Les secrets de l'algorithme Cordic

Le problème

Le fonctionnement d'une calculatrice, lorsqu'on lui demande de calculer un logarithme, relève du secret. On peut cependant supposer que celle-ci travaille à partir d'un petit tableau de valeurs de la fonction logarithme. L'algorithme Cordic (Cordiante Rotation Digital Computer), développé par J.E. Volder en 1959 et encore très utilisé, est l'exemple le plus connu de méthodes qui, à partir d'un nombre fini de valeurs connues d'une fonction, donnent une valeur approchée de la fonction pour *toute* valeur de la variable.
Quitte à partir d'un tableau de valeurs pour calculer $\ln(x)$, on pourrait penser s'appuyer sur les valeurs de $\ln(n)$ avec n entier. Il n'en est rien : les valeurs de départ seront $\ln(10)$, $\ln(2)$, $\ln(1{,}1)$, $\ln(1{,}01)$, $\ln(1{,}001)$, $\ln(1{,}0001)$ et $\ln(1{,}00001)$.
Pourquoi ? Pour un réel x compris entre 1 et 10, on peut approcher $10 / x$ par un produit du type $2^{k_1} \times 1{,}1^{k_2} \times 1{,}01^{k_3} \times \ldots \times 1{,}00001^{k_6}$.

Complément culturel

Logarithme et algorithme sont deux mots presque identiques (ils sont anagrammes et seuls trois parmi les dix lettres ont leur place modifiée). De là à penser que le calcul d'un logarithme est une question d'algorithme, il n'y a pas beaucoup de chemin à parcourir.

Les programmes

Python

La table de logarithmes est donnée par les listes A et B. Dans la liste A, on a stocké les valeurs de la variable x, et dans B, on a stocké des valeurs approchées de $\ln x$ correspondantes. La variable X approche de plus en plus $10 / x$.
La variable Y approche, quant à elle, de plus en plus $\ln x$.

le programme

```
>>> def cordic(x):
        i=0
        A=[2,1.1,1.01,1.001,1.0001,1.00001,1.000001]
        B=[0.69314718056,0.0953101798,0.00995033085,0.00099950033,0.0000999
        X=1
        Y=2.302585093
        while i<=6:
                k=0
                while X*(A[i])**k<=10/x:
                        k=k+1
                k=k-1
                X=X*(A[i])**k
                Y=Y-k*B[i]
                i=i+1
        return Y
```

le résultat

```
>>> cordic(2.952)
1.0824833894399997
```

Scratch

Principe : soit à déterminer le logarithme népérien de 2,135.

On encadre $X = \frac{10}{2{,}135}$ de plus en plus finement.

• D'abord entre 2^2 et 2^3 :

$\ln 10 - 2\ln 2 < \ln(2{,}135) < \ln 10 - 3\ln 2$.

• On poursuit : $2^2 \times 1{,}1 < X < 2^2 \times 1{,}1^2$.

$\ln 10 - 2\ln 2 - 2\ln 1{,}1 < \ln 2{,}135 < \ln 10 - 2\ln 2 - \ln 1{,}1$.

• $2^2 \times 1{,}1 \times 1{,}01^6 < X < 22 \times 1{,}1 \times 1{,}01^7$.

$\ln 10 - 2\ln 2 - \ln 1{,}1 - 7 \times \ln(1{,}01) < \ln(2{,}135) < \ln 10 - 2\ln 2 - \ln 1{,}1 - 6 \times \ln(1{,}01)$.

• $2^2 \times 1{,}1 \times 1{,}01^6 \times 1{,}001^2 < X < 2^2 \times 1{,}1 \times 1{,}01^6 \times 1{,}001^3$…

En mettant en mémoire ln10, ln2, ln(1,1), ln(1,01), ln(1,001) ln(1,0001) et ln(1,00001), on obtiendra une valeur approchée de ln(2,135).

On trouvera le programme correspondant en page 36. Ci-dessous figure celui qui permet de constituer les « listes de référence » à partir desquelles on applique l'algorithme Cordic.

AlgoBox

Les valeurs de référence de x et de $\ln(x)$ sont entrées respectivement dans les listes Listeréférencex et Listeréférencelnx.

```
VARIABLES
    Listeréférencex EST_DU_TYPE LISTE
    Listeréférencelnx EST_DU_TYPE LISTE
    x EST_DU_TYPE NOMBRE
    i EST_DU_TYPE NOMBRE
    lnx EST_DU_TYPE NOMBRE
    approxde10surx EST_DU_TYPE NOMBRE
    k EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    Listeréférencex[1] PREND_LA_VALEUR 2
    Listeréférencex[2] PREND_LA_VALEUR 1.1
    Listeréférencex[3] PREND_LA_VALEUR 1.01
    Listeréférencex[4] PREND_LA_VALEUR 1.001
    Listeréférencex[5] PREND_LA_VALEUR 1.0001
    Listeréférencex[6] PREND_LA_VALEUR 1.00001
    Listeréférencelnx[1] PREND_LA_VALEUR 0.69314718
    Listeréférencelnx[2] PREND_LA_VALEUR 0.09531017
    Listeréférencelnx[3] PREND_LA_VALEUR 0.00995033
    Listeréférencelnx[4] PREND_LA_VALEUR 0.0009950033
    Listeréférencelnx[5] PREND_LA_VALEUR 0.000099995
    Listeréférencelnx[6] PREND_LA_VALEUR 0.00000999995
    LIRE x
    i PREND_LA_VALEUR 1
    lnx PREND_LA_VALEUR 2.302585093
    approxde10surx PREND_LA_VALEUR 1
    TANT_QUE (i<=6) FAIRE
        DEBUT_TANT_QUE
        k PREND_LA_VALEUR 0
        TANT_QUE (approxde10surx*pow(Listeréférencex[i],
            DEBUT_TANT_QUE
            k PREND_LA_VALEUR k+1
            FIN_TANT_QUE
        k PREND_LA_VALEUR k-1
        approxde10surx PREND_LA_VALEUR approxde10surx
        lnx PREND_LA_VALEUR lnx-k*Listeréférencelnx[i]
        i PREND_LA_VALEUR i+1
        FIN_TANT_QUE
    AFFICHER lnx
FIN_ALGORITHME
```

Prolongements

Les programmes des pages précédentes fonctionnent uniquement pour x compris entre 1 et 10. Nous vous proposons de les étendre pour tout $x > 0$.

Pour $x = A \times 10^k$ supérieur à 10, on utilisera la relation $\ln(A \times 10^k) = \ln A + k \times \ln(10)$.

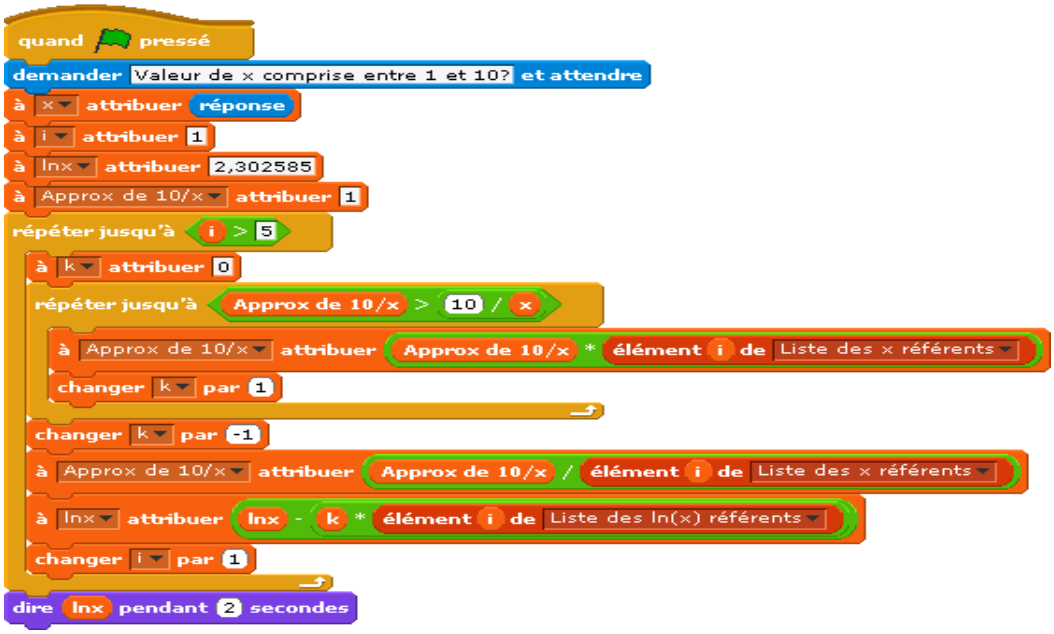
Pour $x < 1$, on utilisera $1 / x > 1$ et $\ln x = -\ln(1 / x)$.

Le programme vous est donné en page suivante.

Les secrets de l'algorithme Cordic

Suite du programme Scratch

La place manquant en page 35, voici le programme Scratch annoncé :



Prolongements (suite)

Voici l'extension du programme Python de la page 34 à tout $x > 0$.

le programme **Python**

```
>>> def extensioncordic(x):
        ln10=2.302595093
        if x>=10:
                A=x
                k=0
                ln10=2.302595093
                while A >10:
                        A=A/10
                        k=k+1
                return cordic(A)+k*ln10
        if x<1:
                return - cordic(1/x)
        if x==1:
                return 0
        if (x>1) and (x<10):
                return cordic(x)
```