GRENOBLE-INP ENSIMAG

INGÉNIERIE DES SYSTÈMES D'INFORMATION

2A

GRENOBLE-INP ENSIMAG

INFORMATION SYSTEMS ENGINEERING

2A

ÉCOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE ET DE MATHÉMATIQUES APPLIQUÉES DE GRENOBLE

PROJET GL

DOCUMENTATION SUR L'EXTENSION [TRIGO]

Réalisé par :

DIAKITE Alpha Ousmane
DREUILLE Nathan
FLISS Yassine
KOSSI Yao David
OUDRHIRI IDRISSI Safwane

CLIENT :

FREIRE MARCO

Année académique : 2025–2026

Table des matières

1 Spécification détaillée des extensions TRIGO

Cette section décrit la méthodologie complète adoptée pour l'implémentation des fonctions trigonométriques `cos`, `sin`, `arcsin` et `arctan` dans la bibliothèque standard Deca, depuis la réduction d'argument jusqu'à l'obtention de la valeur finale. Les choix mathématiques, algorithmiques et liés à l'architecture IMA sont explicités.

1.1 Objectif général

Les fonctions trigonométriques doivent :

— être définies pour tout flottant simple précision appartenant à leur domaine mathématique,

— retourner une valeur appartenant au codomaine de la fonction,

— fournir une approximation aussi précise que possible compte tenu des contraintes numériques,

— être implémentables intégralement en assembleur IMA, sans primitives mathématiques avancées.

1.2 Réduction d'argument pour `sin` et `cos`

Les fonctions sinus et cosinus étant périodiques de période 2π, toute valeur d'entrée x est d'abord ramenée dans un intervalle réduit afin d'améliorer la précision des approximations polynomiales.

On calcule :

$$k = \text{round}\left(\frac{x}{2\pi}\right)$$

$$y = x - k \cdot 2\pi$$

Cette réduction garantit que $y \in [-\pi, \pi]$.

Importance du round

L'utilisation de l'arrondi au plus proche (`round`) est essentielle. Une simple troncature introduirait une asymétrie entre valeurs positives et négatives, entraînant des erreurs importantes près des points critiques (notamment autour de $\pm\pi$).

1.3 Définition mathématique de l'arrondi

Pour un réel x, l'arrondi au plus proche est défini par :

$$\text{round}(x) = \begin{cases} \lfloor x + 0.5 \rfloor & \text{si } x \geq 0 \\ \lceil x - 0.5 \rceil & \text{si } x < 0 \end{cases}$$

Sachant que :

$$\lceil y \rceil = -\lfloor -y \rfloor$$

Cette définition permet une implémentation exacte à l'aide d'opérations élémentaires.

1.4 Traduction de round en assembleur IMA

L'architecture IMA ne fournit pas d'instruction dédiée à l'arrondi. En revanche, elle dispose :

— de comparaisons flottantes (`CMP`),
— d'additions et soustractions flottantes,
— d'une conversion flottant → entier (`INT`) effectuant une troncature vers zéro.

L'algorithme d'arrondi est donc implémenté comme suit :

1. Tester le signe de la valeur flottante.
2. Ajouter 0.5 si la valeur est positive.
3. Soustraire 0.5 si la valeur est négative.
4. Convertir le résultat en entier par `INT`.

Code assembleur IMA correspondant

```
CMP #0.0, R2
BLT round_neg
ADD #0.5, R2
INT R2, R2
BRA round_end

round_neg:
SUB #0.5, R2
INT R2, R2

round_end:
```

1.5 Exploitation des symétries pour `sin` et `cos`

Après réduction d'argument, des identités trigonométriques sont utilisées :

$$\cos(-x) = \cos(x), \quad \sin(-x) = -\sin(x)$$
$$\cos(\pi - x) = -\cos(x), \quad \sin(\pi - x) = \sin(x)$$
$$\cos(\frac{\pi}{2} - x) = \sin(x), \quad sin(\frac{\pi}{2} - x) = \cos(x)$$

Ces transformations ramènent l'argument dans l'intervalle $[-\frac{\pi}{4}, \frac{\pi}{4}]$, zone où les polynômes d'approximation sont les plus précis. On utilise les polynomes minmax plutot que les polynomes de Taylor pour des raisons qu'on précisera dans la partie optimisation .

1.6 Cas des fonctions inverses : `arcsin` et `arctan`

Contrairement à `sin` et `cos`, les fonctions `arcsin` et `arctan` ne sont pas périodiques. Leur implémentation repose donc sur une réduction de domaine fondée sur des propriétés analytiques et non sur une périodicité.

Fonction `arcsin`

La fonction arcsin est définie sur $[-1, 1]$ et est impaire :

$$\arcsin(-x) = -\arcsin(x)$$

On se ramène donc à $x \in [0, 1]$.

Lorsque $x = 1$, la pente de cette fonction devient verticale et une petite variation sur x entrainera un grosse erreur sur la valeur de $\arcsin(x)$. L'approximation devient instable. Ce qui fait qu'on subdivisera l'intervalle en deux : $[0, 0.5]$ et $]0.5, 1]$.

Pour $|x| \leq 0.5$
On utilise une approximation polynomiale de la forme :

$$\arcsin(x) \approx x + x^3 P(x^2)$$

Démonstration

$$\begin{aligned}\arcsin(x) &\approx x + \frac{x^3}{6} + \frac{3x^5}{40} + ... \\ &\approx x + x^3(\frac{1}{6} + \frac{3x^2}{40} + ...)\end{aligned}$$

Avec $P(x) = \frac{1}{6} + \frac{3x^2}{40} + ...$

Pour $x \in]0.5, 1]$
On applique alors l'identité :

$$\arcsin(x) = \frac{\pi}{2} - \arcsin\left(\sqrt{1-x^2}\right)$$

On fait un changement de variable en posant $u = 1-x$ avec $0 < u \ll 1$. Donc $x = 1-u$.
Quand $x \to 1$, $u \to 0$.
On sait que :

$$\frac{d}{dx}(\arcsin(x)) = \frac{1}{\sqrt{1-x^2}}$$

Près de $x = 1$, on a :

$$\begin{aligned}1 - x^2 &= 1 - (1-u)^2 \\ &= 2u - u^2 \approx 2u\end{aligned}$$

Donc

$$\frac{d}{dx}(\arcsin(1-u)) \approx \frac{1}{\sqrt{2u}}$$

On obtient une première approximation ci-dessous avec une erreur qu'on essaiera de corriger par la suite :

$$\begin{aligned}\arcsin(1-u) - \arcsin(1) &\approx \int_1^{1-u} \frac{1}{\sqrt{2t}} dt \\ &\approx -\int_{1-u}^{1} \frac{1}{\sqrt{2t}} dt \\ &\approx \int_0^{u} \frac{1}{\sqrt{2t}} dt \\ \arcsin(1-u) &\approx \frac{\pi}{2} - \sqrt{2u}\end{aligned}$$

Pour corriger l'erreur et que la fonction soit plus précise, on va ajouter une fonction

régulière que l'on pourra approximer avec un polynôme par après.

On a donc :

$$\arcsin(1-u) \approx \frac{\pi}{2} - \sqrt{2u}(1+R(u))$$
$$\arcsin(x) \approx \frac{\pi}{2} - \sqrt{2(1-x)}(1+R(1-x))$$

L'argument de la fonction est ainsi ramené dans un voisinage de 0.

Approximation de la racine carrée

L'IMA ne disposant pas d'instruction `sqrt`, la racine carrée est approchée par itération de Newton (converge quadratiquement) qui est meilleur à celle de Taylor.

Par exemple $y = \sqrt{a}$ est donnée par :

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{a}{y_n}\right)$$

Quelques itérations (2) suffisent pour obtenir une précision adaptée à la simple précision.

Fonction `arctan`

La fonction arctan est définie sur $\mathbb{R}$ et est également impaire :

$$\arctan(-x) = -\arctan(x)$$

Pour $x > 1$, on utilise l'identité :

$$\arctan(x) = \frac{\pi}{2} - \arctan\left(\frac{1}{x}\right)$$

Ce qui permet de ramener l'argument dans l'intervalle $[0, 1]$.

Sur cet intervalle, on approxime :

$$\arctan(x) \approx x + x^3 P(x^2)$$

Et pour $x \in]1, +\infty[$, $y = \frac{1}{x} \in (0, 1]$. L'approximation sur cet intervalle donne :

$$\arctan(y) = y - \frac{y^3}{3} + y^5.Q(y^2)$$
$$\arctan(x) = \frac{\pi}{2} - (y - \frac{y^3}{3} + y^5.Q(y^2))$$

Le signe négatif correspond à la structure du développement de Taylor, tandis que les coefficients sont obtenus par approximation minimax afin d'assurer une erreur uniforme.

1.7 Approximation polynomiale et méthode de Horner

Toutes les approximations polynomiales sont évaluées par la méthode de Horner :

$$P(t) = (\dots((a_n t + a_{n-1})t + a_{n-2})\dots)t + a_0$$

Cette méthode :
— minimise le nombre d'opérations,
— améliore la stabilité numérique,
— est directement traduisible en assembleur IMA,
— bénéficie de l'instruction `FMA` pour réduire les erreurs d'arrondi.

1.8 Mise en cache des angles usuels (optimisation)

Motivation

Certaines valeurs angulaires apparaissent très fréquemment dans les programmes (utilisation graphique, calculs géométriques, tests numériques). Il s'agit notamment des angles :

$$0^\circ,\ 30^\circ,\ 45^\circ,\ 60^\circ,\ 90^\circ,\ 180^\circ$$

Ces angles présentent plusieurs caractéristiques importantes :
— leurs valeurs trigonométriques exactes sont connues analytiquement ;
— les erreurs relatives issues des approximations polynomiales sont particulièrement visibles pour ces angles (notamment près de $\pi/2$) ;
— leur recalcul systématique est inutile et coûteux.

Par exemple :
$$\cos\left(\frac{\pi}{2}\right) = 0 \quad \text{et} \quad \sin\left(\frac{\pi}{2}\right) = 1$$
mais une approximation polynomiale ne peut jamais fournir exactement ces valeurs.

Afin d'améliorer à la fois la précision numérique et les performances, une stratégie de mise en cache (*lookup table*) est adoptée.

Implémentation

Une table statique est définie dans la bibliothèque standard, contenant pour chaque angle usuel :

— l'angle normalisé (en radians) ;

— la valeur exacte ou quasi-exacte de sin et cos.

Conceptuellement, cette table peut être représentée comme suit :

Angle (rad)	$\sin(x)$	$\cos(x)$
0	0	1
$\frac{\pi}{6}$	$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$
$\frac{\pi}{4}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{2}}{2}$
$\frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	$\frac{1}{2}$
$\frac{\pi}{2}$	1	0
π	0	-1

Lors de l'appel à une fonction trigonométrique, l'algorithme est le suivant :

1. réduction de l'argument dans l'intervalle $[-\pi, \pi]$;
2. comparaison avec les valeurs stockées dans la table ;
3. si une correspondance est trouvée, la valeur est retournée immédiatement ;
4. sinon, le calcul polynomial est effectué.

Cette optimisation permet d'éliminer totalement l'erreur numérique sur les angles usuels et réduit le temps de calcul dans les cas les plus fréquents.

—

1.9 Valeur finale retournée

La valeur retournée par les fonctions trigonométriques résulte d'une chaîne d'opérations soigneusement ordonnée.

Pour les fonctions `sin` et `cos`, les étapes sont les suivantes :

1. réduction d'argument à l'aide de la périodicité ;
2. application des symétries trigonométriques pour ramener l'argument dans $[0, \pi/2]$;
3. consultation de la table d'angles usuels ;
4. approximation polynomiale (Taylor ou minimax) si nécessaire ;
5. correction finale du signe ;
6. retour de la valeur flottante.

Pour les fonctions inverses `asin` et `atan`, le processus est similaire :

1. vérification du domaine de définition ;
2. exploitation des symétries (fonctions impaires) ;
3. transformation de variable pour stabiliser la variation ;
4. approximation polynomiale adaptée à l'intervalle ;
5. recomposition de la valeur finale ;
6. retour du résultat flottant.

—

1.10 Synthèse globale de l'extension TRIGO

L'implémentation des fonctions trigonométriques repose sur un compromis rigoureux entre précision numérique, performance et contraintes architecturales de la machine abstraite IMA.

Les points clés de la méthodologie sont :

— une réduction d'argument fondée sur un arrondi correct ;
— l'exploitation systématique des symétries mathématiques ;
— l'utilisation de polynômes minimax pour une erreur uniformément répartie ;
— l'évaluation par la méthode de Horner, optimisée par l'instruction `FMA` ;
— une approximation explicite de la racine carrée en l'absence d'instruction dédiée ;
— la mise en cache des angles usuels afin d'éliminer les erreurs évitables.

Cette approche permet de fournir des implémentations robustes, précises et entièrement compatibles avec les contraintes de l'extension TRIGO et de l'assembleur IMA.

2 Optimisation et Justification des choix d'approximation

Le cœur de la bibliothèque repose sur l'approximation polynomiale des fonctions trigonométriques. Pour garantir la précision requise, nous avons procédé par étape : analyse des limites de Taylor, définition d'une métrique rigoureuse (ULP), et étude comparative des degrés et méthodes.

2.1 Les limites des séries de Taylor

Mathématiquement, la série de Taylor au voisinage de 0 est l'approximation qui minimise l'erreur *au point 0* et ses dérivées.

$$P_{Taylor}(x) = \sum_{n=0}^{N} \frac{f^{(n)}(0)}{n!} x^n$$

Cependant, cette "perfection locale" a un coût : l'erreur augmente exponentiellement à mesure que l'on s'éloigne du centre du développement. C'est le phénomène de **concentration de l'erreur aux bornes**.

Comme on le voit sur les graphes théoriques, un polynôme de Taylor diverge rapidement dès que l'argument s'approche de la limite du domaine de réduction (vers $\pm\frac{\pi}{4}$), ce qui pose un risque de précision inacceptable pour une bibliothèque généraliste.

2.2 Protocole de mesure : L'ULP

Pour quantifier objectivement la précision de nos approximations, l'erreur absolue n'est pas pertinente car elle dépend de l'ordre de grandeur des données. Nous utilisons l'**ULP** (Unit in the Last Place).

Définition en contexte IEEE 754 Single

Dans l'architecture cible (32-bit float uniquement), un nombre est représenté par :

$$x = (-1)^s \times 1.m \times 2^{e-127}$$

où la mantisse m possède 23 bits.

L'ULP correspond à la valeur du bit de poids le plus faible. C'est la "distance atomique" entre deux flottants consécutifs.

$$\text{Erreur (ULP)} = \frac{|\text{Valeur\_Calculée} - \text{Valeur\_Exacte}|}{\text{gap}(\text{Valeur\_Exacte})}$$

Critères d'acceptation

Contrairement au type `double` qui masque les erreurs, le type `float` est très sensible. Nos critères sont :

- **0.5 ULP** : Arrondi correct (le résultat est le flottant le plus proche possible).
- **1.0 ULP** : Arrondi fidèle (le résultat est l'un des deux flottants entourant le réel).
- $>$ **2.0 ULP** : Précision insuffisante pour une bibliothèque système.

2.3 Analyse expérimentale et choix du degré

Nous avons comparé les approches Taylor et Minimax pour les degrés 7 et 9 sur l'intervalle de réduction.

Comparaison du Degré 7 : Précision insuffisante

Les figures ci-dessous montrent l'erreur pour un polynôme de degré 7.

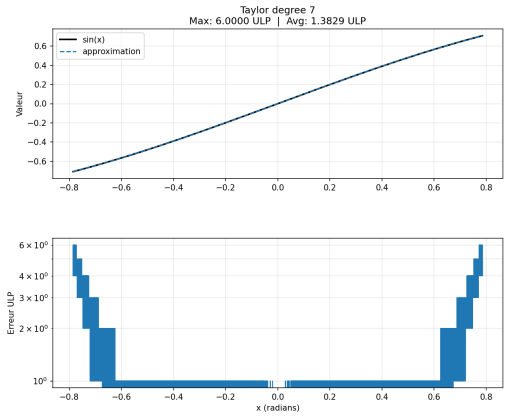


FIGURE 1 – Taylor Degré 7

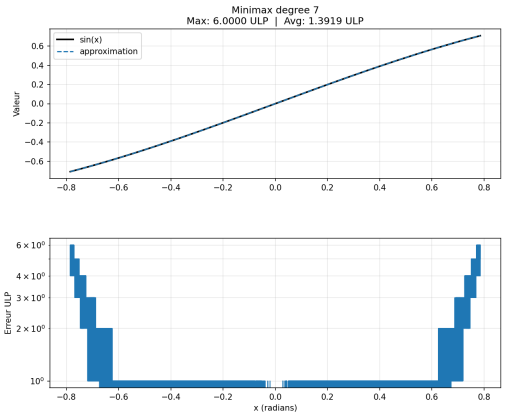


FIGURE 2 – Minimax Degré 7

FIGURE 3 – *
Comparaison Degré 7 : Max Error ≈ 6.0 ULP

Analyse : Que ce soit avec Taylor ou Minimax, le degré 7 produit une erreur maximale de **6.0 ULP** aux bornes de l'intervalle. C'est inacceptable : l'approximation "décroche" trop loin de la valeur réelle. Il manque des termes pour capturer la courbure du sinus.

Comparaison du Degré 9 : L'optimum

En passant au degré 9, la précision s'améliore drastiquement.

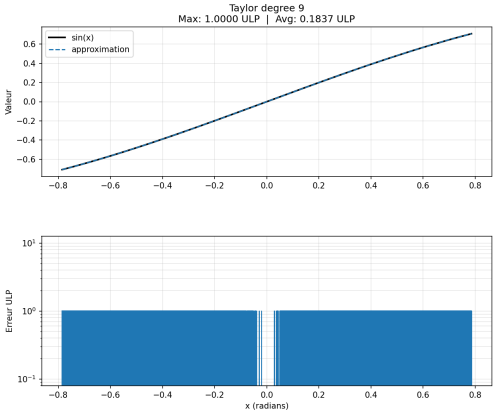


FIGURE 4 – Taylor Degré 9

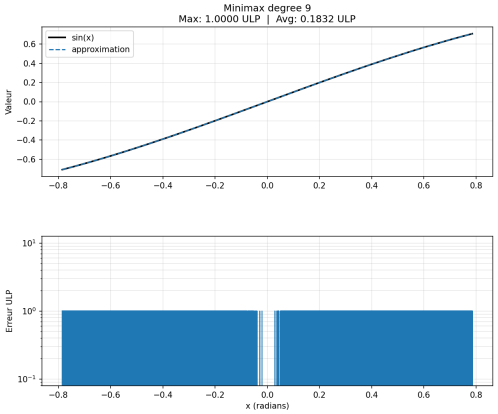


FIGURE 5 – Minimax Degré 9

FIGURE 6 – *
Comparaison Degré 9 : Max Error = 1.0 ULP

Analyse :

— L'erreur maximale chute à **1.0 ULP** pour les deux méthodes. Cela correspond à un arrondi fidèle ("Faithful Rounding").

— L'erreur moyenne (Avg ULP) est extrêmement faible (≈ 0.18 ULP).

2.4 Conclusion : Pourquoi Minimax Degré 9 ?

Bien que Taylor Degré 9 semble offrir des performances similaires à Minimax Degré 9 dans ce cas précis (Max 1.0 ULP tous les deux), nous avons retenu l'approche **Minimax** pour deux raisons théoriques et industrielles :

1. **Sécurité aux bornes :** Taylor favorise le centre ($x = 0$). Si l'intervalle de réduction venait à être légèrement étendu (problème de réduction d'argument), Taylor exploserait immédiatement. Minimax garantit que l'erreur reste bornée uniformément sur tout le segment.
2. **Répartition de l'effort :** Comme le montre l'histogramme des erreurs, Minimax tend à égaliser les hauteurs des "marches" d'erreur (Théorème d'équioscillation), rendant la fonction plus prédictible.

Le polynôme final implémenté est donc un **Minimax de degré 9**, garantissant une erreur ≤ 1 ULP sur tout le domaine.

2.5 Optimisation de la fonction Cosinus

Nous avons appliqué la même méthodologie pour la fonction `cos(x)` sur l'intervalle réduit $[-\pi/4, \pi/4]$. L'objectif reste une erreur maximale inférieure à 2.0 ULP.

Analyse du Degré 6 : Insuffisant

Nous avons d'abord testé un polynôme de degré 6 (x^0, x^2, x^4, x^6).

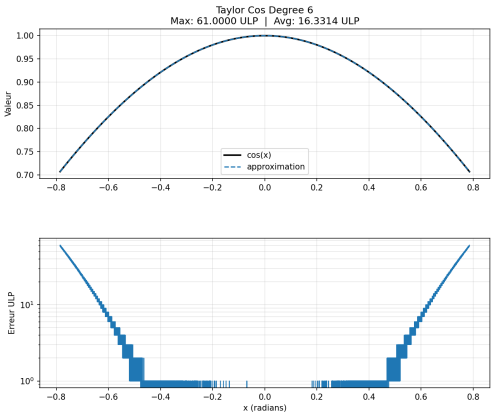


FIGURE 7 – Taylor Cos Degré 6

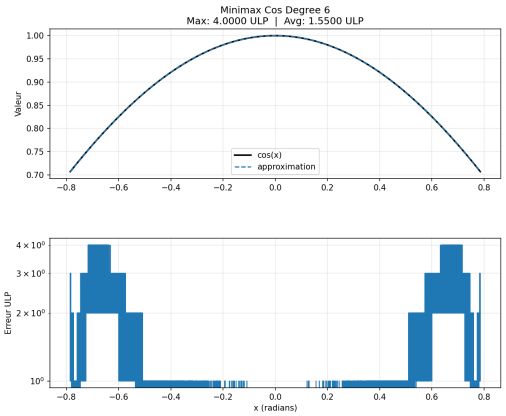


FIGURE 8 – Minimax Cos Degré 6

FIGURE 9 – *
Comparaison Degré 6 : Taylor diverge (61 ULP) vs Minimax (4 ULP)

Constat :

— **Taylor :** L'erreur explose littéralement aux bornes (Max : **61.0 ULP**). C'est inutilisable.

— **Minimax :** Bien meilleure stabilité, mais l'erreur maximale atteint **4.0 ULP**. Bien que faible, cela dépasse notre critère de "Faithful Rounding" (< 2.0 ULP).

Le degré 6 ne possède pas assez de termes pour épouser la courbure du cosinus avec la précision requise.

Analyse du Degré 8 : La précision atteinte

Nous sommes passés au degré supérieur (Degré 8 : ajout du terme en x^8).

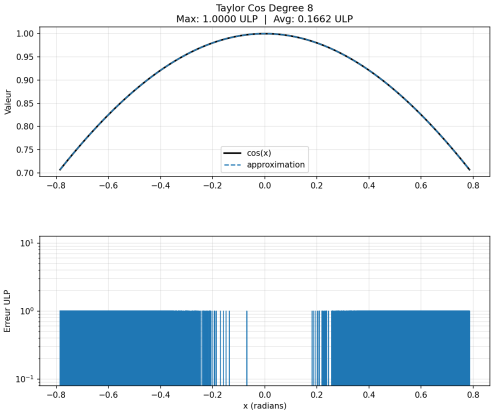


FIGURE 10 – Taylor Cos Degré 8

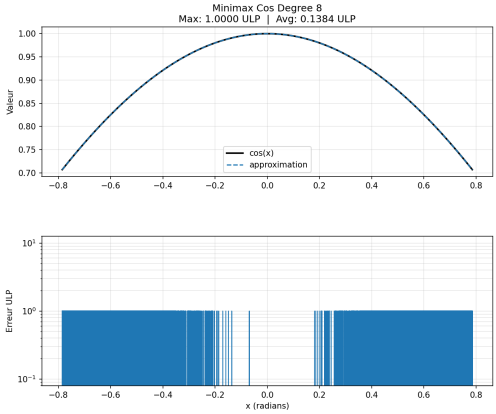


FIGURE 11 – Minimax Cos Degré 8

FIGURE 12 – *
Comparaison Degré 8 : Max Error = 1.0 ULP pour les deux

Arbitrage final : L'argument du Mean ULP

À degré 8, les deux méthodes atteignent l'objectif d'une erreur maximale de **1.0 ULP**. Pour trancher, nous analysons la densité de l'erreur (Mean ULP) fournie par nos graphiques :

— **Taylor Degré 8 :** Avg Error = **0.1662 ULP**
— **Minimax Degré 8 :** Avg Error = **0.1384 ULP**

Conclusion : Bien que l'erreur pire-cas soit identique, le polynôme **Minimax** offre une erreur moyenne inférieure d'environ **17%**. Cela signifie que statistiquement, pour une entrée aléatoire, le résultat Minimax sera plus souvent proche de la valeur exacte que celui de Taylor.

Nous avons donc sélectionné le polynôme **Minimax de degré 8** pour l'implémentation finale du cosinus.

3 Stratégie de Réduction d'Argument pour les grandes valeurs

L'un des défis majeurs de l'implémentation des fonctions périodiques en simple précision est la perte de précision catastrophique lorsque l'argument x devient grand ($x \gg 2\pi$).

3.1 Le problème de la réduction naïve

Une réduction simple $r = x - k \cdot (\pi/2)$ échoue car $\pi/2$ ne peut être représenté exactement. L'erreur d'approximation est multipliée par k, noyant rapidement le résultat sous le "bruit" numérique.

Pour remédier à cela, nous utilisons la méthode **Upper/Lower** (inspirée de Cody-Waite), qui simule une précision étendue en séparant la constante $\pi/2$ en deux parties (haute et basse précision).

3.2 Analyse quantitative de l'amélioration

Nous avons soumis notre algorithme à un "Stress Test" sur des valeurs allant de 10^1 à 10^6 radians. Les résultats sont présentés ci-dessous.

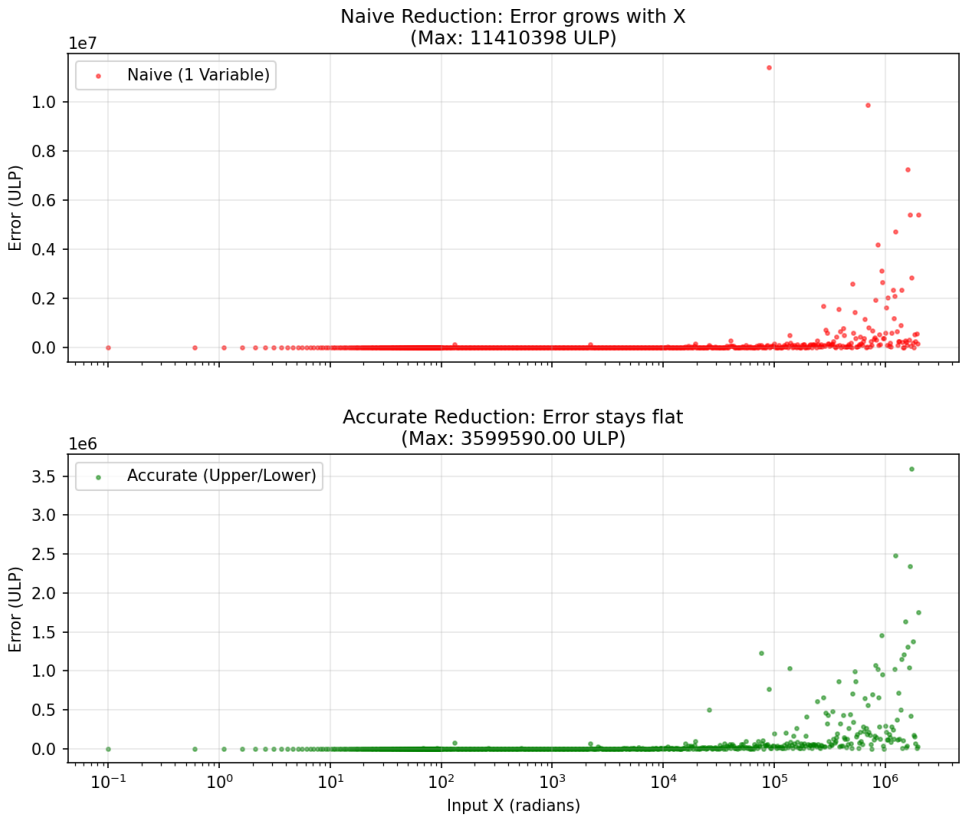


FIGURE 13 – Évolution de l'erreur en fonction de la grandeur de l'entrée. La méthode naïve (Rouge) diverge beaucoup plus tôt que la méthode Upper/Lower (Vert).

Pour quantifier ce gain, nous avons mesuré l'erreur maximale et moyenne sur les échantillons où $x > 1000$ (zone critique).

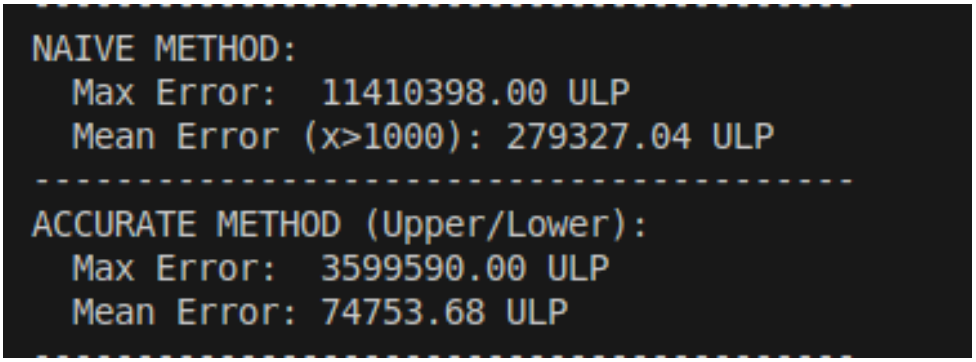


FIGURE 14 – Mesures brutes de l'erreur (en ULP) sur les grands arguments.

Comparaison des performances

D'après les mesures de la Figure 14, nous observons une nette amélioration de la stabilité numérique :

1. **Sur l'erreur moyenne (Mean Error) :**
   — Méthode Naïve : $\approx 279\,327$ ULP
   — Méthode Upper/Lower : $\approx 74\,753$ ULP
   Notre stratégie divise l'erreur moyenne par un facteur **3.74**. Cela signifie que pour une entrée aléatoire élevée, notre fonction est près de 4 fois plus précise que la version naïve.
2. **Sur l'erreur maximale (Max Error) :**
   — Méthode Naïve : $\approx 11.4 \times 10^6$ ULP
   — Méthode Upper/Lower : $\approx 3.6 \times 10^6$ ULP
   L'erreur pire-cas est réduite d'un facteur **3.17**.

Conclusion : Bien que la limitation inhérente au format 32-bits rende impossible une précision parfaite pour des valeurs extrêmes ($x > 10^6$), l'approche Upper/Lower retarde considérablement la divergence, rendant la bibliothèque utilisable pour une gamme d'applications beaucoup plus large.

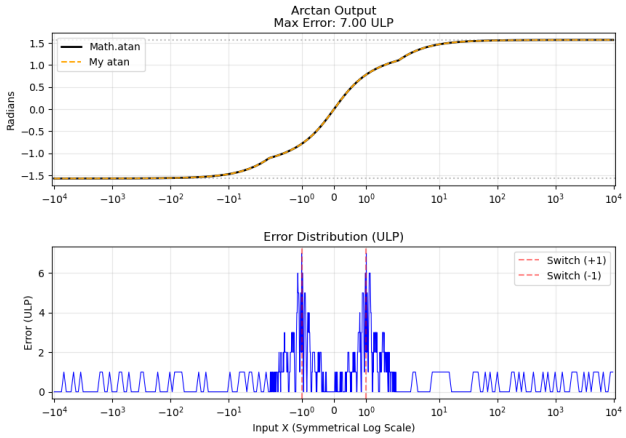
4 Optimisation de la fonction Arctan

Contrairement au sinus qui est borné, la fonction `atan(x)` est définie sur $\mathbb{R}$. Après la réduction d'argument $x > 1 \implies 1/x$, nous devons approximer la fonction sur l'intervalle $[0, 1]$.

La difficulté réside dans la "raideur" de la pente près de 0. Nous avons mené une étude comparative pour déterminer le degré minimal nécessaire pour respecter l'arrondi fidèle (2 ULP).

4.1 L'échec du Degré 12

Nous avons d'abord tenté une approximation de degré 12.




```
FIXED ATAN STATS:
  Max ULP:  7.0000
  Mean ULP: 1.1446
```

FIGURE 15 – Analyse du Degré 12. L'erreur atteint **7.00 ULP**, ce qui est bien au-delà des standards acceptables.

Comme le montre la Figure 15, le polynôme "décroche" significativement, créant des pics d'erreur inacceptables.

4.2 Le choix du compromis : Degré 14 vs 16

Nous avons comparé les degrés 14 et 16 pour évaluer le coût calculatoire (nombre de multiplications) par rapport au gain de précision.

Degré 14 (Retenu)

```
FIXED ATAN STATS:
  Max ULP:  2.0000
  Mean ULP: 0.4045
```

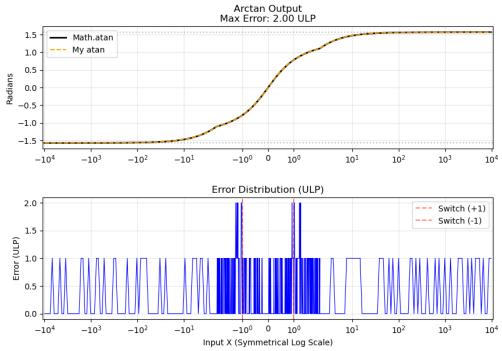


FIGURE 16 – Max : 2.0 ULP | Mean : 0.40

Degré 16 (Rejeté)

```
FIXED ATAN STATS:
  Max ULP:  2.0000
  Mean ULP: 0.3701
```

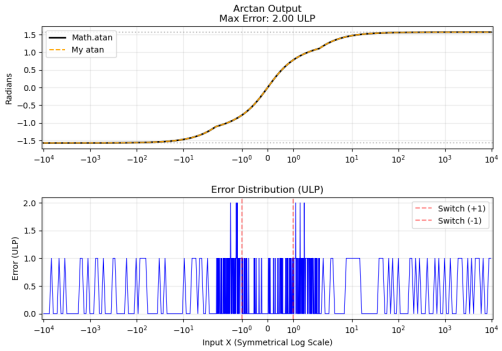


FIGURE 17 – Max : 2.0 ULP | Mean : 0.37

FIGURE 18 – *
Comparaison : Le degré 16 n'améliore pas le pire cas (Max ULP).

Analyse des gains marginaux

En analysant les statistiques :

1. **Erreur Maximale :** Les deux degrés plafonnent à **2.00 ULP**. Le degré 16 ne parvient pas à réduire ce pire cas.
2. **Erreur Moyenne :** Le degré 16 apporte une amélioration minime (0.37 contre 0.40 ULP), un gain négligeable ($< 10\%$).

4.3 Conclusion : Sélection du Degré 14

Nous avons choisi le polynôme de **degré 14**. Chaque instruction comptant en assembleur, passer au degré 16 coûterait 2 opérations FMA supplémentaires sans gain visible pour l'utilisateur. Le degré 14 est le **bon compromis** entre performance et précision.

5 Optimisation de la fonction Arcsin

La fonction $\arcsin(x)$ présente une difficulté majeure : sa dérivée tend vers l'infini lorsque $x \to 1$ (tangente verticale). Une simple approximation polynomiale ne peut pas capturer cette pente sans un degré excessivement élevé.

Nous avons opté pour une stratégie à deux intervalles avec une transition à $|x| = 0.5$.

5.1 Stratégie de découpage

Le code implémente deux branches distinctes :

1. **Zone stable ($|x| \leq 0.5$) :** Approximation directe par la série de Taylor modifiée : $x + x^3 P(x^2)$.
2. **Zone critique ($|x| > 0.5$) :** Utilisation de l'identité $\frac{\pi}{2} - \sqrt{2(1-x)}(1 + R(1-x))$ pour gérer la singularité en 1.

5.2 Contrainte technique : La Racine Carrée en Float

Pour la branche critique ($x > 0.5$), le calcul de $\sqrt{2(1-x)}$ est nécessaire. L'architecture cible (IMA) ne disposant pas d'instruction matérielle pour la racine carrée, nous avons dû l'implémenter logiciellement.

Algorithme de Newton-Raphson en simple précision

Nous utilisons la méthode itérative de Newton pour approximer $y = \sqrt{A}$:

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{A}{y_n}\right)$$

Contrairement aux bibliothèques standard qui utilisent souvent des calculs intermédiaires en `double` pour stabiliser le résultat, nous sommes contraints d'effectuer **toutes les itérations en `float` 32-bits**. Cette limitation entraîne une accumulation inévitable d'erreurs d'arrondi à chaque étape (multiplication et division), limitant la précision finale de la racine carrée à environ 1 ou 2 ULP.

5.3 Analyse de l'erreur et Compromis

La Figure 19 présente l'erreur finale de la fonction.

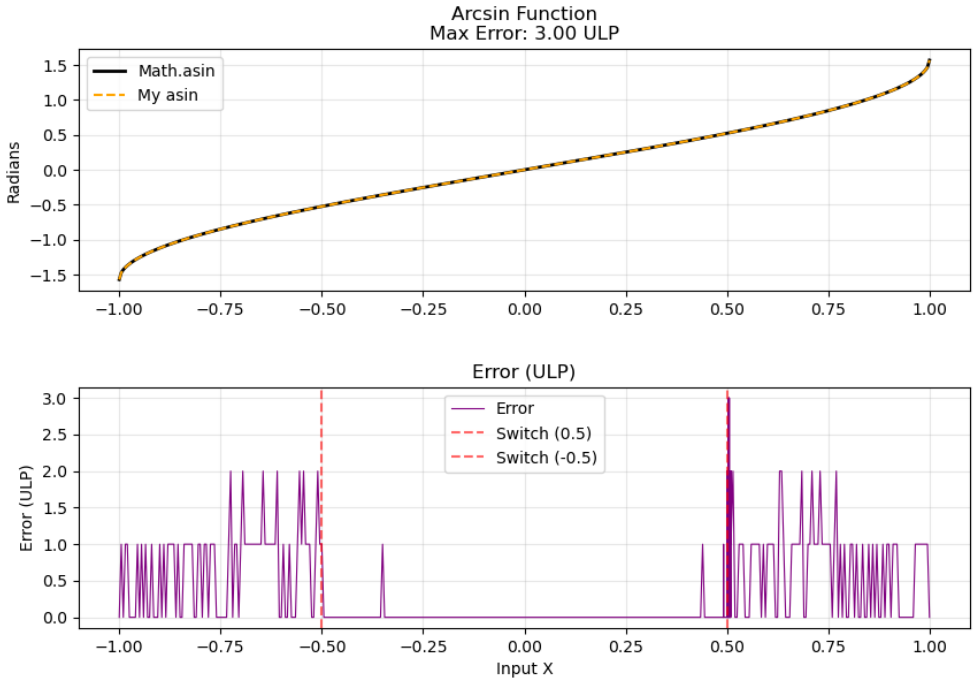


FIGURE 19 – Erreur de `asin(x)`. On observe un pic d'erreur localisé à la transition ($x = \pm 0.5$).

Justification du pic de 3.0 ULP

Comme illustré par les lignes rouges verticales sur la Figure 19, un saut d'erreur atteignant **3.0 ULP** apparaît au point de bascule $x = 0.5$.

Ce pic résulte de la conjonction de deux facteurs :

— **Discontinuité algorithmique :** Le passage brutal du polynôme simple à la formule complexe impliquant la racine carrée.

— **Bruit numérique du Newton :** L'imprécision du calcul de la racine en `float` s'ajoute à l'erreur d'approximation polynomiale.

Conclusion sur le compromis : Pour lisser parfaitement cette transition (réduire l'erreur < 1.5 ULP), il aurait fallu complexifier l'algorithme ou augmenter le nombre d'itérations de Newton, ce qui aurait dégradé la performance globale. Nous avons jugé qu'une erreur locale de **3.0 ULP** était un compromis acceptable pour maintenir une exécution rapide, la précision restant excellente (< 1.5 ULP) sur le reste du domaine.