Projet 1
Méthodes de calcul numérique / Limites de la machine

Groupe 3 - Equipe 6
Responsable : rjeribi
Secrétaire : mmarcos002
Codeurs : clarragueta, ibechoual

Résumé : Le but de ce projet consiste à évaluer les problèmes qui peuvent apparaître lors de l'utilisation d'opérations élémentaires, voire d'algorithmes plus poussés, sur des nombres flottants.

1 Présentation du projet

Le projet se déroulera en deux parties distincts, toutes deux mettant en évidence les différences entre les résultats obtenues par les opérations réalisées par la machine, par rapport aux résultats réels attendu.

- La première partie met en évidence certains exemples pour lesquelles la précision machine est insuffisante.
- La deuxième partie donne des exemples d'algorithmes utilisés pour des calculs pour lesquelles la puissance de calcul est moindre, et où il faut donc optimiser les calculs pour obtenir une précision optimale pour un coût de calcul modéré.

Motivation du projet

Le projet est une application directe du cours qui visait à nous présenter le codage des nombres et les différentes opérations en machine.
Le codage en bits permet de représenter tous types de nombres et d'opérations, mais leur précision est limitée par le nombre de bit, car une machine est finie.
Connaitre les erreurs que peuvent provoquer le codage en bit par rapport aux opérations arithmétiques réelles est très important et ce dans de nombreux cas. (On ne peut pas se permettre une grande approximation si on travaille dans le spatial par exemple.)

Avancement du projet

L'ensemble de ce projet s'est déroulé sur un intervalle de deux semaines, pour un total de 3 séances en groupe complet. Le projet est décomposé en 2 parties, et le travail a été réaliser par tandems. Nous avons commencé par nous repartir les questions de la première partie entre nous. Nous avons cependant terminer par tous travailler sur la première question pour mettre en oeuvre la représentation machine, sur laquelle nous avons rencontré des difficultés.
Après avoir terminer la partie 1, nous avons réparti différemment le travail. Cette fois, un groupe travaillait sur la partie 2, pendant que l'autre commençait à écrire le rapport.

2 Description du code

Partie 1

La première question est juste une approximation de nombres réels sur 4 décimales où 6 décimales que l'on notera p. Le principe est simple, mais pour éviter tous problèmes, je divise ma fonction en 3 en fonctions de la valeur de x. Soit $x < 1$ dans ce cas on calcul la puissance négative que l'on stocke dans une variable que l'on note $puisx$ de x, on multiplie x par $10^{p+puisx}$ puis on prend la partie entière et l'arrondi puis le multiplie par $10^{-(p+puisx)}$ pour avoir x sous représentation à p décimales. Pour les autres cas, la méthode est plus au moins la même ce qui change la valeur de la puissance de 10.

La deuxième question nous propose de mettre en oeuvres les opérations élémentaires que sont l'addition et la multiplication en prenant en compte la représentation machine mise en place à la question précédente.
Les fonctions permettant la mise en place de ces opérations utilisent donc naturellement la fonction rp décrite précédemment.

- Pour l'addition, la fonction renvoie la représentation machine de la somme des deux nombres x et y mis en paramètres avec p la précision souhaitée.
  Le résultat est donc $rp((x + y), p)$.
- Pour la multiplication, la méthode est similaire, on renvoie cette fois ci la représentation machine du produit des nombres x et y, avec la précision p.
  Le résultat est donc $rp((x * y), p)$

La troisième question nous propose de mettre en oeuvre une fonction permettant d'obtenir l'erreur relative induite par la somme en représentation machine selon deux entiers x et y, et une précision p. La formule utilisé pour calculer cette erreur nous est donné et représente, en utilisant les fonctions que nous avons déjà implémenté :

$$\delta_s(x, y, p) = \frac{|(x + y) - rp((x + y), p)|}{|(x + y)|}$$

La fonction calcul donc cette formule et nous renvoie sa valeur.

La quatrième question nous propose cette fois de mettre en oeuvre la fonction permettant d'obtenir l'erreur relative induite par la multiplication en représentation machine entre deux entiers x et y, et une précision p.
Une fois encore, la formule pour calculer cette erreur nous est donné et représente donc en utilisant nos fonctions :

$$\delta_p(x, y, p) = \frac{|(x * y) - rp((x * y), p)|}{|(x * y)|}$$

La fonction calcul donc cette formule et nous renvoie sa valeur.

La dernière question de la partie 1 est un affichage d'un tracé de courbe, on fixe x de manière à ce qu'il soit intéressant, et on prend un tableau de valeur y, on applique respectivement les fonctions multiplication machine et addition machine et on stocke toutes les valeurs dans *tab*. Puis on trace la courbe de y en fonction de *tab* à l'aide des modules matplotlib.pyplot comme le montre les figures 1 et 2.

Partie 2

Dans cette partie nous nous interresserons à l'implémentation d'algorithmes utilisés dans des conditions où de gros moyens informatiques (puissance de calcul etc..) ne sont pas à disposition pour faire des calculs, que l'on retrouve notemment dans une calculatrice.

Logarithme nepérien de 2

Nous avons implémenté le calcul de $\mathbf{ln(2)}$ à l'aide de la série $\ln(2) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}$. Pour implémenter la fonction nous avons utilisé la fonction implémentée dans la partie 1 **rp()** donc à l'entrée de la fonction nous avons mis un entier p qui représente le nombre de chiffres significatifs.
La figure 4 et la figure 5 montre différentes valeures de l'erreur ln(2) en fonction de p et nous constatons que plus la valeur de p est grande plus l'erreur est grande.

Nous avons choisit d'évaluer l'erreur relative sur le calcul de ln(2) à l'aide de la méthode indiquée par le sujet en changeant l'ordre de calcul des termes de la somme. Ainsi, nous pourrons évaluer quelle méthode est la plus précise. (Voir figure 4 et 5).

Il est ressorti de cette expérience que l'erreur relative est nettement moindre lorsque qu'on somme les termes dans l'ordre naturel, c'est à dire 1 à n. cela peut être due par le fait que la famille $\frac{(-1)^{n+1}}{n}$ pour n appartenant à N n'est pas sommable.

Algorithmes CORDIC

D'après les ressources fournies, une calculatrice classique utilise la représentation sur 8 octets des nombres flottants, qui se décomposent en un bit de signe, une matisse sur 52 bits et un exposant. Cette représentation en virgule flottante est avantageuse de par le large spectre de nombre qu'elle peut représenter (environ de $-3.402823 * 10^(38)$ à $3.402823 * 10^(38)$, et du fait qu'elle est très bien gerée par les calculatrices pour les opérations mathématiques élémentaires. Cependant, cette représentation manque de précision sur les nombres ce qui peut entrainer des erreurs de calcul, de plus, elle engendre des erreurs d'arrondis. C'est ici qu'entrent en jeu les algorithmes CORDIC.
Ces algorithmes sont utiles dans le calcul des fonctions trigonométriques en certaines valeurs, cette technique de calcul est implémentée dans les calculatrices en raison des avantages qu'elle présente : elle est avantageuse en complexité spatiale puisque seulement deux tableaux de 5 et 7 éléments doivent être stockés. La méthode est basée sur une série de rotations complexes en coordonées pour obtenir une estimation des fonctions trigonométrique et exponentielles. De ce fait les algorithmes CORDIC sont particulièrement adaptés à des environnements informatiques où les ressources spatiales et temporelles sont comptées, comme les calculatrices de poche.

3 Exemples

Les résultats des 3 premières fonctions sont présentés sur la Figure 1.
Pour la 1er question, les fonctions ont été testés sur des exemples de plusieurs formes, avec différentes précision (p décimales). Ces nombres sont choisi pour pouvoir prendre en compte le plus de cas possibles, à savoir des "grands" entier, des nombres à virgules supérieurs à 1, et des nombres à virgules inférieurs à 1.

La deuxième et la troisième questions ont été testés sur des exemples trouvé sur les diapositives de cours, nous permettant de comparer les erreurs relatives obtenue par la fonctions, avec celles des dites diapositives.

```
Question 1:
Nombre réel     Sur 4 décimales        Sur 6 décimales
3.1415927213        3.142                  3.14159
10507.1823          10510                  10507.2
0.0001857563        0.0001858              0.000185756

Question 2:
D'après le cours en base 10 et p = 4 0.1056 +m 0.4985x10e-5 = 0.1056. Notre résultat de la même opération : 0.1056
D'après le cours en base 10 et p = 4 0.3456 xm 0.2921 = 0.1009. Notre résultat de la même opération : 0.1009

Question 3:
La valeur d'erreur de l'addition machine face au réel pour l'opération 3.981786542 +m 2356789.873 est:  8.746849582923177e-05
La valeur d'erreur de la multiplication machine face au réel pour l'opération xm est :  2.4956605763640586e-05
```

Figure 1 – Résulats des fonctions de la partie 1.

Les figures 2 et 3 représentent les graphiques obtenue selon la méthode présenté Section 2 avec, respectivement $x = 3.981786542$ et $x = 767576457.87865$

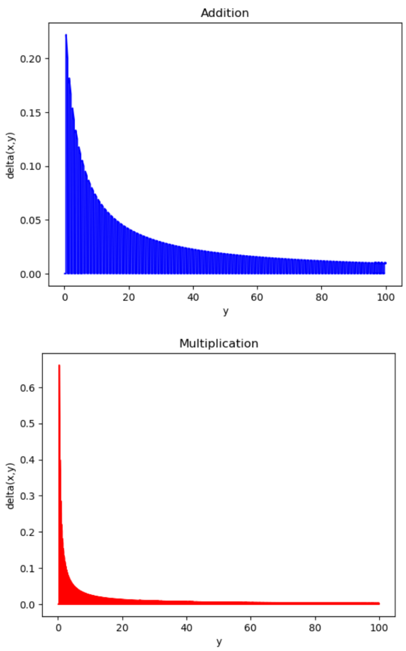

FIGURE 2 – Affichage partie 1 pour x = 3.981786542

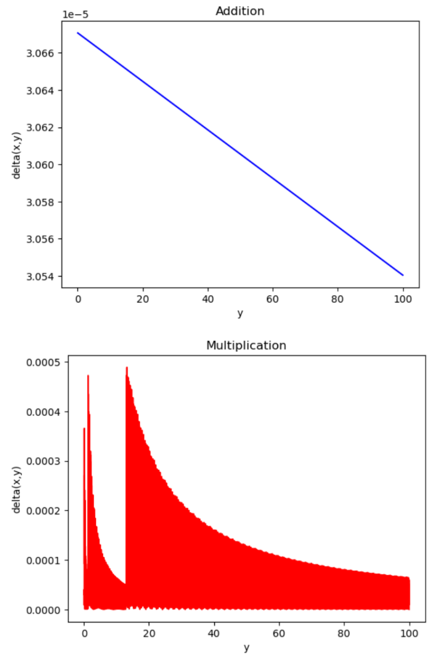

FIGURE 3 – Affichage partie 1 pour x = 767576457.87865

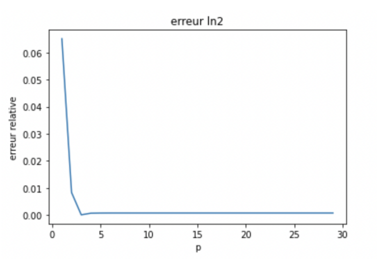

FIGURE 4 – Termes de 1 à n

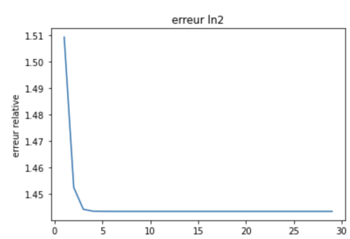

FIGURE 5 – Termes de n à 1

4 Commentaires

Ilyes Bechoual

Bien que la première question semblait d'une facilité déconcertante, il nous a fallu du temps pour la coder, la difficulté était dans l'approximation exacte choisie et l'arrondi. De nombreux cas se démarquaient ce qui nous a obligé à les distinguer. La suite découlant de cette fonction il ne fallu pas beaucoup de temps pour les finir. La répartition des tâches ne fut pas difficile et chacuns a touché à tout que ce soit le code ou bien le rapport. Néanmoins une difficulté a travaillée en tandem, le code n'est pas adapté à cela.

Matteo Marcos

Beaucoup de soucis au niveau de l'implémentation de la première fonction, permettant de simuler la représentation machine. La méthode utilisé dans cette fonction aurait pu être différente, utilisant une somme de puissances de 10, mais nous n'avons pas chercher à mettre en place cette implémentation.

Ramy jeribi

Chacun a participé dans les différentes étapes du projet de l'écriture du code à l'écriture du rapport. La premiére question et la compréhension de la méthode **CORDIC** nous a pris un peu de temps. Mais la réalisation des autres fonctions et la phase de test et de vérification se sont faites assez rapidement en répartissant les taches.

César Larragueta

Malgré la simplicité apparente du problème auquel répond la première fonction, son implémentation a nécéssité un effort de réflexion de la part de tout le groupe. Une fois cette difficulté surmontée le projet s'est assez bien déroulé de lui même avant de se confronter aux algorithmes cordic dont la compréhension n'est pas triviale non-plus.

5 Annexe

```python
#Question 1:

def rpdec(x, p):        #this function is for the decimal number.
    puisx= -1
    aux = x
    while (aux < 1):    #we calculate the negative power of x.
        puisx+=1
        aux *= 10
    npui = puisx + p    #npui here gives us the power of 10 wee need to multiply to get only p terms the rest is not hard to undestand.
    partent = x * math.pow(10, npui)
    partent = int(round(partent, 0))
    partent *= math.pow(10, -npui)
    return partent


def rp(x, p):       #rez is short for result.
    if (x < 1):
        return rpdec(x,p) #creating an another function to reduce the length and to make it more comprehensible.
    partdecimal = x % 1        #here we take the decimal part of x.
    partent = round(x,0)       #here the integral part.
    puisx =0
    aux = x
    while (aux >= 1):
        puisx+=1                #puisx is the maximal power of x.
        aux //= 10
    npui = p - puisx            #npui is here to separate case which are if puisx > p and if puisx < p.
    if (npui > 0):       #if puisx < p then :
        partdecimal *= math.pow(10, npui )
        partdecimal = round(partdecimal,0)
        partdecimal *= math.pow(10, -npui ) #we take the decimal part we need by rounding it.
        rez = partent + partdecimal     #then we adding it to the integral part because we know that puisx < p.
    else:
        partent *= math.pow(10, npui )  #here we know that puisx > p then we take the integral part we need by rounding it.
        partent = round(partent,0)
        partent *= math.pow(10, -npui )
        partent = int(round(partent,0))     #this is security in case of problems.
        rez = partent
    return rez
print( "Question 1: ")
print("Nombre réel  Sur 4 décimales     Sur 6 décimales")
print(3.1415927213,"      ", rp( 3.1415927213, 4),"                ",rp( 3.1415927213, 6))
print(10507.1823,"        ", rp( 10507.1823, 4), "                ", rp( 10507.1823, 6))
print(0.0001857563, "      ", rp( 0.0001857563, 4),"            ", rp( 0.0001857563, 6))
print()
```

Figure 6 – Code de rp