

Devoir 2 (solutions): Méthode de la transformée inverse

Exercice 1

(a)

```
rand_gen <- function(x, trans = identity) {
  trans_vect <- Vectorize(trans)

  counter <- as.table(rep(0L, length(x)))
  names(counter) <- trans_vect(seq(0L, length.out = length(x)))

  p <- cumsum(x) / sum(x)

  function(n) {
    ret <- trans_vect(sapply(runif(n), function(y) sum(y > p)))
    counter <-< counter + table(factor(ret, names(counter)))

    ret
  }
}
```

(b)

```
rand_trans <- function(sim, trans) rand_gen(environment(sim)[["x"]], trans)
```

(c)

La fonction suivante fonctionne pour un nombre arbitraire d'arguments (ce qui n'était pas nécessaire pour répondre à la question).

```
rand_sum <- function(...) {
  xs <- lapply(list(...), function(s) environment(s)[["x"]])

  n <- Reduce(max, sapply(xs, length))

  new_x <- Reduce(x = xs, init = numeric(n),
    f = function(x, y) {
      y <- c(y, numeric(n - length(y)))
      x + y
    })

  rand_gen(new_x)
}
```

(d)

```
rand_stats <- function(sim) proportions(environment(sim)[["counter"]])
```

(e)

```
rand_hist <- function(sim) barplot(rand_stats(sim), space = 0L)
```

Exercice 2

```
exo2 <- function(chiffres = c(2L, 2L, 5L, 6L, 6L, 6L, 7L, 9L),
                 lb = -1e10,
                 ub = 1e10,
                 nb_iters = 1e5L) {
  op_match <- list(`+`, `-`, `*`, `/`, `^`)

  res <- replicate(n = nb_iters, simplify = FALSE, expr = {
    n <- sample(seq_along(chiffres)[-1L], 1L)
    chiffres_utilises <- sample(chiffres, n)

    ## Pour échantillonner les opérations à effectuer, on procède
    ## par conditionnement: (1) on échantillonne le nombre
    ## d'exponentiations, de multiplications/divisions et
    ## d'addition/soustractions et (2) on échantillonne trois
    ## séquences d'opérations pour chacune de ces catégories. De
    ## cette manière, on n'a pas à se préoccuper de la priorité
    ## des opérations.
    ops_rep <- c(rmultinom(1L, n - 1L, rep(1L, 3)))
    ops_utilises <- c(rep(5L, ops_rep[[1L]]),
                     sample(3L:4L, ops_rep[[2L]], replace = TRUE),
                     sample(1L:2L, ops_rep[[3L]], replace = TRUE))

    res <- as.numeric(chiffres_utilises[[1L]])
    toRemove <- logical(n - 1L)
    for (k in 1L:(n - 1L)) {
      candidat <-
        op_match[[ops_utilises[k]]](res, chiffres_utilises[[k + 1L]])
      if (identical(ceiling(candidat), floor(candidat)))
        res <- as.numeric(candidat)
      else
        toRemove[k] <- TRUE
    }

    list(x = res,
         chiffres = chiffres_utilises[c(TRUE, !toRemove)],
         ops = ops_utilises[!toRemove])
  })
```

```

## Enlève les nombres situés à l'extérieur des limites.
res <- Filter(x = res, f = function(t) t[["x"]] <= ub & t[["x"]] >= lb)

## Enlève les doublons. Pour ce faire, la première étape est
## d'ordonner les résultats.
res_trans <- do.call(rbind, res)
xs <- as.numeric(res_trans[,1L])
xs_order <- order(xs)
xs <- xs[xs_order]

## Trouve les indices des doublons.
doublons <- which(head(xs, -1L) == tail(xs, -1L)) + 1L
n <- length(xs) - length(doublons)
if (identical(doublons, integer(0L)))
  ## Hack pour palier à un comportement de R (subset par -integer(0))...
  doublons <- -seq_along(xs)

## Ajuste les chiffres et les opérations.
chiffres <- res_trans[,2L][xs_order]
ops <- res_trans[,3L][xs_order]

## Retourne le résultat.
lapply(1L:n, function(k) list(x = xs[-doublons][k],
                               chiffres = chiffres[-doublons][k][[1L]],
                               ops = ops[-doublons][k][[1L]]))
}

```

La liste retournée contient assez d'informations pour reconstruire chacune des décompositions. Par exemple, la fonction suivante permet d'afficher à l'écran un élément de la suite et sa décomposition.

```

exo2_decode <- function(nb) {
  op_match <- list("+", "-", "*", "/", "^")

  cat(nb[["x"]], " = ", sep = "")

  cat(nb[["chiffres"]][[1L]])
  for (k in seq_along(nb[["ops"]]))
    cat(" ",
        op_match[nb[["ops"]][k]],
        " ",
        nb[["chiffres"]][k + 1L],
        sep = "")

  cat("\n")
}

```

Exemples

```
res <- exo2()  
for (n in sample(res, 10, replace = TRUE))  
  exo2_decode(n)
```

$$32256 = 2^6 * 7 * 2 * 6 * 6$$

$$117646 = 7^6 + 6 - 9$$

$$4091 = 2^2 * 6 - 5$$

$$4218754 = 5^7 * 6 * 9 - 2 + 6$$

$$72015 = 7^2 * 2 * 5 * 6 - 9 - 6$$

$$279924 = 6^6 * 6 + 2 - 7 - 9 + 2$$

$$117660 = 7^6 + 9 + 2$$

$$11679 = 6^5 * 9 / 6 + 7 + 6 + 2$$

$$2480058 = 9^5 * 2 / 2 * 7 * 6$$

$$256 = 6 * 6 * 7 - 2 + 6$$