Demonstration of the Rcpp package for R

Dave Millar

dave.millar@uwyo.edu

R for Ecologists

11/15/17

This is a brief demonstration of running iterative simulations using R functions and again using the Rcpp package for R. The **Rcpp** package allows C++ code to be incorporated into an R script, which can dramatically speed up processing time for computationally intense analyses. C++ is a more verbose, complex programming language than R, which can be intimidating for sure (definitely the case for me!). However, if it can be used sparingly for limited aspects of your analyses, incorporating it into your R code can be a tractable problem, particularly if you are able to consult with someone who has a strong computer programming background.

For our simulations, lets say we roll a 20-sided die iteratively until the sum of the die rolls is greater than or equal to five different threshold values (M).

We want to determine the mean and standard deviation of the number of die rolls if M = 50, 250, 4000, 30000, and 150000.

First, let's develop this as a model using functions in R. Here, I started out by clearing anything stored in memory,

```
rm(list=ls())
```

and call some tidyverse packages.

```
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
```

Next, let's write a few functions in R to help determine the summary stats described above.

The first function below simulates rolling a 20-sided die (d20) until the sum of the rolled values is equal to or greater than M, which is specified by the user. This function records the summed value of the rolls as well as the number of rolls required for the summed value to reach M.

```
d20_roll_func <- function(M){

  # vector to represent the 20-sided die (d20)
```

```
  d20 <- seq(1,20,1)

  # set initial values for the summed roll values and roll counts to zero
  d20_roll_sum <- 0
  d20_roll_count <- 0

  # roll the d20 until the threshold (M) is reached
  while (d20_roll_sum < M){

    d20_roll_sum <- d20_roll_sum + sample(d20, size = 1,replace = TRUE)

    d20_roll_count <- d20_roll_count + 1

  }

  # report the sum of the rolled values and the number of rolls
  d20_ouput <- c(d20_roll_sum,d20_roll_count)
  return(d20_ouput)

}
```

In order to determine the necessary summary statistics, let's write a second function that iteratively runs d20_roll_func, which is nested within it, for a number of times specified by the user. This function ultimately generates values for a statistical sample set from which mean and standard deviation can be calculated.

```
iterate_roll_func <- function(itr,M){

  # matrix to store output
  itr_output <- data.frame(matrix(nrow=itr, ncol=2))

  # iterate the d20 rolls function
  for(i in 1:itr) {

        #fill each row in the itr_output df with results from d20_roll_func
        itr_output[i,] <- d20_roll_func(M)

        }

  # set column names for the final output df
  # (summed roll value and roll count)
  names(itr_output)=c("summed_val", "roll_ct")
  return(itr_output)

}
```

Next, let's define each of the five M values,

```
M1 <- 50
M2 <- 250
M3 <- 4000
M4 <- 30000
M5 <- 150000
```

and run the analyses while recording the run time.

```
# start timing analyses
runtime <- proc.time()

# simualte all the d20 rolls (1000 iterations per M)
M1_itr <- iterate_roll_func(1000,M1)
M2_itr <- iterate_roll_func(1000,M2)
M3_itr <- iterate_roll_func(1000,M3)
M4_itr <- iterate_roll_func(1000,M4)
M5_itr <- iterate_roll_func(1000,M5)

# stop the timer
proc.time() - runtime
```

```
##    user  system elapsed
##   77.75    0.01   77.95
```

Using the R function approach took over a minute to run the analyses using 1000 iterations per M value.

Next, let's consolidate the data into one long format data frame.

```
all_data <- cbind.data.frame(M1_itr$roll_ct,
                             M2_itr$roll_ct,
                             M3_itr$roll_ct,
                             M4_itr$roll_ct,
                             M5_itr$roll_ct)

names(all_data) <- c("M1","M2","M3","M4","M5")

all_data <- all_data %>% gather("M1","M2","M3","M4","M5",
                                key = "threshold", value = "value")
```

Then compute and tabulate all the relevant summary statistics.

```
results_table <- all_data %>%
               group_by(threshold) %>%
               summarize(mean = mean(value),sd = sd(value))
results_table
```

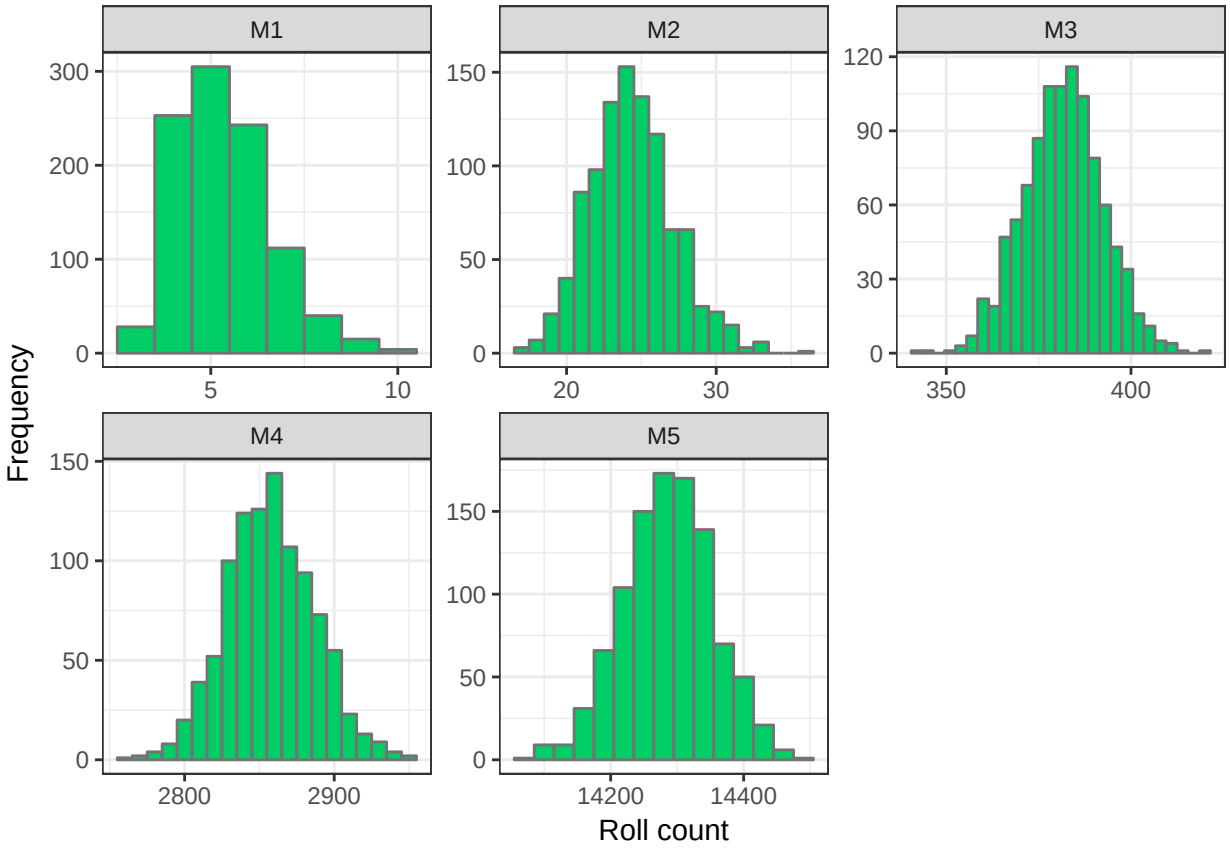
```
## # A tibble: 5 x 3
##   threshold      mean        sd
##       <chr>     <dbl>     <dbl>
## 1        M1     5.358  1.276505
## 2        M2    24.369  2.822198
## 3        M3   381.585 10.774361
## 4        M4  2857.682 29.547689
## 5        M5 14286.911 67.805506
```

Let's also visualize some of the data by creating histograms for the roll counts for each M value.

```
# plot histgrams of roll counts for each M value
# NOTE: binwidths are adjusted manually based on
#       the number of roll counts for aesthetic purposes
hist_rc_plot <- ggplot(all_data, aes(value)) +
  geom_histogram(data = subset(all_data, threshold == "M1"), binwidth=1,
                 color="grey45", fill="springgreen3") +
  geom_histogram(data = subset(all_data, threshold == "M2"), binwidth=1,
                 color="grey45", fill="springgreen3") +
```

```
  geom_histogram(data = subset(all_data, threshold == "M3"), binwidth=3,
                 color="grey45", fill="springgreen3") +
  geom_histogram(data = subset(all_data, threshold == "M4"), binwidth=10,
                 color="grey45", fill="springgreen3") +
  geom_histogram(data = subset(all_data, threshold == "M5"), binwidth=30,
                 color="grey45", fill="springgreen3") +
  facet_wrap(~ threshold, scales="free") +
  xlab("Roll count") +
  ylab("Frequency") +
  scale_x_continuous(breaks= scales::pretty_breaks(n=3)) +
  theme_bw()

hist_rc_plot
```



Now let's try this again using **Rcpp**, which requires a C++ compiler, which can be obtained by downloading Rtools, available here.

Let's start by again clearing anything stored in memory.

```
rm(list=ls())
```

The following is C++ source code for the necessary functions. There is one function for counting the number of die rolls needed for the summed roll to reach M, and a corresponding function for iteration.

```
#include <Rcpp.h>
using namespace Rcpp;

//---------------------------------------
// Function to count the number of rolls needed for the summed values to reach M
// [[Rcpp::export]]
int roll_count_func(int M){

  // set initial values for the summed roll values and roll counts to zero
  int d20_roll_sum = 0;
  int d20_roll_count = 0;

  // roll the d20 until the threshold (M) is reached
  while (d20_roll_sum < M){

    int currentRoll = rand() % 20 + 1;

    d20_roll_sum += currentRoll;

    d20_roll_count += 1;

  }

  // report the sum of the rolled values and the number of rolls
  return d20_roll_count;

}

//---------------------------------------
// Function to iterate roll_count_func
// [[Rcpp::export]]
NumericVector itr_count_func(int itr, int M){

  // vector to store output
  NumericVector itr_output(itr);

  // iterate the d20 rolls function
  for (int i=0; i<itr; i++)  {

    //itr_output[i] = M;
    itr_output[i] = roll_count_func(M);

  }

  return itr_output;
}
```

Now that the C++ source code is compiled, let's rerun and time the analyses as we did using the functions coded in R.

```
# define each of the summed value thresholds of interest
M1 <- 50
```

```
M2 <- 250
M3 <- 4000
M4 <- 30000
M5 <- 150000

# start timing analyses
runtime <- proc.time()

# simualte all the d20 rolls counts (1000 iterations per M)
M1_count <- itr_count_func(1000,M1)
M2_count <- itr_count_func(1000,M2)
M3_count <- itr_count_func(1000,M3)
M4_count <- itr_count_func(1000,M4)
M5_count <- itr_count_func(1000,M5)

# stop the timer
proc.time() - runtime
```

```
##    user  system elapsed
##    0.24    0.00    0.23
```

Optimizing the code by doing the heavy lifting in C++ increased the run speed by about two orders of magnitude!

Let's just verify the results by putting all the simulation output into a single data and creating a table with the summary statistics as was done with the R coding approach.

```
all_data <- cbind.data.frame(M1_count,
                             M2_count,
                             M3_count,
                             M4_count,
                             M5_count)

names(all_data) <- c("M1","M2","M3","M4","M5")

all_data <- all_data %>% gather("M1","M2","M3","M4","M5",
                                key = "threshold", value = "value")
```

```
results_table <- all_data %>%
               group_by(threshold) %>%
               summarize(mean = mean(value),sd = sd(value))
results_table
```

```
## # A tibble: 5 x 3
##   threshold      mean        sd
##       <chr>     <dbl>     <dbl>
## 1        M1     5.392  1.339451
## 2        M2    24.506  2.718337
## 3        M3   381.199 10.561863
## 4        M4  2859.217 29.713478
## 5        M5 14289.672 66.394402
```

The results are almost identical to that of the R code approach, which verifies the optimized approach works. The histogram plots also check out, see below.

```
# plot histgrams of roll counts for each M value
# NOTE: binwidths are adjusted manually based on
```

```
#       the number of roll counts for aesthetic purposes
hist_rc_plot <- ggplot(all_data, aes(value)) +
  geom_histogram(data = subset(all_data, threshold == "M1"), binwidth=1,
                 color="grey45", fill="springgreen3") +
  geom_histogram(data = subset(all_data, threshold == "M2"), binwidth=1,
                 color="grey45", fill="springgreen3") +
  geom_histogram(data = subset(all_data, threshold == "M3"), binwidth=3,
                 color="grey45", fill="springgreen3") +
  geom_histogram(data = subset(all_data, threshold == "M4"), binwidth=10,
                 color="grey45", fill="springgreen3") +
  geom_histogram(data = subset(all_data, threshold == "M5"), binwidth=30,
                 color="grey45", fill="springgreen3") +
  facet_wrap(~ threshold, scales="free") +
  xlab("Roll count") +
  ylab("Frequency") +
  scale_x_continuous(breaks= scales::pretty_breaks(n=3)) +
  theme_bw()

hist_rc_plot
```