TD1 : MPFR

Christophe Pont, LAMPS
christophe.pont@univ-perp.fr
https ://github.com/christof337/AlgoEtCalculScientifique

Licence L3
Algo et Calcul scientifique
18 février 2019

1 Présentation de MPFR

Présentation MPFR est une bibliothèque C utilisée pour des calculs en virgule flottante à précision arbitraire [1] - elle a été développée en France [2]. Elle est entièrement libre, et sous *GNU Lesser GPL*, une licence sensiblement permissive [3].
Initialement intégrée à GMP [4], elle en est désormais une extension. De facto, GMP est toujours nécessaire à l'utilisation de MPFR.
Bien que la bibliothèque ait été initialement développée en C, des interfaces existent pour d'autres langages [5] (attention, la conception est parfois différente) :

— C++
— Java
— Perl
— Python
— Ruby
— R
— ...

Objectif L'intérêt ici est de se familiariser avec cette bibliothèque, et l'utiliser afin de manipuler facilement la précision et les modes d'arrondis permis, en langage C.

Documentation Une documentation de l'interface est disponible en ligne (`http://www.mpfr.org/mpfr-current/mpfr.html`). **Elle servira de référence tout au long de ce TP**.

2 Installation

Si MPFR n'est pas présent sur votre système, vous pouvez le télécharger ici ou là. L'OS recommandé pour l'utilisation de MPFR est bien évidemment GNU/Linux, de par la forte dépendance entre GMP et MPFR. Ceci étant, vous restez libres.

Sous Linux Bien que certaines distributions GNU/Linux l'intègrent nativement, les headers ne sont pas toujours disponibles aux emplacements adéquats au développement.
Se référer à `http://www.mpfr.org/mpfr-current/mpfr.html#Installing-MPFR` pour les instructions d'installation détaillées, et pour tester si l'installation fonctionne.

Sous Windows Vous trouverez tout le nécessaire pour build les packages (ou en télécharger des *prebuilt*) ici `https://github.com/emphasis87/libmpfr-msys2-mingw64`.

Sous OSX Voir macports et récupérer le *Portfile* de MPFR afin de procéder à l'installation (`https://github.com/macports/macports-ports/blob/master/devel/mpfr/Portfile`).

1. `http://www.mpfr.org/#intro`
2. `http://www.mpfr.org/credit.html`
3. `http://www.gnu.org/copyleft/lesser.html`
4. `https://gmplib.org/`
5. `http://www.mpfr.org/#interfaces`

Tester l'installation Si la compilation du simple programme C suivant

```
#include <stdio.h>
#include <mpfr.h>
int main (void)
{
        printf ("MPFR library: %-12s\nMPFR header: %s (based
            on %d.%d.%d)\n", mpfr_get_version (),
            MPFR_VERSION_STRING, MPFR_VERSION_MAJOR,
            MPFR_VERSION_MINOR, MPFR_VERSION_PATCHLEVEL);
        return 0;
}
```

avec `cc -o version version.c -lmpfr -lgmp` s'exécute correctement (et que la version s'affiche bien à l'exécution de `./version`), alors MPFR est correctement installé : vous pouvez passer à la suite.

3 Exercices

Une lecture préliminaire pour comprendre la bibliothèque se trouve ici `http://www.mpfr.org/mpfr-current/mpfr.html#MPFR-Basics`.
L'interface de référence quant à elle est là `http://www.mpfr.org/mpfr-current/mpfr.html#MPFR-Interface`, habituez-vous à y naviguer.

3.1 Exercice 1 : Factorielle

3.1.1 C types

Conseil : vous pouvez trouver le `specifier` *de chaque type ici* ***https://en.wikipedia.org/wiki/C_data_types***. *Pratique pour les* `printf`.

a. Créer un programme C permettant d'afficher la factorielle des n premiers entiers en utilisant le format `int`. Utiliser si possible une fonction récursive.

b. Observer le comportement aux alentours des entiers $[12, 14]$.
Vous pouvez tester les valeurs des factorielles ici si vous avez un doute

c. Réécrire la fonction pour un `long int`, un `float`. Afficher ensuite les factorielles des 36 premiers entiers et concluez.

d. Recommencez avec un type `double`. Tester pour $n \in [170, 172]$

3.1.2 MPFR

Nous allons observer le comportement du même procédé sur des variables `MPFR`.

a. Consulter la documentation pour vous familiariser avec l'initialisation des variables `mpfr_t`. Trouver une méthode de la bibliothèque MPFR permettant de calculer la factorielle directement.

b. Initialiser une variable `mpfr_t` à l'aide de la méthode mpfr_init.
Appeler la méthode de la factorielle sur cette variable (en utilisant comme mode d'arrondi `MPFR_RNDN`).
Comparer les résultats obtenus avec ceux d'un `double` pour $n \in [170, 172]$.

Pour l'affichage, utiliser

```
mpfr_out_str(stdout, 10, 0, mpfr_val_to_print, MPFR_RNDN);
```

où `mpfr_val_to_print` est la variable `mpfr_t` à afficher.

c. À l'aide de la méthode mpfr_init2, choisir arbitrairement la précision de la mantisse.
Afficher les factorielles des 300 premiers entiers pour une mantisse de 200 bits.

d. Concluez sur la flexibilité de la taille de l'exposant avec MPFR.

3.2 Exercice 2 : Modes d'arrondi

Vous trouverez ici : `http://www.mpfr.org/sample.html` un exemple pratique d'utilisation de mpfr.

a. En utilisant cet algorithme, sommer les inverses des factorielles des n premiers entiers à l'aide de MPFR.

b. (Facultatif : MPFR gère les modes d'arrondis suivants : `http://www.mpfr.org/mpfr-current/mpfr.html#Rounding-Modes`. Changer le mode d'arrondi utilisé dans les calculs et observer l'impact sur les résultats.
Cet ordonnancement était-il prévisible ?)

3.3 Exercice 3 : Évaluation de polynome

3.3.1 Méthode directe

Soit le polynôme $p(x) = x^{15} - 30x^{14} + 420x^{13} - 3640x^{12} + 21840x^{11} - 96096x^{10} + 320320x^9 - 823680x^8 + 1647360x^7 - 2562560x^6 + 3075072x^5 - 2795520x^4 + 1863680x^3 - 860160x^2 + 245760x - 32768$.

a. Créer un tableau correspondant à ce polynome à l'aide du code ci-dessous.

```
const double polynome[16] = { 1, -30, 420, -3640, 21840,
    -96096, 320320, -823680, 1647360, -2562560, 3075072,
    -2795520, 1863680, -860160, 245760, -32768 };
```

b. Créer une méthode permettant d'évaluer ce polynome directement, pour une valeur x donnée. Utiliser `mpfr_t` pour les calculs avec l'arrondi `MPFR_RNDN`.

c. Evaluez ainsi $p(x)$ pour $x \in [1.6, 2.4]$ avec un pas de 10^{-4}.
Afficher les résultats dans un fichier (première colonne : valeur de x, seconde colonne : valeur de son évaluation) ; idéalement avec une séparation par tabulation. Ploter ensuite le fichier pour observer le comportement de l'évaluation aux alentours de 2, avec Gnuplot [6] par exemple.

6. `http://gnuplot.sourceforge.net/`

Les oscillations observées rendent difficile la détection de la racine de p par une méthode comme celle de la dichotomie par exemple.

A l'aide du graphique, prospectez la valeur d'une des racines du polynôme.

3.3.2 Méthode de Horner

Horner propose un algorithme plus efficace pour évaluer un polynome, évitant de recalculer les puissances de x successives.
Le principe est de factoriser les x un à un.
Soit

$$P(X) = a_n X^n + ... + a_1 X + a_0 \in \mathbb{R}[X]$$

La réduction de Horner donne

$$a_n x^n + ... + a_1 x + a_0 = ((...(a_n x + a_{n-1})x + a_{n-2})...)x + a_0$$

L'algorithme d'évaluation devient alors :

$y \leftarrow a_n$
for $i \leftarrow (n-1), 0$ **do**
  $y \leftarrow x.y + a_i$
end for

Soit en C :

```
void evaluatePolynomeHorner(mpfr_t acc, const double xDouble
    ) {
mpfr_t x;

mpfr_init2(x, PRECISION);

mpfr_set_d(x, xDouble, MPFR_RNDN);
mpfr_set_d(acc, polynome[0], MPFR_RNDN);

for (size_t ind = 1; ind <= DEGRE; ++ind) {
//              acc = x * acc + polynome[ind];
                mpfr_mul(acc, acc, x, MPFR_RNDN);
                mpfr_add_d(acc, acc, polynome[ind], MPFR_RNDN);
}

mpfr_clear(x);
}
```

a. Implémenter l'algorithme de Horner pour l'évaluation du polynome. L'appliquer au même intervalle que précédemment. Observer la différence.

3.3.3 Factorisation

En réalité, le polynome étudié peut-être factorisé en $(x-2)^{15}$.

a. Utiliser cette dernière formule pour calculer directement les évaluations sur le même intervalle que précédemment.

b. Concluez sur l'efficacité des méthodes (et l'intérêt de trouver les racines lorsque possible) en terme de temps de calcul et d'exactitude.

3.3.4 MPFR

a. Utiliser `mpfr_init2` (au lieu de `mpfr_init`) pour augmenter la précision jusqu'à 60 (au lieu de 53).

b. Observer l'évolution des oscillations, même avec une méthode directe.

c. Changer la précision à 8000. Observer l'évolution sur le temps de calcul.
Sur un procédé itératif, on aurait une déperdition conséquente en terme de temps de calcul... et de mémoire !