Exercice 2 :

La fonction à intégrer dans cette exercice est :

$$g(x) = \frac{exp(x) - 1}{exp(1) - 1}$$

Paramètres de chaques estimateurs (méthode, technique d'échantillonnage, nombre de points et tout ce que vous jugerez nécessaire)

2.2 MC Noir/Blanc (NB):

La méthode de Monte-Carlo par "tirage noir ou blanc" est une méthode qui permet d'échantilloner une loi en regardant si une variable aléatoire est en dessous de la valeur de l'intégrale d'une fonction $f(x)$ ou non.

Les paramètres de cette fonction sont :

- m majorant de $f(x)$
- x qui suit une loi uniforme sur [a,b] (a,b borne de f(x))
- y qui suit une loi uniforme sur [0,m]

Le résultat attendu doit suivre une loi du type :

$$I = m * (b - a) * \frac{ns}{n}$$

avec ns= nombre de succès.

Ici, la loi uniforme a été choisit pour gérer les variables x aléatoires.

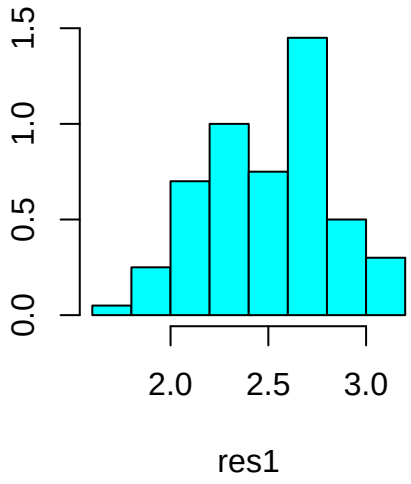

Figure 1: Histogramme de l'estimateur MC par tirage noir ou blanc

dire des trucs dessus

2.3 MC avec densité auxiliaire SIMPLE (uniforme) :

La méthode de Monte-Carlo par MC simple est une méthode qui permet d'échantilloner à partir de variables aléatoires X et d'une fonction de densité de support $h(x)$ tel que :

$g(x) = h(x) * fx(x)$

Ici, la loi uniforme a été utilisé pour générer les variables x aléatoire, et $h(x)$ a été calculé à partir de :

$$h(x) = (b - a) * g(x)$$

tel que

- a et b représentent les bornes
- g(x) la fonction a intégrer

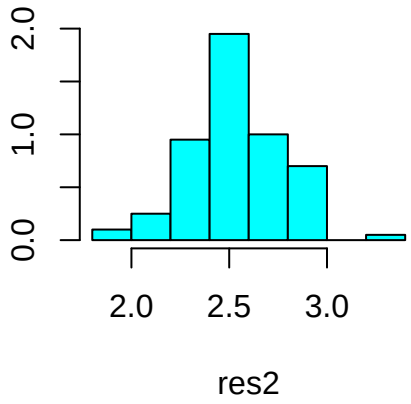

Figure 2: Histogramme de l'estimateur MC avec densité auxiliaire simple

dire des trucs dessus # truc a dire ici #

2.4 MC avec densité auxiliaire en BETA:

La méthode de Monte-Carlo avec densité auxiliaire suivant l'importance a été implémenté ici sur une loi Bêta de paramètres ($\alpha = 2$, $\beta = 1$) comme loi d'échantillonage.

ablablater un peu plus

Histogramme MC Beta :

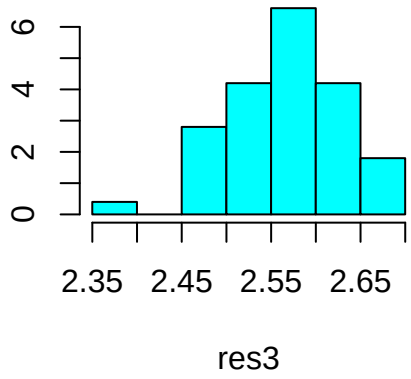

Figure 3: Histogramme de l'estimateur MC avec densité auxiliaire en beta

dire des trucs ici

2.4 MC avec densité auxiliaire en BETA:

Maintenant qu'on a vu que la simple inversion des valeurs de alpha et bêta dans la variable auxiliaire suffit à augmenter grandement le MSE, on va chercher à déterminer les valeurs optimales de alpha et bêta, celles qui maximisent le MSE

On commence par créer une fonction de calcul du MSE adpatée aux fonctions de type Bêta:

```
MSE_BETA = function(vect_param)
{
  my_alpha = vect_param[1] ; my_beta = vect_param[2]
  nbRep = 10000
  g_support = function(scal) { return( expm1(scal)/expm1(1) ) }

  vect_res = rep(0, nbRep)

  for (rep in 1:nbRep)
  {
    vect_res[rep] = MC_beta(alpha=my_alpha, beta=my_beta,
                            k=2, 200, g_support)
  }
  soluce = (exp(2)-3)/expm1(1)
  return( sum( (vect_res-soluce)**2)/nbRep )
}
```

Ensuite, on procède à un 1er quadrillage visuel grossier, pour rétrécir l'espace dans lequel on recherche les alpha et bêta optimaux:

```
pas = 1
range_alpha = seq(pas, 5, by=pas)
range_beta = seq(pas, 3, by=pas)
long_alpha = length(range_alpha)
long_beta = length(range_beta)

graphics.off()
par(mfrow = c(long_alpha, long_beta), mar = rep(2, 4))
for (my_alpha in range_alpha)
{
  for (my_beta in range_beta)
  {
    curve(expm1(x)/expm1(1), from=0, to=2, xname="x",
          main=paste("alpha=", my_alpha, "beta=", my_beta),
          xlab="alpha", ylab="beta")
    curve(dbeta(x/2, my_alpha, my_beta)/2, from=0, to=2, xname="x", add=T)
  }
}
```

En allant voir les formules compliquées sur la loi Bêta sur Wikipédia, on sait déjà que alpha et bêta doit tous les 2 être **strictement** positifs. Ensuite, on peut voir que pour bêta >= 2, les densités auxiliaires perdent en proximité avec la densité de référence. Donc on fera varier bêta sur]0 ; 2[Pour alpha, c'est un peu plus compliqué, ça l'air de mîmer g de mieux en mieux, au fur et à mesure que alpha augmente.

On va mainteant essayer de déterminer les valeurs numériques optimales de alpha et bêta, en essayant de trouver celles pour lesquelles le MSE est minimal. Pour ça, on utilise une méthode de "grille", en créant un tableau (matrice) contenant le MSE associé à chaque couple (alpha ; bêta):

```
system.time(
{
  pas_beta = 0.5
  pas_alpha = 1
  range_alpha = seq(pas_alpha, 5, by=pas_alpha)
  range_beta = seq(pas_beta, 2-pas_beta, by=pas_beta)

  mat_res = matrix(data=0, nrow=long_alpha, ncol=long_beta)
  i = 1

  for (my_alpha in range_alpha)
  {
    j = 1
    for (my_beta in range_beta)
    {
      vect_param = c(my_alpha, my_beta)
```

```
      mat_res[i, j] = MSE_BETA(vect_param)
      j = j + 1
    }
    i = i + 1
  }
})
```

```
##    user  system elapsed
##  21.699   0.016  21.722
```

```
#library(rgl) #mininstall -c r r-rgl
#x = scan()

#persp3d(range_alpha, range_beta, 1/mat_res, xlab="alpha", ylab="beta")
#persp(range_alpha, range_beta, 1/mat_res,  xlab="alpha", ylab="beta",
      #ticktype="detailed")
```

En plottant l'inverse des MSE plutôt que les MSE directement en Z, il semble qu'on amplifie les écarts. On voit donc clairement apparaître sur le plot 3D un "pic", de coordonnées (alpha=3 ; bêta=1), qui semblent être les valeurs optimales des paramètres alpha et bêta

REMARQUE: On pourrait regarder les valeurs prises autour de 1 et 3, pour savoir si c'est exactement ces valeurs ou plutôt des trucs à virgules

EN UTILISANT OPTIM:

```
#res_final = optim(c(2, 1), fn=MSE_BETA, method=c("BFGS"))
```

2.5 Comparaison des méthodes

On commence par créer les vecteurs associés à chacune des 4 méthodes (NB, SIMPLE, BETA(alpha=2,beta=1) et BETA(alpha=1,beta=2)):

```
print( paste( "MOY_NB =", mean(vect_res_NB), "| MSE_NB =", MSE(vect_res_NB) ) )
```

```
## [1] "MOY_NB = 2.55015384579401 | MSE_NB = 0.0638034589108445"
```

```
print( paste( "MOY_SIMPLE =", mean(vect_res_SIMPLE),
              "| MSE_SIMPLE =", MSE(vect_res_SIMPLE) ) )
```

```
## [1] "MOY_SIMPLE = 2.55224784972724 | MSE_SIMPLE = 0.0217398471069271"
```

```
print( paste( "MOY_BETA_2_1 =", mean(vect_res_BETA_2_1),
              "| MSE_BETA_2_1 =", MSE(vect_res_BETA_2_1) ) )
```

```
## [1] "MOY_BETA_2_1 = 2.5536315380898 | MSE_BETA_2_1 = 0.00222370573972782"
```

```
print(paste("MOY_BETA_1_2 =", mean(vect_res_BETA_1_2),
            "| MSE_BETA_1_2 =", MSE(vect_res_BETA_1_2)  ) )
```

```
## [1] "MOY_BETA_1_2 = 2.54814570768159 | MSE_BETA_1_2 = 0.817700685839614"
```

On voit bien ici, que même si les valeurs de I calculées en faisant la moyenne pour 100000 répétitions sont équivalentes, les précisions sur le résultat, ou autrement dit les fluctuations d'échantillonnage changent (quantifiées par le MSE). On voit donc que ça va en diminuant du NB au BETA_2_1, ce qui indique que SIMPLE est meilleure que NB d'un facteur 3, mais moins bonne que BETA_2_1 d'un facteur 10 BETA_1_2 est à part, avec le plus mauvais MSE des 4 méthodes.

Il est possible d'expliquer **qualitativement** les résultats des 3 méthodes MC par "échantillonnage", en les classant par MSE et en affichant leurs densités de probabilité:

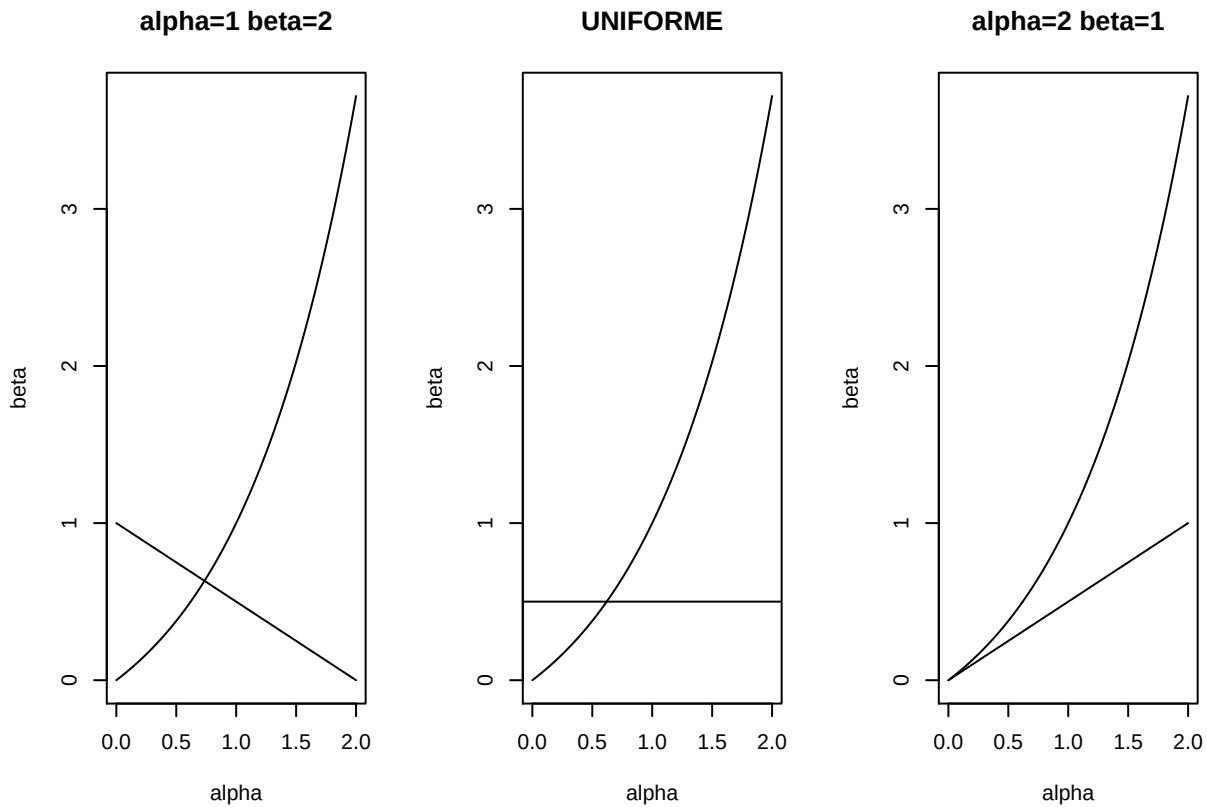

Figure 4: Belles courbes

On a vu précédemment que sur ce type de méthodes par “importance variable” des valeurs, c’est la densité de probabilité de la variable auxiliaire qui détermine l’importance accordée à chaque valeur de l’intervalle support. De ce fait, plus la densité auxiliaire “mîme” fidèlement la densité de g (fonction de référence), plus on accorde une grande importance aux valeurs de g(x) les plus grandes, celles qui contribuent le plus à l’aire, meilleure sera l’approximation. Ainsi pour BETA_1_2, la densité auxiliaire s’oppose à la tendance de la densité de référence, accordant donc plus d’importance aux valeurs inférieures à 1, qui sont pourtant celles qui contribuent le moins à l’aire, puis que g est croissante sur [0, 2]. D’où son mauvais MSE. De même, avec sa densité de probabilité plate, la variable auxiliaire donne une importance équivalente à toutes les valeurs de l’intervalle, d’où son MSE intermédiaire C’est BETA_2_1 qui “épouse” le mieux g, d’où son MSE meilleur que les 2 autres.