'erreur est plutôt constante selon les valeurs.

5.3 CORDIC pour Arctan

x	Erreur sur atan
-10000.0	2.7117125E-8
-1000.0	-3.3519847E-9
-100.0	7.756267E-8
-10.0	7.418564E-8
-1.0	8.146034E-8
-0.1	2.7655027E-8
-0.01	1.48009915E-8
-0.001	4.4105494E-8
-0.0001	4.8021654E-8
-0.00001	3.3843207E-8
0.0	-2.3125994E-8
0.00001	-3.3842298E-8
0.0001	-4.801438E-8
0.001	-4.4105494E-8
0.01	-1.48009915E-8
0.1	-3.503184E-8
1.0	-1.11262665E-7
10.0	-8.362797E-8
100.0	-7.832553E-8
1000.0	3.2909495E-9
10000.0	-2.712689E-8

Avec une erreur d'ordre maximum E-8, la fonction Arctan est très précise avec une erreur très constante.

5.4 ULP

x	Erreur sur ULP
10000000.0	0.0
1000000.0	0.0
100000.0	0.0
10000.0	0.0
1000.0	0.0
100.0	0.0
10.0	0.0
1.0	0.0
0.1	0.0
0.01	0.0
0.001	0.0
0.0001	0.0
0.0	0.0

L'erreur ici est constante et nulle, ce qui démontre une fonction de précision exacte.

6 Efficacité

L'efficacité est normalement en temps constant, sauf pour cosinus et sinus sur de grandes valeurs et ULP, vu que la complexité est en $O(\log_2(x)$.
Le temps indiqué dans les tableaux est en nanosecondes.

6.1 CORDIC pour sinus et cosinus

x	CORDIC sinus	CORDIC cosinus
-1.4	19376	17353
-1.2	19085	19697
-1.0	19827	18975
-0.8	19075	19096
-0.6	19345	19386
-0.4	20438	18544
-0.2	20258	19015
0.0	19376	19907
0.2	20087	19036
0.4	19527	18545
0.6	19115	17883
0.8	19797	19416
1.0	19566	18684
1.2	19577	19125
1.4	19577	20458

Comme attendu, le temps d'exécution est fixe sur cette intervalle, avec une moyenne aux alentours de 19000 nanosecondes, soit 19 millisecondes.

x	CORDIC sinus	CORDIC cosinus
1.0	31900	31679
10.0	72264	19677
100.0	21370	20328
1000.0	36558	37580
10000.0	193550	186367
100000.0	1689546	1792117
1000000.0	3792172	1947376
10000000.0	5322852	16240353

Comme attendu, sur de grandes valeurs la compléxité passe en $log_2(x)$, dû au

temps de ramener l'argument sur l'intervalle, étant un processus assez lent (on enlève un π à la fois).

6.2 CORDIC pour Arctan

Pour Arcsinus, nous attendons un temps constant.

x	CORDIC atan
0.0	17994
0.0000001	27000
0.000001	18354
0.00001	21830
0.0001	14166
0.001	13936
0.01	15378
0.1	14838
1.0	15178
10	14237
100	14887
1000	15900
10000	9597

Le temps est donc bel et bien en temps constant peu importe l'argument, comme attendu avec CORDIC.

6.3 CORDIC pour Arcsin

Le temps attendu est encore une fois constant.

x	CORDIC atan
-1.0	23374
-0.8	23594
-0.6	23023
-0.4	27382
-0.2	26310
0.0	27582
0.2	26160
0.4	26360
0.6	27211
0.8	26941
1.0	26090

attendu, le temps d'exécution est constant sur les valeurs et assez proche des résultats des autres fonctions. Arcsinus demandant quelques actions en plus, il est normal qu'il soit plus élévé

6.4 ULP

Comme ULP nécessite une recherche d'intervalle en divisant par deux à chaque itération, nous ne retrouvons pas avec un temps constant mais un temps variable, grandissant lorsque l'argument se rapproche de 0 ou de la valeur maximale. On attend une complexité proportionnelle au logarithme.

x	ULP
1.0E8	4930
1.0E7	2084
1000000.0	972
100000.0	1192
10000.0	1082
1000.0	972
100.0	972
10.0	1062
1.0	1212
0.1	1273
0.01	1303
1.0E-4	1383
1.0E-5	1453
1.0E-6	1533
1.0E-7	1593
1.0E-8	1744
1.0E-9	1814
1.0E-10	1783
1.0E-11	1904
0.0	451

Le temps d'exécution est en accord avec la prédiction. 0 est beaucoup moins long car il est reconnu dès le début et renvoie automatiquement le ulp correspondant à 0.

7 Axes d'amélioration

Plusieurs améliorations pour les algorithmes avec un peu plus de temps :

L'une des principales faiblesses de ces algorithmes est dans le cosinus et le sinus, quand il s'agit de calculer avec un argument supérieur à $\frac{\pi}{2}$. La méthode que nous utilisons est très peu efficace et très peu précise. Il existe des algorithmes afin d'exécuter la même action, mais plus efficace et plus précise.

Sur les fonctions implémentées avec CODRIC, il est possible qu'il y ait trop d'itérations afin d'avoir une valeur assez précise, ce qui rallonge le temps d'exécution.

L'instruction FMA n'a pas été implémenté, ce qui aurait pu rendre plus rapide les algorithmes à l'exécution.

CORDIC n'est peut être pas la méthode la moins coûteuse pour l'implémenation des fonctions, une autre méthode aurait peut être été plus adapté et plus efficace.

La méthode pour ulp n'est sûrement pas la meilleure, un peu plus de recherche aurait mené à la découverte d'un algorithme plus efficace.

8 Conclusion

Le cahier des charges est dans l'ensemble respecté, toutes les fonction sont implémentées et marchent. La précision et l'efficacité sont satisfaisantes même si elles pourraient être améliorées comme dit ci dessus. Les résultats sont concluant, même si ceux ci font pâle figure face à certains modules comme Java.lang.Math qui a été utilisé dans les tests. Nous n'avons peut être pas accordé assez de temps à l'extension ce qui aurait permi d'implémenter des fonctions d'un peu meilleure qualité et efficacité.

9 Bibliographie

— un guide pour utiliser CORDIC pour calculer Arctan

— la page Wikipédia de CORDIC

— guide pour utiliser CORDIC pour calculer Arcsinus

— pour comprendre ce qu'est ulp