Notations : un nombre flottant normalisé f est de la forme

$$f = \pm \left(\sum_{k=1}^{r} \frac{a_k}{b^k} \right) b^e = \pm m b^{e-r}, \quad m = \sum_{k=1}^{r} a_k b^{r-k},$$

- $b > 1$ est un entier (la base), la plupart du temps 2, le même pour tous les flottants
- r est le nombre de chiffres (ou bits) significatifs, le même pour tous les flottants (sauf en multi-précision)
- $m < b^r$ est la mantisse de f, un entier positif dont l'écriture en base b est $a_1, \ldots, a_r$ avec $a_1 \in]0, b-1]$.
- e est l'exposant de f, $L \leq e \leq U$.

La norme IEEE-754 (langages de programmation, logiciels de calcul numérique...) utilise $b = 2$, $r = 53$, $L = -1021$, $U = 1024$).
La saisie d'un nombre flottant se fait en écrivant la mantisse en base 10 avec un point décimal, suivit éventuellement de la lettre `e` et de l'exposant en base 10, même si le flottant est stocké en base 2. Par exemple `6.02e23` pour le nombre d'Avogadro. Attention, dans la plupart des languages `10^10` est un entier exact, à ne pas confondre avec le flottant `1e10`.

Exercice 1 Ecriture en base b d'un rationnel Soit b un entier supérieur ou égal à 2. Soit p et q deux entiers strictement positifs tels que $p < q$.
Soit $\frac{p}{q} = \sum_{k=1}^{+\infty} \frac{a_k}{b^k}$ le développement en base b de $\frac{p}{q}$.

1. Montrer que la suite (a_k) est obtenue de la façon suivante :

   $r_0 = p$, a_1 est le quotient de la division euclidienne de $r_0 b$ par q et r_1 le reste.

   Si $(r_0, \cdots, r_n)$ et $(a_1, \cdots, a_n)$ sont définis alors a_{n+1} est le quotient de la division euclidienne de $r_n b$ par q et r_{n+1} est son reste.
2. Montrer que la suite (a_k) est périodique à partir d'un certain rang.
3. Donner le developpement de $x = \frac{1}{10}$ en base 2. Soit $\widehat{x}$ le flottant arrondi normalisé avec 12=11+1 bits de mantisse correspondant à x. Calculer $\widehat{x}$. Même question pour $y = 3/10$. Faire la somme $\widehat{x} + \widehat{y}$.
4. Expliquez le résultat de l'opération `0.3-(0.1+0.1+0.1)` dans Python/Octave/Xcas.

Exercice 2 Durant la première guerre du Golfe, une batterie anti-missile Patriot a raté l'interception d'un missile Scud qui causa la mort de 28 personnes. Nous allons essayer de comprendre pourquoi.
L'ordinateur de la batterie Patriot représente les nombres par des nombres à virgule fixe en base 2 avec une précision de 23 chiffres après la virgule. L'horloge de la batterie

compte le temps en dixième de seconde. Pour obtenir le temps en seconde, le programme multiplie le temps donné par l'horloge par dix.
1) En utilisant l'écriture en base 2 de $\frac{1}{10}$ trouvée dans l'exercice précédent, donner une estimation de l'erreur d'approximation de $1/10$ par la représentation des nombres de la batterie Patriot.
2) La batterie fonctionne pendant 100 heures. Donner une estimation de l'erreur introduite dans le temps en secondes.
3) Sachant qu'un missile Scud voyage à 1676 mètres par seconde, donner l'erreur de distance commise par la batterie anti-missile.

Exercice 3 Donner la valeur exacte de chaque expression suivante pour $x = 10^{20} + 1$ et $y = 10^{20}$

$$x - y \quad , \quad \binom{1000}{2} = \frac{1000!}{2(998!)} \ .$$

On donne ces calculs à faire à une machine. Sa réponse correspondra-t-elle à l'expression exacte ? (Discuter selon le mode de calcul, exact ou approché, et selon la précision). Même questions pour la valeur de

$$\left(\sum_{k=1}^{N} \frac{1}{n} \right) - \ln(N)$$

pour N grand.

Exercice 4 Les nombres suivants peuvent se calculer selon deux manières notées A et B. Quelle est l'expression la plus judicieuse pour le calcul approché sur machine ?

— $A = 1 - \cos^2(10^{-8}), \quad B = \sin^2(10^{-8})$
— $A = \sum_{n=1}^{10^9} \frac{1}{n}, \quad B = \sum_{n=0}^{10^9-1} \frac{1}{10^9-n}$.
— $A = e^{-10} \approx \sum_{n=0}^{30} \frac{(-10)^n}{n!}, \quad B = \frac{1}{e^{10}} \approx \left(\sum_{n=0}^{30} \frac{10^n}{n!}\right)^{-1}$

Proposez une méthode de calcul approché précise pour calculer :

$$\frac{1}{\sqrt{10^{10}+1} - \sqrt{10^{10}-1}}$$

Exercice 5 : évaluation d'un polynôme en un réel. On veut calculer la valeur en x d'un polynome $\sum_{k=0}^{n} a_k X^k$. On suppose que $a_0, \ldots, a_n, x$ sont représentés exactement par des nombres flottants (on pourra aussi réfléchir aux erreurs introduites si la représentation n'est pas exacte !).

1. On fait le calcul de la manière suivante : $s_0 = a_0, u_1 = x, v_1 = a_1 u_1$ et pour $k \geq 1$

$$s_k = s_{k-1} + v_k, \quad u_{k+1} = u_k x, \quad v_{k+1} = u_{k+1} a_{k+1}$$

   Combien a t-on fait de multiplications et d'additions ?

2. Méthode de Hörner

   Le principe est d'écrire $P(X) = \sum_{k=0}^{n} a_k X^k = a_0 + X(a_1 + X(\cdots + X(a_{n-1} + X a_n)))$.

   On pose pour $0 \leq k \leq n$,

   $$p_k = a_k + a_{k+1}x + \cdots + a_n x^{n-k}$$

   On a alors $P(x) = p_0$. On calcule les termes par une récurrence descendante
   $p_n = a_n$ et $p_{k-1} = a_{k-1} + xp_k$
   Combien a t-on fait de multiplications et d'additions ?

3. Comparer la rapidité d'exécution en utilisant `timeit(...)` (en Python, après avoir importé le package time) ou `time()` (en Xcas).

4. *Difficile* : Estimation des erreurs par ces deux méthodes (attention, question délicate !).

   On note u l'erreur d'arrondi/troncature machine. On rappelle que si x et y sont des flottants, si $* \in \{+, \times\}$, alors il existe ϵ tel que $x \circledast y = (1+\epsilon)(x*y)$ et $|\epsilon| \leq u$.
   En utilisant la première méthode, on a : $\widehat{s_0} = a_0; \widehat{u_1} = x; \widehat{v_1} = a_1 \otimes \widehat{u_1}$ et pour $k \geq 1$,

   $$\widehat{s_k} = \widehat{s_{k-1}} \oplus \widehat{v_k}, \quad \widehat{u_{k+1}} = \widehat{u_k} \otimes x, \quad \widehat{v_{k+1}} = \widehat{u_{k+1}} \otimes a_{k+1}$$

   La valeur donnée par la machine est donc $\widehat{s_n}$.
   On note $\gamma_n(u) = (1+u)^{n+1} - 1$. Montrer par récurrence que

   $$|\sum_{k=0}^{n} a_k x^k - \widehat{s_n}| \leq \gamma_n(u) \ \sum_{k=0}^{n} |a_k||x|^k$$

   (On pourra montrer $|\hat{v}_n/v_n| \leq (1+u)^n$, $|\hat{v}_n - v_n| \leq \gamma_{n-1}(u)|v_n|$, $|\hat{s}^n| \leq (1+u)^{n+1} \sum |a_k x^k|$). Peut-on en déduire une information sur l'erreur relative du résultat renvoyé, en faisant éventuellement des hypothèses sur les signes ?
   Les processeurs de calcul flottant utilisent souvent une représentation plus précise que le standard IEE-754, avec 64 bits de mantisse (+1 implicite) au lieu de 52 bits (+1 implicite). Si on effectue les calculs intermédiaires de cette manière (ou plus généralement en multi-précision), qu'y gagne-t-on ?
   En notant $\widehat{p_k}$, le flottant obtenu par la méthode de Hörner, on peut montrer que

   $$|P(x) - \widehat{p_0}| \leq \sum_{k=0}^{n} \gamma_{2k+2}(u)|a_k||x|^k$$

   Cette majoration sera meilleure que celle obtenue par la première méthode si les termes $a_k x^k$ décroissent vers 0, mais elle reste du même type.

Exercice 6 On souhaite calculer l'intégrale

$$I_n = \int_0^1 \frac{x^n}{10+x} dx$$

pour $n \in \mathbb{N}$.

1. Calculer I_0.
2. Montrer que pour tout $n \in \mathbb{N}^*$,
$$I_n = \frac{1}{n} - 10 I_{n-1}.$$
En déduire une procédure pour calculer I_n pour n quelconque.
3. Utiliser la procédure de (2) pour calculer en flottant I_5, I_{10}, I_{20}. Les estimations obtenues vous paraissent-elles raisonnables (noter par exemple que, pour tout n, $0 \leq I_n \leq 1$) ?
4. Montrer que l'on peut renverser la procédure ci-dessus :
$$I_{n-1} = \frac{1}{10n} - \frac{I_n}{10}.$$
A partir d'une estimation grossière de I_{30}, utiliser cette procédure inverse pour estimer I_{20}. Comparer le résultat avec la valeur de l'intégrale fournie par Xcas.

Exercice 7 Pour calculer la valeur numérique de la dérivée d'une fonction f suffisamment régulière, on peut utiliser l'une des formules suivantes :
$$f_1 = \frac{f(x+h) - f(x)}{h}, \quad f_2 = \frac{f(x+h) - f(x-h)}{2h}$$
Mais si h est trop petit, alors $x + h = x$ en flottants, et même si $x + h \neq h$, on va introduire des erreurs relatives importantes en faisant la différence entre deux termes proches si h est trop petit.

1. Justifiez les formules f_1 et f_2 en faisant un développement de Taylor de f en x.
2. On suppose que les flottants normalisés sont représentés avec une erreur relative de ε proche de `1e-15`. Déterminer en fonction de h l'erreur absolue pour $f(x+h) - f(x)$ et $f(x+h) - f(x-h)$ en supposant que f, f' et f'' sont de taille proche de 1 au voisinage de x. En déduire l'erreur relative.
3. Pour quelle valeur de h obtient-on une erreur optimale pour les approximation f_1 et f_2 (on tiendra uniquement compte de la première puissance de h non nulle dans le reste de la première question) ? On se limitera donc à des valeurs de h supérieures ou égales.
4. Expérimentez avec des fonctions dont on connait l'expression de la dérivée, par exemple $\sqrt{\exp(\sin(x))}$ en $x = 0$.
5. Proposez une formule approchant la dérivée seconde de f en x faisant intervenir $f(x-h), f(x), f(x+h)$ et discutez la valeur de h optimale. Expérimentez.

Problème 8 (différences finies en dimension 1) Soit A la matrice tridiagonale de taille N donnée par :

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \mathbf{0} \\ & \ddots & \ddots & \ddots & \\ \mathbf{0} & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$

1. Montrer que

$$u_k = (\sin(\frac{k\pi}{N+1}), ..., \sin(\frac{kj\pi}{N+1}), ..., \sin(\frac{kN\pi}{N+1}))$$

   est vecteur propre de A et calculer la valeur propre associée. En déduire que si $\tilde{C}$ est une matrice diagonale à coefficients positifs, alors $A + \tilde{C}$ a toutes ses valeurs propres strictement positives et est inversible.

2. On veut résoudre numériquement

$$-u''(x) + c(x)u(x) = f(x), u(0) = 0, u(1) = 0, c(x) \geq 0, \quad x \in]0, 1[$$

   On discrétise $[0, 1]$ en $N + 1$ intervalles de même longueur $h = 1/(N + 1)$ et on approche $u(nh)$ par $u_n, n \in [1, N]$ (avec $u(0) = u_0 = 0$ et $u(1) = u_{N+1} = 0$). On pose $f_n = f(nh), \vec{f} = (f_1, ..., f_n)$, $c_n = c(nh)$ et C la matrice de diagonale $(c_1, ..., c_N)$. Déterminer le système linéaire que vérifie $\vec{u} = (u_1, ..., u_N)$ en fonction de $A/h^2, C, \vec{f}$.

3. Effectuer la résolution numérique par une méthode de résolution directe. On pourra utiliser, en Python, `A:=np.zeros((N,N))` pour créer une matrice dense de taille N, puis `A[k,k]=2.0` pour mettre la diagonale à 2, etc, et la fonction `np.linalg.solve`.

4. Observez le temps de calcul en fonction de N, par exemple pour $c(x) = x, f(x) = (1 + 2x - x^2)e^x + x^2 - x$
   Représenter graphiquement l'erreur avec la solution exacte $u(x) = (1 - x)(e^x - 1)$ (on pourra utiliser `from matplotlib import pyplot as plt` puis la fonction `plt.plot` en Python).

5. Méthodes itératives : on peut utiliser la méthode de Jacobi pour obtenir une valeur approchée de $\vec{u} = (u_1, ..., u_N)$ plus rapidement lorsque N est grand. On écrit $\tilde{A} = D + (\tilde{A} - D)$ où D est la partie diagonale de $\tilde{A}$, puis le système $\tilde{A}\vec{u} = b$ en remplaçant $\vec{u}$ par deux termes successifs d'une suite récurrente v_k (à valeurs dans $\mathbb{R}^N$) :

$$Dv_{k+1} + (\tilde{A} - D)v_k = b \;\Rightarrow\; v_{k+1} = v_k + D^{-1}(b - \tilde{A}v_k)$$

   Il peut alors être judicieux d'utiliser une matrice creuse pour stocker $\tilde{A}$, quel est le nombre d'opérations à faire par itération ?

On peut justifier la convergence de v_k lorsque $C = 0$ (et la vitesse de convergence) en regardant la norme euclidienne de la matrice $I - D^{-1}\tilde{A}$ en se plaçant dans la base orthonormée de vecteurs propres de $\tilde{A}$. Dans le cas général, on peut appliquer le théorème de convergence de la section sur les méthodes alternatives au pivot du cours. Voir aussi la méthode de Gauss-Seidel et relaxation.