

Informatique – 1A

Calcul de la fonction sinus par sa série de Taylor

La Prépa des INP

2023-2024

1 But du TP et algorithme

Les fonctions Python autorisées pour ce TPL sont celles vues en cours et celles décrites dans l'aide-mémoire. Les fonctions trigonométriques ne seront utilisées que pour évaluer et visualiser l'erreur.

L'objectif de ce TP est d'écrire une fonction qui prend un nombre (flottant ou entier) en argument, représentant un angle en radians, et renvoie une approximation du sinus de ce nombre sous forme de nombre flottant.

La fonction devra être écrite, commentée et testée.

Le TP est à faire *en binôme*. Pour des raisons pratiques, les deux membres du binôme doivent *autant que possible* faire partie du même groupe : A, B, C ou D.

On utilisera une série de Taylor¹ qui permet d'approximer une fonction indéfiniment dérivable par un polynôme (vous verrez les détails en deuxième année) :

$$\begin{aligned}\sin(x) &= \sum_{n=0}^{+\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \\ &\approx \sum_{n=0}^N (-1)^n \frac{x^{2n+1}}{(2n+1)!} \quad \text{pour } N \text{ suffisamment grand} \\ \sin\left(\frac{\pi}{2} - x\right) = \cos(x) &= \sum_{n=0}^{+\infty} (-1)^n \frac{x^{2n}}{(2n)!} \\ &\approx \sum_{n=0}^N (-1)^n \frac{x^{2n}}{(2n)!} \quad \text{pour } N \text{ suffisamment grand}\end{aligned}$$

Ces deux séries convergent rapidement au voisinage de 0 (i.e. on obtient une bonne approximation pour des valeurs de N raisonnables), mais ne sont pas utilisables directement en pratique pour les grandes valeurs de x . On peut cependant utiliser les formules trigonométriques de base pour ramener le calcul de $\sin(x)$ pour x quelconque à un calcul de sinus sur l'intervalle $[0, \frac{\pi}{2}]$:

- $\forall x \in \mathbb{R}, \sin(-x) = -\sin(x)$ (permet de se ramener à $x \geq 0$, pas strictement nécessaire mais simplifie les calculs)
- $\forall x \in \mathbb{R}, \forall k \in \mathbb{N}, \sin(x + 2k\pi) = \sin(x)$ (permet de se ramener à $x \in [0, 2\pi]$)
- $\forall x \in \mathbb{R}, \sin(x + \pi) = -\sin(x)$ (permet de se ramener à $x \in [0, \pi]$)

1. http://fr.wikipedia.org/wiki/S%C3%A9rie_de_Taylor

- $\forall x \in \mathbb{R}, \sin(\pi - x) = \sin(x)$ (permet de se ramener à $x \in [0, \frac{\pi}{2}]$)
- $\forall x \in \mathbb{R}, \sin(\frac{\pi}{2} - x) = \cos(x)$ (permet d'utiliser le développement limité de $\cos$ plutôt que celui de $\sin$ lorsqu'on est plus proche de $\frac{\pi}{2}$ que de 0).

Pour $n = 10$ et $x = \pi/2$, $\frac{x^{2n+1}}{(2n+1)!}$ vaut environ 2.5×10^{-16} , qui est l'ordre de grandeur de précision que l'on peut attendre pour une valeur proche de 1 avec des nombres flottants Python. Les termes suivants de la série seront donc négligeables. Pour les valeurs de x plus petites, on peut attendre une meilleure précision absolue car la valeur du résultat sera également plus petite (on travaille en nombre flottant, donc l'important est l'erreur relative et non l'erreur absolue), mais les termes de la série sont également plus petits. On pourrait donc poser $N = 10$. Cependant, nous allons montrer qu'une astuce permet de réduire ce nombre de termes, et nous montrerons que $N = 7$ est suffisant pour atteindre cette précision.

2 Code fourni

On fournit un fichier Jupyter `sin.ipynb` contenant le squelette du code :

- Les premières cellules contiennent les fonctions `sin1`, `cos1`, `sin2`, qui pour l'instant ne font rien d'intéressant : `sin1(x)` renvoie toujours `x` et `sin2` se contente d'appeler `sin1`. Vous y ajouterez les fonctions `sin3` et `sin4`. Ces fonctions correspondront à quatre implémentations du calcul de sinus et une implantation du calcul de cosinus.
- Une cellule qui contient quelques tests, à compléter.
- Une cellule contenant la fonction `setup_trigo_axis` que vous n'avez pas besoin d'utiliser directement mais qui permet de dessiner des graphiques plus jolis dans les cellules suivantes. Il vous faut simplement exécuter cette cellule.
- Une cellule qui permet de tracer les courbes des fonctions `sin1`, `sin2`, `sin3`, `sin4`, et de les comparer avec la courbe de la fonction `sin` fournie avec Python (plus précisément, `math.sin`). Observez bien les commentaires dans ce fichier, qui permettent de tracer les différentes courbes.
- Une cellule qui permet de visualiser plus finement les erreurs.

3 Travail demandé

3.1 Premier contact

Ouvrez la plateforme Jupyter de Gricad <https://jupyterhub.u-ga.fr/> et uploader le fichier `sin.ipynb`. Cliquez ensuite sur ce fichier pour l'ouvrir. Vous pouvez exécuter toutes les cellules, seule celle de test vous retourne une erreur. Les dernières cellules affichent un graphique. On reconnaît la courbe du sinus calculé avec `math.sin`, mais pour l'instant les courbes représentant les fonctions `sin1` et `sin2` sont la droite `y = x`.

3.2 Première version : itérations simples

Implémentez la fonction `sin1`, qui calcule les 8 premiers éléments de la série (de $n = 0$ à 7) et renvoie le résultat. Tracez la courbe.

La courbe devrait être proche de celle de `math.sin` pour les valeurs de x proches de 1, mais les deux courbes divergent pour les grandes valeurs.

On peut faire varier le nombre d'itérations : essayez avec des valeurs de N plus grandes ou plus petites que 7 (par exemple, 1, 2, 5, 15) et voir le résultat sur les courbes.

Pour voir plus finement l'erreur faite par `sin1` par rapport à `math.sin`, utilisez la dernière cellule qui trace les courbes des fonctions `sin1(x) - math.sin(x)` et `sin2(x) - math.sin(x)`. Il affiche deux graphiques : l'un sur une échelle logarithmique permet de voir assez finement les erreurs, et l'autre sur une échelle linéaire.

Vous pouvez éditer les cellules pour modifier l'échelle du graphique : des constantes `xmin`, `xmax` et `ymax` y sont définies et donnent les valeurs de x et y à afficher.

Vérifiez que votre fonction donne les bons résultats sur l'intervalle $[0, \frac{\pi}{2}]$, graphiquement et avec des tests dans la section « Tests » ci-dessous.

3.3 Deuxième version : réduction à l'intervalle $[0, \frac{\pi}{2}]$ avant itération

Sans modifier la fonction `sin1`, implémentez maintenant la fonction `sin2` qui réduit le calcul à un calcul dans l'intervalle $[0, \frac{\pi}{2}]$ avant de faire les itérations. L'idéal est d'appeler la fonction `sin1` depuis la fonction `sin2` pour éviter d'avoir à recopier le calcul fait par `sin1`.

À titre d'exemple, la réduction à $x \geq 0$ vous est donnée : si x est négatif, on stocke -1 dans la variable `signe` et on change le signe de x . Après le calcul, on multiplie le résultat par `signe`, donc par -1 si nécessaire.

Pour se ramener dans l'intervalle $[0, 2\pi]$, on peut utiliser l'opérateur modulo de Python (`%`), qui marche sur les flottants.

Tracez à nouveau les courbes. En principe, la courbe de `sin2` devrait être visuellement superposée avec la courbe `math.sin`. Bien sûr, cette superposition visuelle n'est pas suffisante pour mesurer l'erreur. En effet, les calculs sont fait avec une précision relative de l'ordre de 10^{-15} , ce qui sur un graphe de 10 cm de haut correspondrait à une différence d'ordonnée plus petite qu'un atome ! Vérifiez que l'erreur est petite pour toutes les valeurs de x (cette fois, l'échelle logarithmique vous permet de voir finement les erreurs).

3.4 Amélioration de la précision : faire la somme dans le bon ordre

Une particularité des nombres flottants par rapport aux nombres réels manipulés en mathématiques est que l'addition n'est pas commutative. Par exemple :

```
print(1 + 1e-16 - 1) # Affiche 0.0
print(1 - 1 + 1e-16) # Affiche 1e-16
```

Lorsqu'on calcule une addition, Python fait une erreur d'arrondi qui est de l'ordre de 10^{-16} fois le plus grand élément de la somme. On a donc intérêt à sommer d'abord les plus petits éléments, pour que les premières additions fassent une erreur d'arrondi la plus petite possible, et terminer par les plus grands. Dans le cas de notre série, il faut mieux donc partir de $n = N$ et descendre jusqu'à $n = 0$ plutôt que l'inverse.

Implémentez cette variante, et comparez les précisions de la nouvelle version, et de l'ancienne (en partant de $n = 0$).

3.5 Troisième version : utiliser `cos` pour les valeurs proches de $\frac{\pi}{2}$

Implémentez la fonction `cos1` qui calcule le cosinus en utilisant son développement limité, sur le modèle de `sin1`, ainsi que la fonction `sin3` qui utilise l'identité $\sin(\frac{\pi}{2} - x) = \cos(x)$ et utilise `cos1` pour calculer le sinus.

Tracez à nouveau les courbes, et comparez l'erreur de `sin2` et `sin3` selon les valeurs de x . Que constatez-vous ?

3.6 Quatrième version : le meilleur des deux

Pouvez-vous déterminer approximativement une valeur « pivot » $x_0 \in [0, \frac{\pi}{2}]$ telle que pour $x < x_0$ l'une des deux fonctions `sin2` et `sin3` donne presque toujours une erreur moins grande, et pour $x \geq x_0$ ce soit l'autre fonction qui donne presque toujours une erreur moins grande. On évaluera approximativement cette valeur en visualisant les courbes d'erreur.

Implémentez la fonction `sin4` qui calcule le sinus en utilisant, selon la valeur de x , soit `sin2` soit `sin3`, et tracez son erreur, qui devrait être presque toujours inférieure à celle de `sin2` et `sin3`.

3.7 Tests

Complétez la partie tests et ajoutez-y des tests pour vérifier que vos fonctions `sin1`, `sin2`, `sin3` et `sin4` fonctionnent correctement. Testez des petites et des grandes valeurs (en vous rappelant qu'un nombre flottant Python strictement positif peut aller approximativement de 10^{-308} à 10^{308} , donc 0,1 n'est pas si petit, et 1000 n'est pas si grand pour un ordinateur ...).

En utilisant l'algorithme ci-dessus, vous n'obtiendrez probablement pas une bonne précision pour toutes les valeurs de x . Préciser dans votre rapport les limitations de votre implémentation.

Vous pouvez pour vos tests utiliser `math.sin` (qui nécessite un `import math`) comme référence. Une solution est d'afficher vos résultats en utilisant `print`. Une autre (suggérée dans le squelette) est d'utiliser la fonction Python `assert`, qui vérifie que son argument est vrai, et arrête le programme avec une erreur si ce n'est pas le cas.

On peut aussi vérifier des propriétés souhaitables des fonctions, comme $2 \sin(x)^2 \approx \sin(2x - \frac{\pi}{2}) + 1$.

4 Rendu

On demande un fichier au format Jupyter (.ipynb). Le fichier sera obligatoirement nommé ainsi `TPL1_Groupe_nom1_prenom1_nom2_prenom2.ipynb`, `Groupe` étant à remplacer par le numéro de votre groupe (A, B, C ou D), et `nom1` et `nom2` (`prenom1` et `prenom2`) étant à remplacer par le nom (prénom) des 2 membres de votre équipe.

Ce fichier devra inclure l'ensemble du programme Python (fonctions `sin1`, `sin2`, `sin3`, `sin4`, et programmes de test), les tracés de courbes, ainsi que des cellules qui présentent et analysent les résultats obtenus : y a-t-il des bugs connus ? la précision est-elle bien celle attendue ? ... Pour ajouter une nouvelle cellule qui contient du texte dans Jupyter, cliquez sur « + », puis sélectionnez « Markdown » à la place de « Code ».

4.1 Fraude

Il est interdit d'utiliser le TP d'une autre équipe, ou de mettre à disposition son TP à une autre équipe. Vous êtes encouragés à aller chercher de l'information sur internet, mais pas autorisés à copier-coller du code que vous n'avez pas écrit vous même. La sanction en cas de fraude est la note 0/20 au TP. En cas de doute, n'hésitez pas à demander conseil à vos enseignants.