DIRO
IFT 2425

EXAMEN INTRA

Max Mignotte

DIRO, Département d'Informatique et de Recherche Opérationnelle, local 2377
Http : //www.iro.umontreal.ca/∼mignotte/ift2425/
E-mail : mignotte@iro.umontreal.ca

Date : *23/02/2023*

I	Erreur/Amplification d'erreur en arithmétique flottante (25 pts)
II	Erreur en arithmétique flottante (34 pts)
III	Recherche des racines d'une équation (59 pts)
Total	118 points.

TOUS DOCUMENTS PERSONNELS, CALCULATRICES ET CALCULATEURS AUTORISÉS

I. Erreur/Amplification d'erreur en arithmétique flottante (25 pts)

1. L'exercice suivant va mettre en évidence les légères erreurs (de flottement) qui se produisent (notamment loin derrière la virgule) lorsqu'on travaille avec des nombres flottants sur ordinateur et qui peuvent s'accumuler dramatiquement si on n'y fait pas attention.

   Dans un premier temps, rappelons brièvement comment on convertit un nombre flottant inférieur à 1.0 en notation flottante en binaire (base 2) ; par exemple $u_0 = 0.3$. Une des façons possibles est d'appliquer l'algorithme Algo1 (cf. question (d)) utilisant la relation de récurrence $u_{n+1} = 2\,u_n$ ramenée *modulo* 1. Par *modulo* 1, on veut dire que si u_n dépasse 1, on enlève 1 de façon que u_n soit toujours ramené dans l'intervalle $[0.0\ ,\ 1.0[$. On obtient ainsi, pour $u_0 = 0.3$, le schéma itératif suivant : $\underline{\mathbf{0}}.3 \rightarrow \underline{\mathbf{0}}.6 \rightarrow 0.2$ (modulo 1, c'est : $1.2 - \underline{\mathbf{1}}$) $\rightarrow \underline{\mathbf{0}}.4 \rightarrow \underline{\mathbf{0}}.8 \rightarrow 0.6$ $(1.6 - \underline{\mathbf{1}}) \rightarrow 0.2$ $(1.2 - \underline{\mathbf{1}})$ $\rightarrow \underline{\mathbf{0}}.4 \rightarrow \underline{\mathbf{0}}.8 \rightarrow 0.6(1.6 - \underline{\mathbf{1}})\ldots$ avec la répétition de la même séquence indéfiniment et on obtient donc $0.3_{10} = 0.0\,1001\,1001\,1001 = 0.0\,\overline{1001}$.

   (a) Donnez la conversion binaire de $u_0 = 0.6$ en utilisant cette méthode et comme notation une barre au-dessus d'un éventuel motif de plusieurs bits qui se répéteraient à l'infini. Trouvez aussi pour une notation flottante simple précision IEEE, l'erreur d'affectation de ce flottant et une borne supérieure $\epsilon\, u$ de l'erreur pour tout nombre flottant du type $0.xxxxx\ldots$ (*i.e*, inférieur à 1).
   <8 pts>

   (b) Soit le programme C sur ordinateur utilisant des flottants et réalisant le schéma itératif **Algo1** $\triangleright$ $u_{n+1} = 2\,u_n = f(u_n)$ ramené *modulo* 1.
   Utilisons un modèle mathématique pour expliquer où pourrait provenir le problème numérique de l'algorithme Algo1 ci-dessus. En utilisant la propagation des erreurs générées par la fonction $f(x)$ et le fait que u_0 est entaché d'une imprécision ou d'une erreur initiale de ϵ_u, montrez qu'au-delà du 20 ou 25-ième bit, ou d'un nombre assez grand de bits après la virgule, on aura inévitablement des erreurs numériques très importantes.
   <6 pts>

   (c) Essayons maintenant de comprendre informatiquement ce qui se passe lorsqu'on fait sur ordinateur le schéma itératif de **Algo1** $\triangleright$ $u_{n+1} = 2\,u_n = f(u_n)$ ramené *modulo* 1 en notation flottante **en partant de** $u_0 = 0.6$.
   À partir de la valeur binaire de 0.6 représentée en flottant (simple précision) expliquez ce que fait concrètement l'ordinateur sur u_n quand il exécute l'instruction en C codant l'opération $u_{n+1} = 2\,u_n$ (au niveau de l'UAL). En déduire ce qui va se passer pour u_k, représentée en notation flottante (avec 24 bits pour la mantisse) avec $k > 24$.
   <5 pts>

   (d) Soit donc le programme simple en langage C codant **Algo1** (cf. question : 1.)

```
int i ; float u=0.6 ;
printf("0 :%1.3f",i,u) ;
For(i=1 ; i<30 ; i++)
.   {   u=2.0*u ;
.       if (u>=1)   u=u-1 ;
.       printf("%d :%1.3f",i,u) ;   }        ◁ Algo1
```

   En prenant en considération ce qui a été dit/vue et démontré dans les questions précédentes. Corrigez ce code pour que l'Algo1 soit codé correctement afin qu'il fasse ce qu'on lui demande de faire.
   Rq : Essayer de réfléchir à une procédure (*i.e.*, une suite d'instructions) qui, ajouter à certains endroits de ce code, permettrait de conserver la même précision (pour u) à chaque itération de l'algorithme).
   <6 pts>

Réponse

$\boxed{\textbf{1.(a)}}$

En utilisant le schéma itératif très simple de l'ALGO1 $\triangleright$ $u_{n+1}=2\,u_n=f(u_n)$ ramené *modulo* 1, ou tout simplement en s'apercevant que 0.6_{10} correspond à $0.3_{10}\times 2_{10}$ (avec $0.3_{10}=0.0\,\overline{1001}_2$, donnée par l'énoncé), on doit simplement décaler de 1 bit vers la gauche pour trouver (sans calcul) la conversion binaire de 0.6_{10} et on trouve ainsi $0.6_{10}=0.\overline{1001}_2$.

Pour une notation IEEE flottante simple précision, l'erreur d'affectation du nombre $0.6_{10} = 0.\overline{1001}_2$ est 0. ...24 bits à zéro puis... $\overline{1001}10011001$ etc. et son erreur d'affectation est donc :

$$
\begin{aligned}
\epsilon &= (2^{-25}+2^{-28}) + (2^{-29}+2^{-32}) + (2^{-33}+2^{-36}) + \dots \\
&= (2^{-25}+2^{-28})\,(1+2^{-4}+2^{-8}+\dots) = (2^{-25}+2^{-28})\,(\frac{1-2^{-\infty}}{1-2^{-4}}) \\
&= \frac{9}{2^{28}}\cdot\frac{16}{15} = \frac{3}{5\times 2^{24}} \approx 3.58\times 10^{-8}
\end{aligned}
$$

Et une borne supérieure ϵ_u de l'erreur pour tout nombre flottant du type $0.xxxxx\dots$ (*i.e*, inférieur à 1) est 0. ...24 bits à zéro puis...11111111...etc., soit une infinité de 1 après la 25 i-ème position (après la virgule) soit une borne supérieure d'erreur d'affectation $\epsilon = 2^{-24}$ (cf. cours).
<8 pts>

$\boxed{\textbf{1.(b)}}$

Avec la relation itérative $u_{n+1}=f(u_n)=2\,u_n$, une erreur ou imprécision sur u_n de ϵ_n est amplifiée par 2. En partant donc de u_0 avec une imprécision de $\epsilon_{u_0}=\epsilon_0$, celle-ci sera donc amplifiée de 2 sur u_1, puis de $2\times 2=2^2$ sur u_2, puis cette erreur sur u_2 sera amplifié de $2\times 2\times 2=2^3$ sur u_3, ainsi de suite...

Au final, en partant de u_0 avec une imprécision de ϵ_0, cette erreur sera amplifiée, par 2^k pour estimer, par l'algorithme ALGO1, u_k, *i.e.* le k-ième bit après la virgule du nombre que l'on désire convertir.

Par exemple, pour $k=25$, ce calcul aura une borne supérieure de l'erreur de $2^{25}\,\epsilon=33\,554\,432\,\epsilon$. *i.e.*, que ϵ sera amplifié de ≈ 33.5 millions et une amplification de plus d'un milliard a lieu dès le $30(=k)$ i-ème bit après la virgule.

Cette relation itérative n'est donc pas, en flottant, stable numériquement et cette méthode numérique pour estimer numériquement la conversion binaire d'un nombre flottant en base deux ne doit pas être implémentée directement par l'opération ou par ces deux simples lignes de code $u_{n+1}=2\,u_n=f(u_n)$ ramené *modulo* 1.
<6 pts>

$\boxed{\textbf{1.(c)}}$

À partir de $0.6_{10} = 0.\overline{1001}_2$, l'instruction en C : $u_{n+1} = 2\,u_n$ (multiplication par deux d'un flottant) va décaler $u_0 = 0.\overline{1001}_2 = 1001\,1001\,1001\,1001\,1001\,1001$ d'un cran vers la droite ce qui va donner $u_1 = 1.001\,1001\,1001\,1001\,1001\,1001\,0_2$ puis l'opération RAMENÉE EN MODULO 1 fera l'opération $\triangleright$
$u_1=0.001\,1001\,1001\,1001\,1001\,1001\,0_2$. Il n'y a plus que 23 bits informatif derrière la virgule suivie d'un zéro.

- Pour $k=10$, la multiplication par 2^{10} va décaler u_0 de 10 crans vers la droite ce qui va donner $u_1=0.1\,1001\,1001\,1001\,\underbrace{0000000000}_{\text{faux}}{}_2$ et il n'y a plus que 13 bits informatifs derrière la virgule suivie de 10 zéros.
- Pour $k=20$, la multiplication par 2^{20} va décaler u_0 de 20 crans vers la droite ce qui va donner $u_{20}=0.001\,\underbrace{00000000000000000000}_{\text{faux}}{}_2$ et il n'y a plus que 3 bits informatifs derrière la virgule suivie de 20 zéros.
- Pour $k>24$,

la multiplication par 2^{24} va décaler u_0 de 24 crans vers la droite donnant inévitablement $u_{>24} = 0.$ et $u_{>24} = 0.\underbrace{000000000000000000000000}_{\text{faux}}$

Bref, à chaque itération de l'algorithme, *i.e.*, à chaque décalage ou multiplication par 2 (dans le calcul de $u_{n+1} = f(u_n) = 2\,u_n$, on perdra de la précision (précisément, 1 chiffre après la virgule) dans ce que l'on veut obtenir et cela conduira inévitablement (après la 24 i-ème multiplication en IEEE simple précision et après la 52 i-ème multiplication en IEEE double précision, etc.) après un certain nombre d'itérations, à un désastre numérique.

<5 pts>

$\boxed{\textbf{1.(d)}}$

Les instructions que l'on doit ajouter pour que le programme marche sont indiqués en gras

int i ; float u=0.6 ;
$\boxed{\textbf{int PREC=1000000 ;}}$
printf("0 :%1.3f",i,u) ;
For(i=1 ; i<30 ; i++)
. { u=2.0*u ;
. if (u>=1) u=u-1 ;
. $\boxed{\textbf{u=(int)(u*PREC) ;}}$
. $\boxed{\textbf{u=(float)(u/PREC) ;}}$
. *printf("%d :%1.3f",i,u) ; }*

Une possibilité est de récupérer, après l'opération $u_{n+1} = 2\,u_n$ ramenée *modulo* 1, la valeur flottante u courante et la multiplié par PREC puis convertir ce résultat intermédiaire en un entier (instruction *u=(int)(u*PREC) ;* (*i.e., en le castant*). Pour $u = 0.2$ et PREC= 1000, on obtient $u = 2000$ qui sera un entier représenté parfaitement sans erreur d'affectation puis ensuite de diviser ce résultat par PREC(=1000) pour le ramener en flottant avec une précision de trois chiffres après la virgule dans ce cas.

Dans le cas où PREC= 1000000, on a une précision fixe de 6 chiffres après la virgule. On peut monter jusqu'à 7 chiffres de précision avec PREC(= $1E7$).

Ainsi, à chaque itération, on aura u (*i.e.*, les différents termes de la suite) , qui seront calculés avec la même précision, en terme de nombre de chiffres après la virgule (avec PREC= 1000000, avec 6 chiffres après la virgule).

Dans le cas non-corrigé le programme donnera (faussement) :
0 : 0.600 1 : 0.200 2 : 0.400 3 : 0.800 4 : 0.600 5 : 0.200 6 : 0.400 7 : 0.800 8 : 0.600 9 : 0.200 10 : 0.400 11 : 0.800 12 : 0.600 13 : 0.200 14 : 0.400 **15** : 0.801 **16** : 0.602 **17** : 0.203 **18** : 0.406 **19** : 0.812 **20** : 0.625 **21** : 0.250 **22** : 0.500 **23** : 0.000 **24** : 0.000 **25** : 0.000 **26** : 0.000 **27** : 0.000 **28** : 0.000 **29** : 0.000 **30** : 0.000...... (erreur visible à partir de la 15 i-ème itération).

Dans le cas corrigé le programme donnera :
0 : 0.600 1 : 0.200 2 : 0.400 3 : 0.800 4 : 0.600 5 : 0.200 6 : 0.400 7 : 0.800 8 : 0.600 9 : 0.200 10 : 0.400 11 : 0.800 12 : 0.600 13 : 0.200 14 : 0.400 115 : 0.800 16 : 0.600 17 : 0.200 18 : 0.400 19 : 0.800 20 : 0.600 21 : 0.200 22 : 0.400 23 : 0.800 24 : 0.600 25 : 0.200 26 : 0.400 27 : 0.800 28 : 0.600 29 : 0.200 30 : 0.400......

<6 pts>

Nota -1- : Par contre, si l'on fait le même programme sur une calculette, il ne se produit pas d'erreurs. Pourquoi ? Parce qu'une calculette n'est pas programmée de la même façon qu'un ordinateur. La conversion en binaire se fait chiffre par chiffre sur quatre bits. Ainsi avec 6 qui s'écrit 0110 sur 4 bits, le nombre 0.6 devient 0.0110 et les calculs sont ensuite faits à partir de ce nombre qui lui est exact.

Nota -2- : Certains étudiants ont bien compris qu'il y avait ici une perte de précision (due a cette multiplication par deux [ou décalage de bit fait par l'UAL] d'un nombre à virgule qui devrait se représenter avec un nombre infini de bits et qui ne pouvait utiliser que les 24 bits de la mantisse d'une représentation flottante simple précision) et m'ont dit que puisque $0.6_{10} = 0.\overline{1001}_2$, il suffisait de remplacer tous les 4 décalages, les 4 zéros dus a ce décalage par 1001. Oui, Ok dans ce cas particulier ou $x = 0.6_{10}$, mais comment aurait-on fait si on avait eu une valeur binaire sans motif de répétition de bits ?

Nota -3- : La transposition en *double* ne fait que retarder le problème et ne le résoud pas.

II. Erreur/Amplification d'Erreur en Arithmétique Flottant (34 pts)

1. On doit calculer sur ordinateur, en représentation flottante simple précision, l'expression suivante :

$$x = \lim_{n \to \infty} (1 + 1/n)^n$$

Montrer tout d'abord que cette expression tend analytiquement vers e et indiquer ce qui va se passer lorsque cette expression sera évaluée en représentation flottante simple précision pour différente valeurs croissantes de n. En utilisant votre raisonnement précédent, tracer brièvement et grossièrement la courbe qui serait le résultat de l'évaluation de cette expression en représentation flottante simple précision pour différentes valeurs croissantes de n [en abscisse] avec une échelle logarithmique).
<10 pts>

2. Dans le cadre de la résolution d'une équation du second degré du type $ax^2 + bx + c = 0$, on utilise habituellement la résolution analytique en utilisant la célèbre formule :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

(a) Pour de grandes valeurs de b, cette formule peut conduire à des erreurs numériques dramatiques. Expliquer quel est ce problème et proposer une expression mathématiquement équivalente qui permettrait d'éviter ce problème.
<6 pts>

(b) Une autre possibilité est de reprendre l'expression (1) qui est sensible aux erreurs numériques et de lui appliquer un développement limité sur $(1-\epsilon)^{1/2}$. Utiliser cette stratégie pour estimer la solution de la racine comprise entre [0 1] à 10^{-20} près de l'équation $ax^2 + bx + c = 0$ avec $a = c = 1.0$ et $b = -10^7$.
<8 pts>

(c) Réarranger l'équation $f(x) = ax^2 + bx + c = 0$ avec $a = c = 1.0$ et $b = -10^7$ sous la forme d'un point fixe itératif qui convergerait vers la solution comprise dans l'intervalle [0 1] en partant de $x_0 = 0$ et faire trois itérations. Au bout de chacune de ces trois itérations, donner la précision obtenue de cette estimation.
<10 pts>

Réponse

$\boxed{1}$

Pour $n = 30$, on obtient $x \approx 2.674318776$, soit e avec 1 cse après la virgule. Avec $n = 300$, on obtient $x \approx 2.713765155$, soit e avec 2 cse après la virgule. Avec $n = 100000$, on obtient $x \approx 2.7181268237$, soit e avec 3 cse après la virgule sur ma calculatrice CASIO *fx-7400G Plus*.

Pour montrer que x converge vers $e = 2.718282$, il existe plusieurs solutions. On peut par exemple poser $y = 1/n$ et $z = \ln(x)$. Par ces deux changements de variables, $\lim_{n\to\infty}(1+1/n)^n = \lim_{y\to 0} \exp\{y^{-1} \cdot \ln(1+y)\}$

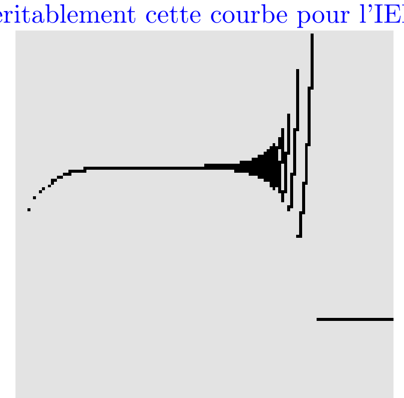
et par un simple développement limité de $\ln(1+y)$ ou par la règle de l'Hôpital sur cette expression, on obtient $\lim_{y\to 0}\exp\{1/(1+y)\} = \lim_{n\to\infty}(1+1/n)^n = e$.

Il nous faut donc un n suffisamment élevé pour respecter la limite et obtenir une bonne estimation de $x=e$ mais il ne faut pas que n soit trop grand sinon on a le problème de l'addition de deux nombres d'ordre de grandeur différent (erreur de décalage) que l'on retrouve aussi lorsque la fonction $(1+\epsilon)^n$ sera approchée (par l'ordinateur) par un développement limité.

$$\begin{aligned} Dx = (1+(1/n))^n \approx \quad 1 \quad &+ \quad n\cdot(1/n)^1 \;+\; [n(n-1)/2!]\cdot(1/n)^2 \\ &+ \quad [n(n-1)(n-2)/3!]\cdot(1/n)^3 \;+\; [n(n-1)(n-2)(n-3)/4!]\cdot(1/n)^4 \\ &+ \quad o(\epsilon^4) \end{aligned}$$

Bref, on peut en déduire le comportement de l'estimation de cette expression en représentation flottante simple précision pour différentes valeurs croissantes de n ; de $n=1$ à une valeur de n_{opt} plus grande, on va s'approcher de e (presque linéairement) puis ensuite, l'erreur due à l'addition de deux nombres d'ordre de grandeur différent (erreur de décalage) sera la plus forte, et pour des valeurs de n croissantes à partir de n_{opt}, on perdra de plus en plus de précision ; le terme à la puissance p sera oublié dans Dx (car trop petit) par les autres termes, puis ensuite, n augmentant, le terme de puissance $[p-1]$ sera oublié (car devenu à son tour trop petit dans Dx), puis le terme de puissance $[p-2]$, etc. et pour des valeurs de n les plus élevées, disons $n=n_l$, il ne restera plus que le premier terme de la série, à savoir 1 et x sera faussement estimé par $x=1$ puis stagnation de la courbe. De $x\approx e$ (pour $n=n_{opt}$) à $x=1$ (pour $n=n_l$), pour des valeurs croissantes de $n>n_{opt}$; la courbe aura donc des erreurs imprévisibles et chaotiques.
<10 pts>

Nota : Voici à quoi ressemble véritablement cette courbe pour l'IEEE simple précision :



On s'approche de e (presque linéairement), puis chaos ou bruit ; due à l'addition de deux nombres d'ordre de grandeur différent (et approximation par le developpement limité), puis stagnation à 1 (le plus petit terme est totalement oublié) :

2.(a)

• Pour de très très fortes valeurs de b, on se retrouve avec le problème d'OVERFLOW.

• Pour de fortes valeurs de b, on se retrouve avec le problème de la soustraction de deux nombres approximés quasi égaux pour l'expression $[-b+\sqrt{b^2-4ac}]$ qui amplifie considérablement les erreurs numériques qui entrainent l'annulation de cse (chiffres significatifs exacts). On peut néanmoins réécrire cette expression pour qu'elle soit mathématiquement équivalente et ne fasse plus intervenir ce problème en multipliant le numérateur et le dénominateur par le conjugué du numérateur de la façon suivante (cf. Intra H09) :

$$\frac{-b+\sqrt{b^2-4ac}}{2a} = \frac{\left(-b+\sqrt{b^2-4ac}\right)}{2a}\cdot\frac{\left(b+\sqrt{b^2-4ac}\right)}{\left(b+\sqrt{b^2-4ac}\right)} = \frac{-b^2+b^2-4ac}{2a(b+\sqrt{b^2-4ac})} = \frac{-2c}{b+\sqrt{b^2-4ac}}$$
<6 pts>

Nota : Sur ma calculatrice *CASIO fx-7400G PLUS :*, pour $a=1$, $c=1$ et $b=10^6$, j'obtiens faussement avec la formule traditionnelle, la solution $x_1=0$ et justement $x_1=-10^{-6}$ avec la formule équivalente ci dessus.
Mais aussi on se souvient que la calculette n'est pas programmée de la même façon qu'un ordinateur (voir remarque de la question I.(d)). Avec un ordinateur, cela ne marche plus en flottant à partir de $b=10000$. J'obtiens faussement avec la formule traditionnelle, la solution $x_1=0$ et justement $x_1=0.0001$ avec la formule équivalente ci-dessus.

2.(b)

En utilisant le fait que $(1-\epsilon)^{1/2} = 1-(\epsilon/2)-(\epsilon^2/8)+\ldots$ et puisque $b<0$, la racine la plus proche de zéro est donnée par la formule $[(-b-\sqrt{b^2-4ac})/2a]$, on a :

$$\frac{-b-\sqrt{b^2-4ac}}{2a} = \frac{-b-|b|\sqrt{1-4ac/b^2}}{2a} = \frac{-b}{2a} - \frac{|b|}{2a}\left(1-\frac{1}{2}\Big(\frac{4ac}{b^2}\Big)-\frac{1}{8}\Big(\frac{4ac}{b^2}\Big)^2+O\big(.\big)^3\right) = \frac{c}{|b|} - \frac{a\,c^2}{b^3}+O\big(.\big)^3$$

$$\text{Avec : } \quad a=c=1.0 \quad \text{et} \quad b=-10^7 : \quad \text{solution} = \frac{-b-\sqrt{b^2-4ac}}{2a} = \underbrace{10^{-7}+10^{-21}}_{\text{solution de la racine à } 10^{-20} \text{ près}} \quad \left(+10^{-49}+10^{-105}\ldots\right)$$

<8 pts>

2.(c)

f est continue et décroissante sur $J=[0.0,1.0]$ ($f'(x)=-10^7x<0$ pour $x\in J$). De plus, on a $f(0)=1$ et $f(1)<0$, il existe donc une racine r unique dans J. De plus,

$$\begin{aligned} f(x) &= x^2-10^7x+1=0 \\ x &= \frac{x^2+1}{10^7} = g(x) \end{aligned}$$

et $|g'(x)| = 2\,10^{-7}x<1 \quad \forall x\in J$ La fonction $g(x)$ est donc contractante sur J et la convergence est assurée puisque le premier élément $r_0=0$ de la suite $r_{n+1}=g(r_n)$ est aussi dans J. En partant de $r_0=0$, on a, $r_n=g(r_{n-1})$ et ;

$$\begin{aligned} r_1 &= 10^{-7} \\ r_2 &= 10^{-7}+10^{-21} \\ r_3 &= 10^{-7}+10^{-21}+10^{-49} \end{aligned}$$

On obtient pour chaque itération, autant de terme pour approximer cette racine que le numéro de l'itération et on obtient les mêmes termes de la question précédente. Pour la première itération, on approche la racine à 10^{-20} près, pour la deuxième itération, on approche la racine à 10^{-48} près, pour la troisième itération, on approche la racine à 10^{-104} près.
<10 pts>

Nota : Au lieu d'un point fixe, on pourrait utiliser la méthode itérative de Newton :

$$r_{n+1} = r_n - \frac{f(r_n)}{f'(r_n)} = r_n - \frac{r_n^2-10^7r_n+1}{2r_n-10^7}$$

Et on aurait obtenu la meme chose : $r_1=10^{-7}$, $r_2=10^{-7}+10^{-21}$, $r_3=10^{-7}+10^{-21}+10^{-49}$, etc.

III. Recherche des racines d'une équation (59 pts)

On se propose de trouver numériquement dans R une valeur approchée de la racine de la fonction :

$$f(x) = 3^x - 6x = 0 \tag{2}$$

1. Méthode du point fixe
   (a) Montrer qu'il existe une racine unique r pour cette Eq. (2) dans l'intervalle $J = [0.0\ ,\ 1.0]$ et donner une forme $x = g_1(x)$, mathématiquement équivalente de $f(x)=0$ (Eq. (2)), pour laquelle la suite itérative $r_{n+1}=g_1(r_n)$ converge vers cette racine lorsque, le premier élément de cette suite est $r_0 = 0.5$ (milieu de l'intervalle J) et donner les 5 premières estimées $r_1, \dots, r_5$ en partant de r_0.
   <12 pts>
   (b) Montrer qu'il existe une racine unique r pour cette Eq. (2) dans l'intervalle $K = [2.0\ ,\ 3.0]$ et donner une forme $x = g_2(x)$, mathématiquement équivalente de $f(x)=0$ (Eq. (2)), pour laquelle la suite itérative $r_{n+1}=g_2(r_n)$ converge vers cette racine lorsque, le premier élément de cette suite est $r_0 = 2.5$ (milieu de l'intervalle K) et donner les 5 premières estimées $r_1, \dots, r_5$ en partant de r_0.
   <12 pts>
   (c) Essayer de trouver une technique qui permettrait de prédire laquelle des deux suites itératives (question (a) ou (b)) est celle associée à la convergence la plus rapide (et sans itérer ces suites).
   <5 pts>
   (d) Donner une forme $x = g_3(x)$, mathématiquement équivalente de $f(x)=0$ (Eq. (2)), pour laquelle la suite itérative $r_{n+1}=g_3(r_n)$ ne converge vers aucune de ces deux racines (question (a) ou (b)) lorsque, le premier élément de la suite est r_0 fixé au milieu de l'intervalle considéré et expliquer mathématiquement pourquoi.
   <7 pts>
2. Méthode de Newton
   (a) Donner la relation $r_{n+1} = g_{\text{newt.}}(r_n)$ intervenant dans la méthode itérative de Newton pour la résolution itérative de la racine r de l'équation $f(x) = 0$.
   <5 pts>
   (b) Montrer théoriquement qu'avec $r_0 = 0.5$ et $r_0 = 2.5$, la relation itérative de Newton converge vers respectivement la première et deuxième racine.
   <5 pts>
   (c) Calculer les 3 premières estimées $r_1, \dots, r_3$ en partant de $r_0 = 0.5$ puis en partant de $r_0 = 2.5$.
   <8 pts>
   (d) Obtenez une valeur approchée de la racine de $f(x)$ comprise dans J en utilisant l'expansion en série de Taylor de $f(x)$ jusqu'au terme en x^2 (en supposant cette racine proche de 0). Utiliser ensuite cette valeur pour estimer r_0 et calculer les 2 premières estimées r_1, r_2 de la suite itérative $r_{n+1} = g_{\text{newt.}}(r_n)$ en partant de ce r_0.
   <5 pts>

Réponse

$\boxed{\textbf{1.(a)}}$

L'étude des variations de la fonction f sur $J = [0.0\ , 1.0]$ montre que la fonction est continue et décroissante (car $f'(x) = \ln(3)\cdot 3^x - 6 < 0$) sur J, donc monotone. De plus, on a $f(0)=1.0$ et $f(1)=-3$ donc $f(0)f(1)<0$ et, puisque la fonction f est monotone sur J, il existe donc une racine r unique dans cet intervalle J.

<4 pts>

De plus $f(x) = 0$ est équivalent à l'équation $x = 3^x/6 = g_1(x)$ avec :

$$|g_1'(x)| = \left|\frac{\ln(3)}{6}\, 3^x\right| < 1 \quad \forall\, x \in J = [0.0\, , 1.0]$$

Plus précisément, $g_1'(x) = \frac{\ln(3)}{6}\, 3^x$ est croissante sur $[0.0\, , 1.0]$ et sa valeur maximale est donc $g_1'(1.0) = 3\ \ln(3)/6 \approx 0.549307 < 1$.

La fonction $g_1(x)$ est donc contractante sur J et la convergence est assurée puisque le premier élément $r_0 = 0.5$ de la suite $r_{n+1} = g_1(r_n)$ est aussi dans J et c'est bien la seule façon de mettre $f(x) = 0$ sous la forme $x = g_1(x)$ qui soit contractante sur J (cf. question suivante).

<4 pts>

En partant de $r_0 = 0.5$, on a, $r_n = g_1(r_{n-1})$ (avec $g_1(x) = \frac{\ln(3)}{6}\, 3^x$), et ;

$$\begin{aligned} r_1 &= 0.288675134 \\ r_2 &= 0.228866270 \\ r_3 &= 0.214311616 \\ r_4 &= 0.210912042 \\ r_5 &= 0.210125794 \end{aligned}$$

qui va converger vers la valeur $r = 0.209889988$ (dès la 15-ième itération).

<4 pts>

1.(b)

$f'(x) = \ln(3) \cdot 3^x - 6 > 0$ sur K, donc uniquement croissante et donc monotone. De plus, on a $f(2) = -3.0$ et $f(3) = 9$ donc $f(2)f(3) < 0$ et, puisque la fonction f est monotone sur K, il existe donc une racine r unique dans cet intervalle K.

<4 pts>

De plus $f(x) = 0$ est équivalent à l'équation $x = \ln(6x)/\ln(3) = g_2(x)$ avec :

$$|g_2'(x)| = \left| \frac{1}{6x \cdot \ln(3)} \right| < 1 \quad \forall\, x \in J = [2.0\,,\, 3.0]$$

Plus précisément, $g_2'(x) = \frac{1}{6x \cdot \ln(3)}$ est décroissante sur $[2.0\,,\, 3.0]$ et sa valeur maximale est donc $g_2'(2.0) = 1/(6 \times 2 \times \ln(3) < 1$.

La fonction $g_2(x)$ est donc contractante sur K et la convergence est assurée puisque le premier élément $r_0 = 2.5$ de la suite $r_{n+1} = g_2(r_n)$ est aussi dans K et c'est bien la seule façon de mettre $f(x) = 0$ sous la forme $x = g_2(x)$ qui soit contractante sur K (et cette forme n'est pas contractante sur J, cf. question précédente).

<4 pts>

En partant de $r_0 = 2.5$, on a, $r_n = g_2(r_{n-1})$ (avec $g_2(x) = \ln(6x)/\ln(3)$), et ;

$$\begin{aligned} r_1 &= 2.464973521 \\ r_2 &= 2.452130349 \\ r_3 &= 2.447375361 \\ r_4 &= 2.445608579 \\ r_5 &= 2.444951232 \end{aligned}$$

qui va converger vers la valeur $r = 2.444561393$ (dès la 19-ième itération).

<4 pts>

Nota

$f(x) = 0$ est équivalent aussi à l'équation $x = (\ln(6) + \ln(x))/3 = g_2(x)$ qui converge aussi vers la deuxième racine.

1.(c)

On pourrait, par exemple, essayer de trouver une borne supérieure du nombre d'itérations nécessaires de la méthode du point fixe pour arriver à la racine souhaitée avec la précision voulue comme dans l'examen Intra H20 III.(d) dans lequel on a :

$$|r_n - r| \leq \left(g'_{\max}(\zeta)\right) |r_{n-1} - r| \ \leq \ \left(g'_{\max}(\zeta)\right)^2 |r_{n-2} - r| \ \leq \ldots \ \leq \ \left(g'_{\max}(\zeta)\right)^n (r_0 - r)$$

Avec r ; la racine, r_n ; la n-ième itération de la suite et $g'_{\max}(\zeta)$; la valeur maximale pour $\zeta \in$ l'intervalle considéré permet de deduire une borne supérieure du nombre d'itérations nécessaires pour obtenir une valeur approchée de la racine souhaitée avec une précision voulue. En fait, plus cette valeur s'approche de zéro, plus on converge rapidement.

Pour la suite de la question (a), on a $g'_1(\zeta = 1) \approx 0.55$ et pour la suite de la question (b), on a $g'_1(\zeta = 2) \approx 0.08$ ce qui montre que la suite de la question (b) à une convergence plus rapide. On peut être plus précis en calculant cette valeur pour la racine. On obtient $g'_1(\zeta = 0.2101) \approx 0.23$ et $g'_2(\zeta = 2.4450) \approx 0.19$ qui nous indique aussi que la suite de la question (b) à une convergence plus rapide. Les deux suites ont une convergence linéaire (car $g'(r) \neq 0$) et cette valeur joue le role de constante asymptotique.

<5 pts>

1.(d)

$f(x) = 0$ est équivalent aussi à l'équation $x = 3^x - 5x = g_3(x)$ qui ne converge pas, car :

$$|g'_3(x)| = \Big|\ln(3) \cdot 3^x - 5| > 1 \quad \forall x \in J = [0.0\,,\,1.0] \quad \text{et aussi} \quad \forall x \in K = [2.0\,,\,3.0]$$

<7 pts>

Nota -1-

$f(x) = 0$ est équivalent aussi aux équations :

- $x = x - 1 + \exp\left[3^x - 6x\right] \ \ = g_3(x)$
- $x = x + \ln[3^x/6x] \ \ = g_3(x)$
- $x = 3^x - 5x \ \ = g_3(x)$
- $x = 2\,.\,3^{1-x} \ \ = g_3(x)$

qui ne convergent pas pour les deux racines et pour laquelle on montre facilement aussi que $|g'_3(x)| > 1 \ \forall x \in K = [0.0\,,\,3.0]$.

Nota -2-

$f(x) = 0$ est équivalent aussi à l'équation $x = (1/6) \cdot 3^x \ \ = g_4(x)$ qui converge vers la premiere racine quand $r_0 = 0.5$ et diverge quand $r_0 = 2.5$ pour la deuxième racine.

2.(a)

La fonction $f(x)$ est dérivable sur $\mathbb{R}$ et la méthode itérative de Newton permet d'écrire :

$$r_{n+1} = r_n - \frac{f(r_n)}{f'(r_n)} = r_n - \frac{3^{r_n} - 6r_n}{(\ln 3) \cdot 3^{r_n} - 6}$$

<5 pts>

2.(b)

Dans ce cas, le plus simple est de montrer que la fonction $f(x)$, sur l'intervalle $J = [0, 1]$ et $K = [2, 3]$, intervalles dans lequel on prendra r_0 et dans lequel une racine unique s'y trouve (cf. questions précédente) ne présente pas d'*extrema*, *i.e.*, de valeurs pour laquelle, $f'(x)$ s'annule.

Dans notre cas, $f'(x) = (\ln 3) \cdot 3^x - 6$ ne s'annule pas sur $J = [0, 1]$ et $K = [2, 3]$; Elle ne s'annule seulement sur $J = [1, 2]$!. Donc aucun problème et convergence assurée.

<5 pts>

$\boxed{\textbf{2.(c)}}$

En partant de $r_0 = 0.5$, on a, $r_n = g_{\text{newt.}}(r_{n-1})$ et :

$$\begin{aligned} r_1 &= 0.190528805 \\ r_2 &= 0.209829528 \\ r_3 &= 0.209889988 \end{aligned}$$

qui converge assez vite vers la première racine $r = 0.209889988$ optimale en 3 itérations seulement !
<4 pts>
En partant de $r_0 = 2.5$, on a, $r_n = g_{\text{newt.}}(r_{n-1})$ et :

$$\begin{aligned} r_1 &= 2.447108153 \\ r_2 &= 2.444567055 \\ r_3 &= 2.444561393 \end{aligned}$$

qui converge assez vite vers la seconde racine $r = 2.444561393$ optimale en 3 itérations seulement !
<4 pts>

$\boxed{\textbf{2.(d)}}$

$$\begin{aligned} f(x) = 3^x - 6x &= 0 \\ 1 + (\ln 3)\, x + [(\ln 3)^2/2]\, x^2 - 6x + O(x^2) &= 0 \\ \Big((\ln 3)^2/2\Big) \cdot x^2 + [(\ln 3) - 6] \cdot x + 1 \ + O(x^2) &= 0 \\ 0.60347448x^2 - 4.901387711 + 1 &= 0 \end{aligned}$$

$$\begin{aligned} \Leftrightarrow \quad x_{1,2} &= \frac{4.901387711 \pm \sqrt{21.60970357}}{2} \\ &= 0.126381937 \quad \text{ou} \quad 4.775005774 \end{aligned}$$

Donc, en partant de $r_0 = 0.126381937$, on a, $r_n = g_{\text{newt.}}(r_{n-1})$ et :

$$\begin{aligned} r_1 &= 0.20883748 \\ r_2 &= 0.209889806 \end{aligned}$$

qui converge vite vers la première racine $r = 0.209889988$ en 3 itérations.
<5 pts>

Nota
J'ai compté juste aussisi on utilisait l'expansion en série de Taylor de $f(x)$ jusqu'au terme en x, on obtient $1 + (\ln 3)x - 6x{=}0$, et $x = 0.204023851$ et en partant de $r_0 = 204023851$, on a, $r_n = g_{\text{newt.}}(r_{n-1})$ et :

$$\begin{aligned} r_1 &= 0.209884313 \\ r_2 &= 0.209889988 \end{aligned}$$

qui converge vite vers la première racine $r = 0.209889988$ en 2 itérations aussi.