Profiling score-calculation methods

Matthew Farrugia-Roberts (@matomatical)

Generating the symbols (batch)

```
WHEEL <- c("DD",  "7", "BBB", "BB",  "B",  "C",  "0")
PROBS <- c(0.03, 0.03,  0.06,  0.1, 0.25, 0.01, 0.52)
```

```
sample_symbols <- function(n=1) {
    matrix(sample(WHEEL, size=3*n, replace=TRUE, prob=PROBS), ncol=3)
}
```

Calculating the prizes

Method 1: Book approach

```
score.book <- function(symbols) {
    diamonds <- sum(symbols == "DD")
    cherries <- sum(symbols == "C")

    # identify case
    # since diamonds are wild, only nondiamonds
    # matter for three of a kind and all bars
    slots <- symbols[symbols != "DD"]
    same <- length(unique(slots)) == 1
    bars <- slots %in% c("B", "BB", "BBB")

    # assign prize
    if (diamonds == 3) {
      prize <- 100
    } else if (same) {
      payouts <- c("7" = 80, "BBB" = 40, "BB" = 25,
        "B" = 10, "C" = 10, "0" = 0)
      prize <- unname(payouts[slots[1]])
    } else if (all(bars)) {
      prize <- 5
    } else if (cherries > 0) {
      # diamonds count as cherries
      # so long as there is one real cherry
      prize <- c(0, 2, 5)[cherries + diamonds + 1]
    } else {
      prize <- 0
    }

    # double for each diamond
    prize * 2^diamonds
}
```

```r
score.book.loop <- function (symbols) {
    v <- numeric(nrow(symbols))
    for (i in 1:nrow(symbols)) {
        v[i] <- score.book(symbols[i,])
    }
    v
}
```

Method 2: Counting and branching

```r
score.count <- function(symbols) {
    # count symbols
    dd <- sum(symbols == "DD")
    x7 <- sum(symbols == "7")
    b3 <- sum(symbols == "BBB")
    b2 <- sum(symbols == "BB")
    b1 <- sum(symbols == "B")
    cc <- sum(symbols == "C")
    # calculate prize (higher prizes detected before lower ones)
    if (dd == 3) {
        prize <- 100
    } else if (x7 + dd == 3) {
        prize <- 80
    } else if (b3 + dd == 3) {
        prize <- 40
    } else if (b2 + dd == 3) {
        prize <- 25
    } else if (b1 + dd == 3) {
        prize <- 10
    } else if (cc + dd == 3) {
        prize <- 10
    } else if (b3 + b2 + b1 + dd == 3) {
        prize <- 5
    } else if (cc > 0 && cc + dd == 2) {
        prize <- 5
    } else if (cc == 1) {
        prize <- 2
    } else {
        prize <- 0
    }
    prize * (2 ^ dd)
}
score.count.loop <- function (symbols) {
    v <- numeric(nrow(symbols))
    for (i in 1:nrow(symbols)) {
        v[i] <- score.count(symbols[i,])
    }
    v
}
```

Method 3: Book approach, vectorised

```r
score.book.fast <- function(symbols) {
  # Step 1: Assign base prize based on cherries and diamonds ---------
  ## Count the number of cherries and diamonds in each combination
  cherries <- rowSums(symbols == "C")
  diamonds <- rowSums(symbols == "DD")

  ## Wild diamonds count as cherries
  prize <- c(0, 2, 5)[cherries + diamonds + 1]

  ## ...but not if there are zero real cherries
  ### (cherries is coerced to FALSE where cherries == 0)
  prize[!cherries] <- 0

  # Step 2: Change prize for combinations that contain three of a kind
  same <- symbols[, 1] == symbols[, 2] &
    symbols[, 2] == symbols[, 3]
  payoffs <- c("DD" = 100, "7" = 80, "BBB" = 40,
    "BB" = 25, "B" = 10, "C" = 10, "0" = 0)
  prize[same] <- payoffs[symbols[same, 1]]

  # Step 3: Change prize for combinations that contain all bars ------
  bars <- symbols == "B" | symbols ==  "BB" | symbols == "BBB"
  all_bars <- bars[, 1] & bars[, 2] & bars[, 3] & !same
  prize[all_bars] <- 5

  # Step 4: Handle wilds ---------------------------------------------

  ## combos with two diamonds
  two_wilds <- diamonds == 2

  ### Identify the nonwild symbol
  one <- two_wilds & symbols[, 1] != symbols[, 2] &
    symbols[, 2] == symbols[, 3]
  two <- two_wilds & symbols[, 1] != symbols[, 2] &
    symbols[, 1] == symbols[, 3]
  three <- two_wilds & symbols[, 1] == symbols[, 2] &
    symbols[, 2] != symbols[, 3]

  ### Treat as three of a kind
  prize[one] <- payoffs[symbols[one, 1]]
  prize[two] <- payoffs[symbols[two, 2]]
  prize[three] <- payoffs[symbols[three, 3]]

  ## combos with one wild
  one_wild <- diamonds == 1

  ### Treat as all bars (if appropriate)
  wild_bars <- one_wild & (rowSums(bars) == 2)
  prize[wild_bars] <- 5

  ### Treat as three of a kind (if appropriate)
  one <- one_wild & symbols[, 1] == symbols[, 2]
```

```
  two <- one_wild & symbols[, 2] == symbols[, 3]
  three <- one_wild & symbols[, 3] == symbols[, 1]
  prize[one] <- payoffs[symbols[one, 1]]
  prize[two] <- payoffs[symbols[two, 2]]
  prize[three] <- payoffs[symbols[three, 3]]

  # Step 5: Double prize for every diamond in combo ------------------
  unname(prize * 2^diamonds)
}
```

Method 4: Counting and vectorised overwriting

```
score.count.fast <- function(symbols) {
    # counts of symbols in each sample
    dd <- rowSums(symbols == "DD")
    x7 <- rowSums(symbols == "7")
    b3 <- rowSums(symbols == "BBB")
    b2 <- rowSums(symbols == "BB")
    b1 <- rowSums(symbols == "B")
    cc <- rowSums(symbols == "C")
    # calculate prize (higher prizes later to override lower ones)
    prize = integer(nrow(symbols)) # defaults to a number of 0s
    prize[cc == 1]                  <-   2
    prize[cc > 0 & cc + dd == 2]    <-   5
    prize[b3 + b2 + b1 + dd == 3]   <-   5
    prize[cc + dd == 3]             <-  10
    prize[b1 + dd == 3]             <-  10
    prize[b2 + dd == 3]             <-  25
    prize[b3 + dd == 3]             <-  40
    prize[x7 + dd == 3]             <-  80
    prize[dd == 3]                  <- 100
    # apply diamonds doubling effect
    prize * (2 ^ dd)
}
```

Test that all outputs are the same

Construct all inputs

```
combos <- expand.grid(sym1 = WHEEL, sym2 = WHEEL, sym3 = WHEEL)
```

Construct all outputs

```
symbols <- as.matrix(combos[,c("sym1","sym2","sym3")])
# loop-based methods
combos$score.book       <- score.book.loop(symbols)
combos$score.count      <- score.count.loop(symbols)
# vectorised methods
combos$score.book.fast  <- score.book.fast(symbols)
combos$score.count.fast <- score.count.fast(symbols)
head(combos)
```

```
##   sym1 sym2 sym3 score.book score.count score.book.fast score.count.fast
## 1   DD   DD   DD        800         800             800              800
## 2    7   DD   DD        320         320             320              320
```

```
## 3  BBB   DD   DD        160         160             160             160
## 4   BB   DD   DD        100         100             100             100
## 5    B   DD   DD         40          40              40              40
## 6    C   DD   DD         40          40              40              40
```

All outputs should be equal

```
combos$equal <- combos$score.book == combos$score.count &
                combos$score.book == combos$score.book.fast &
                combos$score.book == combos$score.count.fast
all(combos$equal)
```

```
## [1] TRUE
```

If there are any non-equal rows, display them here

```
combos[!combos$equal,]
```

```
## [1] sym1             sym2             sym3             score.book       score.count
## [6] score.book.fast  score.count.fast equal
## <0 rows> (or 0-length row.names)
```

Profiling

```
library(ggplot2)
library(microbenchmark)
```

```
all_symbols <- as.matrix(combos[,c("sym1","sym2","sym3")])
results <- microbenchmark(
    score.book.loop(all_symbols),
    score.count.loop(all_symbols),
    score.book.fast(all_symbols),
    score.count.fast(all_symbols),
    times=100L
)
```

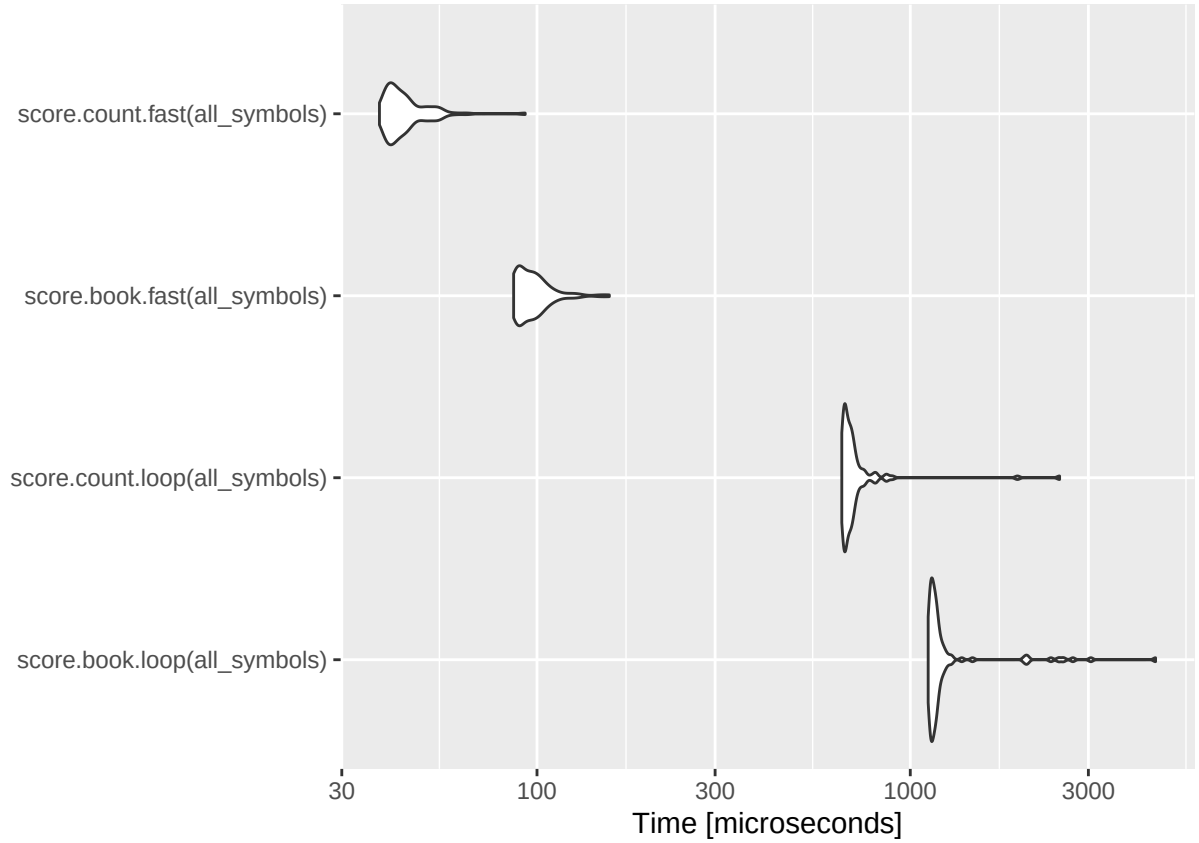
```
## Warning in microbenchmark(score.book.loop(all_symbols), score.count.loop(all_symbols), : less
## accurate nanosecond times to avoid potential integer overflows
```

```
print(results)
```

```
## Unit: microseconds
##                          expr      min        lq       mean    median        uq      max neval
##   score.book.loop(all_symbols) 1116.594 1136.8275 1302.72047 1159.1930 1195.6625 4540.463   100
##  score.count.loop(all_symbols)  655.467  666.7625  726.57371  685.1920  704.8720 2511.824   100
##   score.book.fast(all_symbols)   86.592   89.0520   98.06544   95.3455  101.8850  156.251   100
##  score.count.fast(all_symbols)   37.843   40.2210   44.60554   42.4760   45.7765   92.988   100
```

```
ggplot2::autoplot(results)
```

```
## Coordinate system already present. Adding new coordinate system, which will
## replace the existing one.
```

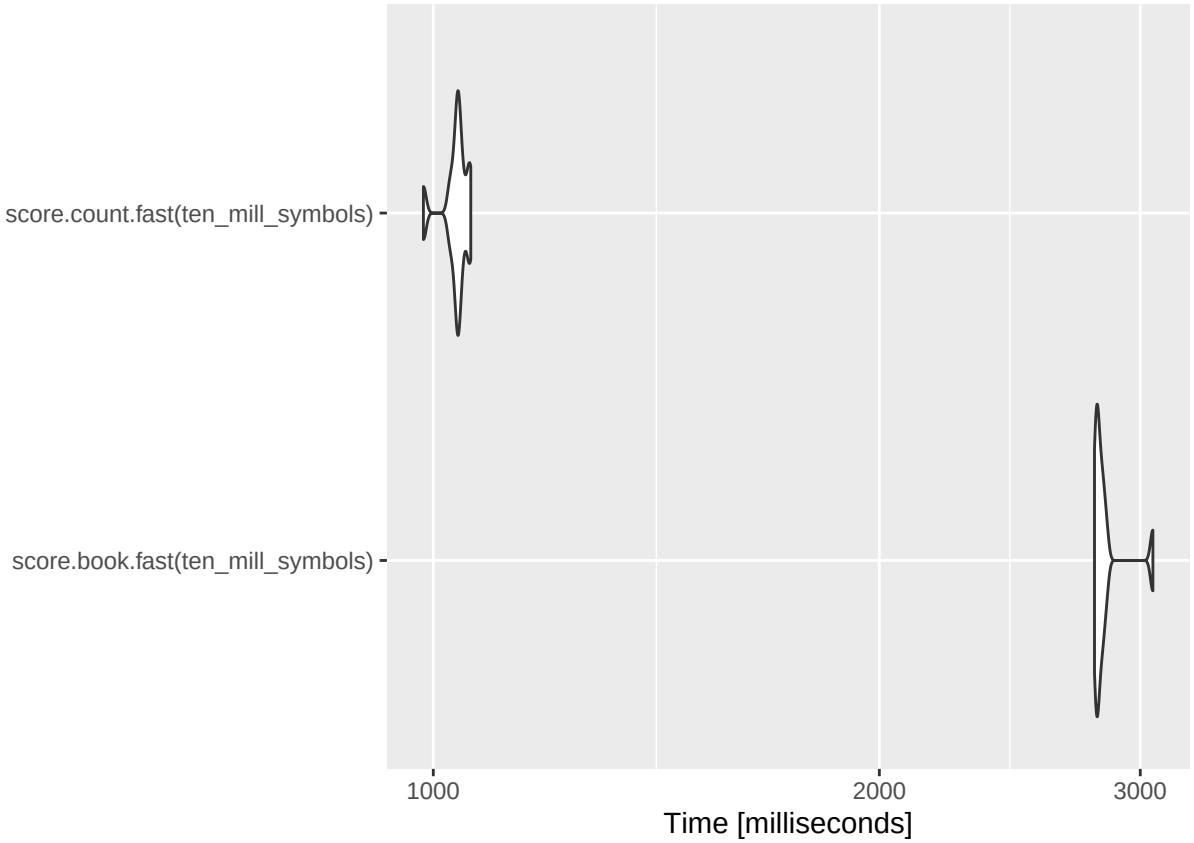



```
ten_mill_symbols <- sample_symbols(10000000)
results <- microbenchmark(
    score.book.fast(ten_mill_symbols),
    score.count.fast(ten_mill_symbols),
    times=10L
)
print(results)
```

```
## Unit: milliseconds
##                                 expr       min       lq     mean   median       uq      max neval
##   score.book.fast(ten_mill_symbols) 2794.1207 2803.084 2838.820 2811.412 2829.872 3059.580    10
##  score.count.fast(ten_mill_symbols)  984.7332 1032.958 1036.428 1039.382 1046.628 1059.957    10
```

```
ggplot2::autoplot(results)
```

```
## Coordinate system already present. Adding new coordinate system, which will
## replace the existing one.
```


score.count.fast(ten_mill_symbols)
score.book.fast(ten_mill_symbols)
1000
2000
3000
Time [milliseconds]