Informatique Appliquée au Calcul Scientifique 1

Séance 1

Représentation des nombres en machine

Table des matières

Introduction à l'informatique appliquée au calcul scientifique

Qu'est-ce que le calcul scientifique ?
Le calcul scientifique est une discipline qui utilise les techniques de calcul numérique pour résoudre des problèmes scientifiques et techniques complexes. Il combine les mathématiques appliquées, l'informatique et les connaissances spécifiques à divers domaines scientifiques pour modéliser et simuler des phénomènes physiques, biologiques, économiques ou sciences appliquées au sens large.

La Démarche du Calcul Scientifique

La démarche du calcul scientifique repose sur plusieurs étapes clés :

1. **Modélisation Mathématique** : Traduire le problème réel en un modèle mathématique.
2. **Analyse Mathématique** : Étudier le modèle pour comprendre ses propriétés et comportements.
3. **Méthodes Numériques** : Développer et utiliser des algorithmes pour obtenir des solutions approximatives du modèle.
4. **Implémentation Informatique** : Programmer les méthodes numériques sur des ordinateurs pour effectuer des calculs intensifs.
5. **Validation et Vérification** : Comparer les résultats numériques avec des données expérimentales ou des solutions analytiques pour vérifier leur exactitude.

Les Méthodes Numériques
Les méthodes numériques sont des techniques utilisées pour approximativement résoudre des problèmes mathématiques complexes. Ces méthodes comprennent notamment :

- Les méthodes des différences finies pour les équations différentielles.
- Les méthodes de Monte Carlo pour les intégrations complexes.
- Les algorithmes d'optimisation pour trouver les maxima ou minima de fonctions.
- Les méthodes de matrices et d'algèbre linéaire pour résoudre des systèmes d'équations linéaires.

Ces méthodes sont implémentées par des programmes informatiques capables de traiter de grandes quantités de données et d'effectuer des calculs à grande vitesse.
Dans tous les cas, il faudra avoir conscience des erreurs générés par la méthode utilisée afin de les maitriser et de les adapter aux contraintes du projet.

Objectifs du Calcul Scientifique
Les principaux objectifs du calcul scientifique sont :

- **Prédiction** : Prédire le comportement de systèmes complexes dans des conditions variées et dans des cas où on ne peut pas réaliser complétement une expérience (rupture d'une centrale nucléaire, crash d'avion,…).
- **Optimisation** : Trouver les meilleures solutions aux problèmes d'ingénierie et de gestion (optimisation de turbines ou de pièces diverses, …).
- **Analyse de Données** : Interpréter et analyser de grandes quantités de données provenant d'expériences ou de simulations (dans le domaine médical avec les données génomiques, analyse des données d'un capteur, …).
- **Visualisation** : Représenter graphiquement les résultats pour une meilleure compréhension (visualisation des données climatiques, imagerie médicale, …).

Exemple d'Application

Un exemple typique d'application du calcul scientifique est la simulation climatique. En utilisant des modèles mathématiques de l'atmosphère et des océans, les scientifiques peuvent prédire les changements climatiques futurs. Ils utilisent des méthodes numériques pour résoudre les équations différentielles complexes qui décrivent la dynamique des fluides et la thermodynamique. Les simulations informatiques permettent de tester des scénarios variés et de visualiser les impacts potentiels des changements climatiques.

I. Rudiments de programmation et Python

Les différents types de variables :

Pour connaitre le type d'une variable : `type(variable)`

Entier (integer ou int) : 5 ; 0 ; -7.
Réel (float) : -1.2, 2.2345 ; pi
Chaîne de caractères : "Bonjour" ; "12azerty".
Liste ou tableau : [12, -5, "Bonjour", [1 ; 12.2]]
Les éléments d'une chaîne de caractère ou d'une liste sont indexés (début à 0). Pour récupérer un élément d'une liste ou d'une chaîne de caractère : `liste[0]`.

Affectation de données à une variable :

Pseudo-code	Python
x prend la valeur 5 x← 5	`x = 5`

Écriture (sortie) de données :

Pseudo-code	Python
Afficher Hello World	`print("Hello World")`

Remarque :
La méthode F-string peut être plus pratique pour afficher la valeur des variables et contrôler plus simplement leur affichage :

print()	F-string

`nom = "Helene"`
`âge = 25`
`print("Nom:", nom, "Âge:", âge)`
	`nom = "Helene"`
	`âge = 25`
	`print(f"Nom: {nom}, Âge: {âge}")`

	`pi = 3.141592653589793`
	`print(f"Pi avec trois décimales: {pi:.3f}")`
	`# Output: Pi avec trois décimales: 3.142`

Lecture (entrée) de données :

Pseudo-code	Python
Saisir x	`x = input("saisir x : ")`

La valeur rentrée par l'utilisateur va être affecté à la variable.
Attention, par défaut, la valeur est stockée sous forme de chaine de caractères (string ou str). Si nous voulons utiliser le retour utilisateur pour des opérations mathématiques il faudra faire une conversion à l'aide des commandes int() (entier) ou float() (réel) :

```
x = int(input("saisir x : "))
```

Les structures de contrôle :
Attention aux indentations (une tabulation qui permettra d'inclure vos lignes de code dans la structure.

If :

Pseudo-code	Python

Si a est strictement plus grand que b
  Afficher "a est strictement plus grand que b"
Sinon si a est strictement plus petit que b
  Afficher "a est strictement plus petit que b"
Sinon
  Afficher "a est égal à b"
	`if a > b:`
	`  print("a est strictement plus grand que b")`
	`elif a < b:`
	`  print("a est strictement plus petit que b")`
	`else:`
	`  print("a est moins ou égal à b")`

Boucle for :

Pseudo-code	Python
Pour i allant de 0 à 4 Afficher i	`for i in range(5):` `  print(i)`
Pour i allant de 2 à 5 Afficher i	`for i in range(2, 6):` `  print(i)`
Pour i allant de 1 à 9 avec un pas de 2 Afficher i	`for i in range(1, 10, 2):` `  print(i)`

Boucle while :

Pseudo-code	Python

Tant que `a` est supérieur à 0
  Afficher `a`
  Décrementer `a` de 1
	`while a > 0:`
	`  print(a)`
	`  a -= 1  #ou a = a-1`

Création d'une fonction :

Pseudo-code	Python

Définir fonction NomDeLaFonction(paramètre1, paramètre2):
Faire quelque chose avec paramètre1 et paramètre2
Retourner un résultat
	`def additionner(a, b):`
	`  # Ajouter les deux paramètres`
	`  resultat = a + b`
	`  # Retourner le résultat`
	`  return resultat`

Arrondi:
fonction `round(nbre, arrondi voulu)`
ou
`print('{:.3f}'.format(536.1182))` (Retourne une chaine de caractère !)

Importation de bibliothèque :
`import math` puis appeler la fonction voulu `math.pi` par exemple, ou `from math import *` et appel de la fonction voulu `pi`.

À faire :
Exercice 1 :

1. Créer une fonction qui prendra en paramètres les valeurs a, b et c d'un polynôme du 2nd degré et retourne le nombre de solution ainsi que leur(s) valeur(s) arrondi au centième.
   Améliorer la fonction en rajoutant un paramètre qui sera pour gérer l'arrondi.
2. Écrire une fonction `cercle` qui prend le rayon d'un cercle et retourne son périmètre et son aire.

3. Écrire une fonction `mediane` qui prend une liste de nombres et retourne la médiane de ces nombres.
   Pour trier une liste, vous pouvez utiliser la fonction `sorted(liste)`.
   Pour la longueur d'une liste : `len(liste)`.

II. Représentation des nombres en machine

II.1. Zéro, infini et première limite

Nous allons dans cette partie chercher les premières limites simples de l'outil informatique pour l'affichage des nombres.

L'idée sera ici d'approcher le 0 par des puissances successives de 0,1 et de voir à partir de quand notre nombre sera interprété comme un 0 et non plus par sa vraie valeur.
Dans un premier temps, on remarque que $0{,}1^{300}$ (`0.1**300` en Python) est bien représenté mais $0{,}1^{400}$ (`0.1**400` en Python) retourne un 0.
Il existe donc un plus petit nombre non nul que la machine est capable de représenter et nous savons qu'il est compris entre $0{,}1^{300}$ et $0{,}1^{400}$. Notons-le α.

$$\exists \alpha > 0, si\ x\ est\ représentable\ en\ machine\ alors\ |x| > \alpha.$$

Nous savons que $\alpha \in [0{,}1^{300};\ 0{,}1^{400}]$.
Partons à sa recherche…

Pour cela, comme la liste de valeurs à tester est grande, bous allons utiliser un algorithme de dichotomie.

Point Méthode :
La recherche par dichotomie (díkha (« en deux ») et de tomós (« section, coupure »), est une méthode efficace pour trouver la position d'un élément ou un élément dans une liste triée.
Elle fonctionne en divisant la liste en deux parties à chaque étape et en comparant l'élément à rechercher avec l'élément central de la liste.
Par comparaison, on ne garde d'une des 2 parties et on recommence jusqu'à trouver l'élément…

Nous cherchons ici le plus petit α tel que $0{,}1^{\alpha} = 0$ et $0{,}1^{\alpha-1} \neq 0$.
Nous savons que $\alpha \in [0{,}1^{300};\ 0{,}1^{400}]$.
Voici le pseudocode de notre algorithme :

```
Définir fonction trouver_alpha(min_alpha, max_alpha):
    alpha = -1
    Tant que min_alpha <= max_alpha:
        milieu = (min_alpha + max_alpha) // 2
        si 0.1^milieu == 0:
            max_alpha = milieu - 1
        sinon si 0.1^(milieu + 1) == 0:
            alpha = milieu
            retourner alpha
        sinon:
            min_alpha = milieu + 1
    retourner alpha
```

À vous de jouer :
Exercice 2 :
1. Écrire le script Python du code ci-dessus et le tester…

Nous allons faire de même pour trouver la plus grande puissance de 10 représentable en machine.
Pour cela, nous allons utiliser deux tests :
float('inf') qui permet de savoir si le nombre est considéré comme infini et OverflowError qui est une exception qui est levée lorsqu'une opération mathématique produit un résultat trop grand pour être représenté par le type numérique utilisé.

2. Recopier le code suivant et le tester :

```
def trouver_limite_puissance_10():
    min_n = 0
    max_n = 100  # On commence avec une limite haute large
    # Étendre la recherche jusqu'à trouver la limite supérieure
    while True:
        try:
            if float('inf') == 10.0 ** max_n:
                break
            max_n *= 2
        except OverflowError:
            break
    print(min_n, "/", max_n)
    # Dicho pour trouver la plus petite puissance de 10 non représentable
    while min_n <= max_n:
        milieu = (min_n + max_n) // 2
        try:
            if float('inf') == 10.0 ** milieu:
                max_n = milieu - 1
```

```
            else:
                min_n = milieu + 1
        except OverflowError:
            max_n = milieu - 1
    return min_n - 1
```

On a trouvé que 10^{308} est bien représenté en machine mais 2.0×10^{308} retourne un overflow.
$(\frac{1}{10})^{323}$ est bien représenté en machine mais $(\frac{1}{10})^{324}$ est représenté par 0.

I.2. Dichotomie et résolution d'équation

L'algorithme de dichotomie peut être mis en œuvre pour résoudre les équations de type $f(x) = 0$, où f est une fonction continue de $\mathbb{R}$ dans $\mathbb{R}$ grâce au Théorème des Valeurs Intermédiaire (TVI) :

Théorème des Valeurs Intermédiaire / Bolzano :
Soient a et b deux nombres réels, $f : [a;\ b] \longrightarrow \mathbb{R}$ une fonction continue telle que $f(a).f(b) < 0$, alors il existe $c \in]a; b[\ tel\ que\ f(c) = 0$.

Remarques :

- La notion de continuité est ici essentielle.
- $f(a).f(b)$ permet d'avoir $f(a)$ et $f(b)$ de signe contraire.

Application :
Nous pouvons ainsi trouver une approximation de $\sqrt{2}$ en choisissant $f(x) = x^2 - 2$ et l'intervalle $[1\ ;\ 2]$.
Nous y reviendrons un peu plus tard.

II.3. Erreur d'arrondi

Quelques algorithmes pour comprendre :

```
0.1 + 0.2 == 0.3
```

```
somme = 0.0
for i in range(10):
    somme += 0.1
    print(somme)
print(f"0.1 ajouté 10 fois donne {somme}")
```

```
sommebis = 1
for i in range(1, 18):
    somme = sommebis + 0.1**i
    print(f"1+0.1**{i}={somme}")
```

On voit grâce à ces 2 exemples les erreurs d'arrondis inhérents à l'utilisation de l'outil informatique dans les calculs.
Mais pourquoi ces erreurs ?
Cela est dû à la façon dont ils sont représentés.

II.4. Représentation des flottants : norme IEEE 754

II.4.a. Décimale vers binaire (partie entière) :

La méthode la plus courante pour convertir un nombre entier décimal en binaire est la méthode des divisions successives par 2.

Exemple : Convertir 13 en binaire

1. 13÷2=6 (quotient), reste 1
2. 6÷2=3 (quotient), reste 0
3. 3÷2=1 (quotient), reste 1
4. 1÷2=0 (quotient), reste 1

En lisant les restes de bas en haut, on obtient 1101. Donc, $(13)_{10} = (1101)_2$.

II.4.b. Décimale vers binaire (partie décimale) :

La conversion de la partie décimale d'un nombre (après la virgule) se fait par une méthode différente, celle des multiplications successives par 2.

Exemple : Convertir 0.625 en binaire

1. 0,625×2=1,25 => partie entière 1, nouvelle partie décimale 0,25.
2. 0,25×2=0,5 => partie entière 0, nouvelle partie décimale 0,5.
3. 0,5×2=1,0 => partie entière 1, nouvelle partie décimale 0.

En lisant les parties entières dans l'ordre, on obtient 101. Donc, $(0{,}625)_{10} = (0{,}101)_2$.

Exercice 3:

Convertir en binaire les nombres suivants :

a. $(5{,}75)_{10}$ b. $(12{,}375)_{10}$ c. $(10{,}625)_{10}$ d. $(133{,}456)_{10}$

II.4.c. norme IEEE 754:

Représentation en virgule flottante normalisée

On appelle nombre à virgule flottante (ou flottant ou float en anglais) un nombre de la forme :

$$x = s \times m \times b^e$$

où $s \in \{0;\ 1\}$ est le signe de x, m sa mantisse, e l'exposant (entier relatif) et b la base dans laquelle on travaille.

On a ainsi :

$$+5{,}75 \times 10^0 = +575 \times 10^{-2}$$
$$+101{,}11 \times 2^0 = +10111 \times 2^{-2} = +1{,}0111 \times 2^2$$

Réponses exercice 3 :

a. $(5{,}75)_{10} = (101{,}11)_2$ *b.* $(12{,}375)_{10} = (1100{,}011)_2$ *c.* $(10{,}625)_{10} = (1010{,}101)_2$

d. $(133{,}456)_{10} = (10000101{,}01110100011110101110)_2$

Une *représentation en virgule flottante normalisée* est une représentation dans laquelle la mantisse est de la forme $b_0, b_{-1}b_{-2}$ … avec b_0 non nul.

Ainsi en base 2, le premier chiffre avant la virgule de la mantisse est toujours égal à 1.

$+1{,}0111 \times 2^2$ et $+5{,}75 \times 10^0$ sont des *représentations en virgule flottante normalisée.*

La norme IEEE 754

La norme IEEE 754 (Institute of Electrical and Electronics Engineers) est une norme pour le formatage des nombres en virgule flottante, qui est largement utilisée dans les ordinateurs et les systèmes numériques pour représenter les nombres réels. Cette norme définit à la fois les formats de données pour les nombres en virgule flottante et les méthodes pour effectuer les opérations arithmétiques sur ces nombres.

Nous ne la verrons que partiellement en se restreignant au format double précision (64 bits).

nombre normalisé	signe	exposant décalé E	mantisse
$\pm 1, b_{-1}b_{-2}\dots b_{-52} \times 2^e$	un bit (n°63)	$e + 1023$ sur 11 bits	sur 52 bits
$+1{,}0111 \times 2^2$	0	$2 + 1023 = (1025)_{10}$ $= (10000000001)_2$	$10111000 \dots 0$ qui sera représenté par $0111000 \dots 0$

Cas particuliers :

- Le bit de poids fort (n°63) définit le signe. Il est égal à 0 pour les nombres positifs, à 1 pour les nombres négatifs.
- Les 52 bits de n°0 à 51 servent à coder la mantisse. Le premier chiffre de la mantisse étant toujours 1 dans une représentation normalisée en base 2, il n'est pas représenté (on parle de bit caché ou implicite).
- Le zéro est représenté avec une mantisse égale à 0 et un exposant après décalage de 0 ;
- Les infinis ont une mantisse de 0 et un exposant avec que des 1.

On voit ainsi que du fait de la place dans la mémoire et de la façon dont les nombres sont représentés, il peut y avoir des arrondis qui sont faits et ainsi engendrer des erreurs dans les calculs.
Reprenons notre 0,1 + 0,2 qui nous donne un résultat différent de 0,3 en machine.

- **en binaire** : $0.0001100110011001100110011001100110011001100110011001100110011 \dots$
  Soit en écriture normalisé : $1{,}1001100110011001100110011001100110011001100110011 \times 2^{-4}$.
  Ce qui donnera un exposant $de\ -4 + 1023\ =\ 1019$ soit 01111111011 en binaire et une mantisse de $1001100110011001100110011001100110011001100110011010$

0.2 en binaire : $0.0011001100110011001100110011001100110011001100110011001100110011..$
Soit en écriture normalisé : $1{,}1001100110011001100110011001100110011001100110011001100110011 \times 2^{-3}$. Ce qui donnera un exposant $de\ -3 + 1023\ =\ 1020$ soit 01111111100 en binaire et une mantisse de $1001100110011001100110011001100110011001100110011010$

Le fait que nous ne puissions pas faire ce calcul avec les valeurs exactes vient du fait des limitations de la machine qui engendre l'erreur observée.

Exercice 4 :

1. Convertir $-13{,}25$ en format IEEE 754 double précision.
2. Quel est le nombre dont la représentation IEE 754 sur 64 bits est :

s	E	m
1	10000000010	0100000000000000000000000000000000000000000000000000

3. Quel est le plus petit nombre flottant strictement positif ? Donner sa représentation, puis une valeur approchée.
4. Quel est le plus grand nombre flottant ? Donner sa représentation, puis une valeur approchée.

III. Est-ce vraiment important de se soucier de ces erreurs…

Oui…

De gros problèmes ont été causés par des problèmes liés aux limitations du calcul sur machine qui n'ont pas été pris en compte.

III.1. Explosion de la fusée Ariane 5 (1996)

Contexte :

Le 4 juin 1996, la fusée européenne Ariane 5 a explosé seulement 37 secondes après son lancement en raison d'un défaut logiciel.

Cause :

Une conversion incorrecte d'un nombre en virgule flottante 64 bits (double précision) en un entier 16 bits dans le système de guidage inertiel de la fusée. La valeur de la vitesse horizontale, trop élevée pour être contenue dans un entier 16 bits, a provoqué une erreur de débordement et le système de secours souffrait du même problème.

Conséquence :

La perte de la fusée Ariane 5 et de sa cargaison, évaluée à environ 500 millions de dollars.

III.2. Incident de la plateforme pétrolière Sleipner A (1991)

Contexte :

En août 1991, la plateforme pétrolière Sleipner A en mer du Nord a coulé pendant son transfert en mer.

Cause :

Une erreur d'arrondi dans un logiciel de calcul par éléments finis utilisé pour modéliser la flottabilité de la plateforme. Les erreurs d'arrondi ont entraîné des inexactitudes dans les calculs de résistance structurale.

Conséquence :

La plateforme a subi une rupture catastrophique et a coulé, entraînant des pertes de près de 700 millions de dollars.

III.3. Erreur de calcul des missiles Patriot (1991)

Contexte :

Pendant la guerre du Golfe en 1991, un missile Patriot a échoué à intercepter un missile Scud irakien, entraînant la mort de 28 soldats américains et la blessure de 98 autres.

Cause :

Une erreur d'arrondi dans le logiciel de suivi des cibles du missile Patriot. La conversion répétée d'un nombre en virgule flottante à une valeur entière a conduit à une erreur accumulée de 0,34 seconde dans le temps de suivi, ce qui a entraîné une erreur de position de 500 mètres.

Conséquence :
L'échec de l'interception du missile Scud, entraînant des pertes humaines et matérielles importantes.

III.4. Incident du vol 143 d'Air Canada (1983)

Contexte :
Le vol 143 d'Air Canada, également connu sous le nom de "Gimli Glider", a dû effectuer un atterrissage d'urgence sans carburant en raison d'une erreur de calcul du carburant.

Cause :
Une confusion entre les unités métriques et impériales lors de la conversion des quantités de carburant, exacerbée par des erreurs d'arrondi.

Conséquence :
L'avion a perdu les deux moteurs en plein vol, mais l'équipage a réussi à atterrir en toute sécurité sur une ancienne base militaire convertie en piste de course, évitant ainsi des pertes humaines.

III.5. Incident du Pentium FDIV (1994)

Contexte :
Une erreur dans la puce de calcul du processeur Intel Pentium a provoqué des erreurs de division en virgule flottante pour certaines opérations.

Cause :
Un bug dans le matériel de la puce, où certaines divisions produisaient des résultats incorrects en raison d'une table de recherche incomplète dans la puce.

Conséquence :
Intel a dû rappeler les puces défectueuses et offrir des remplacements, ce qui a coûté à l'entreprise environ 475 millions de dollars et a terni sa réputation.

Correction :

Exercice 1 :

```
import math
from math import*
def cercle(rayon):
    perimetre = 2 * pi * rayon
    aire = math.pi * rayon ** 2
    return perimetre, aire

def mediane(liste):
    liste_triee = sorted(liste)
    n = len(liste_triee)
    milieu = n // 2

    if n % 2 == 0:  # Si la longueur de la liste est paire
        mediane = (liste_triee[milieu - 1] + liste_triee[milieu]) / 2
    else:  # Si la longueur de la liste est impaire
        mediane = liste_triee[milieu]

    return mediane

# Exemple d'utilisation
liste = [3, 1, 4, 1, 5, 9, 2, 6, 5]
mediane_liste = mediane(liste)
print(f"La médiane de la liste est: {mediane_liste}")
```


Exercice 2 :

I.

```
def trouver_alpha(min_alpha, max_alpha):
    alpha = -1
    while min_alpha <= max_alpha:
        milieu = (min_alpha + max_alpha) // 2
        if 0.1 ** milieu == 0:
            max_alpha = milieu - 1
        elif 0.1 ** (milieu + 1) == 0:
            alpha = milieu
            return alpha
        else:
            min_alpha = milieu + 1
    return alpha  # Si aucun alpha n'est trouvé dans l'intervalle

# Appel de la fonction avec l'intervalle donné
min_alpha = 300
max_alpha = 400
alpha = trouver_alpha(min_alpha, max_alpha)

print(f"Le plus petit entier alpha tel que 0.1^alpha est différent de zéro et
0.1^(alpha+1) est égal à zéro est : {alpha}")
```

2.

```
def trouver_limite_puissance_10():
    min_n = 0
    max_n = 100  # On commence avec une limite haute large
```

```
    # Étendre la recherche jusqu'à trouver la limite supérieure
    while True:
        try:
            if float('inf') == 10.0 ** max_n:
                break
            max_n *= 2
        except OverflowError:
            break
    print(min_n, "/", max_n)
    # Dicho pour trouver la plus petite puissance de 10 non représentable
    while min_n <= max_n:
        milieu = (min_n + max_n) // 2
        try:
            if float('inf') == 10.0 ** milieu:
                max_n = milieu - 1
            else:
                min_n = milieu + 1
        except OverflowError:
            max_n = milieu - 1

    return min_n - 1

# Appel de la fonction pour trouver la limite
limite_puissance = trouver_limite_puissance_10()

print(f"La plus petite puissance de 10 qui ne peut plus être représentée est
10^{limite_puissance + 1}.")
```


Réponses exercice 4 :

1. $-13{,}25 = -1{,}10101 \times 2^3$

Exposant après décalage : $3 + 1023 = (1026)_{10} = (10000000010)_2$

S | E | F

1 | 10000000010 | 10101000000000000000000 … 0

2. $E = 2^{10} + 2 = 1026$ donc $e = 1026 - 1023 = 3$.

On a une écriture normalisée en base qui est : $-1{,}01 \times 2^3 = -1010 \times 2^0 = (-10)_{10}$.

3. Il est obtenu avec une mantisse nulle et un exposant minimal, c'est-à-dire pour E = 1 donc e = 1 − 1023 = −1022, c'est donc $1.0 \times 2^{-1022} \approx 2.225 \times 10^{-308}$.

4. Il est obtenu avec une mantisse maximale donc 111 . . . 111 et un exposant maximal donc E = 2046 donc $e = 2046 - 1023 = 1023$.

On a une écriture normalisée en base qui est : $1{,}111 \ldots 1 \times 2^{1023} = 1111 \ldots 1 \times 2^{-52} \times 2^{1023}$.

Ce qui donne en base 10 :

$$(2^{52} + 2^{51} + 2^{50} + \cdots + 2^0) \times 2^{-52} \times 2^{1023} = (1 + 2^{-1} + 2^{-2} + \cdots + 2^{-52}) \times 2^{1023}$$
$$= 2 \times (1 - 2^{-53}) \times 2^{1023} = (1 - 2^{-53}) \times 2^{1024} \approx 1{,}797 \times 10^{308}$$

Ainsi, le nombre $1{,}2 \times 10^{309}$ ne peut être représenté par un flottant, on a un « overflow».

Cours de B Moreau