Extension TRIGO - 2ème suivi

Premières idées

- $sin(x) = cos(PI/2 - x)$ → Calculer le **cos** est suffisant pour calculer le **sin.**
- $tan(x) = sin(x)/cos(x)$ → Calculer le **cos** est aussi suffisant pour calculer le **tan.**
- $cos(x + 2k * PI) = cos(x)$ → Calculer le **cos** pour les nombres dans l'intervalle **[-PI,PI]** est suffisant pour calculer les 3 fonctions pour n'importe quelle Input.

Inconvénient

- Le nombre PI a une infinité de chiffres après la virgule, il n'est donc simplement pas possible de le représenter exactement dans la mémoire. En effet, dans IEEE 754, le type est une valeur de 32 bits qui donne une plage de **±1,18 × 10−38** à **±3,4 × 1 038** et une précision d'environ 7 chiffres. Cela signifie qu'on ne pourra représenter précisément PI que par 3,141592. Malheureusement on ne dispose pas du type **double** en Deca pour augmenter la précision. Alors, en plus de l'erreur du résultat de la fonction **cos,** il s'y ajoutera une erreur sur l'input (erreur sur PI). On cherchera donc aussi à ne pas calculer par exemple le cosinus d'un nombre très grand **x** à travers le cosinus de **x % pi.** Calculer les fonctions trigonométriques pour de grandes valeurs est une grosse difficulté, car même pour les bibliothèques standards, ceci peut donner des résultats très bizarres et faux sauf si l'on utilise, comme pour la bibliothèque `mpmath` des tableaux pour représenter les nombres afin d'avoir accès a plus de bits.
- La division des flottants n'est pas exacte , donc calculer la tangente d'un nombre comme quotient du sinus et cosinus va générer, en plus de l'erreur sur les fonctions ***sin et cos,*** une deuxième erreur à cause de la division.

→ Il sera préférable donc d'implémenter indépendemment les fonction **sin, cos et tan.**

1. Taylor Series

Un résultat méthématique très fort qui permet d'approximer les fonctions analytiques est formule suivante de Taylor:

$$f(x) = f(0) + \frac{f^{(1)}(0)}{1!}(x-0) + \frac{f^{(2)}(0)}{2!}(x-0)^2 + \frac{f^{(3)}(0)}{3!}(x-0)^3 + \cdots$$

Pour la fonction sin par exemple, on obtient:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$
$$= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Soit P_n le polynome de Taylor de degré n. La borne d'erreur de Lagrande du polynôme P_n donne le pire scénario pour la différence entre l'estimation (polynome) et la valeur réelle de la fonction:

$$R_n(x) = \frac{M}{(n+1)!}(x-a)^{n+1}$$

où M est la borne supérieure de la (n+1)ème dérivée.

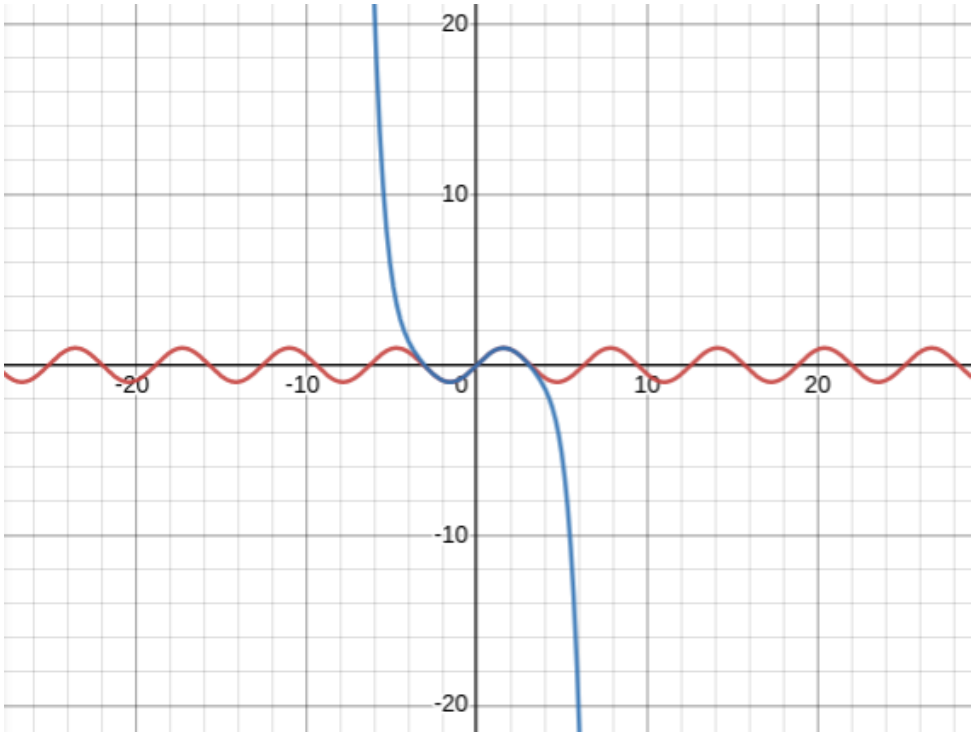
Avantages:

- Vu que le rayon de convergence de cette série est (+ infinity), cette formule est valable pour toute valeur. Il est donc possible d'approximer la fonction pour n'importe quelle valeur.
- L'erreur d'approximation tend vers 0 lorsque $n \to +infinity$. On peut donc obtenir n'importe quelle précision
- On sait faire des multiplications , additions et divisions en Deca.

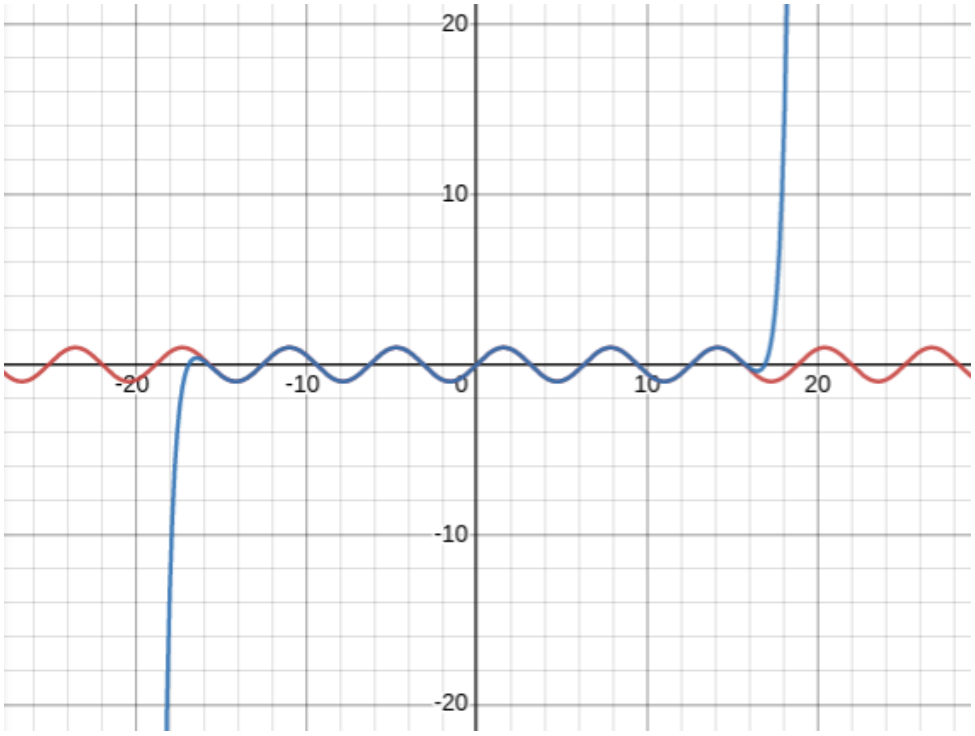
Inconvénients:

On ne peut calculer une série infinie, c'est pourquoi il va falloir choisir un nombre **n** qui limite la somme. Traçons la courbe de la fonction sinus et son développement Taylor:

→ Pour n = 3:



→ Pour n = 20



Même avec n = 20, on n'a pu approximer que des valeurs dont la magnitude est inférieure à 20. La convergence est donc très lente pour des valeurs grandes!

Notons que la formule est très coûteuse → addition, factorielle, puissance. C'est pourquoi cette formule est mieux adaptée pour des valeurs petites de Input.

Séries Fourier-Chebyshev

On va chercher à approximer la fonction cosinus dans l'intervalle [-1, 1]. Pour tout autre intervalle, un simple changement de variable sera suffisant pour revenir au premier cas.

La série Fourier-Chebyshev de la fonction cos est:

$$\cos(x) = J_0(1) + 2\sum_{n\geq 1}(-1)^n J_{2n}(1)T_{2n}(x)$$

où

$$\frac{2}{\pi}\int_{-1}^{1}\frac{\cos(x)T_n(x)}{\sqrt{1-x^2}}\,dx = \frac{2}{\pi}\int_{-\pi/2}^{\pi/2}\cos(\cos x)\cos(nx)\,dx$$

Dans l'intervalle [-1, 1], les polynômes de Chebyshev sont majorés par 1. La précision de l'approximation dépend donc de la vitesse de convergence de $J_{2n}(1)$ vers 0.

Puisqu'on a:

$$J_{2n}(1) = \sum_{l\geq 0}\frac{(-1)^l}{4^{l+n}(2n+l)!}, \qquad |J_{2n}(1)| \approx \frac{1}{4^n(2n)!}$$

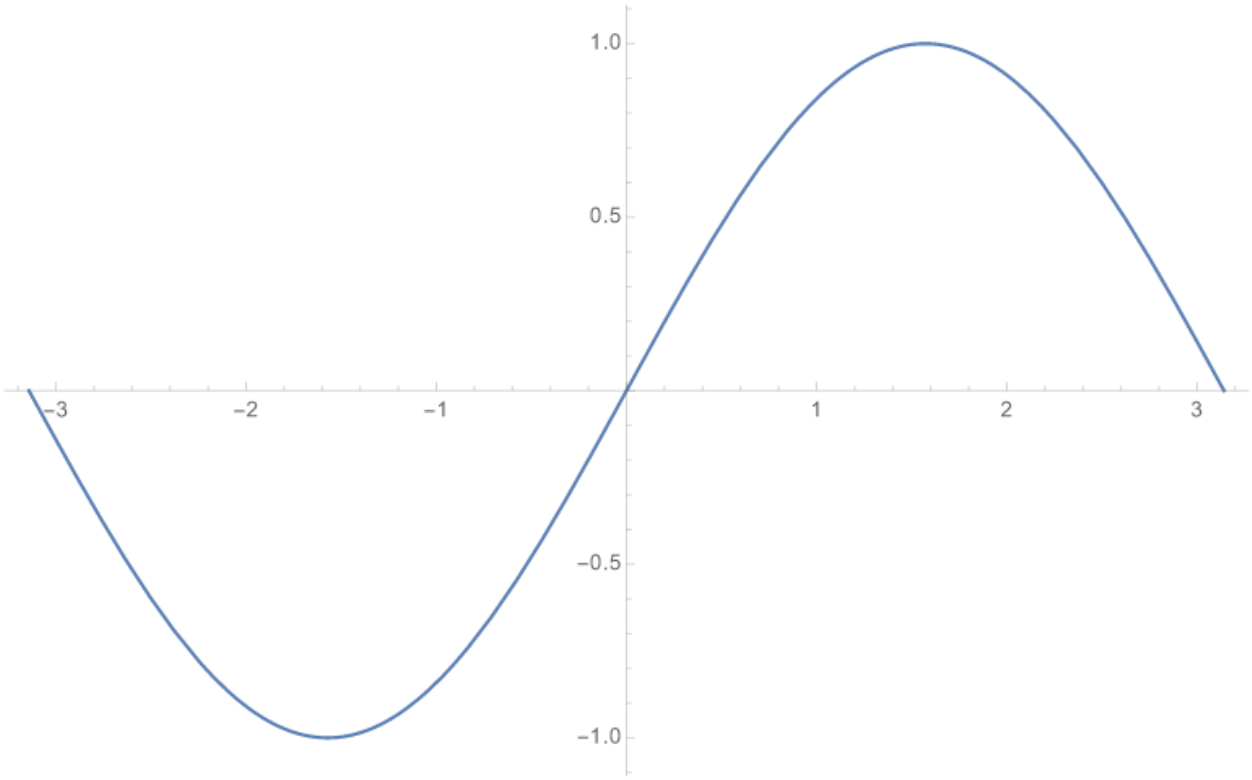
L'approximation de Taylor est meilleur lorsque $|x| \leq 1/2$ tandis que dans l'autre cas, c'est la deuxième qui est plus précise autour des bornes de l'intervalle.

Avantages

- Meilleure précision autour des bornes.
- Convergence plus rapide que Taylor
- Plus adéquat pour calculer la fonction sur un intervalle [a, b]
- Erreur uniforme répartie sur l'intervalle. Pour Taylor, l'erreur est croissante en s'éloignant de 0.

Inconvénients

- Les coefficients de la série sont plus coûteux à calculer.

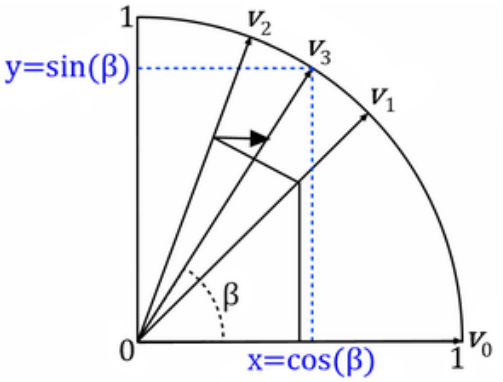


Approximation de la fonction sin par série de Fourier-Chebyshev

Algorithme Cordic

L'algorithme cordic Coordinate Rotation Digital Computer permet d'approximer les fonctions trigonométriques avec une suite d'opérations arithmétiques très simple : addition, soustraction et décalage.

L'algorithme se base sur la matrice de rotation pour approcher le plus possible la valeur du couple (cos(x) , sin(x)) à calculer.



Pour la matrice R_i

$$R_i = \begin{pmatrix} \cos\gamma_i & -\sigma_i \sin\gamma_i \\ \sigma_i \sin\gamma_i & \cos\gamma_i \end{pmatrix}$$

Qu'on peut aussi écrire sous la forme

$$\cos\gamma_i \begin{pmatrix} 1 & -\sigma_i \tan\gamma_i \\ \sigma_i \tan\gamma_i & 1 \end{pmatrix}$$

La force du cordique, c'est qu'on choisit des valeurs de la fonction tangente comme l'inverse des puissances de deux. Ceci dit, on peut récupérer ces valeurs en réalisant uniquement des décalages.

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = K \times \begin{bmatrix} 1 & -\tan\alpha_{n-1} \\ \tan\alpha_{n-1} & 1 \end{bmatrix} \begin{bmatrix} 1 & -\tan\alpha_{n-2} \\ \tan\alpha_{n-2} & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & -\tan\alpha_0 \\ \tan\alpha_0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

$$K = \prod_0^{n-1} \cos\alpha_i$$

Cette suite converge finalement vers la valeur de (cos(x), sin(x)).

Avantages

- Il permet d'obtenir une précision déterminée à l'avance en effectuant un nombre d'itération donné.
- En utilisant des puissances de 10 au lieu que des puissances de 2 (ce qui est contradictoire aux point fort de l'algorithme qui nécessite que des opérations arithmétiques élémentaires) , cette méthode peut générer un peu plus rapidement une valeur précise à 10^(-8) près.

Inconvénients

- Converge lentement vers la valeur voulue comparé aux algorithmes précédents.
- Cet algorithme n'exploite pas la fonctionnalité FMA qui permet de réduire le nombre de cycles nécessaire pour une opération de la forme a + b*c.

- Erreur qui s’ajoute pour le calcul initiale de la valeur K qui s’estompe uniquement en augmentant largement le nombre d’itérations.

Et pour les autres fonctions trigonométriques?

On parlera de la fonction arcsin, mais il en est de même pour arctan et arcos.

Pour calculer la fonction $arcsin$, on peut effectuer un développement de Taylor ou Chebyshev. Comme c’est déjà traité auparavant, on va explorer une deuxième méthode:

Notons **v** la valeur de $arcsin(x)$ pour un x quelconque. Comme trouver **v** revient à trouver la valeur le zéro de la fonction $f(v) = sin(v) - x$, c’est à dire trouver la valeur **v** pour laquelle $sin(v) - x = 0$, ceci peut être résolu par la **méthode de Newton-Raphson,** déjà vu en cours de méthodes numériques de base en 1ère année.

En choisissant un x_0 arbitraire, l’algorithme consiste à calculer à chaque itération x_{k+1} comme suit:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Cette formule peut être utilisée grâce au fait qu’on peut calculer f’(x_k):

- cos’$(x) = -sin(x)$
- sin’$(x) = cos(x)$
- tan’$(x) = 1 + tan^2(x)$

L’intérêt principal de l’algorithme de Newton est sa **convergence quadratique**. Le nombre de chiffres significatifs des itérés double à chaque itération, *asymptotiquement.* Comme le nombre des chiffres significatifs représentabels par un ordinateur est d’environ 15 chiffre décimaux, l’algorithme soit converge en moins de 10 itérations, soit il diverge.

Critère de précision

L’erreur de notre implémentation se fera par l’unité de précision élémentaire sur les nombres flottants **ulp (unit in the last place).** Soit **f(x)** notre approximation de la fonction **sin**, **cos**,…

L’erreur est calculée de la manière suivante: $\frac{|f(x) - sin(x)|}{ulp(f(x))}$

Notre objectif est d’atteindre une précision de **5ulp.**