Documentation sur l'extension.

Projet génie logiciel

Equipe : gl07

Naima AMALOU Yidi ZHU Jiayun Po Zaineb Tiour An Xian

30 Janvier 2020

Table des matières

Introduction

Lors du projet Génie Logiciel, il nous a été demandé d'effectuer un choix parmi dix sujets appelés "extension". Cette partie est plutôt ouverte, et aussi moins guidée. Il fallait qu'on cherche la solution d'implémentation de ces sujets nous même. L'extension que nous avons choisi est l'extension TRIGO où nous devrons concevoir une bibliothèque Math qui contient des fonctions trigonométriques importantes telle que le sinus, le cosinus, l'arc sinus, l'arc tangent et l'ULP. Nous les expliquerons en détails à la partie Analyse du sujet.

1.1 Motivation :

La première raison pour laquelle que nous avons choisi ce sujet est que le calcul de flottants nous intriguait tous. Cette curiosité a vu le jour au projet C de la première année où les tests d'égalité entre flottants généraient des bugs et on se posait toujours la question sur le degré de précision de nos machines. Nous nous avions le même reflex qu'étant étudiante d'une filière scientifique on utilise la trigonométrie souvent dans notre étude et c'est une occasion pour le connaître et l'étudier sa conception en algorithme .

1.2 Analyse du sujet :

L'extension Trigo qu'on devra construire est une bibliothèque standard au format <.decah> en langage Deca qui est à la base la bibliothèque standard <Math.h> du java.
La liste des fonctions trigonométriques et leurs fonctionnalités à implémenter sont :

float **sin** (*float*)	calcule du sinus
float **cos** (*float*)	calcule du cosinus
float **asin** (*float*) [-1.0, 1.0]	calcule de l'arc sinus
float **atan** (*float*)	calcule de l'arc tangente
float **ulp** (*float*)	calcul sur le plus petit pas d'une valeur à la valeur suivante

Objectif :

- Chercher la meilleure approximation possible permise par la représentation des flottants simple précision.

- Gestion des paramètres d'entrée qui entraîne une erreur de math.

1.3 Représentation des flottants en assembleur :

Pendant la période de la recherche pour l'extension, le premier question qu'on se demande est que comment les nombres décimaux sont représentés dans un ordinateur étant su que le calcul d'un nombre décimal est bien plus compliqué parce que un ordinateur reconnaît que des 0 et 1. Pour combler ce problème, plusieurs méthodes sont inventées mais la méthode la plus utilisée s'appelle la virgule flottante. Avec cette méthode, l'ordinateur traduit un nombre avec virgule sous forme binaire. La norme IEEE 754 est celle qui normalise les formats des flottants.

Au fait, un nombre à virgule peut être représenté en exposant de la base 10, par exemple 0.2222 peut être écrit comme 2222E-4. Avec l'ordinateur d'où les nombres sont représentés en binaire, cela donne la même idée. Au lieu d'avoir un exposant de la base 10, avec la norme IEEE, on utilise un exposant de la base 2 sous la forme :

$$ax2^(Exposant)$$

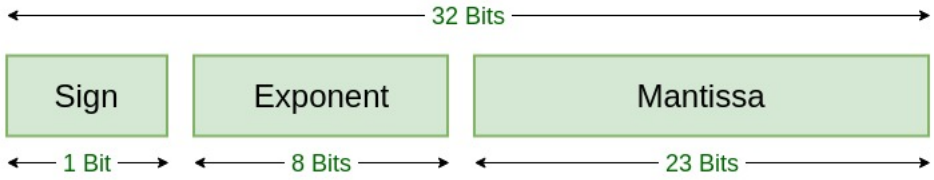
En conséquence, l'écriture d'un nombre décimal en binaire est :

$$1.[partiefractionnairedea]x2^(Exposant)$$

Pour stocker cette écriture dans ordinateur, il nous faut stocker la partie factionnaire (on l'appelle souvent mantisse) et l'exposant de la valeur. Le premier bit de la représentation est le bit de signe qui sert à distinguer un nombre positive et un nombre négative. (0 pour +, 1 pour -) Donc une répresentation IEEE est la suivante :

$$[Bitdesigne].[Exposant].[Mantisse]$$

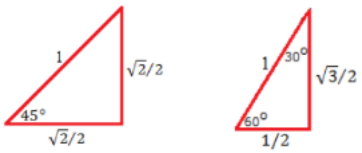
L'exposant répresentable est entre [-126 :127] pour un format simple précision de 32 bits.



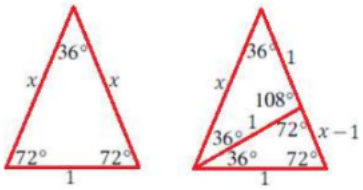
Single Precision
IEEE 754 Floating-Point Standard

1.4 Valeurs exacts de sinus et cosinus :

Avant de se lancer dans la recherche de l'algorithme de sinus et cosinus, on se demande en premier les valeurs exacts de sinus et cosinus en math. D'où vient les valeurs de sinus et cosinus, et ses utilisation.
Les valeurs de sinus et cosinus ou ses angles sont des multiples de trois sont représentable facilement en géométrie.
Par exemple, dans un triangle rectangle, on savait que la valeur de sinus est le rapport de la longueur du côté opposé par la longueur de l'hypoténuse et la valeur de cosinus est le rapport de la longueur du côté adjacent par la longueur de l'hypoténuse. Les angles dans un triangle rectangle (30-60-90 / 45-45-90) sont des angles divisible par trois.



Les restes des angles qui sont multiples de 3 peuvent être représenté géométriquement. Par exemple, triangle isocèles avec angles 72-36-36.



La ratio de x /1 est égale à 1/x-1. Delà, on peut déduire que x^2-x-1=0, et x= (1+ sqrt(5)) /2. En séparant le triangle 36-108-36 en 2, on obtient un triangle de 36-54-90. Donc, avec ceci on pourrait très bien appliquer les formules de sinus et cosinus en géométrie. On obtient :

$$\sin 36° = \frac{\text{opposite}}{\text{hypotenuse}} = \frac{\sqrt{10-2\sqrt{5}}/4}{1} = \frac{\sqrt{10-2\sqrt{5}}}{4},$$

$$\cos 36° = \frac{\text{adjacent}}{\text{hypotenuse}} = \frac{(1+\sqrt{5})/4}{1} = \frac{1+\sqrt{5}}{4}.$$

Pour les angles restés qui ne sont pas de multiple de 3, ses valeurs de cosinus et sinus peuvent être retrouvées à partir du cercle unité ou des séries entières.

Choix d'algorithmes :

Pendant la recherche, nous avons trouvé 3 algorithmes possibles pour chercher le cosinus et sinus. Pour le arctangent et arcsinus, nous nous rendu compte que ces 2 fonctions trigonométriques sont faciles à trouver avec l'algorithme de cos et sin à la main. Dès que nous trouvions ces 2 fonctions trigonométrique de base, nous aurions moins de soucis sur le calcul d'arctangent et arcsinus. Pourtant, nous allons chercher des algorithmes possibles pour trouver ces arctangent et arcsinus sans dépendent de sinus et cosinus.

2.1 Cosinus et Sinus

2.1.1 Table Trigonométrique

La première méthode que nous avons pu trouver est la sauvegarde de la table trigonométrique dans la mémoire du système. La mémorisation des valeurs de cosinus et sinus de 0 à 45 degré est suffisante sachant que le graph de cosinus ou sinus est périodique. Par contre, pour un degré de fraction décimal, le cosinus et sinus de cette valeur sont retrouvés par estimation. Par conséquent, avec cette méthode, nous avons 90 valeurs à enregistrer même avant le calcul. Nous nous demandions si cette méthode vaut le coût. Tout d'abord, l'enregistrement de 90 valeurs est une perte d'espace mémoire. Néanmoins, cette méthode est le plus simple à implémenter et garantie un bon niveau de précision.

2.1.2 Cordic

La deuxième est la méthode de Cordic. L'algorithme de Cordic est simple à implémenter et il n'y a que 32 valeurs de tableau de Cordic à mémoriser. C'est en général une bonne méthode à utiliser dans .deca.
Le Cordic (COordinate Rotation DIgital Computer) est utilisé que pour calculer les résultats de trigonométrique. La plus grande avantage de l'utiliser est que l'algorithme de CORDIC est simple à appliquer qu'on pourrait le construire avec des composantes électroniques de base. Le coût d'application de CORDIC dans la vraie vie est relativement bas, on pourrait trouver l'application de CORDIC dans des calculatrices à poche.
Le Cordic en général emploie la rotation pour retrouver le cos et le sinus qu'on a voulu. Pourtant, la rotation est dur à performer sans la fonction cosinus et sinus. Partant de ce fait, on utilise la pseudo-rotation avec tangent teta et construire l'angle qu'on voudrait avec des petits angles de tangent sachant que le tangent de angle (45- 22.5 -11.25-5.625. . .) est égale à (½, ¼ 1/8 , 1/16 etc. . .)

$\tan(\lvert\theta_i\rvert)$	$\lvert\theta_i\rvert$
$2^0 = 1$	$0.785398 \equiv 45°$
$2^{-1} = 0.5$	$0.4636 \equiv 26.5651°$
$2^{-2} = 0.25$	$0.2449 \equiv 14.0362°$
$2^{-3} = 0.125$	$0.124 \equiv 7.1250°$
$2^{-4} = 0.0625$	$0.0624 \equiv 3.5763°$
$2^{-5} = 0.03125$	$0.03122 \equiv 1.7899°$
$2^{-6} = 0.015625$	$0.015625 \equiv 0.8952°$
$2^{-7} = 0.0078125$	$0.0078125 \equiv 0.4476°$
$2^{-8} = 0.00390625$	$0.00390625 \equiv 0.2238°$

Une rotation à partir d'un coordonnée quelconque (x,y) de l'origine est représenté en mathématique :

$$\begin{aligned} x_{i+1} &= x_i \cos\alpha_i - y_i \sin\alpha_i \\ y_{i+1} &= y_i \cos\alpha_i + x_i \sin\alpha_i \\ \theta_{i+1} &= \theta_i + \alpha_i \end{aligned}$$

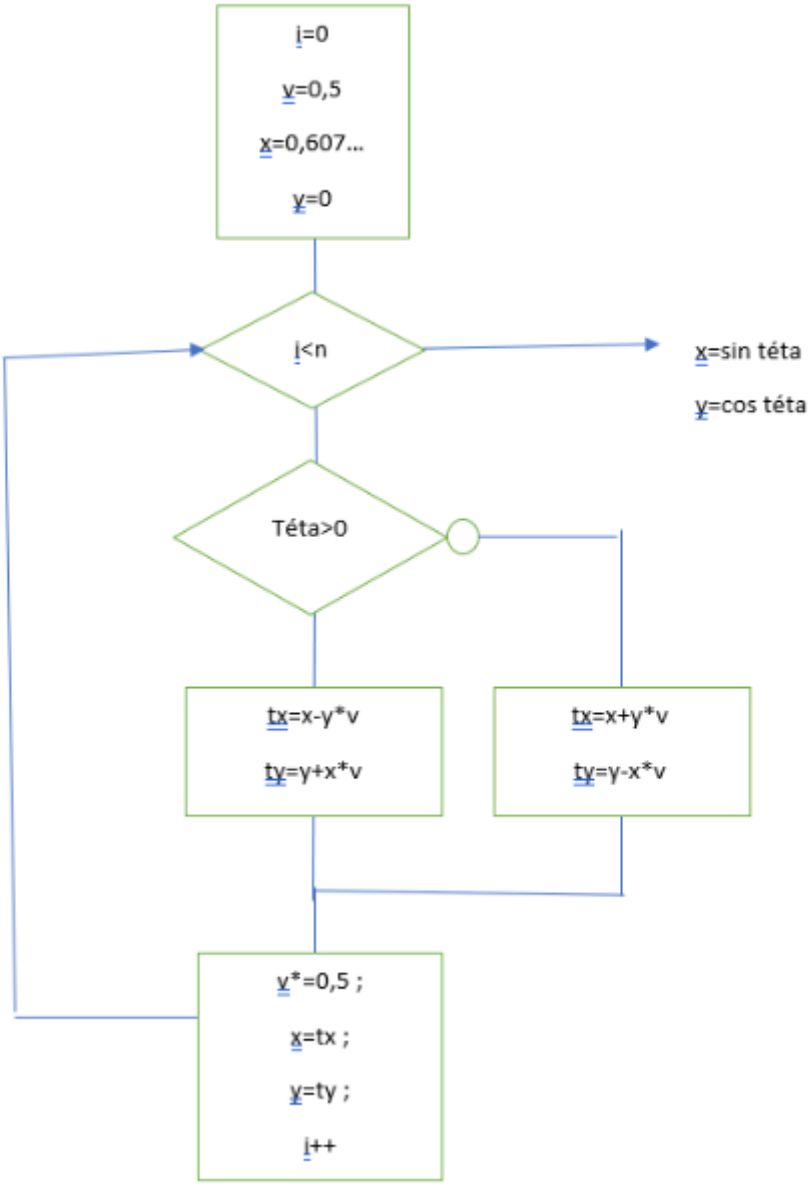
Et les formules de pseudo-rotation sont dérivées de la rotation d'où Et les valeurs de xn et

$$\begin{aligned} x'_{i+1} &= (x_i - y_i \tan\alpha_i) \\ y'_{i+1} &= (y_i - x_i \tan\alpha_i) \\ \theta_{i+1} &= \theta_i - \alpha_i \end{aligned}$$

yn après n fois de pseudo-rotations donnent (d'où K est la facteur à multiplier au cours de n opérations) :

$$\begin{aligned} x_n &= K\left(x_i \cos\sum_{i=0}^{n-1}\alpha_i - y_i \sin\sum_{i=0}^{n-1}\alpha_i\right) \\ y_n &= K\left(y_i \cos\sum_{i=0}^{n-1}\alpha_i + x_i \sin\sum_{i=0}^{n-1}\alpha_i\right) \\ \theta_n &= \theta_i - \sum_{i=0}^{n-1}\alpha_i \end{aligned}$$

L'algorithme de Cordic :

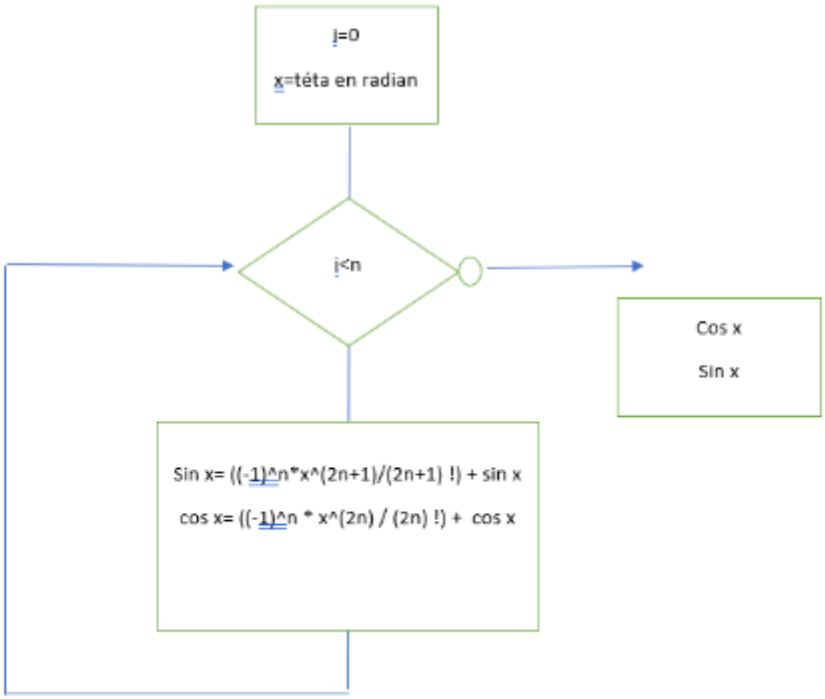


2.1.3 La série de Taylor

En math, une série de Taylor est une représentation de fonction comme une somme des terms à l'infini calculant des valeurs de la dérivation d'une fonction sur un point.
Formule de la série du Taylor :

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

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$



2.2 Arc tangent

Arctangent peut être retrouvé avec la série de Taylor. Ça vient du fait que la dérivée d'Arctangent est :

$$f(x) = \arctan x$$

$$f'(x) = \frac{1}{1+x^2} = \frac{1}{1-(-x^2)}$$

Nous nous rappelons que :

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n$$

En remplaçant le x par le $-x^2$, et le fait intégrale de tout, nous trouvions que :

$$f(x) = \sum_{n=1}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1}$$

pour 1- <= x <= 1 ; Cette méthode est précise et rapide mais cet équation de arc tangent change en fonction de la valeur de x.

$$\arctan x = \begin{cases} \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1} & : -1 \le x \le 1 \\ \frac{\pi}{2} - \sum_{n=0}^{\infty} (-1)^n \frac{1}{(2n+1)\,x^{2n+1}} & : x \ge 1 \\ -\frac{\pi}{2} - \sum_{n=0}^{\infty} (-1)^n \frac{1}{(2n+1)\,x^{2n+1}} & : x \le -1 \end{cases}$$

Autrement dit, pour pouvoir couvrir tous les valeurs possibles de x, nous devrions mettre en place 3 différents algorithme.

Une autre méthode que nous arrivons à penser est de nous servir de les fonctions cosinus et sinus que nous avons trouvé. Pour cette méthode, nous parcourions tous les degrés possibles 0° à 90° de la fonction tangent. Dès que nous trouvions l'arc tangent d'un degré est égale ou légèrement supérieur à la valeur qu'on cherche, on retourne le degré associé.
Les problèmes avec cette méthode est qu'on perte quelques valeurs de précision et le temps d'exécution est beaucoup plus long que celui de Taylor.

2.3 Arc sinus

Comme l'arc tangent, nous avons trouvé deux méthodes qu'on peut mettre en place. La première est l'utilisation de la série de Taylor avec la formule :

$$\begin{aligned}\arcsin x &= \sum_{n=0}^{\infty} \frac{(2n)!}{2^{2n}(n!)^2} \frac{x^{2n+1}}{2n+1} \\ &= x + \frac{1}{2}\frac{x^3}{3} + \frac{1\times 3}{2\times 4}\frac{x^5}{5} + \frac{1\times 3\times 5}{2\times 4\times 6}\frac{x^7}{7} + \cdots\end{aligned}$$

Cette équation est bien plus compliqué que celle de cosinus et sinus, mais le plus gros souci est que le numérateur et le dénominateur deviennent un nombre très grand rapidement et qu'on risque d'avoir un arithmétique overflow avec notre compilateur.

La deuxième méthode qu'on a pu penser est de parcourir tous les degrés possibles 0° à 90° de la fonction sinus. Dès que nous trouvions l'arc sinus d'un degré est égale ou légèrement supérieur à la valeur qu'on cherche, on retourne le degré associé.
Certes, cette méthode partage les mêmes problèmes que celle d'arc tangent.

2.4 ULP

Pour l'ULP, nous avons employé l'algorithme que nous avons trouvé sur le page Wikipédia. Tout d'abord, on multiplie la valeur en entrée par 2 et on répète cette étape jusqu'à on obtient une valeur très grande qui égale à la même valeur plus 1. Pendant cette opération , on compte aussi le nombre de fois de la multiplication qu'on a fait. Ensuite, avec la valeur d'ulp en partant de 1, on divise par 2 pendant nièmes fois d'où le nombre de fois qu'on a fait la multiplication.

```
>>> x = 1.0
>>> p = 0
>>> while x != x + 1:
...     x = x * 2
...     p = p + 1
...
>>> x
9007199254740992.0
>>> p
53
>>> x + 2 + 1
9007199254740996.0
```

Analyse théorique de la précision en Java :

Avant la mise en oeuvre des fonctions trigonométriques sur la bibliothèque standard de decah, nous avons testé quelques algorithmes que nous avons choisis sous le langage Java. Nous avons comparé l'algorithme de la série de Taylor et l'algorithme de Cordic pour les fonctions sinus et cosinus.
Les figures ci-dessous sont les résultats de sinus obtenus en java.

```
Sin: 00,0 (rad: 0,00000000000000) CORDIC: 0,00000000000000 / java: 0,00000000000000
Sin: 05,0 (rad: 0,08726646259972) CORDIC: 0,08715574276119 / java: 0,08715574274766
Sin: 10,0 (rad: 0,17453292519943) CORDIC: 0,17364817769388 / java: 0,17364817766693
Sin: 15,0 (rad: 0,26179938779915) CORDIC: 0,25881904514270 / java: 0,25881904510252
Sin: 20,0 (rad: 0,34906585039887) CORDIC: 0,34202014337876 / java: 0,34202014332567
Sin: 25,0 (rad: 0,43633231299858) CORDIC: 0,42261826180630 / java: 0,42261826174070
Sin: 30,0 (rad: 0,52359877559830) CORDIC: 0,50000000007761 / java: 0,50000000000000
Sin: 35,0 (rad: 0,61086523819802) CORDIC: 0,57357643644008 / java: 0,57357643635105
Sin: 40,0 (rad: 0,69813170079773) CORDIC: 0,64278760978631 / java: 0,64278760968654
Sin: 45,0 (rad: 0,78539816339745) CORDIC: 0,70710678129631 / java: 0,70710678118655
Sin: 50,0 (rad: 0,87266462599716) CORDIC: 0,76604444323788 / java: 0,76604444311898
Sin: 55,0 (rad: 0,95993108859688) CORDIC: 0,81915204441614 / java: 0,81915204428899
Sin: 60,0 (rad: 1,04719755119660) CORDIC: 0,86602540391886 / java: 0,86602540378444
Sin: 65,0 (rad: 1,13446401379631) CORDIC: 0,90630778717733 / java: 0,90630778703665
Sin: 70,0 (rad: 1,22173047639603) CORDIC: 0,93969262093177 / java: 0,93969262078591
Sin: 75,0 (rad: 1,30899693899575) CORDIC: 0,96592582643900 / java: 0,96592582628907
Sin: 80,0 (rad: 1,39626340159546) CORDIC: 0,98480775316507 / java: 0,98480775301221
Sin: 85,0 (rad: 1,48352986419518) CORDIC: 0,99619469824638 / java: 0,99619469809175
Sin: 90,0 (rad: 1,55334303427495) CORDIC: 0,99984769531159 / java: 0,99984769515639
```

```
Sin: 00,0 (rad: 0,00000000000000) TAYLOR: 0,00000000000000 / java: 0,00000000000000
Sin: 05,0 (rad: 0,08726646259972) TAYLOR: 0,08715574274766 / java: 0,08715574274766
Sin: 10,0 (rad: 0,17453292519943) TAYLOR: 0,17364817766693 / java: 0,17364817766693
Sin: 15,0 (rad: 0,26179938779915) TAYLOR: 0,25881904510252 / java: 0,25881904510252
Sin: 20,0 (rad: 0,34906585039887) TAYLOR: 0,34202014332567 / java: 0,34202014332567
Sin: 25,0 (rad: 0,43633231299858) TAYLOR: 0,42261826174070 / java: 0,42261826174070
Sin: 30,0 (rad: 0,52359877559830) TAYLOR: 0,50000000000000 / java: 0,50000000000000
Sin: 35,0 (rad: 0,61086523819802) TAYLOR: 0,57357643635105 / java: 0,57357643635105
Sin: 40,0 (rad: 0,69813170079773) TAYLOR: 0,64278760968654 / java: 0,64278760968654
Sin: 45,0 (rad: 0,78539816339745) TAYLOR: 0,70710678118655 / java: 0,70710678118655
Sin: 50,0 (rad: 0,87266462599716) TAYLOR: 0,76604444311898 / java: 0,76604444311898
Sin: 55,0 (rad: 0,95993108859688) TAYLOR: 0,81915204428899 / java: 0,81915204428899
Sin: 60,0 (rad: 1,04719755119660) TAYLOR: 0,86602540378444 / java: 0,86602540378444
Sin: 65,0 (rad: 1,13446401379631) TAYLOR: 0,90630778703665 / java: 0,90630778703665
Sin: 70,0 (rad: 1,22173047639603) TAYLOR: 0,93969262078591 / java: 0,93969262078591
Sin: 75,0 (rad: 1,30899693899575) TAYLOR: 0,96592582628907 / java: 0,96592582628907
Sin: 80,0 (rad: 1,39626340159546) TAYLOR: 0,98480775301221 / java: 0,98480775301221
Sin: 85,0 (rad: 1,48352986419518) TAYLOR: 0,99619469809175 / java: 0,99619469809175
Sin: 90,0 (rad: 1,55334303427495) TAYLOR: 0,99984769515639 / java: 0,99984769515639
```

En comparant les deux valeurs avec les valeurs de sinus obtenues en java, nous avons trouvé que les valeurs de Taylor sont plus proches aux valeurs exactes. Avec l'algorithme de Cordic,

nous avons obtenu une précision de 10E-9, d'un autre côté, nous avons obtenu une précision de 1E-14 avec la série de Taylor.

Le résultat que nous avons trouvé sur la fonction cosinus est assez proche à celui que nous avons obtenu pour la fonction sinus.

```
Cos: 00,0 (rad: 0,00000000000000) CORDIC: 1,00000000015522 / java: 1,00000000000000
Cos: 05,0 (rad: 0,08726646259972) CORDIC: 0,99619469824638 / java: 0,99619469809175
Cos: 10,0 (rad: 0,17453292519943) CORDIC: 0,98480775316507 / java: 0,98480775301221
Cos: 15,0 (rad: 0,26179938779915) CORDIC: 0,96592582643900 / java: 0,96592582628907
Cos: 20,0 (rad: 0,34906585039887) CORDIC: 0,93969262093177 / java: 0,93969262078591
Cos: 25,0 (rad: 0,43633231299858) CORDIC: 0,90630778717733 / java: 0,90630778703665
Cos: 30,0 (rad: 0,52359877559830) CORDIC: 0,86602540391886 / java: 0,86602540378444
Cos: 35,0 (rad: 0,61086523819802) CORDIC: 0,81915204441614 / java: 0,81915204428899
Cos: 40,0 (rad: 0,69813170079773) CORDIC: 0,76604444323788 / java: 0,76604444311898
Cos: 45,0 (rad: 0,78539816339745) CORDIC: 0,70710678129631 / java: 0,70710678118655
Cos: 50,0 (rad: 0,87266462599716) CORDIC: 0,64278760978631 / java: 0,64278760968654
Cos: 55,0 (rad: 0,95993108859688) CORDIC: 0,57357643644008 / java: 0,57357643635105
Cos: 60,0 (rad: 1,04719755119660) CORDIC: 0,50000000007761 / java: 0,50000000000000
Cos: 65,0 (rad: 1,13446401379631) CORDIC: 0,42261826180630 / java: 0,42261826174070
Cos: 70,0 (rad: 1,22173047639603) CORDIC: 0,34202014337876 / java: 0,34202014332567
Cos: 75,0 (rad: 1,30899693899575) CORDIC: 0,25881904514270 / java: 0,25881904510252
Cos: 80,0 (rad: 1,39626340159546) CORDIC: 0,17364817769388 / java: 0,17364817766693
Cos: 85,0 (rad: 1,48352986419518) CORDIC: 0,08715574276119 / java: 0,08715574274766
```

```
Cos: 00,0 (rad: 0,00000000000000) TAYLOR: 1,00000000000000 / java: 1,00000000000000
Cos: 05,0 (rad: 0,08726646259972) TAYLOR: 0,99619469809175 / java: 0,99619469809175
Cos: 10,0 (rad: 0,17453292519943) TAYLOR: 0,98480775301221 / java: 0,98480775301221
Cos: 15,0 (rad: 0,26179938779915) TAYLOR: 0,96592582628907 / java: 0,96592582628907
Cos: 20,0 (rad: 0,34906585039887) TAYLOR: 0,93969262078591 / java: 0,93969262078591
Cos: 25,0 (rad: 0,43633231299858) TAYLOR: 0,90630778703665 / java: 0,90630778703665
Cos: 30,0 (rad: 0,52359877559830) TAYLOR: 0,86602540378444 / java: 0,86602540378444
Cos: 35,0 (rad: 0,61086523819802) TAYLOR: 0,81915204428899 / java: 0,81915204428899
Cos: 40,0 (rad: 0,69813170079773) TAYLOR: 0,76604444311898 / java: 0,76604444311898
Cos: 45,0 (rad: 0,78539816339745) TAYLOR: 0,70710678118655 / java: 0,70710678118655
Cos: 50,0 (rad: 0,87266462599716) TAYLOR: 0,64278760968654 / java: 0,64278760968654
Cos: 55,0 (rad: 0,95993108859688) TAYLOR: 0,57357643635105 / java: 0,57357643635105
Cos: 60,0 (rad: 1,04719755119660) TAYLOR: 0,50000000000000 / java: 0,50000000000000
Cos: 65,0 (rad: 1,13446401379631) TAYLOR: 0,42261826174070 / java: 0,42261826174070
Cos: 70,0 (rad: 1,22173047639603) TAYLOR: 0,34202014332567 / java: 0,34202014332567
Cos: 75,0 (rad: 1,30899693899575) TAYLOR: 0,25881904510252 / java: 0,25881904510252
Cos: 80,0 (rad: 1,39626340159546) TAYLOR: 0,17364817766693 / java: 0,17364817766693
Cos: 85,0 (rad: 1,48352986419518) TAYLOR: 0,08715574274766 / java: 0,08715574274766
Cos: 90,0 (rad: 1,57079632679490) TAYLOR: 0,00000000000000 / java: 0,00000000000000
```

En comparant les deux valeurs avec les valeurs de cosinus obtenues en java, nous avons trouvé que les valeurs de Taylor sont plus proches aux valeurs exactes. Avec l'algorithme de Cordic, nous avons obtenu une précision de 10E-10, d'un autre côté, nous avons obtenu une précision de 1E-14 avec la série de Taylor.

Implémentation en Deca :

L'implémentation de ces algorithmes en decah est presque le dernier travail que nous avons effectué parce que pour pouvoir compiler la bibliothèque standard de math, cela nous faut un compilateur qui est fonctionnel. De plus, l'inclusion d'une bibliothèque est sous le langage complet. De plus, pendant l'implémentation, nous avons trouvé des 'bugs' du côté compilateur et cela nous ralentisent encore plus notre avancement sur l'extension.

En revanche, avec le langage sans objet, nous arrivions à coder les algorithmes dans un programme principal. Cela nous facilitait l'implémentation dans la blibliothèque standard plus tard. Avec ceci, nous arrivions aussi à avoir un premier aperçu sur la précision que nous allions obtenir pour les fonctions cosinus et sinus.

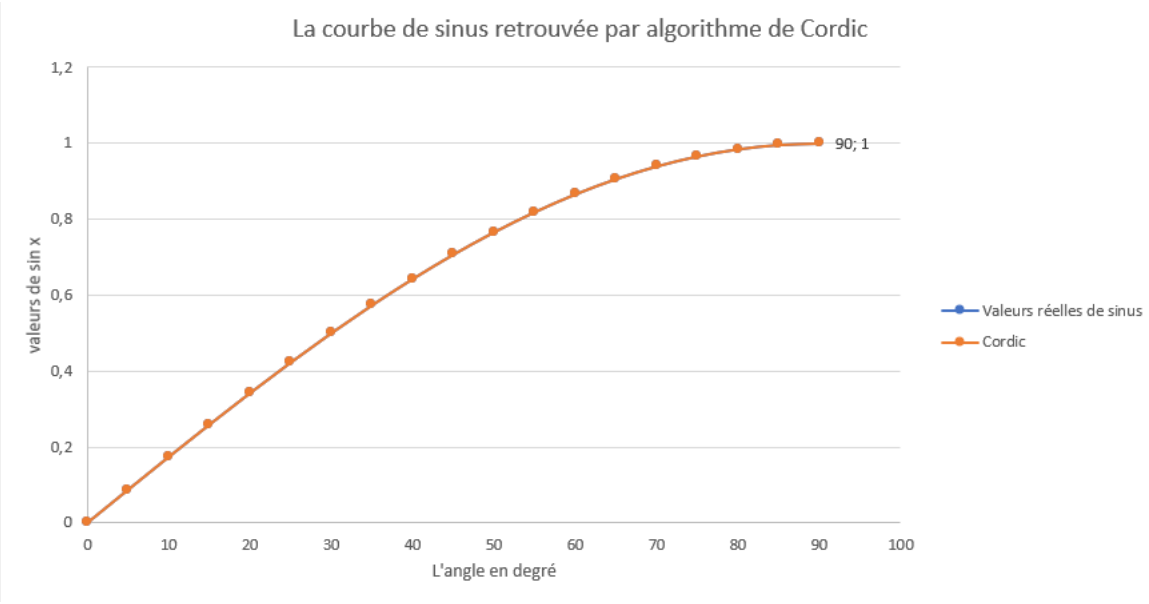
Pour les fonctions de cosinus et sinus, nous avons choisi à implémenter les deux méthodes, donc l'algorithme de Cordic et l'algorithme de Taylor parce que nous n'étions pas sûr lequel s'adapte mieux avec notre compilateur. Avec l'algorithme de Cordic, nous trouvions une précision de 10E-7, mais avec l'algorithme de Taylor, nous avions une précision de 10E-8 pour les fonctions cosinus et sinus.

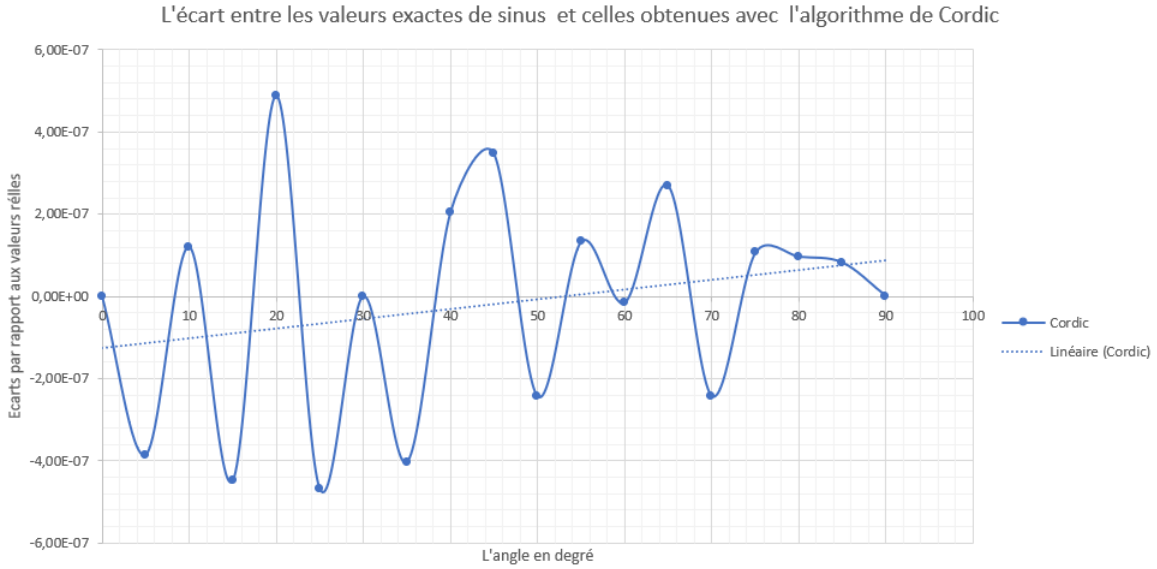
Pour les fonctions arctangent et arcsinus, nous avons choisi de nous servir des fonctions qui sont déjà mises en places, dont les fonctions sinus et cosinus parce que les algorithmes Taylor sur les fonctions arctangent et arcsinus sont bien plus compliquées que celles de cosinus et sinus. En raison du temps insuffisant, nous avons pris cette décision.

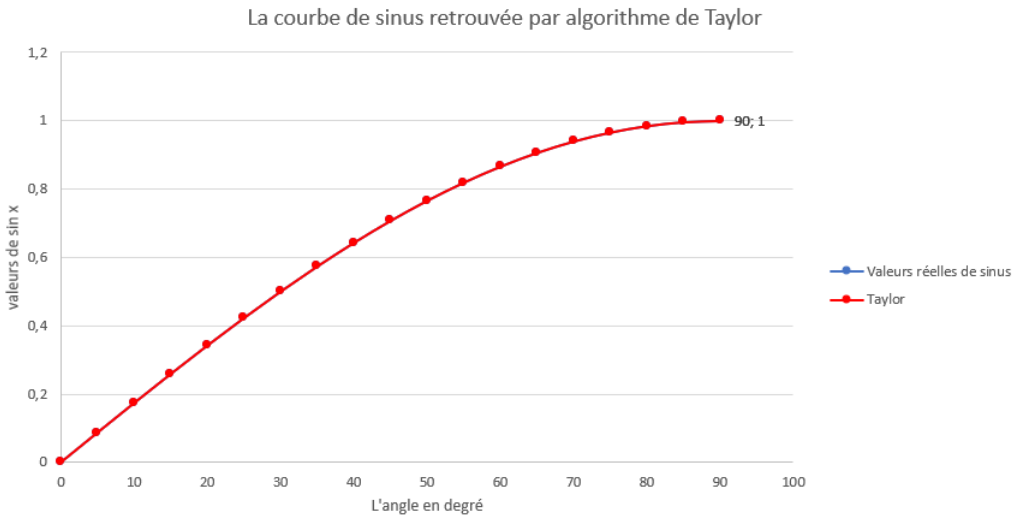
Pour la fonction ULP, nous n'avons pas trouvé de difficulté lors de l'implémentation en langage deca. Nous avons trouvé une précision entre 1E-6 à 1E-43 pour une valeur entre [-100 ; 100].

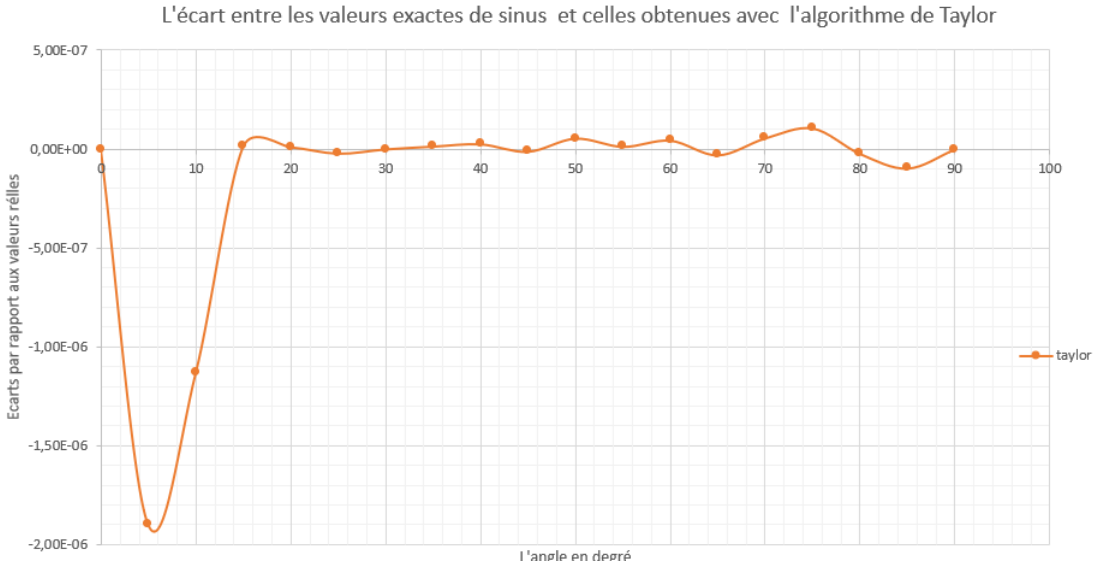
Validation et résultat :

5.1 Sinus





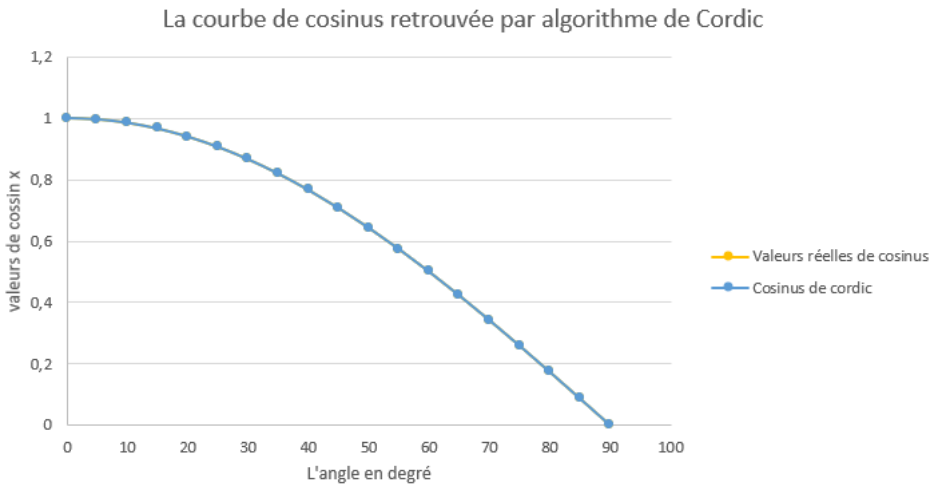


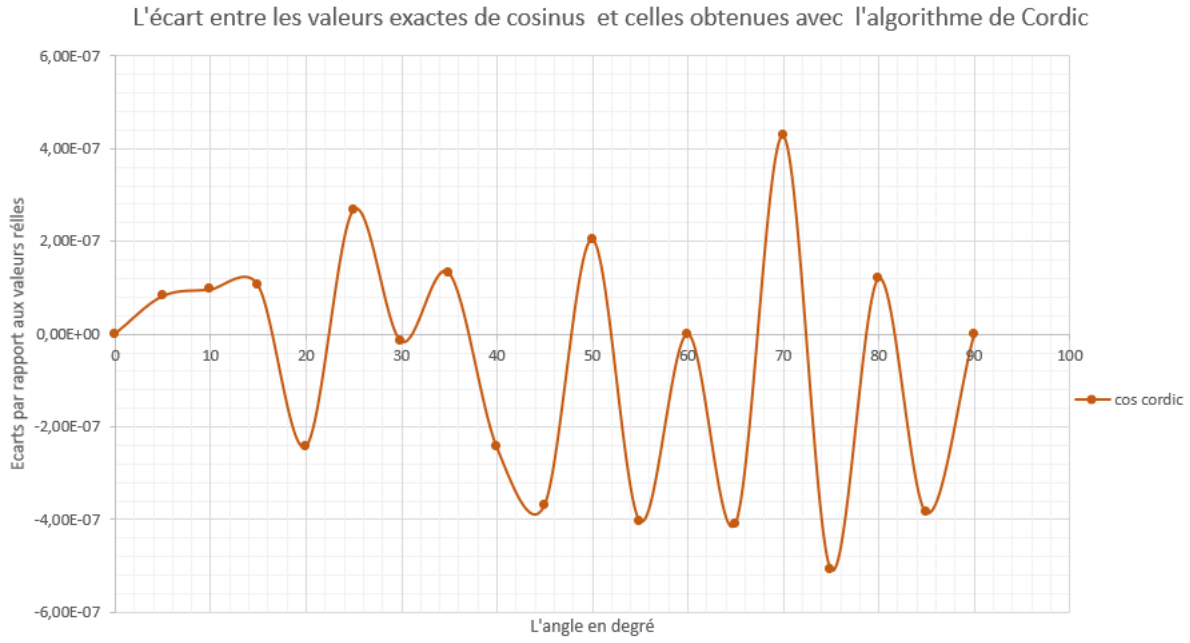


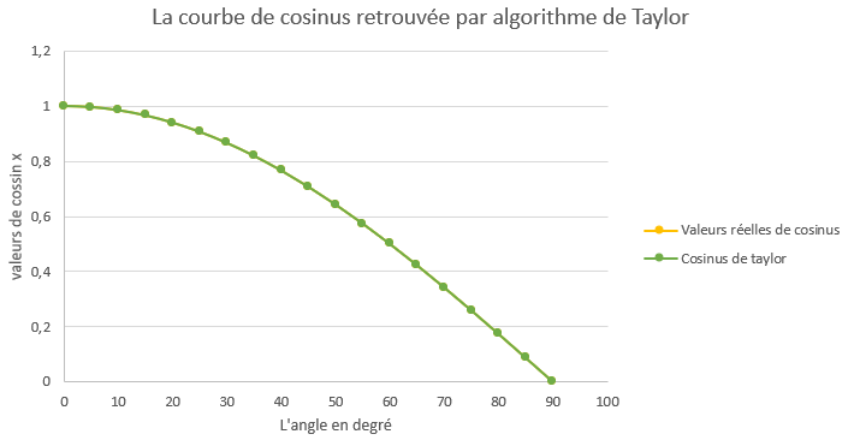
Cosinus teta	Ecart cordic	Ecart taylor
0	0	0
5	8,17811E-08	2,21765E-08
10	9,59181E-08	-2,32911E-08
15	1,05641E-07	-1,35678E-08
20	-2,42742E-07	-4,3232E-09
25	2,67887E-07	-3,01359E-08
30	-1,55436E-08	-1,55436E-08
35	1,32091E-07	-4,67228E-08
40	-2,43652E-07	-6,48383E-08
45	-3,69729E-07	-1,21016E-08
50	2,04454E-07	2,56398E-08
55	-4,03236E-07	1,39969E-08
60	0	-4,76837E-07
65	-4,09053E-07	-5,1425E-08
70	4,27906E-07	-7,87333E-08
75	-5,08106E-07	-9,08732E-08
80	1,2012E-07	-5,86942E-08
85	-3,85744E-07	7,61916E-08
90	-6,12574E-17	-6,12574E-17

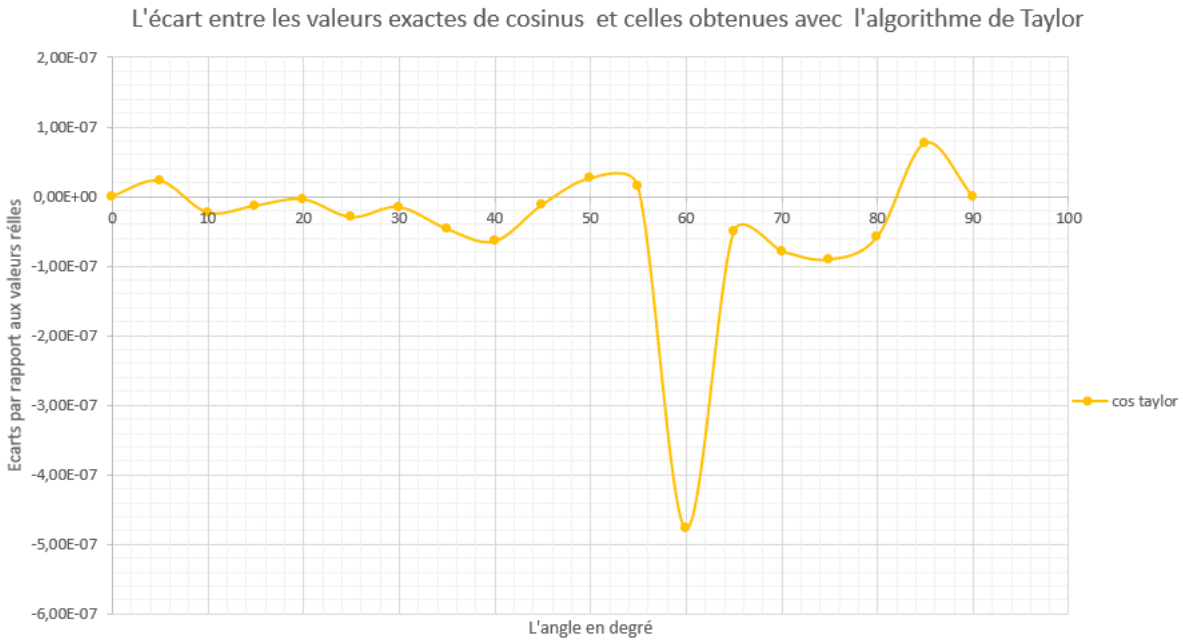
Pour la fonction sinus, nous avons choisi l'algorithme de Taylor. Nous avons obtenu une précision de 1E-8 en moyenne.

5.2 Cosinus


La courbe de cosinus retrouvée par algorithme de Cordic
1,2
1
0,8
0,6
0,4
0,2
0
valeurs de cossin x
0
10
20
30
40
50
60
70
80
90
100
L'angle en degré
Valeurs réelles de cosinus
Cosinus de cordic


L'écart entre les valeurs exactes de cosinus et celles obtenues avec l'algorithme de Cordic
6,00E-07
4,00E-07
2,00E-07
0,00E+00
-2,00E-07
-4,00E-07
-6,00E-07
Ecarts par rapport aux valeurs rélles
0
10
20
30
40
50
60
70
80
90
100
L'angle en degré
cos cordic

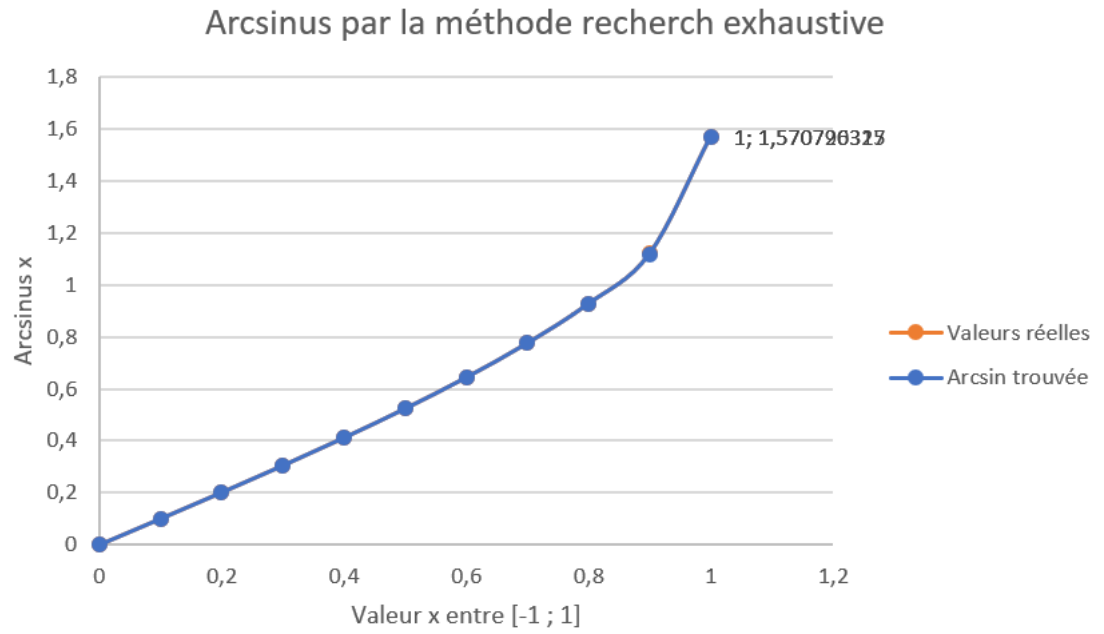


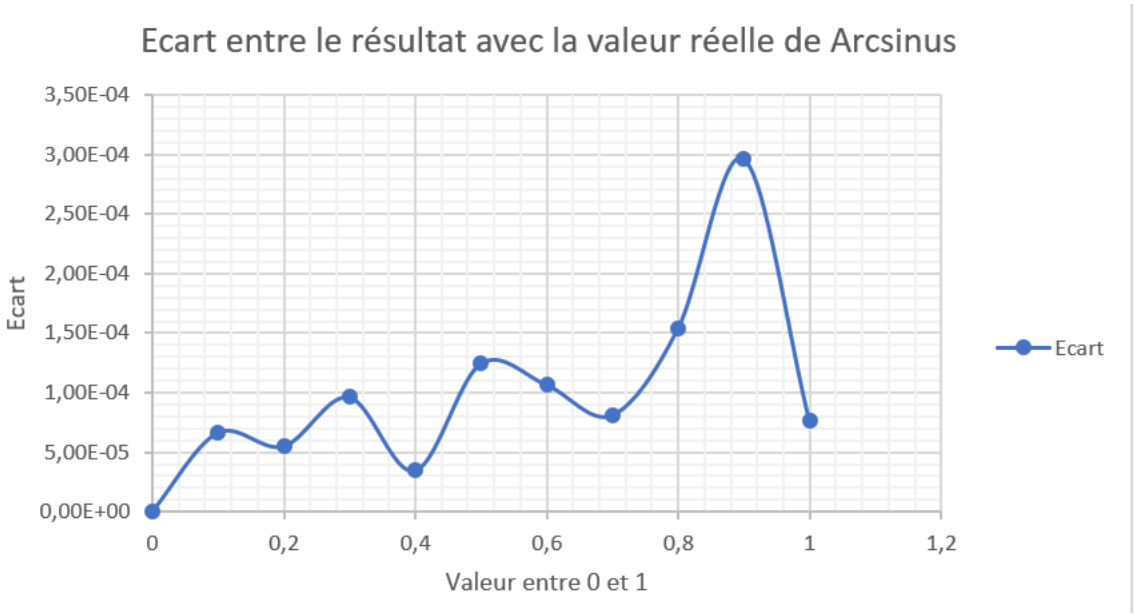


Cosinus teta	Ecart cordic	Ecart taylor
0	0	0
5	8,17811E-08	2,21765E-08
10	9,59181E-08	-2,32911E-08
15	1,05641E-07	-1,35678E-08
20	-2,42742E-07	-4,3232E-09
25	2,67887E-07	-3,01359E-08
30	-1,55436E-08	-1,55436E-08
35	1,32091E-07	-4,67228E-08
40	-2,43652E-07	-6,48383E-08
45	-3,69729E-07	-1,21016E-08
50	2,04454E-07	2,56398E-08
55	-4,03236E-07	1,39969E-08
60	0	-4,76837E-07
65	-4,09053E-07	-5,1425E-08
70	4,27906E-07	-7,87333E-08
75	-5,08106E-07	-9,08732E-08
80	1,2012E-07	-5,86942E-08
85	-3,85744E-07	7,61916E-08
90	-6,12574E-17	-6,12574E-17

Pour la fonction cosinus, nous avons choisi l'algorithme de Taylor. Nous avons obtenu une précision de 1E-7 en moyenne.

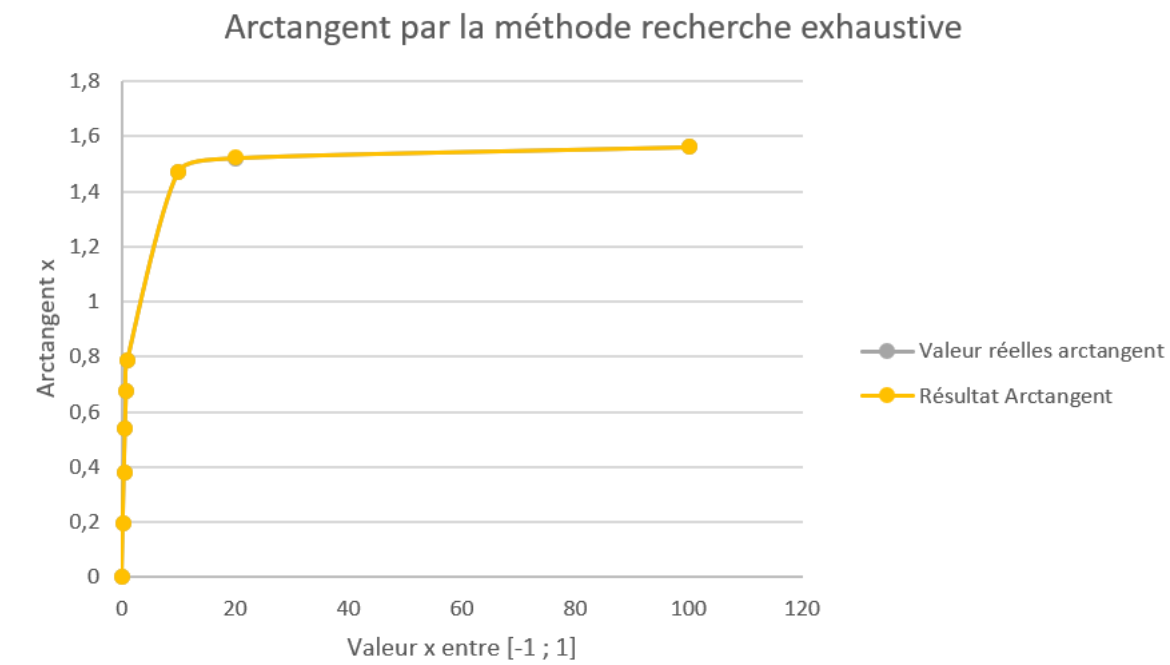
5.3 Arcsinus

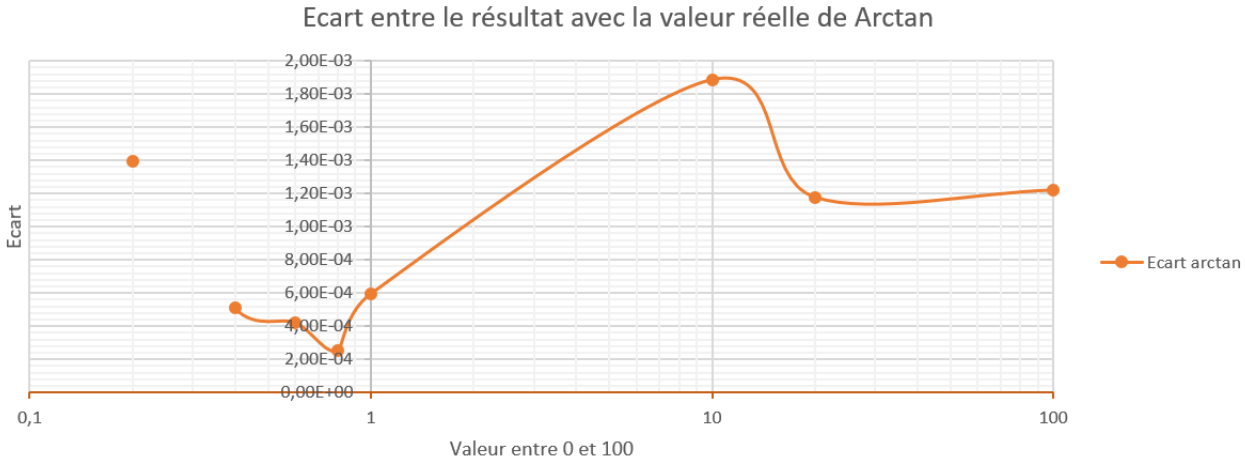




Pour la fonction arc sinus, nous avons choisi la méthode de la recherche exhaustive. Nous avons obtenu une précision de 1.5e-4 en moyenne.

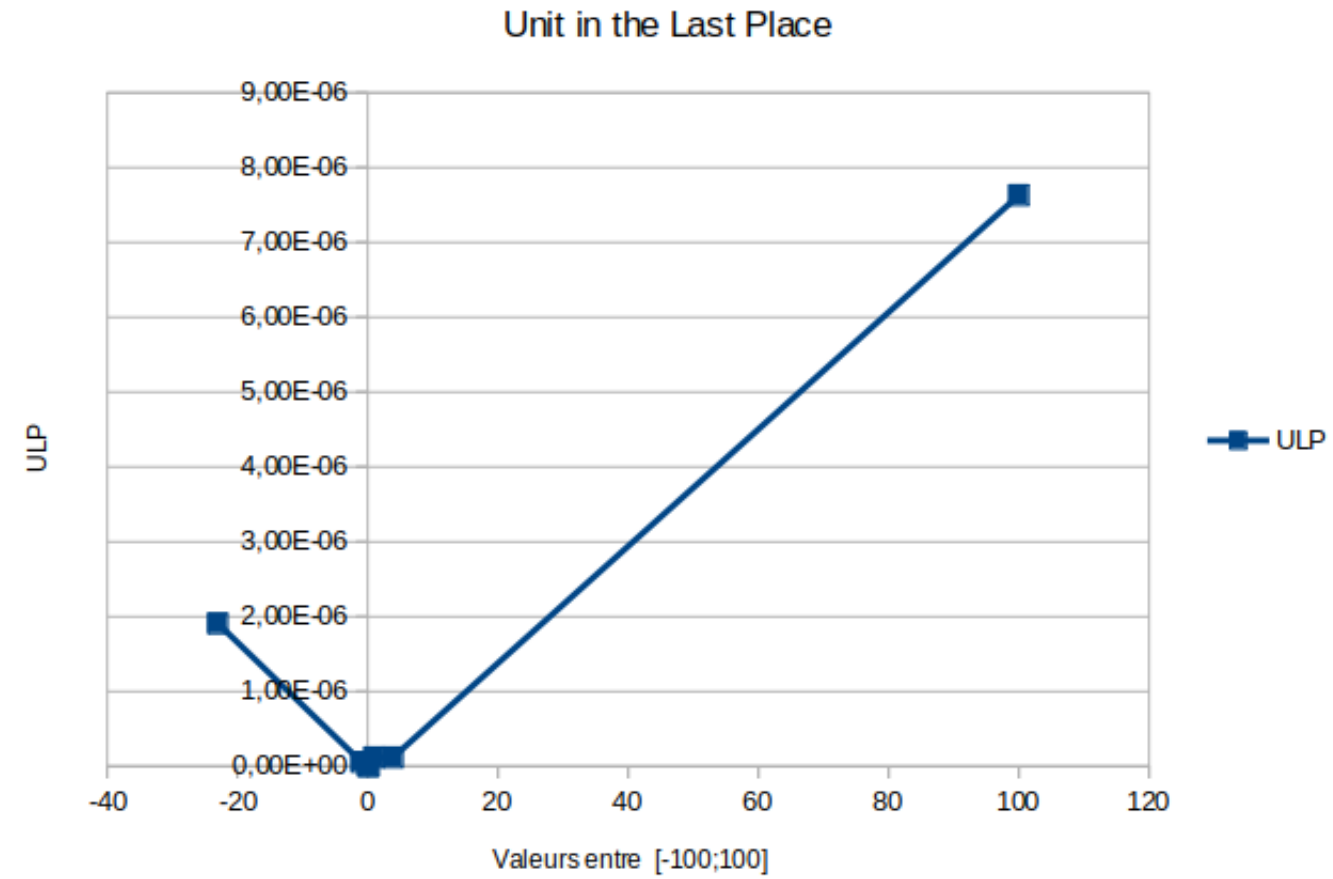
5.4 Arctangent





Pour la fonction arc tangent, nous avons choisi la méthode de la recherche exhaustive. Nous avons obtenu une précision de 1.2e-3 en moyenne.

5.5 ULP



valeur	ULP
-23	1,91E-06
-1	5,96E-08
-0,898	5,96E-08
0	4,48E-44
0,23	7,45E-09
1	1,19E-07
3,898	1,19E-07
100	7,63E-06

Avec l'algorithme que nous avons choisi, nous avons obtenu une précision de 4E-6 en moyenne.

Bibliographie :

- https ://math.stackexchange.com/questions/29649/why-is-arctanx-x-x3-3x5-5-x7-7-dots
- https ://math.stackexchange.com/questions/197874/maclaurin-expansion-of-arcsin-x
- https ://en.wikibooks.org/wiki/Digital_Circuits/CORDIC
- https ://www.apmep.fr/IMG/pdf/cordic.pdf
- http ://www.dcs.gla.ac.uk/ jhw/cordic/
- https ://fr.mathworks.com/help/fixedpoint/examples/compute-sine-and-cosine-using-cordic-rotation-kernel.html
- https ://math.la.asu.edu/ surgent/mat170/Exact_Trig_Values.pdf
- https ://fr.wikipedia.org/wiki/Sinus_(math