Algorithmes et logarithmes

Table des Matières

Algorithmes et logarithmes

I. Algorithmes de Briggs

I. A. Calcul de $\log_{10}(2)$

henry Briggs (anglais 1556-1630) souhaite calculer logarithme de 2 Pour se faire il remarque qu'il suffit de connaître le nombre de chiffres qui composent 2^n et prendre n très grand. En effet si le nombre de chiffres de 2^n est k on a :

$$10^{k-1} < 2^n < 10^k$$

Avec le logarithme en base 10, on a alors

$$\frac{k-1}{n} < \log_{10}(2) < \frac{k}{n}$$

Briggs choisit $n = 10^{14}$, en regroupant ses calculs des puissances de 2 en quatraine :

- Calcul de 2^{10} :

$$2^2 = 4\ ;\ 2^4 = \left(2^2\right)^2\ ;\ 2^8 = \left(2^4\right)^2\ ;\ 2^{10} = 2^8 \times 2^2$$

- Calcul de 2^{100} :

$$2^{20} = \left(2^{10}\right)^2\ ;\ 2^{40} = \left(2^{20}\right)^2\ ;\ 2^{80} = \left(2^{40}\right)^2\ ;\ 2^{100} = 2^{80} \times 2^{20}$$

- Calcul de $2^{1\,000}$:

$$2^{20} = \left(2^{100}\right)^2\ ;\ 2^{400} = \left(2^{200}\right)^2\ ;\ 2^{800} = \left(2^{400}\right)^2\ ;\ 2^{1\,000} = 2^{800} \times 2^{200}$$

- Ainsi de suite jusqu'à 10^{15} pour une précision à 10^{14}.

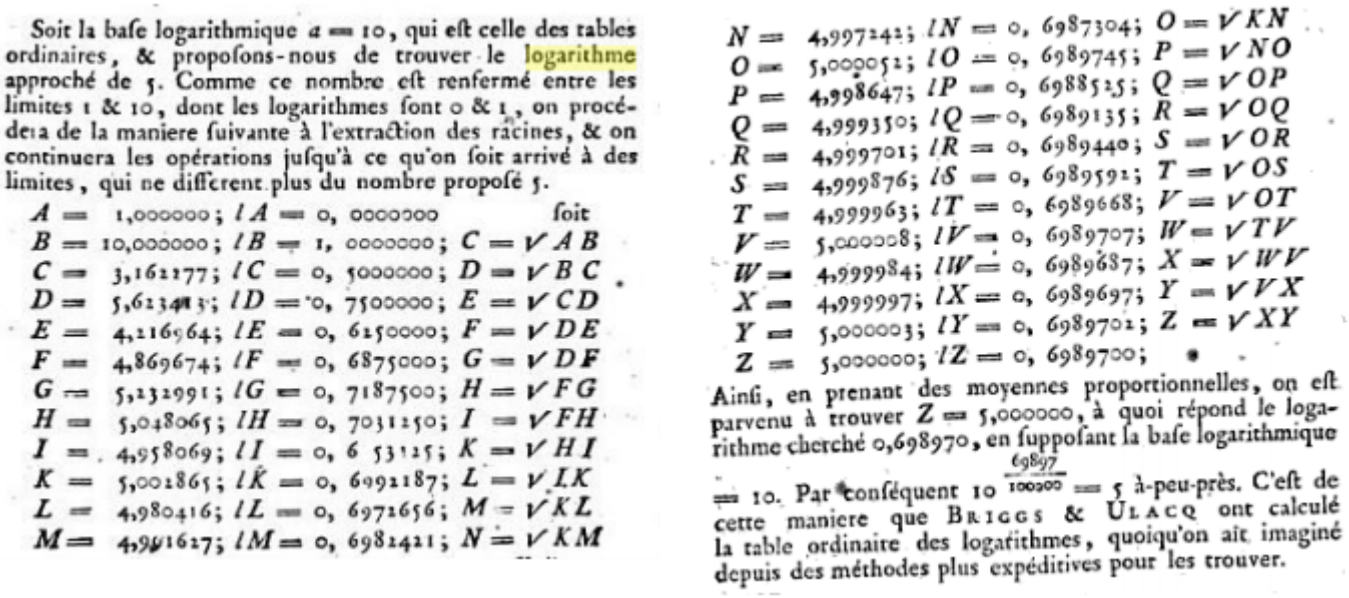
```python
def briggs(n):
    '''
    n=14 pour Briggs
    calcul les valeurs des puissances de 2 par quatraine jusqu'à 10^n
    '''
    i=2
    quatraine=0
    a=2
    while 10**quatraine<10**n :
        quatraine=quatraine+1
        a=a**i
        b=a**2
        c=b**2
        d=a*c
        l=len(str(d))
        a=d
    print('2^(10^',quatraine,') admet ',l,'chiffres')

    return
```

briggs_log2.py

In[1]:Briggs(6)
Out[1]: 2^{10^6} admet 301 030 chiffres.

I. B. Calculs de logarithmes entre 1 et la base B

Dans son Introduction à l'analyse infinitésimale (traduction française de 1796), Léonhard Euler (1707-1783) reprend les travaux de Briggs donne un algorithme pour calculer logarithme de 5.
Cet algorithme se généralise calculer tous les logarithmes décimaux entre 1 et la base $\mathscr{B}$ du logarithme souhaité (c'est le principe de dichotomie).



Soit la base logarithmique $a = 10$, qui est celle des tables ordinaires, & proposons-nous de trouver le logarithme approché de 5. Comme ce nombre est renfermé entre les limites 1 & 10, dont les logarithmes sont 0 & 1, on procédera de la maniere suivante à l'extraction des racines, & on continuera les opérations jusqu'à ce qu'on soit arrivé à des limites, qui ne different plus du nombre proposé 5.

$A = 1,000000$; $lA = 0, 0000000$ soit
$B = 10,000000$; $lB = 1, 0000000$; $C = \sqrt{AB}$
$C = 3,162177$; $lC = 0, 5000000$; $D = \sqrt{BC}$
$D = 5,623413$; $lD = 0, 7500000$; $E = \sqrt{CD}$
$E = 4,216964$; $lE = 0, 6250000$; $F = \sqrt{DE}$
$F = 4,869674$; $lF = 0, 6875000$; $G = \sqrt{DF}$
$G = 5,232991$; $lG = 0, 7187500$; $H = \sqrt{FG}$
$H = 5,048065$; $lH = 0, 7031250$; $I = \sqrt{FH}$
$I = 4,958069$; $lI = 0, 6 53125$; $K = \sqrt{HI}$
$K = 5,001865$; $lK = 0, 6992187$; $L = \sqrt{LK}$
$L = 4,980416$; $lL = 0, 6972656$; $M = \sqrt{KL}$
$M = 4,991627$; $lM = 0, 6982421$; $N = \sqrt{KM}$

$N = 4,997242$; $lN = 0, 6987304$; $O = \sqrt{KN}$
$O = 5,000052$; $lO = 0, 6989745$; $P = \sqrt{NO}$
$P = 4,998647$; $lP = 0, 6988525$; $Q = \sqrt{OP}$
$Q = 4,999350$; $lQ = 0, 6989135$; $R = \sqrt{OQ}$
$R = 4,999701$; $lR = 0, 6989440$; $S = \sqrt{OR}$
$S = 4,999876$; $lS = 0, 6989592$; $T = \sqrt{OS}$
$T = 4,999963$; $lT = 0, 6989668$; $V = \sqrt{OT}$
$V = 5,000008$; $lV = 0, 6989707$; $W = \sqrt{TV}$
$W = 4,999984$; $lW = 0, 6989687$; $X = \sqrt{WV}$
$X = 4,999997$; $lX = 0, 6989697$; $Y = \sqrt{VX}$
$Y = 5,000003$; $lY = 0, 6989701$; $Z = \sqrt{XY}$
$Z = 5,000000$; $lZ = 0, 6989700$;

Ainſi, en prenant des moyennes proportionnelles, on eſt parvenu à trouver $Z = 5,000000$, à quoi répond le logarithme cherché 0,698970, en ſuppoſant la baſe logarithmique $= 10$. Par conſéquent $10^{\frac{69897}{100000}} = 5$ à-peu-près. C'eſt de cette maniere que Briggs & Ulacq ont calculé la table ordinaire des logarithmes, quoiqu'on ait imaginé depuis des méthodes plus expéditives pour les trouver.

Voici une traduction de cet algorithme en Python :

```python
from math import*

def BriggsLog(x,B,p):
    '''
    calcul le logarithme de x en base B  avec une précision de 10^(-p)
    '''
    A=1
    logA=0
    logB=1
    while abs(A-B)>10**(-p):
        if sqrt(A*B)<=x:
            A=sqrt(A*B)
            logA=(logA+logB)/2
        if sqrt(A*B)>x:
            B=sqrt(A*B)
            logB=(logA+logB)/2
        #print(A,logA,B,logB)
    return round(logA,p)
```

briggs_log.py

In[1]:BriggsLog(2,10,14)
Out[1]:0,30102999566398
In[2]:BriggsLog(2,exp(1),14)
Out[2]:0.69314718055994

II. Algorithme de Cordic

II. A. Historique

L'algorithme de CORDIC(COordinate Rotation DIgital Computing), inventé par Jack E. Volder en 1959 est un algorithme de calcul des fonctions trigonométriques et hyperboliques, notamment utilisé dans les **calculatrices**.
Il ressemble à des techniques qui avaient été décrites par Henry Briggs en 1624. Les calculatrices doivent calculer vite et avec peu de mémoire, l'algorithme de CORDIC est très performant pour gérer cette contrainte.

II. B. Principe et algorithme

II. B. 1. Construction de l'algorithme de Cordic sur $[1\ ;\ 10]$

Supposons la table des 11 algorithmes suivant (obtenus à la main) :

x	$\ln(x)$
10	2,302 585 092 994
2	0,693 147 180 560
1,1	0,095 310 179 804
1,01	0,009 950 330 853
1,001	0,000 999 500 333
1,000 1	0000 099 995 000
1,000 01	0,000 009 999 950
1,000 001	0,000 000 999 999
1,000 000 1	0,000 000 100 00
1,000 000 01	0,000 000 010 000
1,000 000 001	0,000 000 001 000
1,000 000 000 1	0,000 000 000 10

Soit un réel x de l'intervalle $[1\ ;\ 10]$.
$\frac{1}{10} \leqslant \frac{1}{x} \leqslant 1 \iff 1 \leqslant \frac{10}{x} \leqslant 10.$

1. $\exists \alpha_1 \in [\![0\ ;\ 3]\!]$ tel que $2^0 \leqslant 2^{\alpha_1} \leqslant \frac{10}{x} \leqslant 2^{\alpha_1+1} \leqslant 10 < 2^3$.
   On a alors $1 \leqslant \frac{10}{x \times 2^{\alpha_1}} \leqslant 2$

2. $\exists \alpha_2 \in [\![0\ ;\ 8]\!]$ tel que $2^0 \leqslant 1,1^{\alpha_2} \leqslant \frac{10}{x \times 2^{\alpha_1}} \leqslant 1,1^{\alpha_2+1} \leqslant 2 < 1,1^8$.
   On a alors $2^{\alpha_1} \times 1,1^{\alpha_2} \leqslant \frac{10}{x} \leqslant 2^{\alpha_1} \times 1,1^{\alpha_2+1}$

3. Ainsi de suite $\exists \alpha_i \in [\![0\ ;\ 10]\!]$ pour i variant de 1 à 10, tels que
   $$2^{\alpha_1} \times 1,1^{\alpha_2} \times \ldots + \left(1+10^{-10}\right)^{\alpha_{10}} \leqslant \frac{10}{x} \leqslant 2^{\alpha_1} \times 1,1^{\alpha_2} \times \ldots \left(1+10^{-10}\right)^{\alpha_{10}+1}$$

4. On pose $y = 2^{\alpha_1} \times 1,1^{\alpha_2} \times \ldots + \left(1+10^{-10}\right)^{\alpha_{10}}$, on a
   $\ln(y) = \alpha_1 \ln(2) + \alpha_2 \ln(1,1) + \ldots + \alpha_{10} \ln(1+10^{-10})$
   La fonction logarithme est croissante sur $\mathbb{R}_+^*$ donc :
   $y \leqslant \frac{10}{x} \leqslant y\,(1+10^{-n})$
   $\ln(y) \leqslant \ln(10) - \ln(x) \leqslant \ln(y) + \ln(1+10^{-n})$
   D'après le relation de Napier, $\ln(1+x) < x$:
   $\ln(y) \leqslant \ln(10) - \ln(x) \leqslant \ln(y) + 10^{-10}$
   $\ln(10) - \ln(y) - 10^{-10} \leqslant \ln(x) \leqslant \ln(10) - \ln(y)$.
   $\ln(10) - \ln(y)$ est donc une approximation de $\ln(x)$ à 10^{-10}

II. B. 2. Algorithme de Cordic pour $x \in [1\ ;\ 10]$

```
#algorithme de cordic

from math import*

#liste de la table initiale (prédéfinie) des logarithmes d'une calculatrice les valeurs sont données à 10^{-12}
L=[round(log(10),12),round(log(2),12),round(log(1.1),12),round(log(1.01),12),round(log(1.001),12),round(log
    (1.0001),12),round(log(1.00001),12),round(log(1.000001),12),round(log(1.0000001),12),round(log(1.00000001)
    ,12),round(log(1.000000001),12),round(log(1.0000000001),12)]

def cordic01(x):
    '''
    x est un nombre réel de l'intervalle [1 ; 10]

    renvoie une approximation à 10^{-10} du logarithme népérien de x connaissant une table initiale de 11
    logarithmes approchés à 10^{-12}.

    ln(x) approchée par ln(10)-ln(y) avec ln(y)=L[0]-(A[0]L[1]+A[1]L[1]+...+A[9])=ln(10)-(a0.ln(2)+a1.ln(1,1)
    +...+a9.ln(1,0000000001))
    et a_i tels que : (1+10^{-i})^a_i<10/(x*2^a_0*...*(1+10^{-i-1})^a_{i-1}<(1+10^{-i})^a_{i+1}
    '''
    global L

    y=L[0]
    A=[]                                #liste des coefficients alpha_i
    for i in range(0,10):
        z=1+10**(-i)
        a=0
        while x*z<=10:                  #permet d'enlever ln(y) au rang i autant de fois que nécessaires
            a=a+1
            x=x*z
            y=y-L[i+1]
            print("x=",x," ; y=",y," ; z=",z," ; xz=",x*z)
        A.append(a)
    print(A)
    return round(y,10)
```

cordic.py

II. B. 3. Algorithme de Cordic pour $x \in [10^{-100}\,;\,10^{100}]$

Soit x un réel de $10^{-100}\,;\,10^{100}]$ représentant l'intervalle $[0\,;\,+\infty[$ d'une calculatrice.

1. Pour $x>0$, $\exists n \in [\![-100\,;\,100]\!]$ tel que $10^n \leqslant x \leqslant 10^{n+1}$, on a $1 \leqslant x10^{-n} \leqslant 10$.

2. On pose $X = 10^{-n}x$ et $X \in [\![1\,;\,10]\!]$.
   Avec l'algorithme de Cordic sur $[1\,;\,10]$ on trouve une valeur approchée de $\ln(X)$, on en déduit une valeur approchée de $\ln(x)$ sachant que $\ln(x) = \ln(X) + n\ln(10)$.

On peut construire la fonction pour déterminer l'algorithme de tout réel x à la suite du programme précédent :

```
def cordic(x):
    '''
    x est un nombre réel de  [10^{-100},10^{100}] représentant ]0;+infty[

    renvoie une valeur approchée de ln(x)
    '''

    global L

    y=L[0]
    n=0
    while x>10:
        x=x/10
        n=n+1
    while x<1:
        x=10*x
        n=n-1
    return round(n*L[0]+cordic01(x),10)
```

cordic.py

III. Algorithme de Brouncker

III. A. Historique

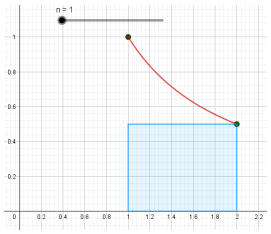
William Brouncker (anglais 1620-1684), était un linguiste et mathématicien.
Docteur de philosophie (université d'Oxford) en 1647, il est l'un des fondateurs et le premier président de la Royal Society, en 1660. En 1662, il devient chancelier de la reine Catherine, puis maître de l'hôpital Sainte-Catherine.

Ses travaux mathématiques portent en particulier sur la rectification (mesure des longueurs) de la parabole et de la cycloïde ainsi que sur la quadrature (mesure des aires) de l'hyperbole. Il est le premier, en Angleterre, à s'intéresser aux fractions continues généralisées, notamment $\frac{4}{\pi}$.

III. B. Principe et algorithme

Dans un repère orthogonal, soit l'hyperbole sur un intervalle $[a\,;\,b]$ avec $0 < a < b$.
La fonction f est la fonction inverse sur l'intervalle $[a\,;\,b]$.
Si on choisit $a = 1$ on détermine alors $\ln(b)$.

1. Pour $n = 1$, on considère l'aire $\mathscr{A}_1$ du rectangle formé par les points de coordonnées $(a\,;\,0)$, $(b\,;\,0)$, $(b\,;\,f(b))$, $(a\,;\,f(b))$.
   $\mathscr{A}_1 = (b-a)f(b) = \dfrac{b-a}{b} = 1 - \dfrac{a}{b}$.
   Pour $a = 1$, $\mathscr{A}_1 = 1 - \dfrac{1}{b}$.



2. Pour n entier supérieur ou égale à 2 et j variant de 1 à $n-1$; i variant de 0 à $2^j - 1$ avec un pas de 2, on considère les rectangles formés par les points de coordonnées :
   - $(a + hi\,;\,f(a + h(i+2)))$
   - $(a + h(i+1)\,;\,f(a + h(i+2)))$
   - $(a + h(i+1)\,;\,f(a + h(i+1)))$
   - $(a + hi, f(a + h(i+1)))$
   - Avec $h = \dfrac{b-a}{2^j}$

Figure pour $n = 2$

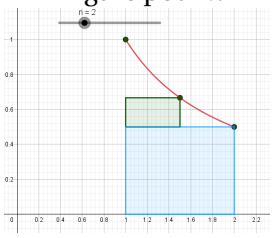


Figure pour $n = 3$

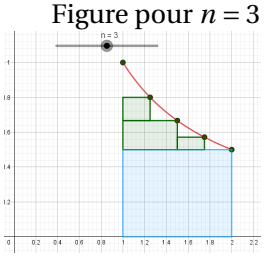
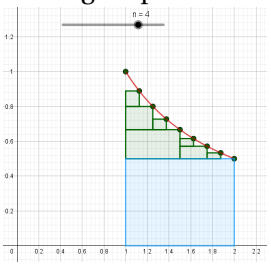


Figure pour $n = 4$



L'aire $\mathscr{A}_i$ d'un rectangle est
$\mathscr{A}_i = h(f(a + h(i+1)) - f(a + h(i+2)))$
$\mathscr{A}_i = h\left(\dfrac{1}{a + h(i+1)} - \dfrac{1}{a + h(i+2)}\right)$
$\mathscr{A}_i = \dfrac{h^2}{a^2 + a(2i+3) + h^2(i+1)(i+2)}$

j	i	i	i	i	h
$j=1$	$i=0$				$\frac{b-a}{2}$
$j=2$	$i=0$	$i=2$			$\frac{b-a}{4}$
$j=3$	$i=0$	$i=2$	$i=4$	$i=6$	$\frac{b-a}{8}$

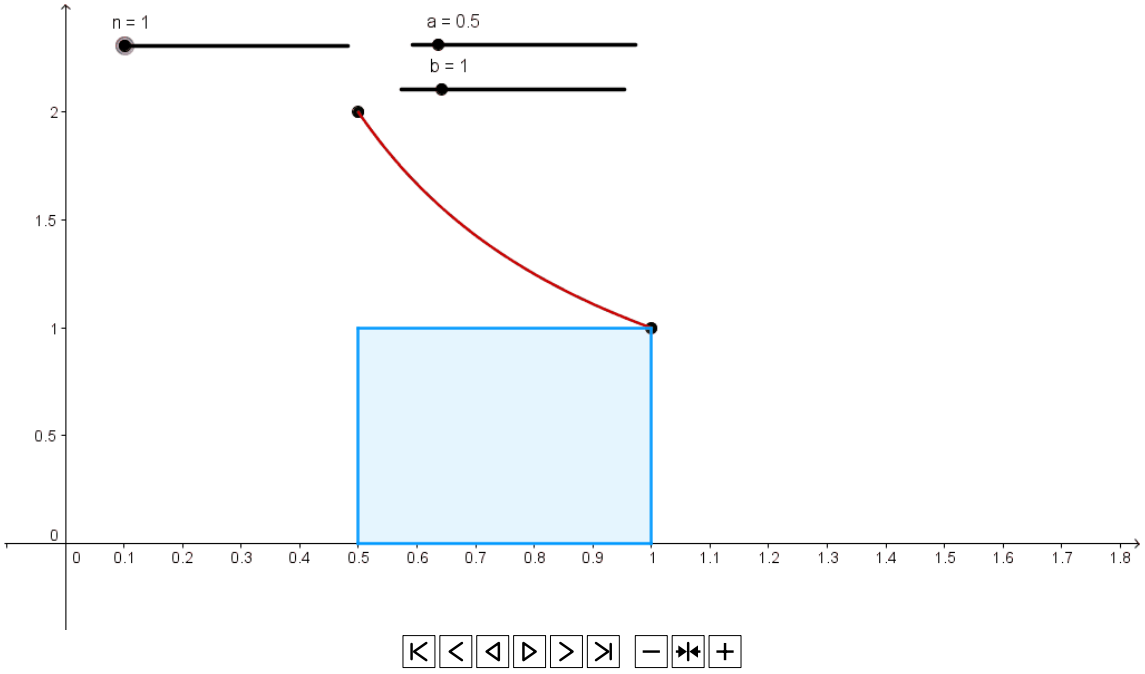
La somme des aires des rectangles converge vers $\ln(b)-\ln(a)$
Pour $a=1$ et $b=2$, on trouve

$h=\dfrac{1}{2^j}$.

$$\mathscr{A}_i=\frac{1}{2^j}\left(\frac{1}{1+\frac{1}{2^j}(i+1)}-\frac{1}{1+\frac{1}{2^j}(i+2)}\right)=\frac{1}{2^j}\left(\frac{2^j}{2^j+i+1}-\frac{2^j}{2^j+i+2}\right)=\frac{1}{(2^j+i+1)(2^j+i+2)}$$

j	$i\,;\mathscr{A}_i$	$i\,;\mathscr{A}_i$	$i\,;\mathscr{A}_i$	$i\,;\mathscr{A}_i$	h
$j=1$	$i=0\,;\dfrac{1}{3\times4}$				$\frac{1}{2}$
$j=2$	$i=0\,;\dfrac{1}{5\times6}$	$i=2\,;\dfrac{1}{7\times8}$			$\frac{1}{4}$
$j=3$	$i=0\,;\dfrac{1}{9\times10}$	$i=2\,;\dfrac{1}{11\times12}$	$i=4\,;\dfrac{1}{13\times14}$	$i=6\,;\dfrac{1}{15\times16}$	$\frac{1}{8}$

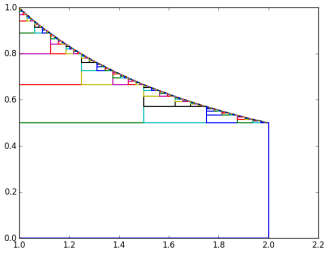
On admet que $\ln(2)=\sum_{k=0}^{\infty}\dfrac{1}{(2k+1)(2k+2)}$



```python
#algorithme de Broukner

import matplotlib.pyplot as plt


def f(x):
    return 1/x

def aire_rectangle(A,B,C,D):
    return (B[0]-A[0])*(D[1]-A[1])

def rectangle(A,B,C,D):
    return plt.plot([A[0],B[0],C[0],D[0],A[0]],[A[1],B[1],C[1],D[1],A[1]] ,linewidth=1.5)

def brouncker(a,b,n):
    A=[a,0]
    B=[b,0]
    C=[b,f(b)]
    D=[a,f(b)]
    rectangle(A,B,C,D)
    s=aire_rectangle(A,B,C,D)
    for j in range(1,n):
        h=(b-a)/(2**j)
        for i in range(0,int(2**(j)),2):
            A=[a+h*i,f(a+h*(i+2))]
            B=[a+h*(i+1),f(a+h*(i+2))]
            C=[a+h*(i+1),f(a+h*(i+1))]
            D=[a+h*i,f(a+h*(i+1))]
            rectangle(A,B,C,D)
            s=s+aire_rectangle(A,B,C,D)
    plt.show()
    return s
```

brouncker.py

In [1]: brouncker(1,2,10)
Out[1]: 0.6926591377284118



In [2]: brouncker(0.5,1,10)
Out[2]: 0.6926591377284118