# TP5 Correction

Clement LAROCHE

5 avril 2019

```r
library(ggplot2)
library(latex2exp)
```

## Un problème de résolution d'équation

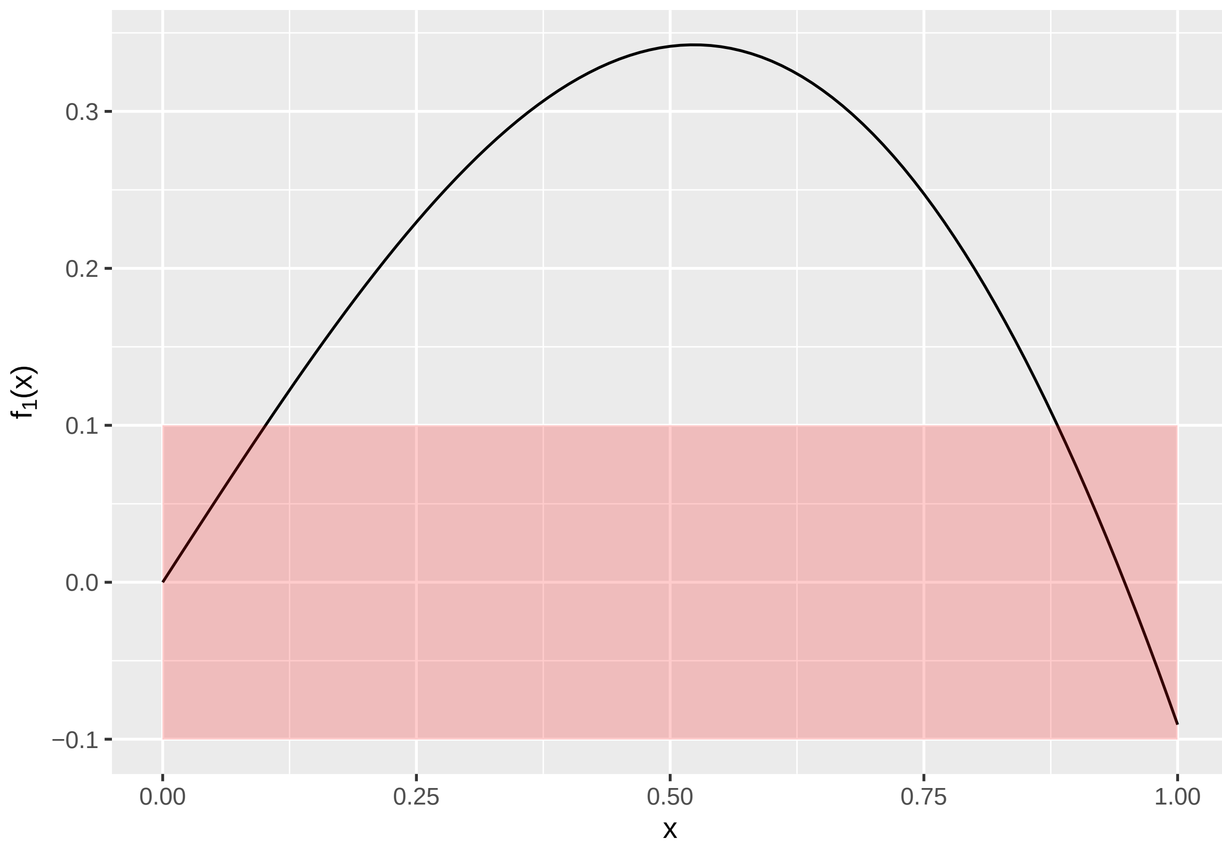
On calcule la dérivée de $f_1$ : $f_1'(x) = 2\cos(2x) - 1$.
A partir de cette donnée, on peut construire le tableau de variation suivant :

| | | | | | |
|---|---|---|---|---|---|
| $x$ | $0$ | | $\frac{\pi}{6}$ | | $1$ |
| $f$ | $0$ | $\nearrow$ | $0.34$ | $\searrow$ | $-0.09$ |
| $f'$ | | $+$ | $0$ | $-$ | |

(les valeurs sont calculées dans `R` et sont arrondies à la deuxième décimale)

Une solution triviale de cette équation est $x_0 = 0$. On peut la retrouver de manière théorique, c'est bien le $x_0$ demandé dans l'énoncé. Comme $f_1$ est strictement décroissante sur $[\frac{\pi}{6}, 1]$ et que $f_1(\frac{\pi}{6}) = \frac{\sqrt{3}}{2} - \frac{\pi}{6} \geq 0$ et $f_1(1) = \sin(2) - 1 \leq 0$, cela veut dire qu'il existe une solution moins triviale $x_1$ dans cet intervalle.
Pour obtenir une première approximation numérique de la valeur numérique de $x_1$, on peut raisonner de la manière suivante. Je souhaite récupérer les abscisses des points $x_i$ tels que $|f(x_i)| < c$ où $c$ représente un seuil que je choisis. Pour l'exemple, je choisis $c = 0.1$.

```r
# axe des abscisses
x <-seq(from = 0,to = 1,by = 0.01)
# application de la fonction à mon axe des abscisses
y <- sin(2*x)-x
# tracé de la fonction et de l'abscisse où la dérivée s'annule
ggplot()+geom_line(aes(x = x,y = y))+
  geom_rect(aes(xmin = 0,xmax = 1,
              ymin = -0.1,ymax = 0.1),alpha = 0.2,fill = "red")+
  xlab("x")+
  ylab(TeX("$f_1(x)$"))
```




Je récupère donc l'abscisse des points qui sont inclus dans le rectangle rouge sur le graphe ci-dessus. Pour le faire, j'utilise les commandes suivantes.

```
# on cherche quelles coordonnées du vecteur des abscisses sont telles que
# f(vecteur à cette coordonnée) soit proche de 0
# cela s'obtient avec la commande
#          which(abs(y[-1]) < 10^-1)

# on affiche donc les valeurs du vecteur des absicces concernées
res <- x[which(abs(y[-1]) < 10^-1)]
res
```

```
##  [1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.88 0.89 0.90 0.91 0.92
## [16] 0.93 0.94 0.95 0.96 0.97 0.98 0.99
```

Pour obtenir une valeur de $x_1$ à 0.1 près, on peut donc arrondir à la première décimale les valeurs du vecteur `res`.

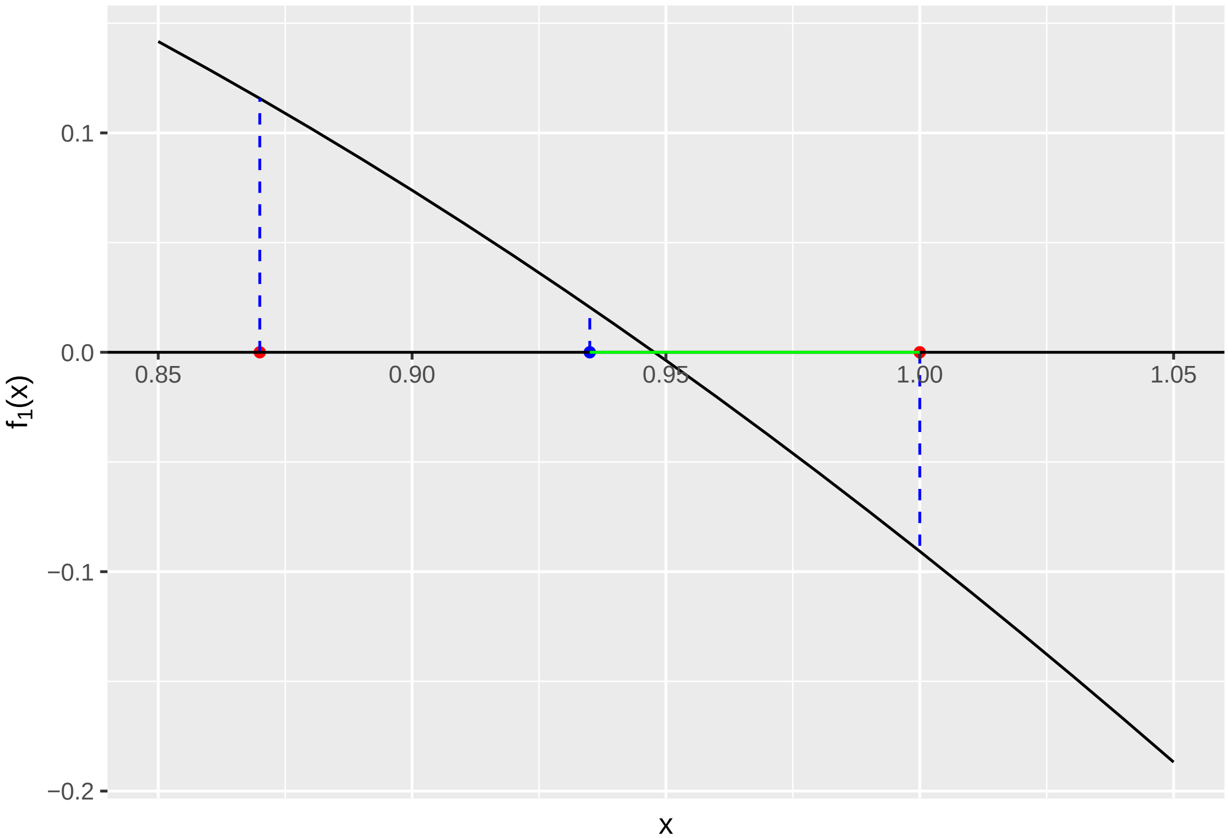
```
unique(round(res,1))
```

```
## [1] 0.0 0.1 0.9 1.0
```

On obtient les valeurs 0, 0.1, 0.9 et 1.
On se rend compte que les valeurs 0 et 0.1 sont les valeurs qui approchent $x_0$ à 0.1 près, elles ne nous intéressent donc pas. La valeur $x_1$ est donc comprise entre 0.9 et 1.

# Méthode d'approximation par dichotomie

**Illustration**




Description d'une itération de la méthode par dichotomie :

1. (en rouge) On part d'un intervalle initial défini par deux points.

2. (en bleu) On calcule le point milieu. On évalue ensuite notre fonction sur les bornes de l'intervalle et au point milieu.

3. (en vert) On actualise notre intervalle. On garde celui dont les bornes ont des images par $f$ de signes opposés.

**Partie code :**

(je ne pars pas de la valeur approchée à 0.1 près de $x_1$, je choisis des valeurs pour illustrer l'encadré de l'énoncé)

```
# définition de la fonction
f1 <- function(x)
{
  sin(2*x)-x
}
# je pars d'un intervalle initial
a = 0.5
b = 1.5
# j'initialise un compteur
n <- 0
```

```r
# je pose un critère d'arrêt
crit = TRUE
while (crit == TRUE)
{
  # calcul du point milieu
  pt <- (a+b)/2
  # actualisation des bornes
  if(f1(a)*f1(pt) < 0)
  {
    b <- pt
  }
  else if(f1(b)*f1(pt) < 0)
  {
    a <- pt
  }
  # vérification du critère d'arrêt
  if(abs(b-a)<10^-10)
  {
    crit <- FALSE
  }
  # mise à jour du compteur
  n <- n+1
}
# affichage d'une des bornes
a
```

```
## [1] 0.9477471
```

```r
# affichage du nombre d'itération nécessaire
n
```

```
## [1] 34
```

Pour retrouver la formule de l'énoncé, on se sert du critère d'arrêt. On nous dit qu'à la $n$-ième itération de l'algorithme, la largeur de l'intervalle est de $2^{-n}$ (si la largeur de notre intervalle initial est de 1 ce qui est mon cas). Pour avoir le $n$ théorique adéquat pour atteindre la précision $10^{-}p$ souhaitée, il suffit juste de trouver le $n$ tel que $2^{-n} = 10^{-p}$. On obtient que :

$$
\begin{aligned}
2^{-n} = 10^{-p} &\iff -n\log(2) = -p\log(p) \\
&\iff n = p\frac{\log(10)}{\log(2)}
\end{aligned}
$$

C'est exactement la formule de l'encadré. Numériquement, on calcule :

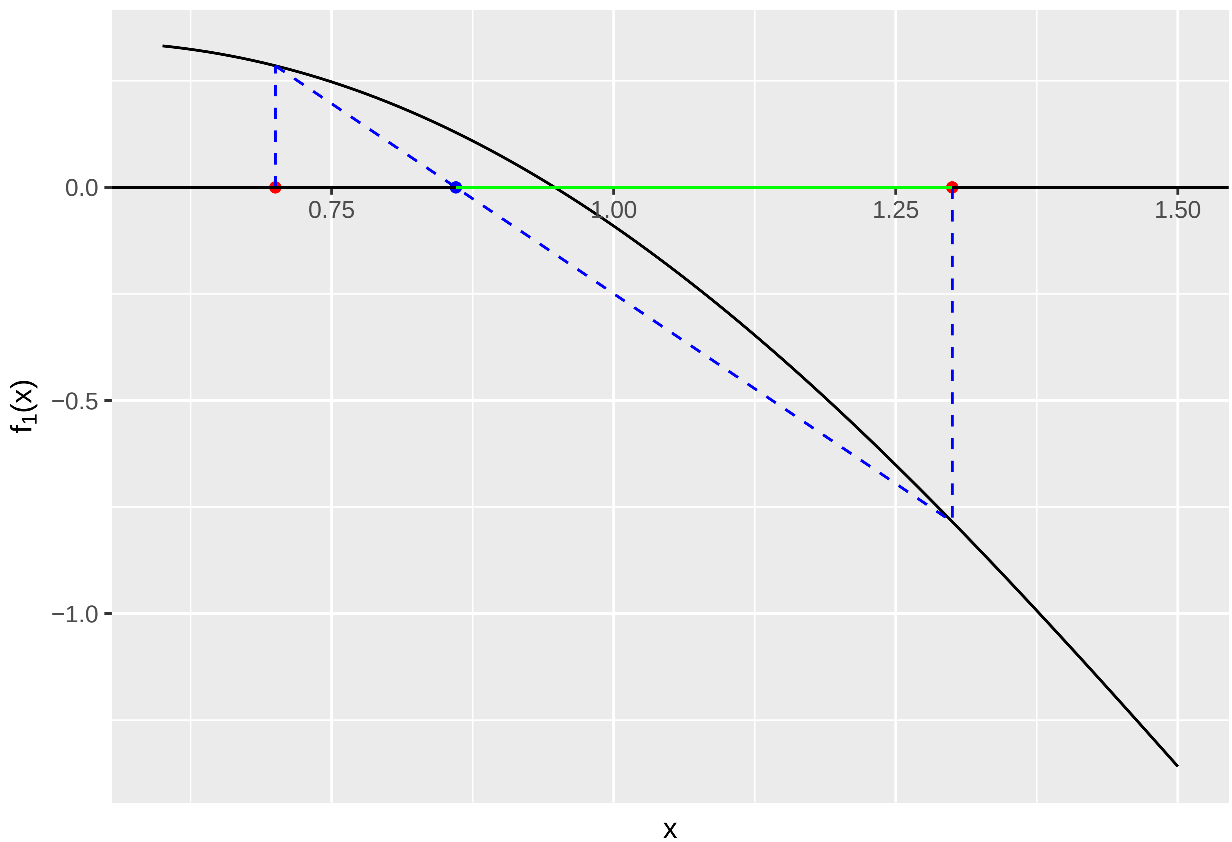
```r
10*log(10)/log(2)
```

```
## [1] 33.21928
```

On est satisfait de ce résultat.

## Méthode de la sécante

#### Illustration




Description d'une itération de la méthode de la sécante :

1. (en rouge) On part d'un intervalle initial $[u_0, u_1]$ défini par deux points $u_0 = 0.7$ et $u_1 = 1.3$.

2. (en bleu) On calcule l'équation de la droite passant par l'image des deux points de l'intervalle. On la trace et on cherche $u_2$ le point d'intersection de cette droite avec l'axe des abscisses.

3. (en vert) J'actualise mes bornes de mon intervalle $[u_0, u_1]$ qui devient l'intervalle $[u_2, u_1]$.

**Attenzione !!!** Cette méthode ne converge pas systématiquement, il faut que votre intervalle initial soit suffisament près du 0 de votre fonction (ici $x_1$).

**Partie code :**

L'équation de cette droite s'obtient assez simplement si on effectue le tracé de cette fameuse sécante. Cette équation s'écrit (dans notre cas particulier pour $f_1$) :

$$y = \frac{f_1(u_1) - f_1(u_0)}{u_1 - u_0} x + f_1(u_1) - u_1 \frac{f_1(u_1) - f_1(u_0)}{u_1 - u_0}$$

On sait que $u_2$ est la valeur où $y = 0$, donc on a que :

$$\frac{f_1(u_1) - f_1(u_0)}{u_1 - u_0} u_2 + f_1(u_1) - u_1 \frac{f_1(u_1) - f_1(u_0)}{u_1 - u_0} = 0 \iff \frac{f_1(u_1) - f_1(u_0)}{u_1 - u_0} u_2 = u_1 \frac{f_1(u_1) - f_1(u_0)}{u_1 - u_0} - f(u_1) \tag{1}$$

$$\iff u_2 = u_1 - f_1(u_1) \frac{u_1 - u_0}{f_1(u_1) - f_1(u_0)} \tag{2}$$

Mettons la méthode en application. On suit les indications de l'énoncé, je pars de la valeur à 0.1 près de $x_1$.

```
# si elle n'est pas en mémoire n'oubliez
# pas de définir votre fonction
f1 <- function(x)
{
  sin(2*x)-x
}

# je définis mon intervalle initial
a = 0.9
b = 1
# j'initialise mon compteur
n <- 0
# critère d'arrêt
crit = TRUE
while (crit == TRUE)
{
  # définition point sécant
  pt <- b - f1(b)*(b-a)/(f1(b)-f1(a))
  # actualisation de mes bornes
  a <- b
  b <- pt
  # définiton du critère d'arrêt
  stop <- abs(a-b)
  if(stop<10^-10)
  {
    crit <- FALSE
  }
  # actualisation du compteur
  n <- n+1
}
# valeur d'une des bornes
a
```

```
## [1] 0.9477471
```

```
# nombre d'itérations
n
```

```
## [1] 6
```

On obtient 6 itérations à peu près. On va illustrer théoriquement ce résultat. Pour retrouver la formule de l'énoncé, rien de plus simple. On vous donne une majoration de la précision de la méthode (**qui marche sous certaines conditions**) qui vaut $\frac{1}{C}\exp(-K\phi^n)$. Il suffit donc de résoudre (comme d'habitude) l'égalité de cette borne avec $10^{-p}$ et isoler $n$ pour réobtenir la formule de l'encadré. On a :

$$
\begin{aligned}
\frac{1}{C}\exp(-K\phi^n) = 10^{-p} &\iff -\log(C) - K\phi^n = -p\log(10) \\
&\iff \phi^n = \frac{p\log(10) - \log(C)}{K} \\
&\iff n = \log\left(\frac{p\log(10) - \log(C)}{K}\right)/\log(\phi)
\end{aligned}
$$

Ce qui est exactement la formule de l'encadré. Illustrons le numériquement.

```
# nombre d'or
psi <- (1+sqrt(5))/2
```

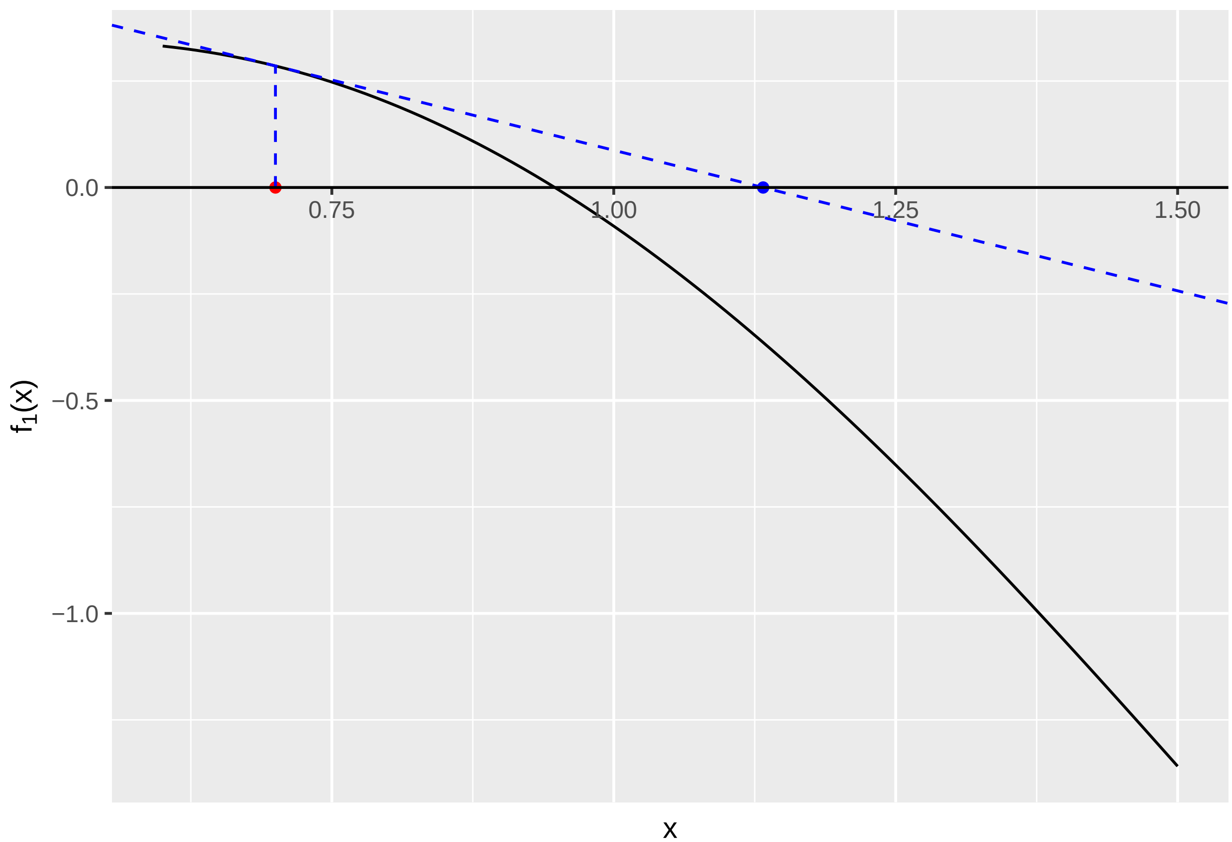
```
# inf de la valeur absolue
# de la dérivée sur l'intervalle
m1 <- min(abs(2*cos(2*seq(from = 0.9,to = 1,by = 0.0001))-1))
# le sup de la valeur absolue de la dérivée
# seconde sur l'intervalle
m2 <- max(abs(-4*sin(2*seq(from = 0.9,to = 1,by = 0.0001))))
# les deux constantes de l'encadré
C <- 0.5*m2/m1
K <- log(C*abs(0.9-pt))/psi-log(C*abs(1-pt))
# le résultat
log((10*log(10)-log(C))/K)/log(psi)
```

```
## [1] 6.576147
```

On est satisfait de ce résultat.

## Méthode de Newton Raphson

### Illustration




Description d'une itération de la méthode de Newton-Raphson :

1. (en rouge) On part **d'un point initial** $u_0$ (et non plus d'un intervalle).

2. (en bleu) On calcule l'équation de la tangente à notre fonction passant par l'image de notre point. On la trace et on cherche $u_1$ le point d'intersection de cette tangente avec l'axe des abscisses.

### Partie code

On met en application Newton-Raphson pour notre fonction. On part de la solution à 0.1 près de $x_1$.

```
# définition de la fonction
f1 <- function(x)
{
  sin(2*x)-x
}
# défintion de la dérivée
fprime1 <- function(x)
{
  2*cos(2*x)-1
}
# point initial
u <- 1
# critère d'arrêt
crit <- TRUE
# initialisation du compteur
n <- 0
while(crit == TRUE)
{
  # calcul du nouveau point
  pt <- u - f1(u)/fprime1(u)
  # evaluation du critère d'arrêt
  if(abs(u-pt) < 10^-10)
  {
    crit <- FALSE
  }
  # mise jour du point
  u <- pt
  # mise à jour du compteur
  n <- n+1
}
# valeur du point
u
```

```
## [1] 0.9477471
```

```
# nombre d'itérations
n
```

```
## [1] 4
```

Pour retrouver la formule de l'encadré, on commence à s'en douter, il va falloir résoudre l'équation suivante :

$$
\begin{aligned}
(C|v_0 - x_1|)^{2^n}/C = 10^{-p} &\iff 2^n \log(C|v_0 - x_1|) - \log(C) = -p\log(10) \\
&\iff n\log(2) + \log(\log(C|v_0 - x_1|)) = \log(\log(C) - p\log(10)) \\
&\iff n = \log\left(\frac{\log(C) - p\log(10)}{\log(C|v_0 - x_1|)}\right)/\log(2)
\end{aligned}
$$

Ce qui est à une coquille près ($v_0$ au lieu de $v_1$) le résultat demandé. Maintenant retrouvons le numériquement.

```
m1 <- min(abs(2*cos(2*seq(from = 0.9,to = 1,by = 0.001))-1))
m2 <- max(abs(-4*sin(2*seq(from = 0.9,to = 1,by = 0.001))))
C <- 0.5*m2/m1
log((log(C)-10*log(10))/log(C*abs(1-u)))/log(2)
```

```
## [1] 3.095552
```

Ce qui est un résultat assez représentatif du `n` obtenu par notre méthode.