ASGR Assignment 1 Questions

Aaron Robotham

1) [5 marks total]

Here we are testing basic and advanced **R** skills. Do the following:

a) [1] Create a function **Rfunc1** that takes the vector inputs a and b and returns $\sqrt{a^{2b}}$. Show the output when a=21:30 and b=(21:20)/200.

b) [1] Create an infix function **skip** that can skip different numbers of elements without throwing an error, e.g. for 1:9 %skip% 3 we should get 1,4,7 without any warnings. Using this function show the results of:

```
set.seed(year)
sample(100,10) %skip% 3
```

c) [3] Write a base **R** function **RMatMult** that does proper matrix multiplication for general matrices (not just square, but generic [n,m]x[m,n]). This should not use %*% and instead explicit loops. Write the same function now called **RcppMatMult** using **Rcpp** code.

Show the result of $A \times B$, where A =matrix(17:24,4,2) and B =matrix(15:8,2,4) using both functions.

Comment of the code speed using **RMatMult**, **RcppMatMult** and %*%.

2) [10 marks total

Load the parquet format geo_heal_10.parquet data and read the associated PDF describing the data.

a) [5] What is the 25th most populous country as per the geo_heal_10.parquet data, and what is the area of this country in square kilometres? How does that compare to expected area, and why might it differ?

b) [5] Create a sensible map showing the terrain height of Great Britain (i.e. the main island of the UK). You can choose the z-scale, but sensible options might be grey or viridis.

---

3) [10 marks total]

You decide you want to start playing Poker, and by far the most popular game is Texas Hold 'Em. This uses a standard deck containing value cards 2-10JQKA and suits Clubs (c) Diamonds (d) Hearts (h) Spades (s) (note, in card notation "10" is written as "T" thus Ts is the 10 of spades and 8d is the eight of diamonds etc). Whilst the odds are easy to find online, we want to construct our own computer simulations to calculate the probability of different outcomes. To get marks for this exercise you must write your own simulation code to produce the results requested below yourself- simply stating the correct answer will get you **zero** marks.

In this variant of the game you try to form the best 5 card hand (look online for details of this, but e.g. https://en.wikipedia.org/wiki/Texas_hold_%27em#Hand_values) out of 2 private cards that only you can use, and 5 shared board cards that you and your opponents can use. Given this set-up, what is the probability (to 0.1% absolute accuracy) that after seeing all 7 cards your best 5 card hand is:

a) [3] Any Flush (5 cards of the same suit, including Straight Flushes and Royal Flushes, e,g,: 2s4s7s9sJs)?

b) [3] Any Straight (cards forming a gap-less run of 5, including Straight Flushes and Royal Flushes, e.g.: 3s4h5d6c7s)? [Careful to treat Aces as both high and low since Ac2d3s4h5d is also a legal straight].

c) [4] A Full House (a three-of-a-kind and a pair, e.g.: 5d5h5sQdQc)? [Careful to check you have not made a better four-of-a-kind].

---

4) [10 marks total]

Here you will need to code up efficient **Rcpp** routines that can check if a number is prime. Your **Rcpp** code *must* be included to get the solutions, since it is trivial to find the answers online (or using various **R** packages).

a) [3] Which of 40,001,407; 40,001,447; 40,001,467; 40,001,473 are prime? (you will lose a mark for each wrong answer, so do not guess!)

b) [5] How many primes have a value less than 200,000,000?

c) [2] What is the 5,000,000th prime number?

As a hint, if you code a sensible strategy using **Rcpp**, you should be able to answer all these questions in well under a minute (at least in terms of computer time).

---

5) [15 marks total]

This question follows on from the introduction to the Mandelbrot set from the course material. To look at this in more detail we should re-write the core algorithm with **Rcpp** since using pure **R** will be far too slow. There are a few tricks to help you on your way:

- We can describe a complex number C as $C = A + Bi$ where A is the modulus of the real part and B is the modulus of the imaginary part, so $C^2 = (A^2 - B^2) + i(2AB)$. This means we do not need to explicitly use complex numbers computationally to find the result, we just have to make use of these identities.

- When doing these iterations, it turns out that if at any point $|C_n| > 2$ (or equally $A^2 + B^2 > 4$) then the solution will certainly diverge. These means if we hit this criterion we can stop the iterations and report that value of C as having a divergent solution.

a) [5] With this knowledge you should code a **Rcpp** function call *bounded* that takes the input A (real part of C) B (imaginary part of C) and N (number of iterations to make). The result should be NA if our distance from the complex origin never exceeds 2, and otherwise it should return the iteration number at which this stopping criterion is reached. To help you check the validity of your code, here are some expected results:

```r
bounded(A=1/2, B=0, N=1e3)
```

```
## [1] 6
```

```r
bounded(A=1/3, B=0, N=1e3)
```

```
## [1] 10
```

```r
bounded(A=1/3.9, B=0, N=1e3)
```

```
## [1] 38
```

```r
bounded(A=1/3.99, B=0, N=1e3)
```

```
## [1] 125
```

```r
bounded(A=1/4, B=0, N=1e3)
```

```
## [1] NA
```

```
bounded(A=0, B=0.9999999, N=1e3)
```

```
## [1] 24
```

```
bounded(A=0, B=0.999, N=1e3)
```

```
## [1] 11
```

```
bounded(A=0, B=1, N=1e3)
```

```
## [1] NA
```

```
bounded(A=-1, B=0.3, N=1e3)
```

```
## [1] 36
```

```
bounded(A=-1.5, B=1e-10, N=1e3)
```

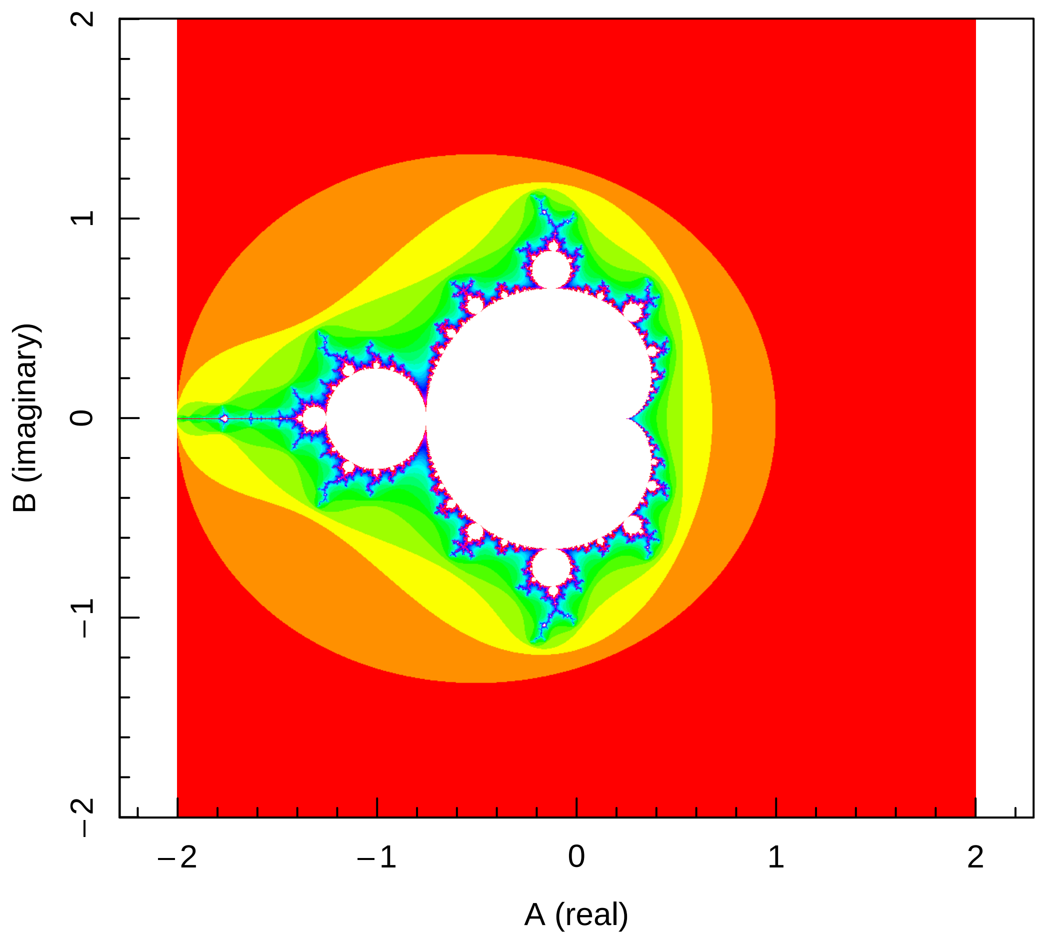
```
## [1] 91
```

b) [5] Getting more sophisticated, the next step is the encode a function where we compute convergence solutions on an arbitrary grid. This should be called *mandel_grid*, and should take inputs *real_lim* (two elements; lower and upper limits for the real component A of the complex number), *imag_lim* (two elements; lower and upper limits for imaginary component B of the complex number), *res* (how many samples to make between limits) and N (number of iterations to make).

To view the results we will create a simple plotting function that brings out some of the attractive features of the Mandelbrot set:

```
R_mandel=function(loc = c(0,0), zoom = 2, res=1e3, N=1e3, index=NULL, plot=TRUE,
                  col=rainbow(1e3), legend=FALSE, ...){
  real_lim = loc[1] + c(-zoom,zoom)
  imag_lim = loc[2] + c(-zoom,zoom)
  output = mandel_grid(real_lim, imag_lim, res=res, N=N)
  if(plot){
    if(zoom > 1e-4){
      output = list(x=seq(real_lim[1],real_lim[2],len=res), y=seq(imag_lim[1],
                imag_lim[2],len=res), z=output)
      magimage(output,col=col, xlab='A (real)', ylab='B (imaginary)', ...)
    }else{
      magimage(output,col=col, xlab='A (real)', ylab='B (imaginary)', ...)
    }
  }
  return(invisible(output))
}
```

For reference here is what we should see if we now generate a default resolution matrix of solutions where the colour represents the iteration number, where white is NA (i.e. it has not met our divergent criterion in 1,000 iterations):



To give a guide of what sort of performance you should be expecting, the above only took 0.3 seconds to run on a single core. For the really adventurous, try to get your code working even faster than this, perhaps by using multiple threads etc. Good luck!

c) [5] If you have a look online you should be able to find list of attractive regions with suggested zoom levels. Re-create 5 of these yourself with the above code you have developed. These should be distinct in terms of the patterns shown.