Algorithmes pour calculer les décimales de $\sqrt{2}$

1 Méthode élémentaire utilisant un développement limité

1.1 Quelques égalités

En utilisant les égalités sur la racine carré $a = (\sqrt{a})^2$, $\sqrt{ab} = \sqrt{a}\sqrt{b}$, $\sqrt{\frac{a}{b}} = \frac{\sqrt{a}}{\sqrt{b}}$ pour des nombres positifs a et b, on vérifie facilement les quelques formules suivantes sur $\sqrt{2}$:

$$\sqrt{2} = \frac{7}{5}\frac{1}{\sqrt{1-\frac{1}{50}}}$$

$$\sqrt{2} = \frac{41}{29}\frac{1}{\sqrt{1-\frac{1}{1682}}}$$

et encore

$$\sqrt{2} = \frac{27720}{19601}\frac{1}{\sqrt{1-\frac{1}{384199201}}}$$

et enfin pour

$$\frac{1}{\sqrt{2}} = \frac{13860}{19601}\frac{1}{\sqrt{1-\frac{1}{384199201}}}$$

1.2 Un peu d'analyse

Les égalités précédentes sur $\sqrt{2}$ peuvent nous permettre de calculer de "bonnes" approximations de ce nombre.
En effet au voisinage de 0, on peut décomposer la fonction $x \mapsto \frac{1}{\sqrt{1-x}}$ en une série entière. Plus précisement on montre que : $\forall x$ tel que $\mid x \mid < 1$ on a :

$$\frac{1}{\sqrt{1-x}} = \sum_{k=0}^{\infty} a_k x^k$$

$$\frac{1}{\sqrt{1-x}} = 1 + \frac{1}{2}x + \frac{1.3}{2!\,2^2}x^2 + \frac{1.3.5}{3!\,2^3}x^3 + \frac{1.3.5.7}{4!\,2^4}x^4 + \cdots$$

c'est-à-dire avec

$$a_k = \frac{\prod_{j=0}^{k-1}(1+2j)}{k!\,2^k}$$

La série converge donc en prenant par exemple

$$x = \frac{1}{384199201}$$

Ce développement en série entière permet de calculer $\sqrt{2}$ en ne faisant que des additions, soustractions, et des multiplications et divisions avec de petits entiers.

1.3 Algorithme élémentaire

1.3.1 Rappel rapide sur les flottants à virgule fixe

Pour comprendre tout l'intérêt des algorithmes proposés, un détour doit être fait sur une méthode pour représenter les "réels" dans les ordinateurs, et qu'on appelle **flottants à virgule fixe**.
Pour la clarté de l'exposé je vais raisonner sur un représentation en base 10 mais n'importe quel base convient. En particulier pour l'implémentation en langage C, l'auteur a utlisé la base 10^8 qui permet de stocker un chiffre sur un entier de type *long* (ie 32 bits).
Cette représentaion repose sur le simple fait que tout nombre décimale n'ayant que p chiffres après la virgule s'écrit sous la forme $\frac{a}{10^p}$ où a est en entier relatif. Si on se fixe un précision (ce qui revient à fixer p), il suffit d'enregistrer l'entier a pour enregistrer le nombre décimal.
De plus l'addition et la soustraction entre nombres décimales à p chiffres après la virgule se fait simplement en faisant les opérations sur les numérateurs :

$$\frac{a}{10^p} + \frac{b}{10^p} = \frac{a+b}{10^p}$$

$$\frac{a}{10^p} - \frac{b}{10^p} = \frac{a-b}{10^p}$$

ainsi que les multplications et divisions par un entier :

$$\left(\frac{a}{10^p}\right) \times d = \frac{a \times d}{10^p}$$

$$\left(\frac{a}{10^p}\right) \div d = \frac{a \div d}{10^p}$$

Par exemple dans cette représentaion, si on travaille avec p chiffres après la virgule, le réel $3,0\cdots01$ avec $p-1$ zéros correspond à l'entier $30\cdots01$ avec $p-1$ zéros.

Pour une explication plus détaillée, je vous renvoie à la page de mon site sur la représentation des nombres.

Ainsi pour calculer $\sqrt{2}$ avec une précision à 10^{-p} près (avec $p = 1000$ par exemple), on est ramené à faire des additions, des soustractions sur des entiers de grande taille (de l'ordre de 10^p) et des multiplications, divisions par de "petits" entiers (en général inférieur à dix millions) comme nous allons le voir dans le prochain paragraphe.

1.3.2 Algorithme utilisant un développement en série entière

En posant $m = 384199201$ et

$$u_k = \frac{\prod_{j=0}^{k-1}(1+2j)}{k!\,2^k\,m^k}$$

Pour obtenir une approximation de $\sqrt{2}$, il suffit de calculer la série partielle

$$\sum_{k=0}^{K} u_k \approx \sqrt{2}.$$

Le nombre K dépend du nombre de décimales voulues. Une méthode simple consiste à calculer u_k par récurrence en se servant des égalités :

$$u_{k+1} = \frac{2k+1}{2(k+1)m} u_k$$

ou encore

$$u_{k+1} = \frac{u_k}{m} - \frac{u_k}{m(2k+2)}$$

On peut utiliser la deuxième égalité, ce qui donne l'algorithme suivant :

$$
\begin{array}{ll}
k \leftarrow 0 & \\
u \leftarrow 1 & \\
s \leftarrow 1 & \\
\text{répéter} & \\
u \leftarrow u \div 384199201 & (1) \\
a \leftarrow u \div (2k+2) & (2) \\
u \leftarrow u - a & (3) \\
s \leftarrow s + u & (4) \\
k \leftarrow k+1 & \\
\text{jusqu'à } u = 0 & \\
s \leftarrow s \times 27720 & (5) \\
s \leftarrow s \div 19601 & (6) \\
\text{afficher } s \approx \sqrt{2} &
\end{array}
$$

Remarques :

- Comme on peut le voir sur l'algorithme précédent, on fait des additions et soustractions sur de grands entiers (lignes (3) et (4)) des divisions par de petits entiers (lignes (1), (2) et (6)) et une multiplication par un petit entier (ligne (5)).
  Pour la ligne (2), on divise bien par un petit entier car pour calculer $\sqrt{2}$ avec 2560000 chiffres après la virgule par exemple, k varie entre 0 et $K = 327409$ et plus généralement pour avoir p chiffres après la virgule, il faut prendre $K \approx \frac{p}{\log(m)}$ qui est un entier petit pour des valeurs de p inférieures à 10^7 (valeurs courantes en pratique).
- J'ai choisi la deuxième égalité pour u_{k+1} car il est plus facile de faire deux divisions avec de petits diviseurs que de faire une multiplication puis une division par des entiers qui dépasseraient la taille d'un chiffre dans la base que j'ai choisi (cad 10^8 dans mon cas).
- L'algorithme a un comportement asymptotique de l'ordre de $O(n^2)$: i.e. pour doubler le nombre de décimales exactes, il faut multiplier par 4 le temps de calcul.
- Puisque $\frac{u_{k+1}}{u_k} \sim \frac{1}{m}$ et $\log(m) \approx 7,8$, on gagne asymptotiquement 7.8 chiffres par itération (ce qui explique la "lenteur" relative de l'algorithme).

Résultats :

Voici le temps mis par le programme de l'auteur :

nombre de décimales exactes	5 000	10 000	100 000	1 000 000	2 560 000
durée du calcul en secondes	0.08	0.3	28.7	3375.4	22680.1

Configuration :

Processeur : Athlon K7 700 Mhz.

Carte mère : ASUS K7M supportant 256 Mo de SDRAM à 100 Mhz.

Système d'exploitation : Mandrake8.2.

Compilateur : gcc-3.0.4.

En pratique :

Pour compiler le programme, tapez dans une console (dans le répertoire contenant le source) :
gmake

Pour exécuter le programme :
./pii p sqrt2 10
où p est le nombre de décimales justes désirées.