ENSIMAG

PROJET GL

GL 59

Documentation de l'extension

WALID BAKIR

WALID ABIDI

ABDELOUAHAB MESTASSI

EL MEHDI SALAH-EDDINE

YASSIN EL KHALDI AHANNACH

Table des matières

1 Introduction **2**

2 Fonctions auxiliaires **2**
2.1 Fonction abs . . . 3
2.2 Fonction fact . . . 3
2.3 Fonction power . . . 3
2.4 Fonction sqrt . . . 4
2.4.1 Méthode de Newton . . . 4
2.4.2 Implémention . . . 4
2.4.3 Dessin de la fonction dans [0,3] . . . 5

3 Choix d'implémentation **6**
3.1 Théorème d'approximation de Weierstrass . . . 6
3.2 Séries de Taylor . . . 7
3.3 Séries de Chebyshev . . . 7

4 Précision de l'implémentation **7**

5 Méthode de Cody-Waite pour la réduction du rang **8**

6 Fonctions principales **9**
6.1 Fonction ulp . . . 9
6.1.1 Représentation des flottants (Norme IEEE-754) . . . 9
6.1.2 Unit of least precision . . . 9
6.1.3 Notre implémentation de la fonction ulp . . . 10
6.2 Fonction cos . . . 10
6.3 Fonction sin . . . 11
6.4 Fonction atan . . . 12
6.5 Fonction asin . . . 14

7 Tests **15**

8 Références **17**

1 Introduction

L'extension TRIGO est une continuation de ce projet dans le but d'implémenter des fonctions trigonométriques de base dans la bibliothèque standard Math. Le langage Deca impose quelques

contraintes et obstacles qui rendent un peu difficile l'implémentation de cette bibliothèque, en particulier la façon dont elle traite les flottants et la façon dont ils sont représentés en Deca. Notre

objectif est de développer une bibliothèque Math avec autant de fonctions que possible tout en se concentrant principalement sur la précision en tenant compte bien sûr de la syntaxe Deca.

2 Fonctions auxiliaires

Afin d'implémenter les fonctions cibles, nous avons besoin de quelques méthodes supplémentaires qui seront utilisées dans les implémentations. Pour cela, nous devons également les implémenter à partir de zéro.

Ces méthodes sont :

1. Valeur absolue
2. Factorielle
3. Puissance
4. Racine carrée

2.1 Fonction abs

Algorithm 1 Algorithme Valeur Absolue

Require: x $\in \mathbb{R}$
  if x ≥ 0 **then**
    $x \leftarrow x$
  else
    $x \leftarrow -x$
  end if

2.2 Fonction fact

Algorithm 2 Algorithme factoriel

Require: $n \geq 0$
  return $p = n!$
  $m \leftarrow n$
  $p \leftarrow 1$
  while $m \geq 0$ **do**
  $p \leftarrow p \times m$
  $m \leftarrow m - 1$
  end while

2.3 Fonction power

Algorithm 3 Puissance ;flottant puissance entier

Require: $n \in \mathbb{N}$
Require: $f \in \mathbb{R}$
  $y \leftarrow 1$
  if $n < 0$ **then**
    $f \leftarrow \frac{1}{f}$
    $n \leftarrow -n$
  end if
  while $n > 0$ **do**

    $y \leftarrow n \times f$

    $n \leftarrow n - 1$
  end while

2.4 Fonction sqrt

2.4.1 Méthode de Newton

Nous souhaitons trouver une racine, α, de $y = f(x)$, étant donné une "estimation initiale" de x_0. L'idée fondamentale de la méthode de Newton est d'utiliser l'approximation de la ligne tangente de la fonction au point $(x_0, f(x_0)$.
La formule point-pente de l'équation de la ligne droite nous donne une approximation de la ligne tangente.

$$\frac{y - y_0}{x - x_0} = f'(x) \tag{1}$$

Ainsi, nous avons une ligne droite avec l'équation.

$$y = f(x_0) + (x - x_0)f'(x) \tag{2}$$

Pour y = 0 :

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{3}$$

Soit cette nouvelle valeur x_1
Maintenant, on continue le processus avec une autre ligne droite pour obtenir

$$x = x_1 - \frac{f(x_1)}{f'(x_1)} \tag{4}$$

ou, de manière générale,

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}} \tag{5}$$

2.4.2 Implémention

Pour obtenir une approximation de la fonction racine carrée, on applique la formule précédente sur $f(x) = \sqrt{x}$.

Nous avons choisi notre estimation initiale pour être toujours x lui-même.

On trouve l'implementation :

Nous avons choisi que l'erreur soit d'environ 0.000001

```java
float sqrt(float f){
    // The Newton method to approfimate square root of f
    float error = 0.000001f;
    float guess = f;
    float newGuess = 0.0f;
    float difference = 100.3f;
    if (f == 0){
        return 0.0f;
    }
    while (difference > error){
        newGuess = guess - (guess*guess - f)/(2*guess);
        difference = newGuess - guess;
        if (difference < 0){
            difference = -difference;
        }
        guess = newGuess;
    }
    return guess;
}
```

FIGURE 1 – la fonction racine carrée implémentée en java

2.4.3 Dessin de la fonction dans [0,3]

En utilisant l'algorithme précédent, qui a été développé en utilisant la méthode de Newton, nous pouvons dessiner le graphique de la fonction dans un intervalle donné. Dans l'intervalle [0,3], on remarque que la forme de cette racine carrée est presque identique à celle utilisée dans java.Math.

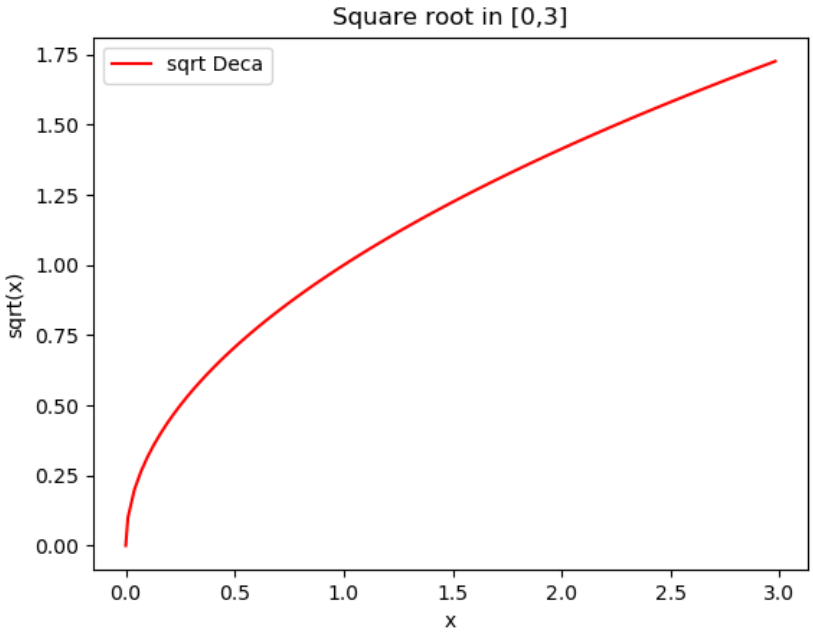


FIGURE 2 – la fonction racine carrée dans [0,3]

3 Choix d'implémentation

3.1 Théorème d'approximation de Weierstrass

Soit f une fonction continue de $[a, b]$ dans $\mathbb{R}$.
Pour tout $\epsilon > 0$, il existe une fonction polynomiale p à coefficients réels telle que pour tout x dans $[a, b]$:

$$|f(x) - p(x)| \leq \epsilon \tag{6}$$

toute fonction continue définie sur un intervalle fermé [a, b] peut être uniformément approchée d'aussi près que souhaité par une fonction polynomiale. Comme les polynômes font partie des fonctions les plus simples et que les ordinateurs peuvent évaluer directement les polynômes, ce théorème présente un intérêt à la fois pratique et théorique, notamment pour l'interpolation polynomiale.

3.2 Séries de Taylor

Ce sont bien les séries "naturelles" auxquelles on pense lorsqu'il s'agit d'approximations polynômiales .

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

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

3.3 Séries de Chebyshev

Les séries Taylor perdent leur précision en s'éloignant de 0 , on opte pour les séries de Chebyshev bien que les coefficients sont cette fois-ci plus compliqué à calculer.

$$sin(\alpha x) = 2 \sum_{k=0}^{+\infty} (-1)^k J_{2k+1}(\alpha) T_{2k+1}(x)$$

$$cos(\alpha x) = 2 \sum_{k=0}^{+\infty} J_{2k}(\alpha) T_{2k}(x)$$

où T_n est le polynôme de Chebyshev de première espèce de degré n et J_k la fonction de Bessel de première espèce.

4 Précision de l'implémentation

Lors de notre implémentation , nous essayons d'avoir la meilleure précision possible , pour cela on calcule à chaque fois l'erreur relative : $r = |\frac{x-\bar{x}}{x}|$ pour une valeur donnée en comparant avec la valeur du sinus déjà connue , et on compare avec l'ulp (unit of least preicsion) de la valeur obtenue.

5 Méthode de Cody-Waite pour la réduction du rang

Les approximations polynomiales sont précises sur des segments , on a donc toujours besoin de ramener notre calcul au segment souhaité à l'aide d'une évaluation de la forme x-kC sauf que cela induit des erreurs dues aux opérations sur les flottants, la méthode de Cody-Waite consiste en la décomposition de notre constante C de manière à se ramener à un calcul de la forme

$$((x - kC_1) - kC_2) - ... - kC_i$$

.

Dans notre cas , on utilise les constantes :

$$C_1 = 50 \times 2^{-5}$$
$$C_2 = 16 \times 2^{-11}$$
$$C_3 = 63 \times 2^{-17}$$
$$C_4 = 26 \times 2^{-23}$$
$$C_5 = 40 \times 2^{-29}$$
$$C_6 = 34 \times 2^{-35}$$
$$C_7 = 5 \times 2^{-41}$$
$$C_8 = 40 \times 2^{-47}$$
$$C_9 = 48 \times 2^{-53}$$
$$C_{10} = 35 \times 2^{-59}$$
$$C_{11} = 19 \times 2^{-65}$$
$$C_{12} = 4 \times 2^{-71}$$
$$C_{13} = 49 \times 2^{-77}$$
$$C_{14} = 38 \times 2^{-83}$$
$$C_{15} = 10 \times 2^{-89}$$
$$C_{16} = 11 \times 2^{-95}$$

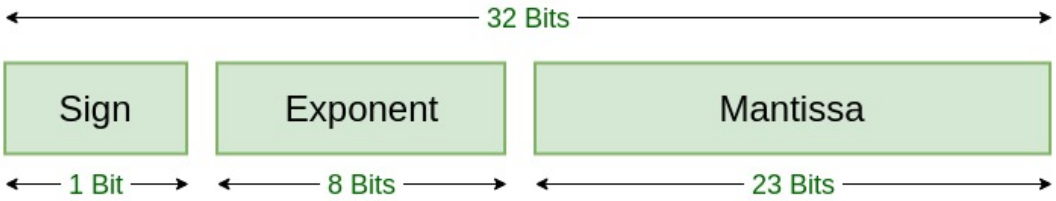
pour $C = \frac{\pi}{2}$

6 Fonctions principales

6.1 Fonction ulp

6.1.1 Représentation des flottants (Norme IEEE-754)

En Deca , les flottants sont codés sur 32 bits suivant la norme IEEE-754 .



Single Precision
IEEE 754 Floating-Point Standard

Un flottant $x = s \times 2^e \times m$ est stocké en mémoire de la manière précisée par la figure ci-dessus , avec :

— s : le signe de x.
— e : l'exposant avant un décalage de 127.
— m : La partie significative.

6.1.2 Unit of least precision

On reprend les notations de x utilisées ci-dessus.

m est limité à un certain nombre de chiffres et, en binaire, il doit généralement être au moins égal à un et inférieur à deux. Le plus petit changement que l'on peut apporter au nombre consiste à modifier le dernier chiffre de m par 1. Par exemple, si m est limité à six chiffres binaires, il a des valeurs allant de 1,00000 à 1,11111, et le plus petit changement que l'on peut lui apporter est 0,00001. Étant donné l'exposant e, un changement de 0,00001 dans m modifie la valeur représentée par $0,00001 \times 2^e$. On appelle cette quantité Unit of Least Precision (ULP).

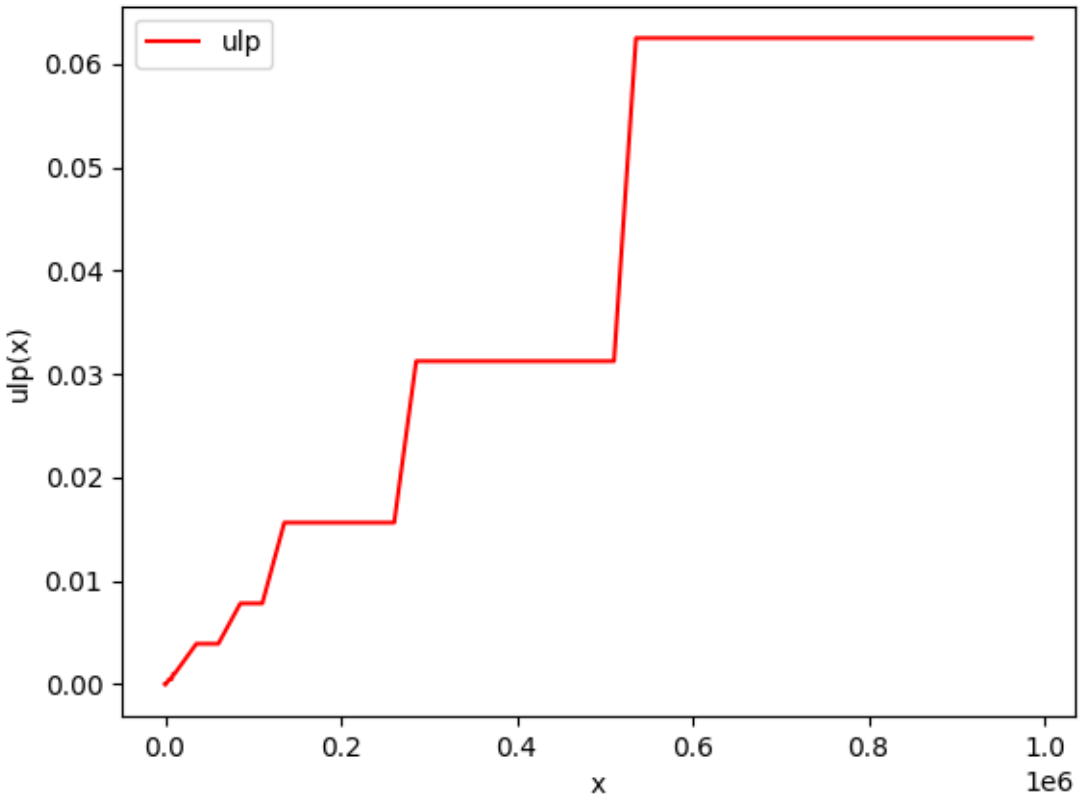


FIGURE 3 – Fonction ulp implémentée

6.1.3 Notre implémentation de la fonction ulp

Nous avons utilisé la formule

$$ulp(x) = 2^{23-k}$$

où k est l'exposant de la plus grande puissance de 2 inférieure à x.

6.2 Fonction cos

Nous avons utilisé une approximation polynômiale utilisant la série de Fourier-Chebyshev sur le segment $[0, \frac{\pi}{2}]$ et fournissant une précision de 7 chiffres après la virgule , les coefficients du polynôme utilisé sont les suivants :

$p_0 = 0.999999953464f$
$p_1 = -0.499999053455f$
$p_2 = 0.0416635846769f$
$p_3 = -0.0013853704264f$
$p_4 = 0.00002315393167f$

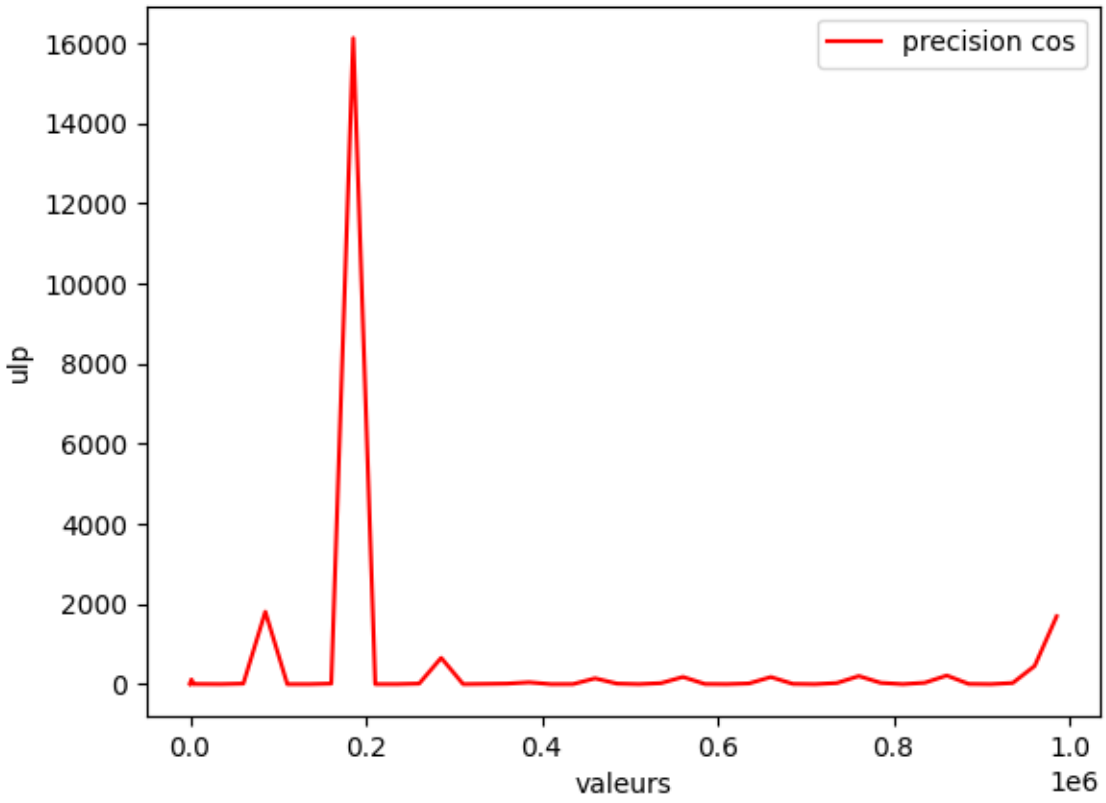


FIGURE 4 – Précision de la fonction cos implémentée

Pour les valeurs dépassant $\frac{\pi}{2}$, on utilise la réduction de Cody and Waite . Et pour les valeurs négatives , on utilise la parité de la fonction cosinus.

6.3 Fonction sin

Nous avons utilisé une approximation polynômiale utilisant la série de Fourier-Chebyshev calculant $sin(\frac{\pi}{2}x)$ sur le segment

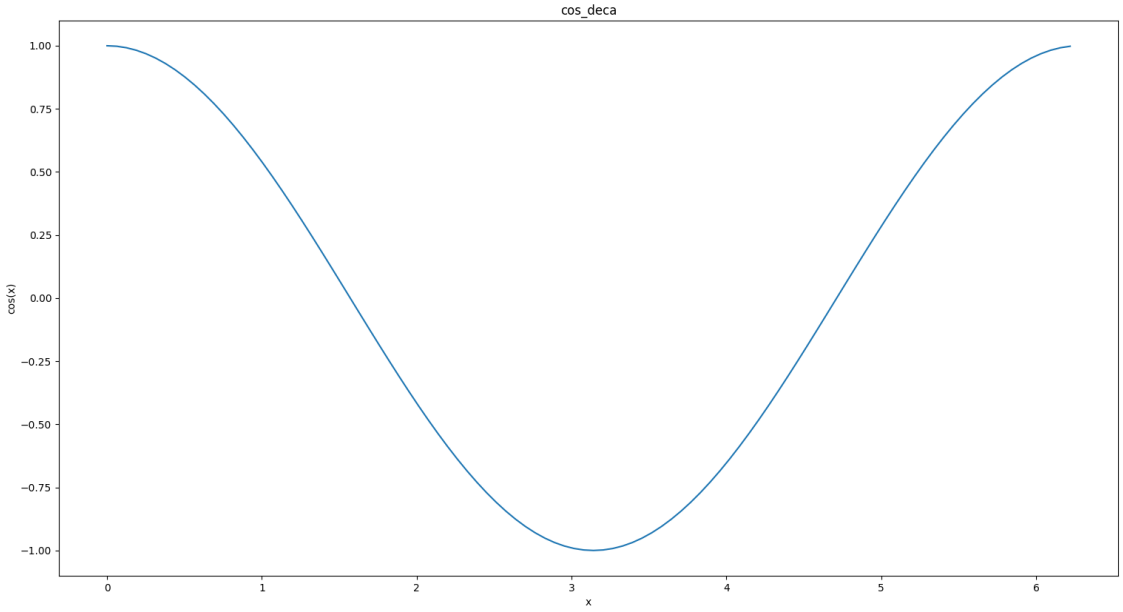


FIGURE 5 – cos deca sur [0,2π]

$[0, 1]$ et fournissant une précision de 6 chiffres après la virgule , les coefficients du polynôme utilisé sont les suivants :

$p_0 = 1.57079632662187f$
$p_1 = -0.645964092652696f$
$p_2 = 0.079692587335023f$
$p_3 = -0.004681620350771f$
$p_4 = 0.000160217246309f$
$p_5 = -0.000003418213039f$

Pour les valeurs dépassant $\frac{\pi}{2}$, on utilise la réduction de Cody and Waite . Et pour les valeurs négatives , on utilise le fait que la fonction sinus soit impaire.

6.4 Fonction atan

Pour les valeurs négatives , on utilise le fait que atan soit impaire.

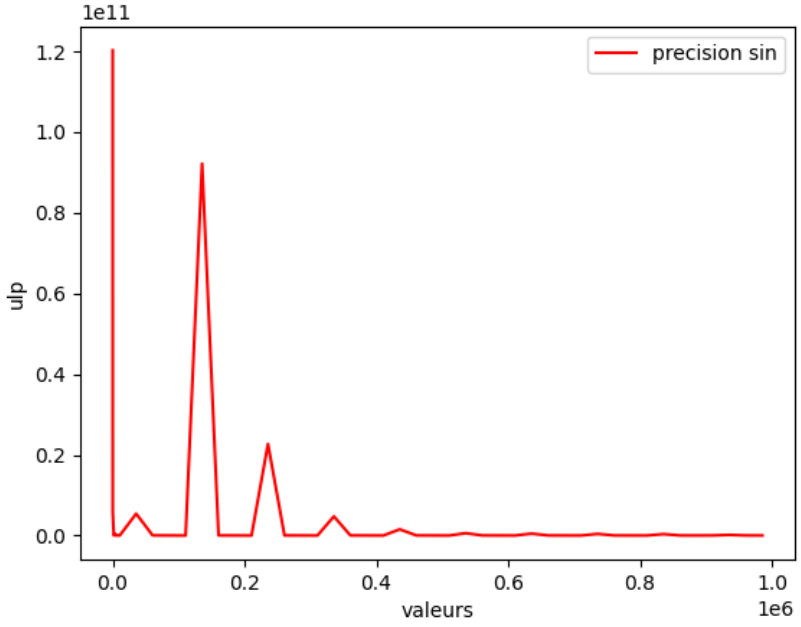


FIGURE 6 – Précision de la fonction sin implémentée

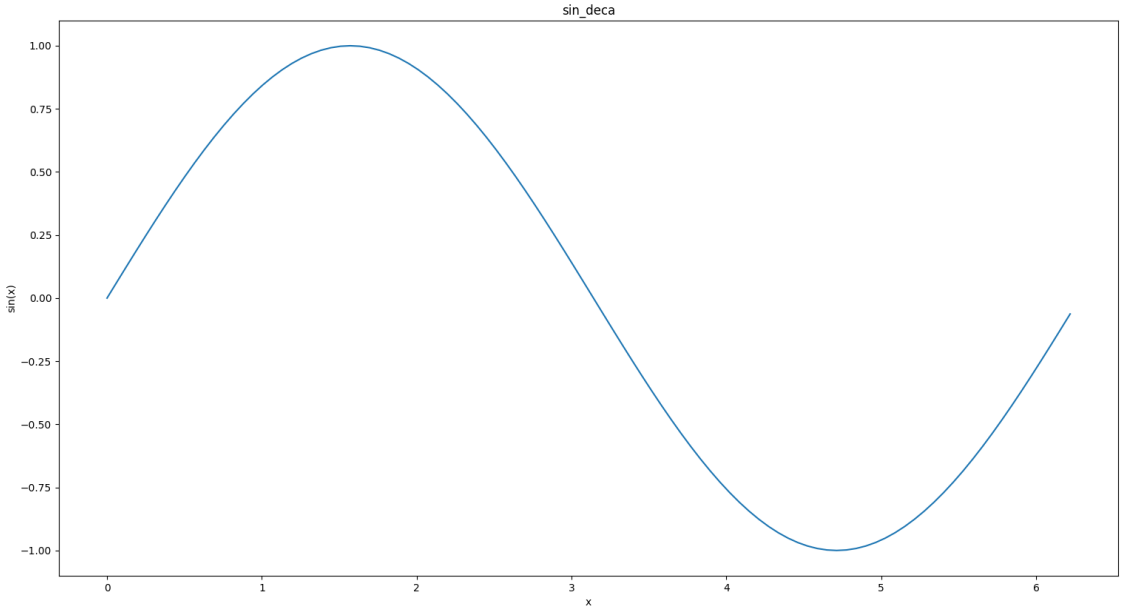


FIGURE 7 – sin deca sur $[0,2\pi]$

sur [0,1] , on utilise le développement en série entière :

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

On prend les 300 premiers termes de la série .

Pour les $x > 1$, on utilise :

$$atan(x) + atan(\frac{1}{x}) = \frac{\pi}{2}$$

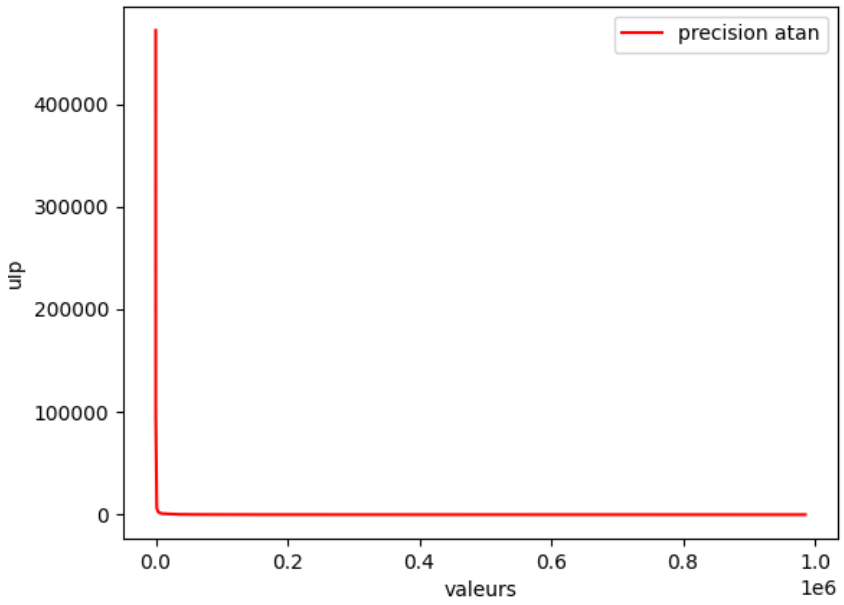


FIGURE 8 – Précision de la fonction atan implémentée

6.5 Fonction asin

Pour les valeurs négatives , on utlise le fait que asin soit impaire .

Pour les $x < \frac{1}{2}$, on utilise :

$$asin(x) = 2 \times atan(\frac{x}{1+\sqrt{1-x^2}})$$

Sinon pour toujours se ramener autour de zéro , on utilise pour les $x > \frac{1}{2}$, on utilise :

$$asin(x) = \frac{\pi}{2} - 2 \times asin(\sqrt{\frac{1-x}{2}})$$

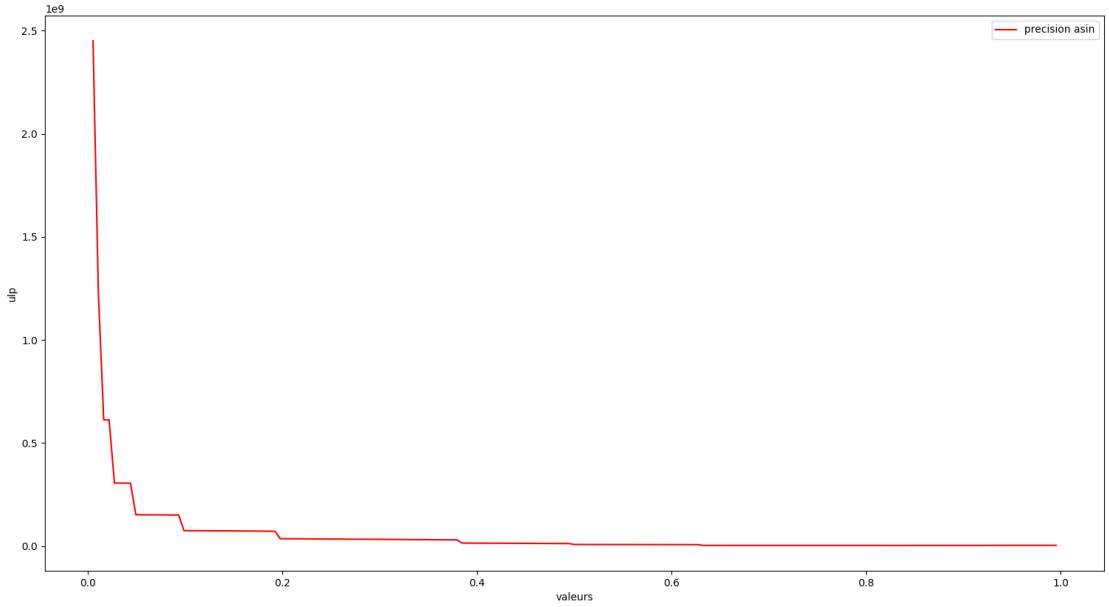


FIGURE 9 – Précision de la fonction asin implémentée

7 Tests

Comme nous avons eu un problème avec la génération de code de notre compilateur, nous n'avons pas pu utiliser Deca pour tester la bibliothèque Math.decah.

Nous avons essayé de tester nos fonctions en utilisant Deca sans objet, mais nous avons découvert que c'était trop peu pratique car nous devions écrire beaucoup de code et séparer de nombreux cas pour calculer une valeur à la fois. Nous avons

donc abandonné cette idée.

Ce que nous avons trouvé le plus proche d'un vrai test Deca, c'est d'écrire un code java de la bibliothèque Math où nous n'utilisons que ce qui est autorisé dans Deca en termes de syntaxe et de types de données.

Cette méthode était plus convaincante car elle nous permettait de comparer nos algorithmes avec les fonctions déjà implémentées dans la bibliothèque java.Math.

Nous avons cependant testé la syntaxe et le contexte de la classe Math.decah, et tout semble fonctionner.

8 Références

1. Elementary Functions by Jean-Michel Muller.

2. Computer Approximations by John F. Hart, E. W. Cheney, Charles L. Lawson, Hans J. Maehly, Charles K. Mesztenyi, John R. Rice, Henry G. Thatcher, Cristoph Witzgall.

3. An Introduction to Numerical Methods and Analysis, by Epperson, James F.