Compte-rendu du TP3
de Calcul scientifique 3

Arnaud CHAPON (chapon@etudiant.unicaen.fr)

5 mars 2006

1 Taille des variables

1.1 Sizes.C

Le programme *Sizes.C* permet de déterminer la taille de plusieurs types utilisés en C++. L'opérateur unaire *Sizeof* utilisé dans l'exemple donne directement la taille en octets de l'opérande correspondant.

Le fichier est enregistré dans le répertoire de travail et compilé à l'aide de la commande :

$ g++ Sizes.C -lm -o Sizes

Il est ensuite exécuté, en tapant la commande :

$ ./Sizes

On peut relever les tailles des types que donne ce petit programme.

Les résultats sont rassemblés dans le tableau 1.

Type	Taille
bool	1
char	1
short (int)	2
int	4
long (int)	4
float	4

Table 1 – Tailles des types avec Sizes.C

1.2 Sizes2.C

Une copie de ce fichier est enregistrée dans le répertoire de travail et modifiée de sorte à afficher les tailles des éléments suivants :
- unsigned long
- double
- char *
- int *
- float*

Comme précédemment, on peut reconstruire un tableau contenant les tailles des types donnés par ce nouveau petit programme.

Les résultats sont rassemblés dans le tableau 2.

Type	Taille
unsigned long	4
double	8
char *	4
int *	4
float *	4

Table 2 – Tailles des types avec Sizes2.C

2 Codage des entiers

2.1 Bits.C

Le programme *Bits.C* calcule la valeur positive minimale qui peut être contenue dans une variable de type *unsigned short*.

Avec 16 bits, la plus grande valeur d'un *unsigned short* est 65535. Ce qui correspond bien à $(2^{16} - 1)$.

2.2 Bits2.C

Le fichier *Bits.C* est modifié de sorte à déterminer la valeur maximale codable avec un *unsigned long*.

La valeur maximale qu'il est possible de coder à l'aide du type *unsigned long* est 4294967295.

2.3 Bits3.C

De même, une autre copie de ce programme, nommée *Bits3.C* permet de déterminer la valeur maximale qui puisse être codée à l'aide d'un *short*. Cette valeur est de 32767.

La représentation en machine du nombre -1 est également donnée par ce programme. Elle est : 11111111.

2.4 Bits4.C

Enfin, une dernière modification, appelée *Bits4.C* permet de trouver la valeur minimale négative qui peut être codée au moyen d'un *short*. Cette valeur est de -32768.

3 Nombres réels en machine

3.1 FloatLimit.C

Le programme *FloatLimit.C* permet de définir approximativement la limite positive maximale codée sur une variable de type *float*.
Le résultat obtenu est d'environ 1E+38.

3.2 FloatLimit2.C

Le programme précédent est modifié de sorte à afficher la limite inférieure strictement positive qu'il est possible de coder avec un *float*.
Cette limite est d'environ 1E-39.

4 Calcul de π

Le programme *Pi1.C* permet un calcul de π de trois façons différentes. Les deux premières méthodes sont basées sur la formule suivante :

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} ...$$

4.1 Méthode 1

Le programme est compilé avec la commande suivante :

$ ./Pi1 method_1

On peut rediriger le flux de sortie vers un fichier au format data, appelé methode_1.dat :

$ ./Pi1 method_1 >> methode_1.dat

Ainsi, il est possible d'utiliser Gnuplot pour tracer, en échelle "log/log" la convegence de la formule, vers π :

gnuplot> set logscale x ; set logscale y
gnuplot> plot 'methode_1.dat'

Le résultat obtenu est représenté sur la figure 1.

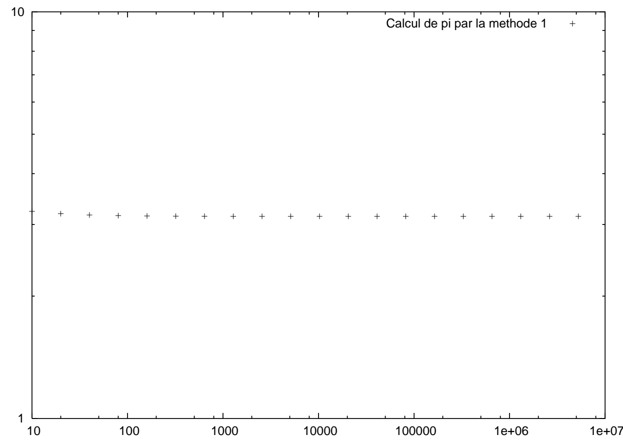

FIGURE 1 – Méthode 1

4.2 Méthode 2

Le programme est compilé avec la commande suivante :

$ ./Pi1 method_2

De la même façon que précédemment, on peut rediriger le flux de sortie vers un fichier au format data, appelé methode_2.dat :

$ ./Pi1 method_2 >> methode_2.dat

Ainsi, il est possible d'utiliser Gnuplot pour tracer, en échelle "log/log" la convegence de la formule, vers π.

Le résultat obtenu est représenté sur la figure 2.

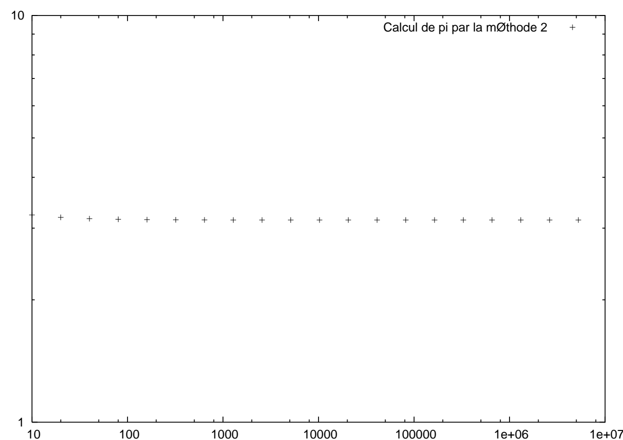

FIGURE 2 – Méthode 2

Superposons ces deux graphes afin de mieux apprécier les différences. Le résultat obtenu est représenté sur la figure 3.

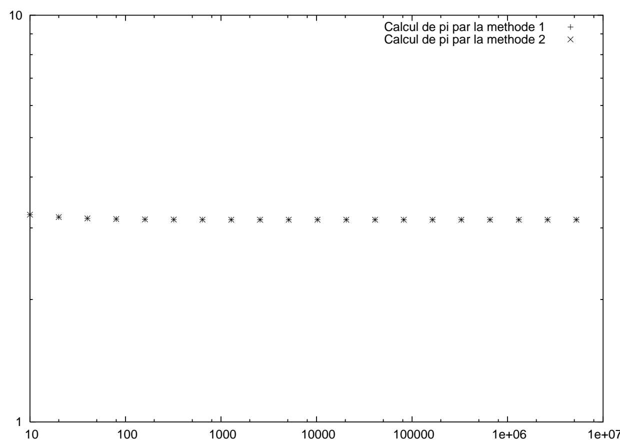

FIGURE 3 – Méthodes 1 et 2 superposées

4.3 Méthode 3

Une autre méthode de calcul de π est basée sur la formule de Machin :

$$\pi = 4f(x_1) - f(x_2)$$

avec :

$$x_1 = \frac{1}{5}$$

$$x_2 = \frac{1}{239}$$

et :

$$f(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + ...$$

Le résultat obtenu par cette nouvelle méthode de calcul est donnée par le graphe de la figure 4.

5 Simulation d'un jeu de dé

Le programme *Die1.C* illustre l'utilisation de la fonction *drand48* en simulant le comportement d'un dé. Cette fonction permet de générer des nombres pseudo-aléatoires.

Le programme est compilé et exécuté. Le tirage donne :

$$1 - 1 - 1 - 3 - 2 - 5 - 4 - 1 - 6 - 4$$

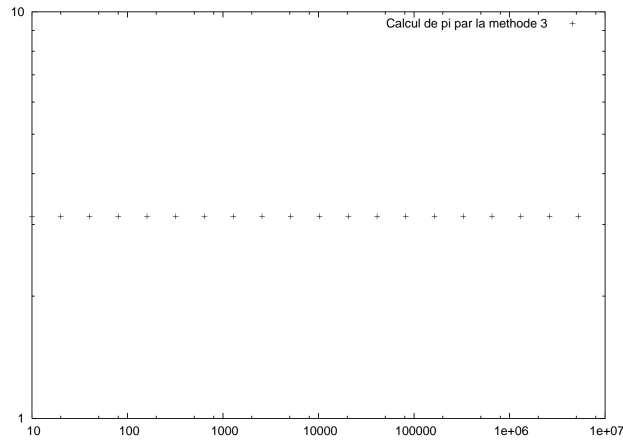

FIGURE 4 – Méthode 3

On peut changer a valeur de l'amorce en agissant sur l'option "seed". On peut également choisir le nombre de tirages en agissant sur l'option "throws".

Par exemple, tapons dans le terminal :

$ ./Die1 throws=50 seed=130385

On pourra alors vérifier l'uniformité de la distribution en étudiant ce qui est proposé en sortie, à savoir :

$$5 - 5 - 2 - 5 - 2 - 5 - 5 - 4 - 3 - 1$$
$$1 - 1 - 2 - 4 - 3 - 6 - 3 - 2 - 6 - 1$$
$$5 - 6 - 4 - 2 - 6 - 4 - 6 - 5 - 6 - 3$$
$$4 - 1 - 2 - 3 - 6 - 2 - 4 - 3 - 2 - 5$$
$$1 - 2 - 2 - 6 - 1 - 5 - 3 - 2 - 6 - 4$$

Comptons le nombre de fois que chaque valeur est sortie. Les résultats sont rassemblés dans le tableau 3.

On vérifie donc bien l'uniformité approximative de ces tirages. Mathématiquement, en effet, sur cinquante tirages, on peut s'attendre à avoir $\frac{50}{6} \approx 8$ sortie de chaque nombre, avec une différence de nombre de sorties de $\sqrt{\frac{50}{6}} \approx 3$.

Considérant ces résultats, on peut donc, ici, effectivement, considérer le tirage uniforme.

Valeur du Tirage	Nombre de Sorties
1	7
2	11
3	7
4	7
5	9
6	9

TABLE 3 – Tirages pseudo-aléatoires avec le programme Die1.C

6 Un peu de statistique

6.1 Stat1.C

Le programme *Stat1.C* prend en flux d'entrée une série de nombre entrés par l'utilisateur. Il compte ces nombres et en calkcule la moyenne.

Exécutons ce programme en entrant les cinquante nombres tirés par le programme *Die1.C* et écrits en section précédente.

Le résultat obtenu est le suivant :

- Nombre d'entrées : 50
- Moyenne : 3.54

On peut rediriger le flot de sortie du programme *Die1* vers le flot d'entrée de *Stat1* au moyen d'untube. On tapera, par exemple :

$ ./Die1 throws=100 sides=10 seed=12345 | ./Stat1

Le résultat obtenu est :

- Nombre d'entrées : 100
- Moyenne : 5.15

6.2 Stat2.C

Le programme *Stat2.C* calcule, en plus de la moyenne, l'écart-type de la série de valeurs lues dans le flot d'entrée.

Exécutons ce programme en entrant à nouveau les cinquante nombres tirés par le programme *Die1.C* et écrits en section précédente.

$ echo " 5 5 2 5 2 5 5 4 3 1 1 1 2 4 3 6 3 2 6 1 5 6 4 2 6 4 6 5 6 3 4 1 2 3 6 2 4 3 2 5 1 2 2 6 1 5 3 2 6 4 " | ./Stat2

Le résultat obtenu est le suivant :

- Nombre d'entrées : 50

– Moyenne : 3.54
– Ecart-type : 1.7404

Une autre méthode consiste à rediriger le flot de sortie du programme *Die1* vers le flot d'entrée de *Stat1* au moyen d'un tube.

Avec les mêmes nombres, on aurait alors juste à écrire :

$ ./Die1 throws=50 seed=130385 | ./Stat2

7 Histogramme à une dimension

7.1 Histo1.C

La classe *Histogram1D* est le cœur du programme *Histo1.C*.

En effet, la classe prend en argument les valeurs minimales et maximale de l'intervalle à considérer, ainsi que la largeur de chaque intervalle.

La fonction *main* s'occupe de lire les saisies de l'utilisateur et en vérifie la validité. Le cas échéant, la fonction fait appel à la classe *Histogram1D* qui "compte" le nombre de sorties d'un même nombre.

La fonction *main* fait ensuite appel à la fonction *dump_histo* afin d'afficher les valeurs dans le flot de sortie.

7.2 Die1.C

Le fichier *Die1.dat* contient les valeurs minimales et maximale de l'intervalle à considérerle nombre d'intervalles voulu et les tirages obtenus par la commande :

$ ./Die1 throws=50 seed=130385

On exécute alors le programme *Histo1* en lui donnant ce fichier pour flot d'entrée. On écrit donc :

$ cat "Die1.dat" | ./Histo1 >> Histo1.dat

Ainsi, le fichier *Histo1.dat* contient ceci :

1 7
2 11
3 7
4 7
5 9
6 9

Il est donc exploitable via Gnuplot. On tape :

gnuplot> set xrange[0 :7] ; set yrange[0 :12]
gnuplot> plot 'Histo1.dat'

On obtient le graphe représenté par la figure 5.

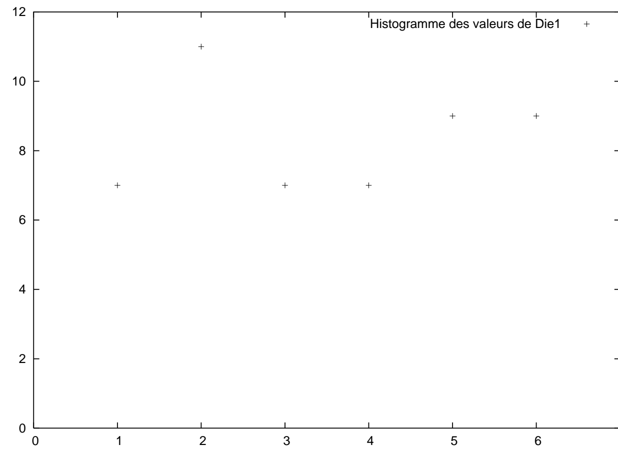

FIGURE 5 – Tirages de Die1

7.3 Die2.C

On modifie maintenant le programme *Die1.C* de sorte que le nouveau programme *Die2.C* affiche la somme de trois tirages successifs d'un dé à six faces.

On compile alors le programme en *Die2* et on exécute *Histo1* en prenant pour flot d'entrée le fichier *Die2.dat* ainsi créé.

$ cat "Die2.dat" | ./Histo1 >> Histo2.dat

Ainsi, le fichier *Histo2.dat* contient ceci :

3 1
4 0
5 4
6 0
7 3
8 2
9 7
10 5
11 6
12 8
13 3
14 4
15 5
16 2
17 0
18 0

Il est donc exploitable via Gnuplot. On tape :

gnuplot> set xrange[2 :19] ; set yrange[0 :9]
gnuplot> plot 'Histo1.dat'

On obtient le graphe représenté par la figure 6.

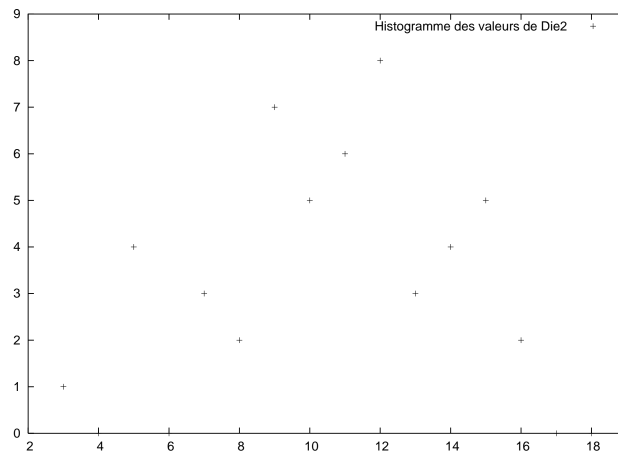

FIGURE 6 – Tirages de Die2

La distribution obtenue semble assez dispersée. En revanche, avec un nombre supérieur de tirages, on pourrait observer une distribution en cloche,

En effet, selon les lois de probabilités, il n'existe qu'une façon de trouver 3 ou 18. Les valeurs centrales sont plus probables, et, en particulier, 10 et 11.