

TP1 Correction

Clément LAROCHE

17 décembre 2018

Exercice 1

Démonstration de la convergence de la série de terme $u_n = \frac{1}{n^2}$.

La série $\sum_{n=1}^{+\infty} \frac{1}{n^2}$ converge si et seulement si la série $\sum_{n=2}^{+\infty} \frac{1}{n^2}$ converge. Le fait de retirer le premier terme de la série n'a pas d'incidence sur la convergence.

On établit le postulat suivant, pour tout $n \geq 2$:

$$\frac{1}{n^2} < \frac{1}{n^2 - n}$$

En effet, le dénominateur de la fraction de droite de l'inégalité est toujours plus petit que celui de gauche pour les valeurs de n considérées.

Si nous réussissons à prouver que la série de terme $u_n = \frac{1}{n^2 - n}$ converge, nous réussirons alors à prouver la convergence de la série qui nous intéresse par test de comparaison.

Observons maintenant que la série de terme $u_n = \frac{1}{n^2 - n}$ est télescopique. En effet pour $N > 0$, on a :

$$\begin{aligned} \sum_{n=2}^N \frac{1}{n^2 - n} &= \sum_{n=2}^N \left(\frac{1}{n-1} - \frac{1}{n} \right) \\ &= \left(1 - \frac{1}{2} \right) + \left(\frac{1}{2} - \frac{1}{3} \right) + \dots + \left(\frac{1}{N-1} - \frac{1}{N} \right) \\ &= 1 - \frac{1}{N} \end{aligned}$$

On conclue donc que, quand $N \rightarrow +\infty$, la série $\sum_{n=1}^{+\infty} \frac{1}{n^2 - n}$ tend vers 1. Elle converge donc. Cela implique que la série de terme général $u_n = \frac{1}{n^2}$ converge également.

De plus, notons que :

$$\begin{aligned} 0 &< \sum_{n=2}^N \frac{1}{n^2} < \sum_{n=2}^N \frac{1}{n^2 - n} = 1 - \frac{1}{N} \text{ ce qui donne quand } N \text{ tend vers l'infini} \\ 0 &< \sum_{n=2}^{+\infty} \frac{1}{n^2} < 1 \text{ en ajoutant le premier terme de la série, on trouve} \\ 1 &< \sum_{n=1}^{+\infty} \frac{1}{n^2} < 2 \end{aligned}$$

On passe maintenant au code :

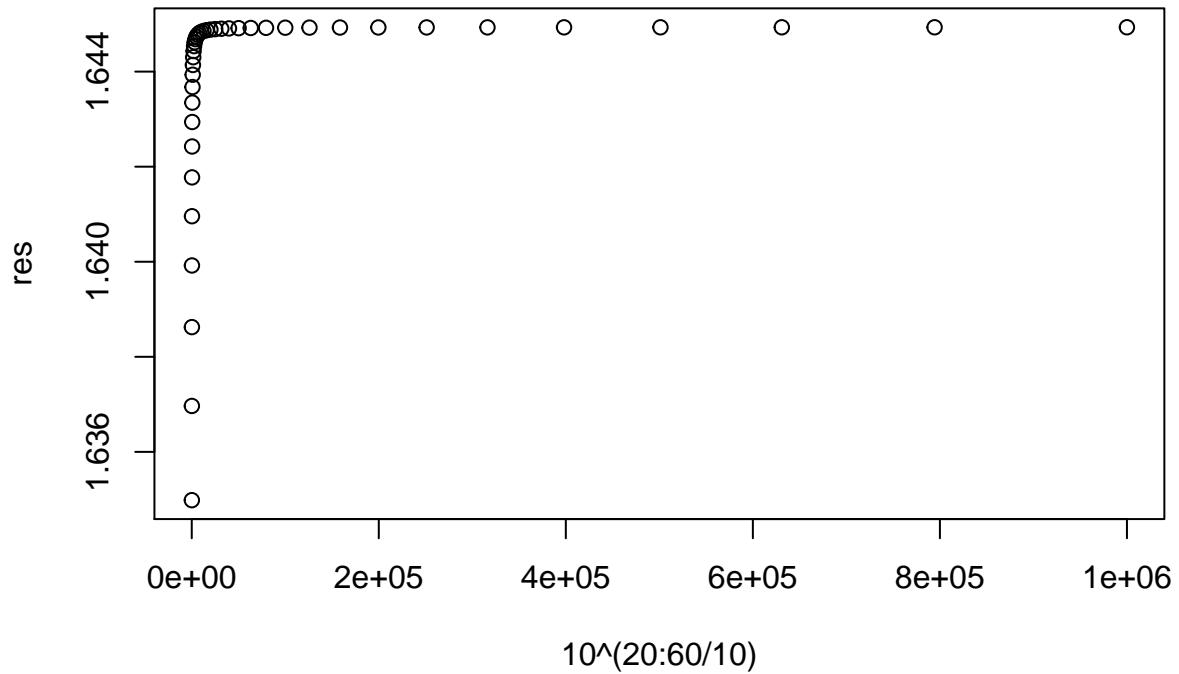
```
# Construction de I_n pour n = 10^6
INum <- rep(x = 1, 10^6)
IDen <- (1:10^6)^2
In <- INum/IDen
# Obtention du résultat
sum(In)
```

```

## [1] 1.644933
# On nettoie son espace de travail
rm(INum, IDen)

# Tracé pour les différentes valeurs de r
## construction avec une boucle des différentes valeurs de In
res <- c()
for(r in 20:60/10)
{
  n = 10^r
  INum <- rep(x = 1,n)
  IDen <- (1:n)^2
  In <- INum/IDen
  res <- c(res,sum(In))
}
## execution du tracé
plot(10^(20:60/10),res)

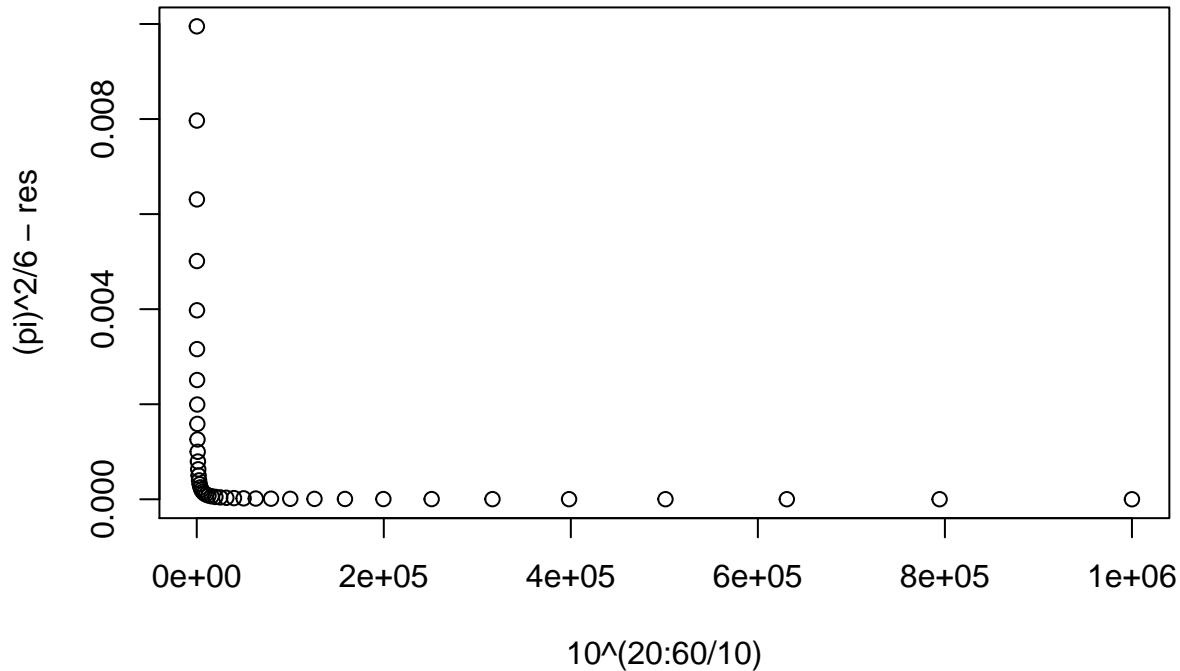
```



```

# tracé du reste de cette série
plot(10^(20:60/10),(pi)^2/6 - res)

```



On se reporte au cours. On y trouve une formule pour la majoration de l'erreur d'approximation. En l'appliquant, on obtient :

$$\begin{aligned}|R_n| &\leq \int_n^{+\infty} \frac{1}{x^2} dx \\&\leq \left[-\frac{1}{x}\right]_n^{+\infty} \\&\leq \frac{1}{n}\end{aligned}$$

L'ordre de grandeur de cette approximation est donc n^{-1} .

Exercice 2

```
## on efface les objets présents dans l'environnement
## de travail ils ne nous serviront plus

rm(list = ls(all = TRUE))

# Création du vecteur x

# x <- readline("Entrez le point d'initialisation : ")
# si l'on veut lui rentrer x nous même
# x <- as.numeric(x)
# on convertit la sortie de la fonction readline, qui
# stocke par défaut les valeurs sous forme de chaîne de
```

```

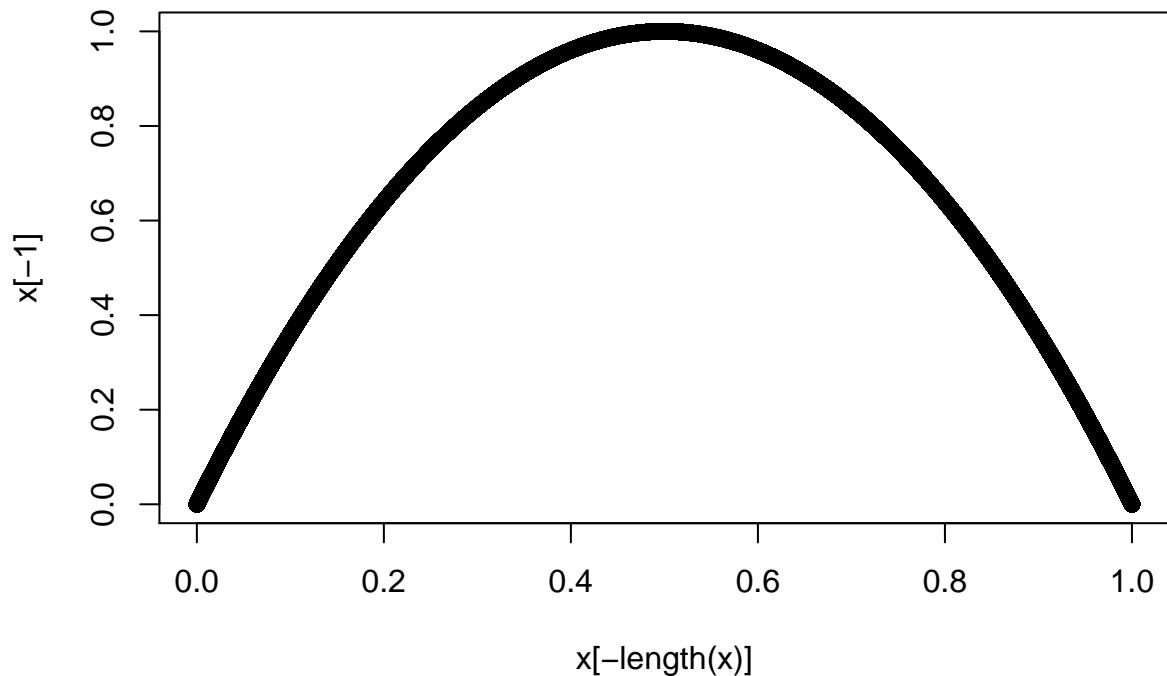
# caractères, en valeur numérique.

# Pour l'exemple je prends x_0 = 0.666
x <- 0.666
# notez que x est ici un scalaire (pour l'instant)
for (n in 1:10^5)
{
  x[n+1] <- 4*x[n]*(1-x[n])
  # Dans cette boucle, je transforme x, qui était un
  # scalaire, en vecteur. R comprend implicitement que je
  # demande à rajouter des coordonnées dans l'objet x.
}

# tracé des points consécutifs

plot(x[-length(x)],x[-1])

```



```

# on se rappelle que la commande x[-num coordonnées]
# retourne le vecteur x sans les coordonnées stockées dans
# num coordonnées. Dans la fonction de tracé plot, je demande
# ici à tracer en abscisse le vecteur (x_1,x_2,...,x_(n-1))
# et en ordonnée le vecteur (x_2,x_3,...,x_n). Cela me permet
# donc de tracer les couples (x_1,x_2), (x_2,x_3), ...,
# (x_(n-1),x_n)

```

Exercice 3

Commençons par supposer que le nombre N n'est pas premier. Il admet donc des diviseurs autres que un et lui-même. Supposons par l'absurde qu'il n'admet aucun diviseur d inférieur à \sqrt{N} (qui n'est pas forcément entière si N n'est pas un carré). Par définition on peut écrire que $N = d \times q$ avec d et q deux entiers naturels. On écarte le cas où $N = 0$ ou 1 . On observe le fait suivant :

$$N = d \times q$$

$N = \sqrt{N} \times \sqrt{N}$ Par définition de la fonction racine.

Comme N est strictement supérieur à 1 , il vient que :

$$q = \frac{N}{d} \text{ et } \sqrt{N} = \frac{N}{\sqrt{N}}$$

Or on sait que : $\sqrt{N} < d \implies \sqrt{N} > q$

Ce qui n'est pas possible car nous avions supposé par l'absurde que N n'admettait pas de diviseur inférieur à sa racine. Le raisonnement par l'absurde nous amène à conclure que si N admet des diviseurs autres que un ou lui-même, au moins l'un d'entre eux sera inférieur à \sqrt{N} .

On conclue donc que pour vérifier si un nombre est premier, il suffit de vérifier si les nombres inférieurs ou égaux à sa racine sont des diviseurs ou non.

```
# Code de l'exercice 3

rm(list = ls(all = TRUE))

# x <- readline("Entrez un nombre entier : ")
# si l'on veut lui rentrer x nous même
# x <- as.numeric(x)
# on convertit la sortie de la fonction readline, qui
# stocke par défaut les valeurs sous forme de chaîne de
# caractères, en valeur numérique.

# exemple
x <- 17
res <- rep(x,floor(sqrt(x))-1)
# je crée ici un vecteur qui contient la valeur x que je
# répète partie entière de racine(x)-1 fois.

if(!0 %in% (res%%(2:floor(sqrt(x))))) {
  # je regarde dans cette condition if le reste division
  # euclidienne entre x et 2,3,4,...,partie entière de racine
  # de x (cf. "res%%(2:floor(sqrt(x)))"). Je cherche si 0 est
  # dans ce vecteur ou pas (cf. "!0 %in%")
  print("Le nombre que vous avez rentré est premier.")
  # J'en déduis que si 0 n'est PAS dans le vecteur, mon
  # nombre x est bien premier
} else {
  print("Le nombre que vous avez rentré n'est pas premier.")
  # si 0 est dans mon vecteur, il n'est donc pas premier
}

## [1] "Le nombre que vous avez rentré est premier."
```

Exercice 4

```
# Code de l'exercice 4

# on crée le vecteur des nombres pairs entre 1 et 10^5
test <- (3:10^2)[3:10^2 %% 2 == 0]

# ATTENTION TEMPS D'EXECUTION LONG

# Goldbach <- function(x)
# {
#   p1 <- 2:(x-2)
#   p2 <- x-(2:(x-2))
#   pos1 <- c()
#   pos2 <- c()
#   for(i in 1:length(p1))
#   {
#     res1 <- rep(p1[i], floor(sqrt(p1[i]))-1)
#     res2 <- rep(p2[i], floor(sqrt(p2[i]))-1)
#     if(!0 %in% (res1%%(2:floor(sqrt(p1[i])))))
#     {
#       pos1 <- c(pos1, i)
#     }
#     if(!0 %in% (res2%%(2:floor(sqrt(p2[i])))))
#     {
#       pos2 <- c(pos2, i)
#     }
#   }
#   pos <- which(pos1 %in% pos2)[1]
#   return(c(p1[pos1[pos]], p2[pos1[pos]]))
# }
#
# apply(X = as.matrix(res), MARGIN = 1, FUN = "Goldbach")

# POUR LE DIMINUER ON PEUT PROCÉDER AINSI

# fonction qui confirme si un nombre est premier ou pas
premieroupas <- function(x)
{
  res <- rep(x, floor(sqrt(x))-1)
  if(!0 %in% (res %% (2:floor(sqrt(x)))))
  {
    res <- TRUE
  }else
  {
    res <- FALSE
  }
  return(res)
} # cette fonction reprend le code de l'exercice 3, j'ai
# juste modifié les sorties pour retourner TRUE si le nombre
# est premier et FALSE sinon.

# fonction qui code la conjecture de Goldbach en utilisant
```

```

# dans son code la fonction précédente
Goldbach <- function(x)
{
  p1 <- 2:(x-2)
  # je crée ici le vecteur 2,3,...,(x-2)
  p2 <- x-(2:(x-2))
  # je crée ici le vecteur (x-2),(x-3),...,2
  # Notez que si vous sommez coordonnée par coordonnée p1
  # et p2 vous réobtenez x à chaque fois.
  # Je veux connaître les indices des coordonnées de p1 qui
  # sont premières (de même pour p2). Je vais donc
  # appliquer ma fonction premieroupas sur chacune des
  # coordonnées des deux vecteurs.
  pos1 <- apply(X = as.matrix(p1), MARGIN = 1, FUN = "premieroupas")
  pos2 <- apply(X = as.matrix(p2), MARGIN = 1, FUN = "premieroupas")
  # j'utilise la fonction apply pour le faire. Je
  # transforme mon vecteur en une matrice colonne (1
  # colonne et le nombre de ligne correspond à la taille de
  # mon vecteur). Je suis obligé car la fonction apply ne
  # s'applique que sur des matrices. Une fois que j'ai ma
  # matrice, l'argument MARGIN de ma fonction me permet
  # d'appliquer ma fonction soit sur les lignes (MARGIN =
  # 1) soit sur les colonnes (MARGIN = 2). Ici, je veux
  # appliquer sur chaque ligne la fonction premieroupas.
  # Enfin, l'argument FUN me permet de spécifier la
  # fonction que je veux appliquer, ici c'est la fonction
  # premieroupas. pos1 (respectivement pos2) sont donc des
  # vecteurs de TRUE et de FALSE avec des TRUE aux
  # positions où p1 (respectivement p2) contenaient des
  # nombres premiers.
  pos1 <- which(pos1 == TRUE)
  pos2 <- which(pos2 == TRUE)
  # je récupère donc les indices de ces positions le
  # vecteur pos1 (respectivement pos2).
  pos <- which(pos1 %in% pos2)[1]
  # je stocke dans pos le premier cas (cf. "[1]" à la fin
  # de la ligne) où je trouve une
  # position dans pos1 qui est également dans pos2
  # j'ai donc une position où je sais que p1 et p2
  # contiennent des nombres premiers tous les deux. Donc si
  # je somme p1 et p2 à cette position, j'ai bien deux
  # nombres premiers dont la somme me redonne x (j'ai
  # construit p1 et p2 pour que ce soit le cas).
  return(c(p1[pos1[pos]], p2[pos2[pos]]))
}

resfin <- apply(X = as.matrix(test), MARGIN = 1, FUN = "Goldbach")
# présentation des résultats
resfin <- as.data.frame(resfin)
colnames(resfin) <- as.character(2+2*1:ncol(resfin))
row.names(resfin) <- c("1er terme de la somme", "2ème terme de la somme")
resfin

## 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42

```

```

## 1er terme de la somme 2 3 3 3 5 3 3 5 3 3 5 3 5 7 3 3 5 7 3 5
## 2ème terme de la somme 2 3 5 7 7 11 13 13 17 19 19 23 23 23 29 31 31 31 31 37 37
## 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80
## 1er terme de la somme 3 3 5 3 5 7 3 5 7 3 3 5 7 3 5 3 3 5 7
## 2ème terme de la somme 41 43 43 47 47 47 53 53 53 59 61 61 61 67 67 71 73 73 73
## 82 84 86 88 90 92 94 96 98 100
## 1er terme de la somme 3 5 3 5 7 3 5 7 19 3
## 2ème terme de la somme 79 79 83 83 83 89 89 89 89 79 97

```

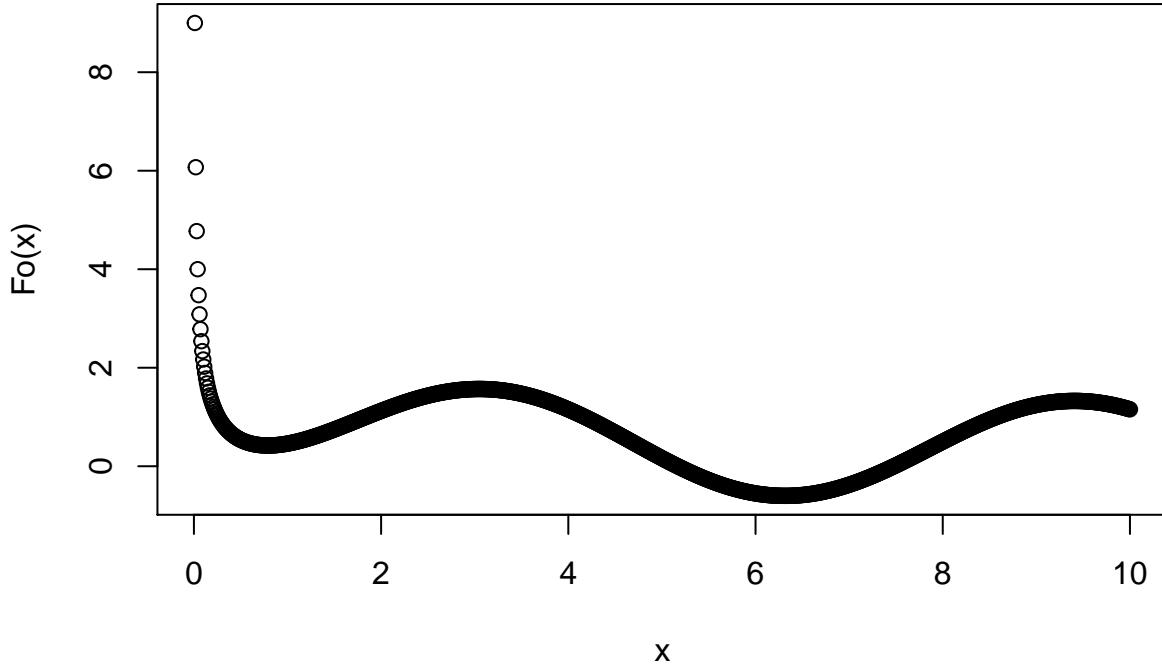
Exercice 5

```

Fo <- function(x)
{
  res <- 1/sqrt(x)-cos(x)
  return(res)
}

pas <- 0.01
x <- seq(from = 0.01,to = 10,by = pas)
plot(x,Fo(x))

```



On va trouver deux fonctions qui encadrent F pour pouvoir mener une étude de fonction. On remarque que, $\forall x \in]0, 10]$:

$$\frac{1}{\sqrt{x}} - 1 \leq \frac{1}{\sqrt{x}} - \cos(x) \leq \frac{1}{\sqrt{x}} + 1$$

Etudions donc la borne inférieure que l'on notera $g(x)$. On a que :

- $g(x)$ est continue sur $[0, 10]$
- $g'(x) = -\frac{1}{2x\sqrt{x}}$, elle strictement négative sur $[0, 10]$, donc g est strictement décroissante sur cet intervalle
- $g(0^+) = +\infty$, $g(10) = \frac{1}{\sqrt{10}} - 1 < 0$ et $g(1) = 0$

On en déduit de cette étude que F est positive sur $[0, 1]$, il n'y a donc pas de valeurs x_0 sur cet intervalle. En revanche, il y en a peut être sur l'intervalle $[1, 10]$. On travaillera à partir de maintenant sur cet intervalle. Reprenons maintenant l'équation $F(x) = 0$. Celle ci implique l'égalité suivante :

$$\frac{1}{\sqrt{x}} = \cos(x)$$

Or $\frac{1}{\sqrt{x}}$ est strictement positive sur l'intervalle d'étude (pour rappel $[1, 10]$). On sait que la fonction cosinus est positive sur les intervalles du type $[-\frac{\pi}{2} + 2k\pi, \frac{\pi}{2} + 2k\pi]$ avec $k \in \mathbb{Z}$. On sait donc que les 0 de la fonction F se trouvent sur l'intervalle $[-\frac{\pi}{2} + 2k\pi, \frac{\pi}{2} + 2k\pi] \cap [1, 10]$ qui n'est autre que $[1, \frac{\pi}{2}] \cup [\frac{3\pi}{2}, \frac{5\pi}{2}]$.

Etudions F' sur $[1, \frac{\pi}{2}]$. Calculons la formule de F' :

$$F'(x) = \sin(x) - \frac{1}{2x\sqrt{x}}$$

On a donc que sur cet intervalle :

$$\sin(1) - \frac{1}{2} \leq F'(x) \leq 1 - \frac{1}{2}(\frac{\pi}{2})^{-\frac{3}{2}}$$

On sait donc que F' est strictement positive sur cet intervalle (utiliser R comme calculatrice pour calculer $\sin(1)$). Donc F est strictement sur cet intervalle. On calcule maintenant $F(1)$.

Fo(1)

```
## [1] 0.4596977
```

On conclue qu'il n'y a pas de solutions x_0 sur l'intervalle $[1, \frac{\pi}{2}]$.

Tournons nous maintenant vers le dernier intervalle restant : $[\frac{3\pi}{2}, \frac{5\pi}{2}]$. On a l'inégalité suivante :

$$-1 - \frac{1}{2}(\frac{3\pi}{2})^{-\frac{3}{2}} < F'(x) < 1 - \frac{1}{2}(\frac{5\pi}{2})^{-\frac{3}{2}}$$

La borne inférieure est négative et la borne supérieure est positive. F' est continue sur cet intervalle, il existe donc un point où F' s'annule. Cherchons maintenant les 0 de F' . On a l'implication suivante :

$$F'(x) = 0 \implies \sin(x) = \frac{1}{2x\sqrt{x}}$$

Or $\frac{1}{2x\sqrt{x}} > 0$ sur notre intervalle d'étude. Et $\sin(x) > 0$ uniquement sur $[2\pi, \frac{5\pi}{2}]$. Donc les potentiels 0 de F' se trouvent sur cet intervalle. Calculons maintenant la dérivé seconde de F sur cet intervalle. On a :

$$F''(x) = \cos(x) + \frac{3}{4}x^{-\frac{5}{2}}$$

Sur l'intervalle $[\frac{3\pi}{2}, \frac{5\pi}{2}]$, F'' est strictement positive. Donc F' est strictement croissante. Et on a $F'(2\pi) < 0$ et $F'(\frac{5\pi}{2}) > 0$. Cela nous assure qu'il existe un seul 0 de la fonction F' sur l'intervalle considéré. Cela implique également que F' est négative sur un certain intervalle $[\frac{3\pi}{2}, z]$ et positive sur $[z, \frac{5\pi}{2}]$. On peut donc en déduire les variations de F . Regardons quelques valeurs numériques dès à présent, on a :

Fo(3/2*pi)

```
## [1] 0.4606589
```

Fo(2*pi)

```
## [1] -0.6010577
```

```
Fo(5/2*pi)
```

```
## [1] 0.3568248
```

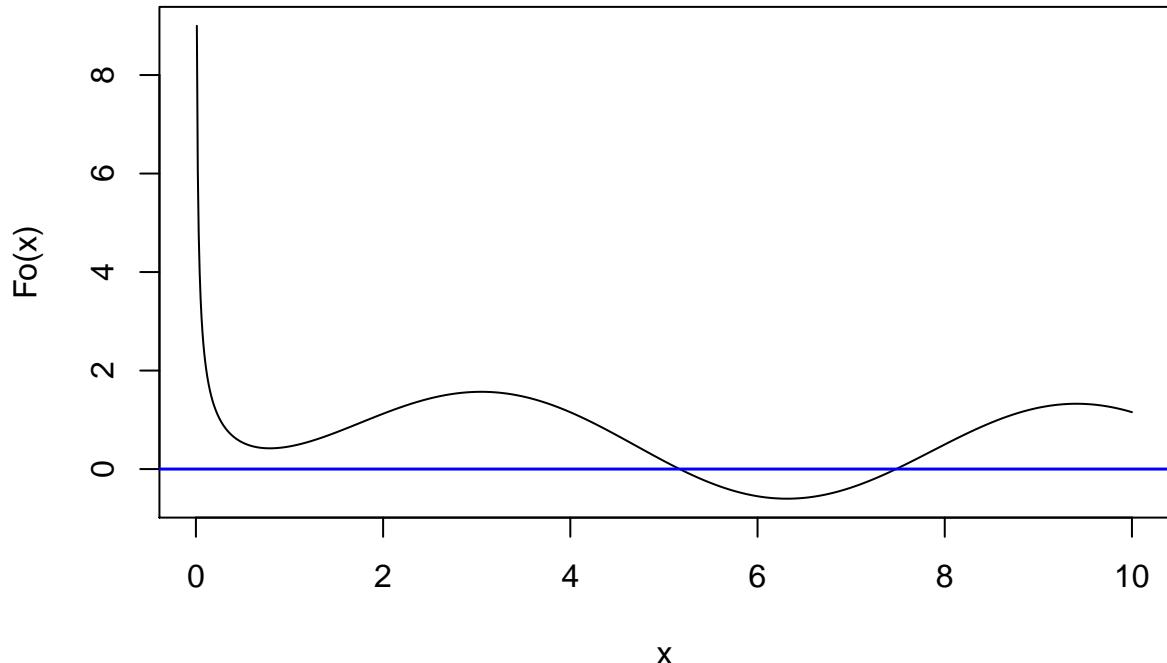
On en déduit que F admet bien deux solutions x_0 telles que $F(x_0) = 0$ sur l'intervalle $]0, 10]$ (on peut même être plus précis que ça).

Pour chercher une approximation de ces valeurs, on procède ainsi :

```
pas <- 0.01
x <- seq(from = 0.01,to = 10,by = pas)
delta <- 10^(-2)
x[which(abs(Fo(x)) < delta)]
```

```
## [1] 5.16 5.17 7.47 7.48 7.49
```

```
# graphiquement cette méthode consiste à récupérer les
# abscisses des points de la courbe de Fo qui sont compris
# entre les deux lignes bleues
{
  plot(x,Fo(x),type="l")
  abline(h = 10^(-2),col = "blue")
  abline(h = -10^(-2),col = "blue")
}
```



On obtient plusieurs valeurs pour l'axe des abscisses à cause la précision de cette méthode. On peut n'obtenir aucune valeur si on prend une valeur de `delta` trop petite. On peut améliorer l'approximation en augmentant le nombre de points dans dans le vecteur `x` ce qui revient à prendre un pas plus petit. Cependant, notez

qu'au bout d'un moment, on sera de nouveau heurté au problème de la précision de la machine.