Part A Simulation and Statistical Programming Hilary 2015

Problem Sheet 4, due Tuesday 10am Week 8.

Please email solutions in a single well-commented R-script to

andreas.anastasiou@jesus.ox.ac.uk

Q1. Here is an algorithm converting a non-negative number 0<x<1 to binary.

Let b be the binary representation of x. If x is zero or one then set b=x. Otherwise compute the first j binary places as follows. Let i=1 and y=2x. If y is greater than or equal one set b[i]=1 and x=y-1, and otherwise set b[i]=0 and x=y. If x is now zero then stop (as there are no more non zero places) and otherwise increase i by one and repeat until i=j and b[j] is evaluated.

(a) Write an R function implementing the algorithm you wrote down in (a). Your function should take as input a non-negative fraction x between 0 and 1 and return the corresponding binary representation, accurate to 56 binary places. Represent the binary number as a vector, so for example decimal 0.125 becomes c(0,0,1) in binary.

(b) At what binary place do R's numerical values for 0.3 and 0.1+0.1+0.1 differ?

Q2. Quick sort is a method for sorting a vector of numbers. A vector `x=c(x1,x2,...,xn)` is given. The function `s=qsort(x)` returns a vector `s` of the same length as `x` with the entries of `x` given in numerically increasing order.

The algorithm works as follows. If the input vector x is empty (`x=c()`, so `n=0`), or has length `n=1`, the function returns `s=x`. Otherwise, pick a pivot (for example `x[1]`). Split the vector into entries smaller than the pivot `x[1]`, say `y=(x[i]: x[i]<x[1], i=2...n)` and greater than or equal `x[1]`, say `z=(x[i]: x[i]>=x[1], i=2...n)`. Call `qsort()` to sort the (possibly empty) vectors `y` and `z`, and return `s=(qsort(y),x[1],qsort(z))`.

(a) Plan and write a recursive R function implementing Quick sort.

(b) Show that the runtime of Quick sort (measured in comparions) is O(n^2) for the worst case input.

Q3. Consider a sequence of observations $x_1,\ldots,x_n$. Let μ_i and σ_i^2 denote the mean and sample variance of the first *i* observations $x_1,\ldots,x_i, i \le n$. How many operations (additions/subtractions or multiplications/divisions) are needed to calculate the sequence of means $\mu_1,\ldots,\mu_n$, if each mean is calculated separately?

(a) Derive an expression for μ_{i+1} in terms of μ_i and x_{i+1} and write an R function that calculates $\mu_1,\ldots,\mu_n$ using this sequential formula. How many operations will this function use?

(b) Now consider the sequence of sample variances $\sigma_1^2,\ldots,\sigma_n^2$. Calculate the number of operations needed to calculate this sequence by evaluating each variance separately, and using a sequential method.

Q4. Consider the normal linear model $Y = X\beta + \varepsilon$ where Y is a vector of n observations, X is an nxp matrix with each column containing a different explanatory variable and ε is vector of n independent normal random errors with mean zero and unknown variance σ^2. The maximum likelihood estimator $\hat{\beta}$ for β is $\hat{\beta} = (X^T X)^{-1} X^T Y$. The sample variance S^2 is $S^2 = \frac{1}{n-1}(X\hat{\beta} - Y)^T (X\hat{\beta} - Y)$. The standard error for $\hat{\beta}_i$ is se($\hat{\beta}_i$)= $S\sqrt{[(X^T X)^{-1}]_{ii}}$.

(a) The `trees` data give Girth, Height and Volume measurements for 31 trees. Fit the normal linear model $y = \beta_1 + x_{girth}\beta_2 + x_{height}\beta_3 + \varepsilon$ using `data(trees)` and `summary(lm(Volume~Girth+Height,data=trees))` and briefly interpret the output. (`trees` is a built-in dataframe – start R and type "trees" at the console)

(b) Write a function of your own (using `solve()` or your solution to Q6, not `lm()`) to fit a normal linear model. Your function should take the 31x1 vector `y=trees$Volume` and the 31x3 matrix `X=cbind(rep(1,31),trees$Girth,trees$Height)` as input and return estimates of $\hat{\beta}$, the residual standard error S and the standard errors of each of the $\hat{\beta}$'s. Check your output against the corresponding results from the `summary(lm())` output in (a).

Q5. Here is an algorithm to compute the QR factorisation of nxp matrix A with p<n into an nxp orthogonal matrix Q and a pxp upper triangular matrix R. |v| is the Euclidean norm of vector v, the square root of the sum of the squares of the elements of v.

Step 1: Create nxp matrix Q of NA's and pxp matrix R of NA's.
Step 2: Set Q[,1]=A[,1]/|A[,1]| and R[1,1]=|A[,1]|
Step 3: If p=1 then we are done; return Q and R.
Step 4: Otherwise (if p>1)

Step 4.1: set R[1,2:p]= $Q[,1]^T$ A[,2:p] and R[2:p,1]=$0_{(p-1)x1}$
Step 4.2: set A'= A[,2:p] – Q[,1]R[1,2:p].

Notice that Q[,1]R[1,2:p] is an outer product of an n component column vector and a (p-1) component row vector so A' is a new nx(p-1) matrix. Either make use the outer() command or, if you use %%, be careful to use drop=F in forming these subset matrices.*

Step 4.3: compute the QR factorisation of A' (A'=Q'R' say).
Step 4.4: set Q[,2:p] = Q' and R[2:p,2:p]=R' and return Q and R.

(a) Implement this algorithm as a recursive function in R. Your function should take as input an nxp matrix A and return two matrices Q and R in a list. State briefly how you checked your function was correct.

(b) Using your QR function, and the R backsolve() command, give a least squares solution to the over-determined system X $\hat{\beta}$ = y where X and y take their values from the trees data in question 5.