Optimisation Globale

Elliot Braud et Elizabeth Gandibleux
M2 ORO

Octobre 2024

1 Introduction

Dans le cadre de ce projet nous nous intéressons à l'étude et à l'approximation de la fonction sinus cardinal, notée $sinc(x) = \frac{\sin(x)}{x}$. La première étape consiste à analyser les propriétés de cette fonction, en particulier ses extrémums et son comportement oscillatoire. Ensuite, l'objectif principal est de développer et d'implémenter des algorithmes d'optimisation, en commençant par l'algorithme de Newton, pour déterminer les valeurs minimales et maximales de $sinc(x)$ sur un intervalle donné $[a, b]$. Pour la suite de ce rapport, nous adoptons la convention selon laquelle la borne inférieure a est toujours inférieure ou égale à la borne supérieure b ($a \leq b$) . L'algorithme final proposé a pour but de fournir l'image la plus précise possible de cet intervalle par la fonction sinus cardinal, en un temps optimal.

2 Problème initial

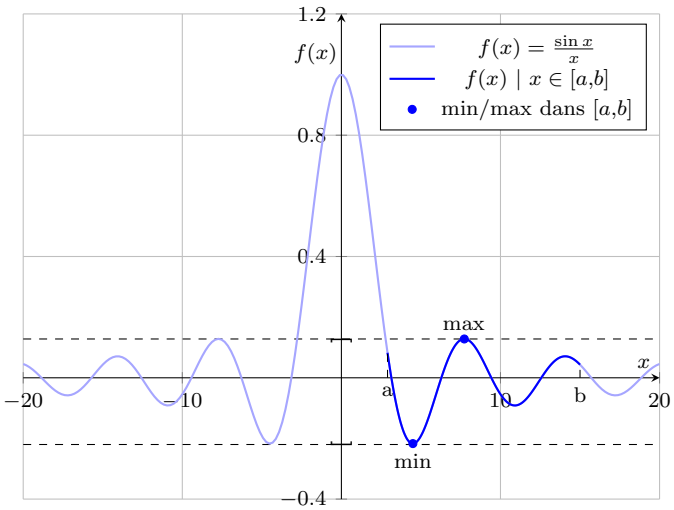

Figure 1: Fonction sinus cardinal avec les valeurs *max* (*resp.* *min*) dans l'intervalle $[a, b]$.

2.1 Propriété de sinus cardinal

Dans cette sous-section nous allons aborder l'étude de la fonction sinus cardinal. Cette dernière s'articule autour de plusieurs propriétés permettant de généraliser à un nombre fini de cas. Tout d'abord, la fonction sinus cardinal est une fonction paire *i.e.* $f(x) = f(-x), \quad \forall x \in \mathbb{R}$. Nous travaillerons ainsi uniquement sur la partie positive de la fonction f. La fonction *sinc* est également π périodique. En d'autres termes, elle va s'annuler pour chaque valeur où x est un multiple de π, *i.e.* $x \bmod \pi \equiv 0$ à l'exception de $x = 0$ où $f(0) = 1$. Une autre conséquence de cette π périodicité est que chaque portion de la courbe adopte d'abord une forme décroissante sur l'intervalle $[0, \pi]$, puis se prolonge en une succession de paraboles tendant vers $+\infty$.

On utilisera la fonction $k(x) = \lceil \frac{x-\pi}{\pi} \rceil$, appelée l'indice de la plage x, pour découper la fonction *sinc* en segment de largeur π. De cette façon, on obtient $k(x) = 0$ quand $x \in [0, \pi]$, sur cet intervalle la fonction est une courbe décroissante. Lorsque $k(x)$ est impaire, la fonction *sinc* ressemble à une parabole orientée vers le bas. À l'inverse, quand $k(x)$ est paire, la fonction *sinc* possède l'allure d'une parabole orientée vers le haut.

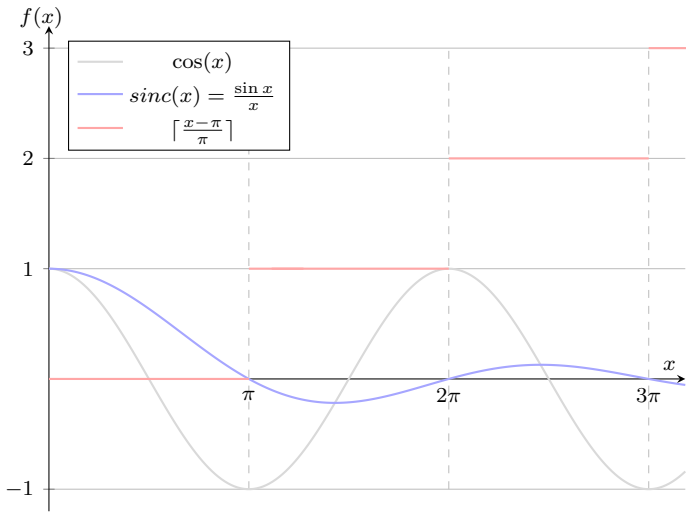

Figure 2: Périodicité de la fonction sinus cardinal.

Une dernière observation importante à propos de la fonction *sinc* est que son amplitude décroît à mesure que x augmente. Ainsi, si la largeur de l'intervalle $[a, b]$ dépasse $(2\pi + 0.5)$, nous redéfinissons la valeur de b en la limitant à $(a + 2\pi + 0.5)$. En effet, pour la fonction sinus cardinal, l'écart maximal entre deux extrema consécutifs est inférieur à $(2\pi + 0.5)$. Réduire l'intervalle $[a, b]$ à cette largeur permet de concentrer l'analyse sur la région où les extrema sont les plus marqués, sans risquer de perdre des informations cruciales. Cette restriction en direction de a, où la fonction présente une amplitude plus élevée (puisque $a \leq b$), permet de se concentrer sur les parties les plus pertinentes de la fonction.

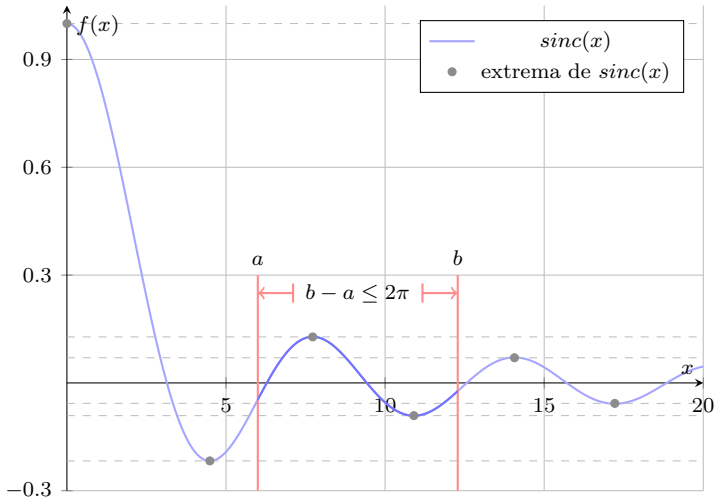

Figure 3: Re-définition de l'intervalle [a,b].

2.2 Les différents cas possibles:

Dans la fonction `approximation_fct_sinus_cardinal(a::Float64, b::Float64)` plusieurs cas sont étudiés; dans l'ordre:

- **Cas 1**: Soit a > b: D'après la convention qui a été prise ultérieurement (voir section 1), ce cas là n'est pas traité. On le considère comme impossible. On retourne par la suite une valeur sentinelle pour a et b.
- **Cas 2**: Soit a et b $\in \mathbb{R}$ tel que a = b alors l'image de l'intervalle qui est un point, par la fonction f est simultanément le minimum et maximum.
- **Cas 3**: Soit $0 \in$ [a,b]: Par étude de la fonction, on connait la valeur du maximum qui est 1. On sait également que si $\frac{3\pi}{2} \leq |a|$ ou $\frac{3\pi}{2} \leq |b|$ alors on sait d'ores et déjà que le minimum globale de la fonction `sinc` est très proche de $\frac{3\pi}{2}$. On utilise donc la `méthode de Newton` (intervalle ou non) pour obtenir cet extrema.
- **Cas 4**: Soit a et b $\in \mathbb{R}$ tel que a < 0 et b < 0.
- **Cas 5**: Soit a et b $\in \mathbb{R}$ tel que a > 0 et b > 0.

Pour les cas 4 et 5, l'algorithme suit un schéma commun avec des ajustements spécifiques selon les valeurs de a et b. Si $a < 0$ et $b < 0$, on utilise la fonction `abs()` pour convertir les bornes en valeurs positives (comme mentionné dans la section 2.1), comme suit `a, b = abs(b), abs(a)`, afin de travailler sur la partie positive de la fonction. Lorsque $a \geq 0$ et $b \geq 0$, l'algorithme vérifie si l'intervalle est supérieur à $(2\pi+0.5)$ et, si tel est le cas, il ajuste b à $(a+2\pi+0.5)$. Ensuite, il identifie les indices des plages correspondant aux bornes ($k(a)$ et $k(b)$) et, en fonction de ces indices, applique la méthode de Newton (classique ou par intervalle) pour calculer les extrema important pour l'analyse.

Si $k(a)$ et $k(b)$ sont égaux, l'extrémum est situé entre a et b. Si $k(a) \neq k(b)$, l'algorithme examine les extrémums dans les plages et ajuste les valeurs minimales et maximales en conséquence, en tenant compte de la parité des indices.

3 Commandes

Au départ d'un REPL Julia, entrer: `include("GO.jl")`

4 Algorithmes

4.1 Méthode de Newton

On part d'un point x et la méthode de Newton essaie de converger vers un zéro de la fonction, c'est-à-dire une valeur où la fonction vaut 0. En général, la méthode converge vers le zéro le plus proche, à condition de choisir un bon point de départ x. La dérivée d'une fonction f s'annule aux points où f atteint un extrémum (maximum ou minimum local). Ainsi, appliquer la méthode de Newton sur la dérivée de f permet d'approcher les points où f a un extrémum. La méthode de Newton est définie par:

$$N(x) = x - \frac{f(x)}{f'(x)} \quad \text{avec} \quad f'(x) \neq 0$$

où : - x : un point de départ,
- $f(x)$: la valeur de la fonction,
- $f'(x)$: la dérivée de la fonction en x.

On définit $f'(x) := \text{sinc}'(x)$ comme suit :

$$\text{sinc}(x) = \frac{\sin(x)}{x} \quad \text{et} \quad \text{sinc}'(x) = \frac{x \cdot \cos(x) - \sin(x)}{x^2}$$

Ainsi, la dérivée seconde est donnée par :

$$f''(x) := \text{sinc}''(x) = \frac{(2 - x^2) \cdot x \cdot \sin(x) - 2x^2 \cdot \cos(x)}{x^4}$$

4.1.1 Détails de l'implémentation

L'équation de Newton est traduite dans le code par la fonction suivante :

```
1 N(x::Float64)::Float64 = return (x - (sinc_d_1st(x) / sinc_d_2nd(x)))
```

Figure 4: Implémentation de l'opérateur de Newton pour une fonction *sinc*, utilisant ses premières et deuxièmes dérivées pour mettre à jour l'approximation de la racine.

Cette méthode est implémentée dans le code suivant :

```
function newton_method(x::Float64, threshold::Float64 = accuracy)::Float64

    tmp::Union{Float64, Nothing} = x-1
    while !(tmp <= x <= tmp + threshold)
        tmp = x
        x = N(x)
    end
    return x
end
```

Figure 5: Implémentation de la méthode de Newton

En pratique, la méthode s'arrête lorsque deux points consécutifs sont suffisamment proches (cas de convergence) ou lorsque la distance entre deux points consécutifs est trop grande (cas de divergence). Ici, cela est géré par la condition d'arrêt dans la boucle `while`, qui compare les valeurs successives de x avec un seuil donné *threshold*.

4.2 Méthode de Newton Interval

La méthode de Newton interval est une version modifiée de la méthode de Newton classique. Elle utilise des intervalles pour s'assurer que les calculs prennent en compte les incertitudes et les erreurs d'arrondi. Cette méthode est particulièrement utile pour garantir que les zéros d'une fonction se trouvent dans un intervalle donné.

4.2.1 Opérateur de Newton

L'itération de la méthode de Newton est définie par :

$$N(f, [x], c) = c - \frac{f(c)}{[z]}$$

où $[z]$ est un encadrement de l'image $f'([x])$. Le processus d'itération est décrit par :

$$\begin{cases} [x]^0 & \leftarrow [x] \\ [x]^{k+1} & \leftarrow [x]^k \cap N(f, [x]^k, c^k), \quad (c^k \in [x]^k) \end{cases}$$

Cette méthode itérative permet de réduire la taille de l'intervalle jusqu'à ce qu'une solution suffisamment précise soit atteinte.

4.2.2 Opérateur d'inflation

Pour éviter des erreurs d'arrondi ou des pertes de précision, un opérateur d'inflation est souvent utilisé. Cet opérateur est défini comme suit :

$$\theta([x]) = m([x]) + \delta([x] - m([x])) + \lambda[-1, 1]$$

où $\delta > 1$ et $\lambda > 0$ sont des paramètres contrôlant l'inflation de l'intervalle.

4.2.3 Détails d'implémentation

Fonction principale La fonction principale de la méthode de Newton interval est définie comme suit :

```
N(f::Function, df::Function, x::tInterval{Float64}, c::Float64 = m(x))
    = return c - f(c)/df(x)
```

Figure 6: Implémentation de l'opérateur de Newton interval

La méthode est décrite telle quelle:

```
function NewtonInterval(x::tInterval, max_iter::Int64 = 1000, τ::Float64 = accuracy)

    i::Int64 = 0
    while (i < max_iter) && (w(x) >= τ)
        x = N(sinc_d_1st, sinc_d_2nd, θ(x))
        i += 1
    end

    return x
end
```

Figure 7: Implémentation de la méthode de Newton interval

Explications des fonctions

- **Fonction N** : Cette fonction applique l'itération de Newton à un intervalle, utilisant un point c (le point médian par défaut) pour mettre à jour l'intervalle x à chaque itération.
- **Opérateur d'inflation** θ : Cet opérateur ajuste l'intervalle en ajoutant une petite quantité autour de la valeur médiane, élargissant ainsi l'intervalle pour éviter les erreurs d'arrondi.
- **Fonction Newton interval** : Exécute l'algorithme de Newton interval, itérant jusqu'à ce que la largeur de l'intervalle soit inférieure à un seuil donné τ ou que le nombre maximal d'itérations soit atteint.

4.2.4 Structure de l'intervalle

Au lieu d'exploiter le package `IntervalArithmetic.jl` de Julia, nous implémentons notre code permettant de réaliser des opérations sur des intervalles et ainsi s'assurer de limiter les erreurs de précisions. Nous définissons ainsi une structure d'intervalle à travers le type `tInterval`:

```
struct tInterval{T}
    l::Union{T, Nothing}  # Borne inferieure
    u::Union{T, Nothing}  # Borne superieure
end
```

Figure 8: Structure de donnée `tInterval`

Ainsi que des opérateurs de base pour les intervalles qui sont définis comme suit:

```
    # Addition d'un scalaire
Base.:+(x::tInterval{T}, v::U) where {T, U} =
    return tInterval{T}(x.l + v, x.u + v)

    # Soustraction d'un scalaire
Base.:-(x::tInterval{T}, v::U) where {T, U} =
    return tInterval{T}(x.l - v, x.u - v)

    # Multiplication par un scalaire
Base.:*(x::tInterval{T}, v::U) where {T, U} =
    return tInterval{T}(x.l * v, x.u * v)

    # Division par un scalaire
Base.:/(x::tInterval{T}, v::U) where {T, U} =
    return tInterval{T}(x.l / v, x./ v)
```

Figure 9: Exemple de définition des opérations arithmétiques sur les intervalles

4.3 Jeux de tests

Le tableau ci-dessous présente plusieurs exemples de résultats obtenus en utilisant les algorithmes de `Newton` et `Newton interval` pour la résolution de certains problèmes. Les colonnes a et b indiquent les intervalles d'entrée, tandis que les résultats intervalles générés par chaque algorithme sont affichés dans leurs colonnes correspondantes:

a	b	`Newton`	`Newton interval`
-3	2	[0.0470400026866224, 1.0]	[0.0470400026866224, 1.0]
-1	120	[-0.21723362821122166, 1.0]	[-0.21723362821122166, 1.0]
1	6	[-0.21723362821122166, 0.8414709848078965]	[-0.21723362821122166, 0.8414709848078965]
8	120	[-0.09132520282305767, 0.12366978082792272]	[-0.09132520282305767, 0.12366978082792272]
5	7	[-0.1917848549326277, 0.09385522838839844]	[-0.1917848549326277, 0.09385522838839844]
1	2	[0.45464871341284085, 0.8414709848078965]	[0.45464871341284085, 0.8414709848078965]

Table 1: 6 Exemples de résultats obtenus avec les algorithmes `Newton` et `Newton interval`

Comme illustré dans le tableau 1 ci-dessus, les deux algorithmes fournissent des résultats similaires, avec des valeurs très proches voir identiques pour les différents intervalles d'entrés testés, ce qui montre leur robustesse dans le cadre de ce projet.

Pour évaluer les temps de calcul, nous avons réalisé un benchmark en utilisant le package `BenchmarkTools.jl` de Julia. Pour un intervalle de valeur $[1, 120]$ (calcul des deux extrémums locaux et ajustement de la borne b), la méthode de Newton prend en moyenne 800 ns pour retourner un résultat, tandis que la méthode de Newton avec intervalles prend en moyenne 6 μs.

5 Conclusion

Dans ce travail, nous avons exploré plusieurs cas possibles pour l'approximation des bornes d'un intervalle dans le cadre de la fonction sinus cardinal. Nous avons développé et implémenté deux algorithmes basés sur la méthode de Newton : la méthode classique et la méthode de Newton intervalle. Ces approches ont permis d'approximer les extrémums locaux de la fonction dans des conditions variées, notamment lorsque les bornes de l'intervalle sont positives, négatives, ou incluent l'origine.

Les différents algorithmes proposés fournissent des solutions précises pour l'analyse de la fonction `sinc` dans des temps de calcul tout à fait raisonnables. De plus, la découpe et l'analyse par morceaux de la fonction garantissent une gestion rigoureuse des cas particuliers ainsi que des incertitudes numériques.

Ces résultats ont été validés à travers des exemples pratiques et des jeux de tests, confirmant la capacité des algorithmes développés à calculer des minima et maxima avec précision.