Erreurs Numériques

Slide 1

Bonjour et bienvenue dans cette première vidéo du cours d'Analyse Numérique, consacrée à la notion d'erreur.

Slide 2

Avant de rentrer dans le vif du sujet, essayons tout d'abord de répondre à la question que vous vous posez certainement : qu'est-ce que l'Analyse Numérique, et à quoi ça sert?

L'Analyse Numérique consiste en la conception et l'étude d'algorithmes de résolution numérique (c'est à dire au moyen d'un ou plusieurs ordinateurs) de problèmes mathématiques. Ces problèmes mathématiques proviennent pour la plupart de la modélisation de phénomènes ou comportements issus du monde réel, au sein de disciplines aussi variées que la physique, la biologie, ou bien encore l'économie.

La notion fondamentale autour de laquelle s'articule l'Analyse Numérique est la notion d'erreur. Le terme erreur désigne ici la différence entre le résultat numérique d'un problème et la valeur théorique ou réelle que l'on est censé obtenir.

Ces erreurs peuvent avoir plusieurs origines. On distingue : - les erreurs de modélisation, liées au fait que la modélisation mathématique d'un problème repose sur un certain nombre d'hypothèses et d'approximations, permettant de simplifier le modèle final, - les erreurs de données, liées à l'imprécision des mesures physiques ou des calculs permettant de donner des valeurs aux différents paramètres d'un modèle, - les erreurs de discrétisation, liées à la transformation du problème mathématique continu (autrement dit d'équations) en un problème numérique discret, permettant de calculer la solution d'un problème en un nombre fini de points, - et enfin les erreurs d'arrondi, liées au fait que les ordinateurs ne peuvent représenter les réels qu'avec un nombre fini de chiffres. Nous reviendrons plus tard sur ce point.

Le but de l'Analyse Numérique sera (entre autres) d'arriver à estimer ces erreurs, de comprendre et identifier leurs origines, et de mettre au point des stratégies permettant de les limiter.

Slide 3

Comment calcule-t-on une erreur?

Prenons l'exemple d'un problème dont on connaît la solution théorique notée x, et dont la résolution numérique conduit à une valeur approchée notée $\tilde{x}$. Il existe deux manières de quantifier l'erreur commise entre la valeur théorique et la valeur numérique. On distingue : - l'erreur absolue, calculée en prenant la valeur absolue de la différence entre x et $\tilde{x}$, - l'erreur relative, calculée en divisant l'erreur absolue par la valeur absolue de la valeur théorique x. On remarquera que l'erreur absolue possède une unité physique (mètres, grammes...) qui est la même que celle de x et $\tilde{x}$, tandis que l'erreur relative est sans unité.

Parmi ces deux erreurs, seule l'erreur relative permet d'estimer correctement la qualité de l'approximation numérique $\tilde{x}$ par rapport à la valeur théorique x. En effet, prenons l'exemple d'un chronomètre dont la précision est limitée aux dizièmes de secondes. L'erreur absolue entre la durée réelle T d'une course et sa durée chronométrée $\tilde{T}$ est donc au maximum d'un dizième de seconde, et ce indépendemment de la course considérée. Mais ce n'est pas le cas de l'erreur relative. Dans le cas d'un marathon couru en approximativement 2 heures 20, l'erreur relative de chronométrage sera au maximum d'environ $1.19 \times 10^{(} - 5)$, soit une valeur suffisamment faible pour permettre a priori de classer sans problème tous les concurrents. En revanche, dans le cas d'un 100 mètres couru en 10 secondes, l'erreur relative sera de l'ordre de 0.01, soit 1%, ce qui pourrait s'avérer très problématique dans le cas fréquent d'une arrivée serrée. La qualité du chronométrage sera ainsi jugée bonne dans le cas du marathon, où l'erreur relative est faible, et mauvaise dans le cas du 100m, où l'erreur relative est nettement plus élevée.

Slide 4

Avant de nous intéresser à la manière dont les nombres réels sont représentés en machine, rappelons la manière dont nous les écrivons traditionnellement, via leur décomposition en base 10. Pour tout réel x, on sait qu'il existe un entier relatif N tel que x est compris entre 10^N et 10^{N+1}, et une suite (finie ou infinie) de chiffres a_i compris entre 0 et 9 tels que x s'écrive comme la somme des $a_i * 10^{N-i}$ pour i allant de 0 à éventuellement l'infini (si le nombre de décimales de x est infini par exemple). On représentera alors x sous la forme $a_0 a_1 ... a_N . a_{N+1} a_{N+2} ...$ où le point est la manière scientifique d'écrire la virgule qui permet de séparer la partie entière de x de sa partie décimale. Exemple : prenons $x = 1.25$, son écriture décimale vient du fait qu'il se décompose en base 10 de la manière suivante : $1.25 = 1 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$. Cette représentation, dite décimale à virgule fixe, est celle que nous utilisons dans la vie de tous les jours.

Il faut savoir que ce type de représentation peut s'étendre à n'importe quelle autre base b où b est un entier positif. En effet, pour tout réel x, il existera de la même manière un entier relatif M tel que x est compris entre b^M et b^{M+1}, et une suite (finie ou infinie) d'entiers naturels α_i compris entre 0 et $b - 1$ tels que x s'écrive comme la somme des $\alpha_i * b^{M-i}$ pour i allant de 0 à éventuellement l'infini. On représentera alors x dans cette

base par $\alpha_0\alpha_1...\alpha_M.\alpha_{M+1}\alpha_{M+2}....$ Exemple : reprenons $x = 1.25$ et écrivons le en base 2. Sa décomposition dans cette base s'écrit $x = 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$ et il s'écrira donc en base 2 $x = 1.01$.

On pourrait penser que cette représentation binaire des nombres réels est celle utilisée dans les ordinateurs, mais il n'en est rien, comme nous allons le voir tout de suite.

Slide 5

Intéressons nous donc maintenant à la manière dont les nombres réels sont représentés et stockés en machine, autrement dit dans la mémoire d'un ordinateur.

La première chose à comprendre est que, du fait du caractère fini de la mémoire d'un ordinateur, on ne peut pas représenter tous les nombres réels de manière exacte. En effet, en admettant que l'on travaille en base 10, il faudrait une mémoire de taille infinie pour représenter exactement le nombre pi, du fait du nombre infini de ses décimales. Les nombres réels seront donc, pour une bonne partie d'entre eux, représentés de manière approchée.

Le principal mode de représentation des nombres réels en machine est la représentation en virgule flottante normalisée. Cette représentation est utilisée aussi bien dans les calculatrices de poche que dans les stations de travail et les supercalculateurs. A tout réel x on associe sa représentation en machine notée $fl(x)$ par l'expression suivante : $fl(x) = \pm m.b^e$ où b est la base de la représentation, m est appelé la mantisse et e l'exposant. Précisons quelques points importants. Tout d'abord, il faut savoir que la plupart des machines utilisent la base 2 pour représenter les réels, afin d'utiliser les unités de stockage binaires. Deuxième précision : la mantisse m est un réel compris entre 1 et b strictement. Cette mantisse sera stockée en mémoire en utilisant son écriture en base b, tel que vu dans la diapositive précédente. Troisième point, et c'est un point important, la mantisse ne peut avoir plus de p chiffres en base b, p étant un entier positif fixé par le format utilisé. On comprend aisément l'intérêt de cette limitation, visant à utiliser une taille de stockage fixe et limitée pour chaque réel. La contrepartie, en revanche, est qu'une grande partie des nombres réels sera stocké de manière approchée, et non exacte. Dernière précision enfin, l'exposant e est un entier relatif, compris entre deux valeurs minimum et maximum.

La représentation d'un nombre réel en machine passera donc par le stockage des écritures en base 2 (ou b plus généralement) de sa mantisse m et de son exposant e. En ce qui concerne les valeurs de p et de e, il existe principalement deux formats de représentation, correspondant chacun à une précision spécifique : la simple précision et la double précision. En simple précision, on a $p = 24$ et e compris entre -126 et +127, ce qui donne une taille de stockage de 32 bits pour chaque réel. En double précision, ces valeurs sont plus élevées : p est égal à 53 et e est compris entre -1022 et +1023, pour une taille de stockage de 64 bits. L'ensemble des réels représentables de manière exacte sera donc plus grand en double précision qu'en simple précision.

Slide 6

Comme nous venons de le voir, il est impossible de représenter tous les réels de manière exacte en machine, et la plupart seront donc représentés de manière approchée. Il faut cependant noter que pour déterminer la représentation machine $fl(x)$ d'un réel x, il existe deux stratégies d'arrondi possibles : - soit par troncature, en ne gardant que les p premiers chiffres de la mantisse écrite en base b, - soit par arrondi au plus près, en prenant le réel représentable de manière exacte le plus proche en valeur absolue de x. Cela revient en pratique à arrondir la mantisse exacte à la mantisse de longueur p la plus proche, comme dans l'exemple affiché. La stratégie mise en oeuvre dans la plupart des ordinateurs est la stratégie d'arrondi au plus près.

Mais quelque soit la stratégie choisie, elle entraîne l'apparition d'une erreur d'arrondi, égale à $|x - fl(x)|/|x|$ pour tout réel x non nul. Cette erreur relative possède une valeur maximum, notée u et appelée unité d'arrondi. En simple précision, cette unité d'arrondi est égale à environ 10^{-7}, contre environ 10^{-16} en double précision. Ces valeurs sont égales (à un facteur 0.5 près) à que ce que l'on nomme la précision machine, qui est le plus grand réel strictement positif ε telle que $1 + \varepsilon$ soit arrondi à 1 en machine.

Ces erreurs d'arrondi peuvent engendrer deux types de problème. Le premier consiste en la perte de chiffres significatifs. Ce phénomène peut être dû à deux mécanismes : - le mécanisme d'absorption, lorsque l'on additionne des nombres d'ordres de grandeur très différents (comme 1 et ε par exemple), - et le mécanisme d'élimination, lorsque l'on soutrait des nombres très proches l'un de l'autre. Ces deux mécanismes, lorsqu'ils sont combinés, peuvent déboucher sur des erreurs de calcul catastrophiques. Prenons l'exemple suivant : si l'on cherche à calculer $1/(\sqrt{1+\varepsilon} - 1)$ via l'algorithme : $x = 1 + \varepsilon$; $y = \sqrt{x} - 1$; $z = 1/y$;, on obtiendra une division par zéro du fait du l'arrondi de x à 1 lors de la première étape. On évitera donc autant que possible d'utiliser des algorithmes de calcul susceptibles de faire apparaître des mécanismes d'absorption et/ou d'élimination.

Le second problème lié à ces erreurs d'arrondi est que ces dernières peuvent se cumuler à chaque opération arithmétique effectuée. En effet, lorsque l'on calcule par exemple $0.1 + 0.2$, on représente d'abord 0.1 et 0.2 par $fl(0.1)$ et $fl(0.2)$, puis on calcule la somme $fl(0.1) + fl(0.2)$, et on stocke enfin le résultat, qui sera ainsi égal à $fl(fl(0.1) + fl(0.2))$. On voit donc que pour une simple addition, on peut avoir jusqu'à trois erreurs d'arrondi, celles-ci pouvant éventuellement se cumuler. Ceci peut devenir très problématique lorsque l'on utilise un algorithme nécessitant d'effecteur un grand nombre d'opérations arithmétiques : les erreurs d'arrondi, pourtant faibles, peuvent alors se cumuler jusqu'à devenir suffisamment significatives pour fausser complètement le résultat.

Slide 7

Terminons cette vidéo par quelques mots sur le conditionnement et la stabilité.

Le conditionnement et la stabilité ont un but commun : analyser la sensibilité de la

solution d'un problème aux différentes erreurs pouvant interférer avec le calcul de celle-ci. Néanmoins, ces deux notions s'inscrivent dans deux contextes différents : - le conditionnement est une notion qui s'applique à un problème théorique (en dehors de tout contexte numérique), et qui mesurera principalement l'influence des erreurs de données sur la précision de la solution finale, - tandis que la stabilité est une notion qui s'applique à un algorithme de résolution numérique d'un problème donné, où l'on essaiera de quantifier la sensibilité de l'ensemble des opérations de l'algorithme aux erreurs d'arrondis principalement.

Plutôt que de rentrer plus dans les détails complexes de ces deux notions, prenons les exemples suivants.

Considérons le problème de la résolution du système $Ax = b$ où A est une matrice carrée quelconque et b un vecteur de même taille. Si l'une des valeurs propres de A est proche de 0, alors la solution exacte de ce problème sera très sensible à la moindre erreur de donnée : une simple petite erreur sur l'un des coefficients de A ou sur l'une des composantes de b entrainera une erreur importante sur la solution obtenue. On dit que ce problème est mal conditionné.

Considérons maintenant le problème du calcul de $y = \sqrt{x+1} - \sqrt{x}$. Ce problème est bien conditionné pour x grand. Pourtant, si l'on applique l'algorithme de calcul numérique suivant : $u = x + 1$; $v = \sqrt{u}$; $w = \sqrt{x}$; $y = v - w$, on observe que l'erreur relative sur le résultat final est d'environ 10% pour de grandes valeurs de x, ce qui démontre une grande sensibilité de la solution aux erreurs d'arrondis. Cet algorithme est ainsi dit instable. Ici, le problème vient du fait que la soustraction de deux réels a et b est un problème mal conditionné : si a et b sont proches, une petite erreur sur l'un des deux entraînera une erreur relative conséquente sur le résultat. Or dans cet algorithme, v et w sont proches, et les erreurs d'arrondi ont alors un impact important sur le calcul de y.

Nous reviendrons ultérieurement dans le cours sur ces notions de conditionnement et de stabilité.

Slide 8

Ceci clôt cette première vidéo, merci pour votre attention.