1-Introduction : EXTENSION TRIGO :

Les performances des fonctions implémentées sont mesurées en deux temps , d'abord en utilisant la structure *double* en *Java*, ensuite en utilisant la structure *float*.

Le temps de calcul est mesuré à l'aide de la méthode *Sytem.nanoTime().* Afin d'être le plus précis possible, on mesure pour un grand nombre d'échantillons (100 000) de 100 entrées le temps nécessaire pour le calcul de l'une des fonctions, et on prend après l'espérance (la moyenne arithmétique), ceci donne le temps pour 100 entrées, ce temps est ensuite divisé par 100 pour obtenir le temps pour une seule entrée.

La précision est assurée à l'aide de la fonction *assertEquals* de la classe *Assertions* .

2-Tableau de performances du CORDIC :

2.1- Structure double :

Le tableau suivant est pour l'implémentation avec la structure *double*.

Fonction	Temps pour une entrée (en ms)	Temps pour 100 entrées (en ms)	Précision
cos	0,00841 ms	0,095 ms	1E-13
sin	0,00800 ms	0,093 ms	1E-13
atan	0,00197 ms	0,082 ms	1E-13
asin	0,00215 ms	0,113 ms	1E-13 *
ulp	0,00010 ms	0,003ms	1E-5

* La précision E-13 de l'*asin* est pour x tel que |x|<0.975, pour x tel que |x|>0.975 la précision diminue tant que |x| -> 1

2.1- Structure float :

Le tableau qui suit est selon l'implémentation avec la structure *float.*

Fonction	Temps pour une entrée (en ms)	Temps pour 100 entrées (en ms)	Meilleur précision
cos	0,00162 ms	0,081 ms	1E-6
sin	0,00084 ms	0,084 ms	1E-6
atan	0,00264 ms	0,264 ms	1E-7
asin	0,00246 ms	0,246 ms	1E-6
ulp	0,00005 ms	0,005 ms	1E-8

* La précision E-6 de l'*asin* est pour x tel que |x|<0.917, pour x tel que |x|>0.917 la précision diminue en s'approchant de 1. (Plus de détail dans le tableau ci-dessous)

*ulp nous garantit une précision de E-6 dans tous les cas, et une précision de E-8 dès que |x|>1

3-Précision détaillée des différentes fonctions :

- cos et sin :

  La précision de ces deux fonctions trigonométriques dépend de l'intervalle de départ :

  - : La précision est de 1E-6
  - $\mathbb{R} \setminus [-9\pi, 9\pi]$: La précision oscille entre 1E-6 ET 1 E-5

  Explication et idées pour amélioration :

  Les fonctions cos et sin sont **2π-**périodiques, et comme l'implémentation repose sur les égalités trigonométriques entre le cos et le sin pour prolonger les deux fonctions sur tout l'axe réel, l'imprécision vient donc de la fonction « *_anglePrincipal* » . *Comme piste d'amélioration, on pourra au lieu de faire une suite de soustraction dans une boucle while, d'abord localiser l'angle d'entrée dans un intervalle de la forme* [**kπ**, **(k+1)π**]*, et économiser une suite de soustractions à une multiplication et une addition en plus de comparaisons (ces dernières ne représentent pas une source d'imprécision).*

  ⇨ Après implémentation de l'idée ci-dessus, malheureusement ça n'a pas été suffisant pour augmenter la précision en dehors de [**-9π**, **9π**] .

- atan :
  - **[-0.274 ,0.274] :** La précision est de 1E-7
  - $\mathbb{R} \setminus$ **[-0.274 ,0.274]** : La précision est 1E-6

- asin :

  La précision l'*asin* pour x tel que |x|<0.917 est 1E-6, ensuite pour x tel que |x|>0.917 la précision diminue en s'approchant de 1. (Plus de détail dans le tableau ci-dessous). On se contente des intervalles positifs ; la précision est pareille pour **[-1,0].**

Intervalle	**[0,0.915]**	**[0.915,0.943]**	**[0.943,0.965]**	**[0.965,0.983]**	**[0.983,0.995]**	**[0.995,1]**
Précision	1E-6	1E-5	1E-4	1E-3	1E-2	1E-1

- ulp :

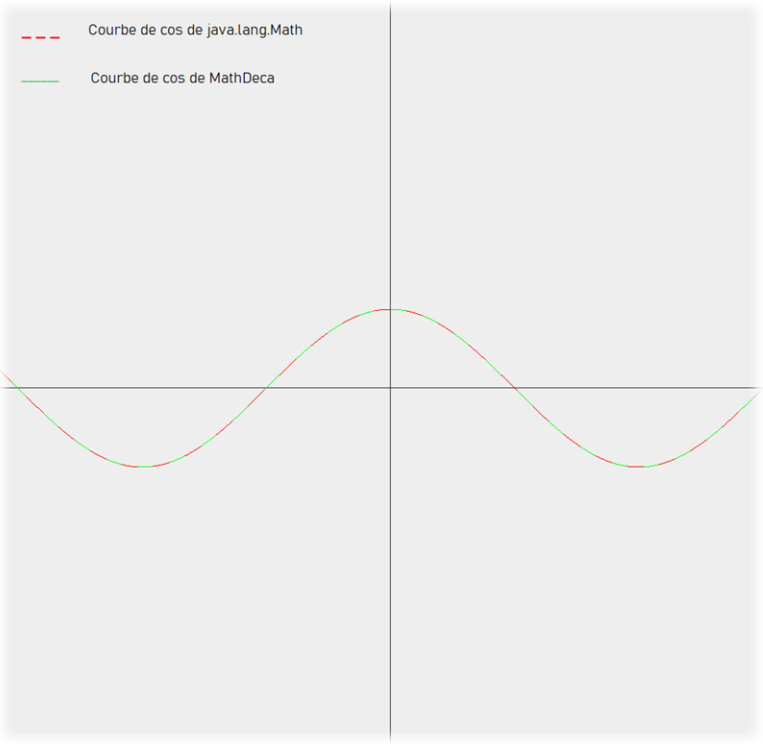
  ulp nous garantit une précision de E-8 dès que |x|>= 1 , et une précision qui varie entre E-6 et E-7 dans]-1,1[

Intervalle	**] -∞ ,-1]**	**]-1,-0.25]**	**[-0.25,0.25]**	**[0.25,1[**	**[1,+∞ [**
Précision	1E-8	1E-7	1E-6	1E-7	1E-8

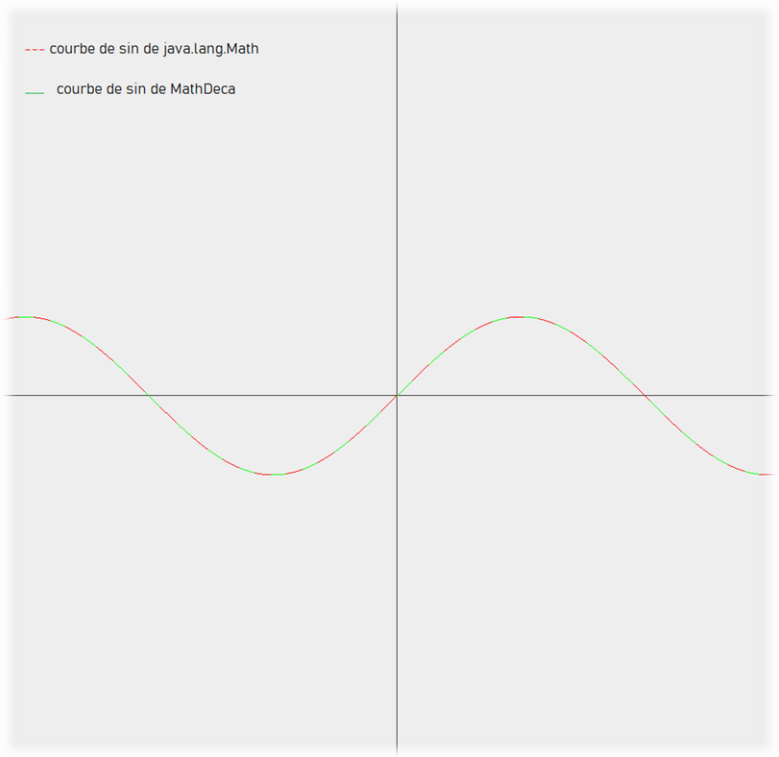
3-Courbe des fonctions:

Les courbes ont été tracé à l'aide des classes de java.awt (abstract window toolkit). On a dû faire notre propre traceur de fonctions, l'unité de ce dernier était de 100 pixels (c'est-à-dire chaque 1 pixel = 0.01). Le traçage est fait sur l'intervalle **[-10,10]** = 1000 pixels pour les deux axes. Les courbes en - - - rouge tiret sont les fonctions de la bibliothèque Math de Java, et en —— **vert continu** les fonctions de notre bibliothèque.

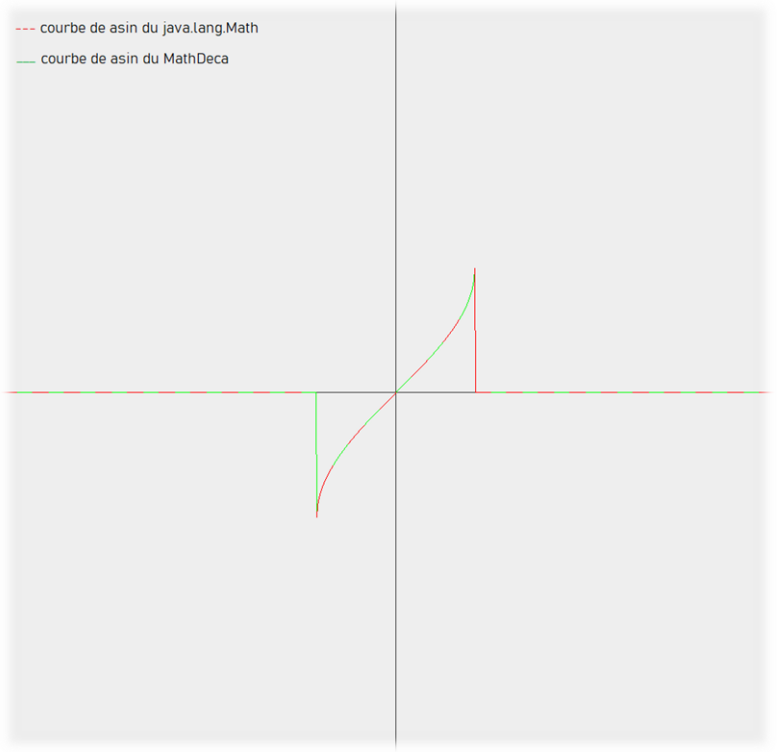
- Fonction cosinus



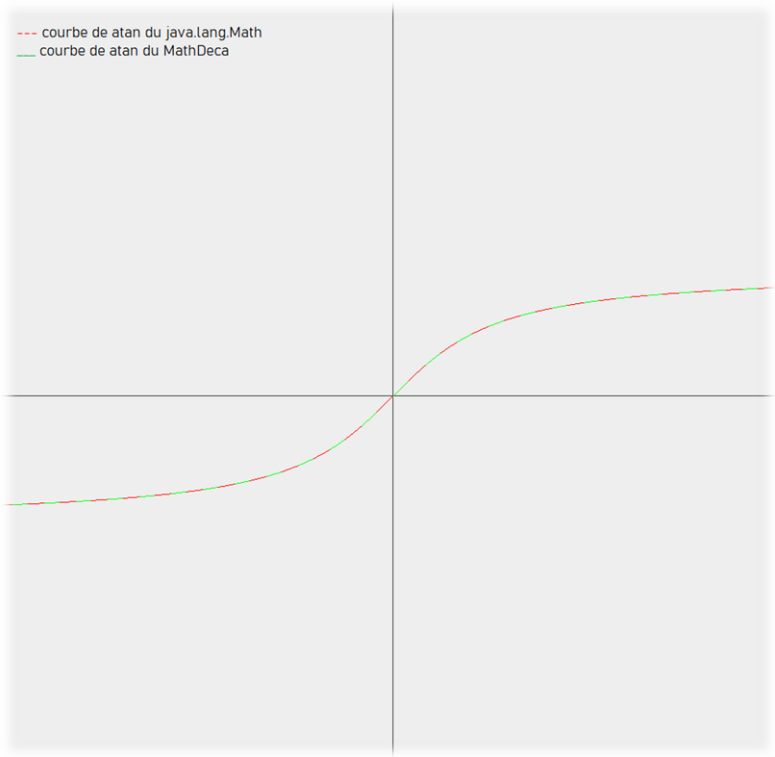
- Fonction sinus



- Fonction tangente



- Fonction arcsinus :



Commentaire :

Les fonctions coïncident parfaitement, et on ne peut distinguer avec l'œil nue la différence entre les deux courbes, c'est d'ailleurs la raison pour laquelle on a décidé de tracer les courbes de la bibliothèques Math de Java en tiret.