**PROGRAMMING EXERCISES FOR R**

by

Nastasiya F. Grinberg & Robin J. Reed

*Programming Exercises for R Jun 7, 2013(21:45) Page i*

*Page ii Jun 7, 2013(21:45) Programming Exercises for R*

**Introduction**

These exercises were originally developed for a second year undergraduate module at the University of War- wick. The exercises are graded—the first two sheets are intended to get users thinking in terms of vector and matrix operations whilst the later sheets involve writing functions. Anyone is free to make a copy or multiple copies of this document or parts of this document.

**Contents**

EXERCISES*Exercises 1. Vectors ....................................................... 1 Exercises 2. Matrices ...................................................... 3 Exercises 3. Simple Functions ............................................... 5 Exercises 4. Harder Functions ............................................... 9 Exercises 5. Data frame, list, array and time series ............................ 13*

ANSWERS

*Answers to Exercises 1 ................................................... 18 Answers to Exercises 2 ................................................... 19 Answers to Exercises 3 ................................................... 21 Answers to Exercises 4 ................................................... 25 Answers to Exercises 5 ................................................... 30*

**Exercises 1. Vectors**

1. Create the vectors:

(a) (1*,*2*,*3*,...,*19*,*20) (b) (20*,*19*,...,*2*,*1) (c) (1*,*2*,*3*,...,*19*,*20*,*19*,*18*,...,*2*,*1) (d) (4*,*6*,*3) and assign it to the name tmp. For parts (e), (f) and (g) look at the help for the function rep.

(e) (4*,*6*,*3*,* 4*,*6*,*3*,...,*4*,*6*,*3) where there are 10 occurrences of 4. (f) (4*,*6*,*3*,* 4*,*6*,*3*,...,*4*,*6*,*3*,*4) where there are 11 occurrences of 4, 10 occurrences of 6 and 10 occurrences of 3. (g) (4*,*4*,...,*4*,* 6*,*6*,...,*6*,* 3*,*3*,...,*3) where there are 10 occurrences of 4, 20 occurrences of 6 and 30 occurrences of 3.

2. Create a vector of the values of *ex* cos(*x*) at *x* = 3*,*3*.*1*,*3*.*2*,...,*6.

3. Create the following vectors:

(a) (0*.*130*.*21*,*0*.*160*.*24*,...,*0*.*1360*.*234) (b)

(2*,* 22 2*,* 23 3*,...,* 225 25

)

4. Calculate the following:

(a)

∑100(*i*3 + 4*i*2). *i*=10(b)

∑25*i*=1

(2*ii* + 3*i*) *i*25. Use the function paste to create the following character vectors of length 30:

(a) ("label 1", "label 2", ....., "label 30").

Note that there is a single space between label and the number s. (b) ("fn1", "fn2", ..., "fn30").

In this case, there is no space between fn and the number following.

6. Execute the following lines which create two vectors of random integers which are chosen with replace-

ment from the integers 0, 1, *...*, 999. Both vectors have length 250.

set.seed(50) xVec <- sample(0:999, 250, replace=T) yVec <- sample(0:999, 250, replace=T) Suppose x = (*x*1*,x*2*,...,xn*) denotes the vector xVec and y = (*y*1*,y*2*,...,yn*) denotes the vector yVec. (a) Create the vector (*y*2 *− x*1*,...,yn − xn−*1).

(b) Create the vector

( cos(*x*sin(*y*1)

2)*,* cos(*x*sin(*y*2)

3)*,...,* sin(*y*cos(*xn−*1)

*n*)

).

(c) Create the vector (*x*1 + 2*x*2 *− x*3*,x*2 + 2*x*3 *− x*4*,...,xn−*2 + 2*xn−*1 *− xn*).

(d) Calculate

*n−*1∑*i*=1

*e−xxi* + *i*+1

10.

7. This question uses the vectors xVec and yVec created in the previous question and the functions sort,

order, mean, sqrt, sum and abs. (a) Pick out the values in yVec which are *>* 600. (b) What are the index positions in yVec of the values which are *>* 600?

*Programming Exercises for R Jun 7, 2013(21:45) Page 1*

*Page 2 Programming Exercises for R Jun 7, 2013(21:45)*

(c) What are the values in xVec which correspond to the values in yVec which are *>* 600? (By corre-

spond, we mean at the same index positions.) (d) Create the vector (*|x*1 *−*  ̄x*|*1*/*2*,|x*2 *−*  ̄x*|*1*/*2*,...,|xn −*  ̄x*|*1*/*2) where ̄x denotes the mean of the vector

x = (*x*1*,x*2*,...,xn*). (e) How many values in yVec are within 200 of the maximum value of the terms in yVec? (f) How many numbers in xVec are divisible by 2? (Note that the modulo operator is denoted %%.) (g) Sort the numbers in the vector xVec in the order of increasing values in yVec. (h) Pick out the elements in yVec at index positions 1*,*4*,*7*,*10*,*13*,....*

8. By using the function cumprod or otherwise, calculate

1 + 23 +

(23

45)

+

(23

67)

+ *···* +

(23 45 *···* 38394) 5

**Exercises 2. Matrices**

1. Suppose

A =

[ 1 1 3 5 2 6 *−*2 *−*1 *−*3] (a) Check that A3 = 0 where 0 is a 3 *×* 3 matrix with every entry equal to 0. (b) Replace the third column of A by the sum of the second and third columns.

2. Create the following matrix B with 15 rows:

B =



Calculate the 3 *×* 3 matrix BTB. (Look at the help for crossprod.)

3. Create a 6*×*6 matrix matE with every entry equal to 0. Check what the functions row and col return when

applied to matE. Hence create the 6 *×* 6 matrix:

0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 4. Look at the help for the function outer. Hence create the following patterned matrix:

0 1 2 3 4 1 2 3 4 5 2 3 4 5 6 3 4 5 6 7 4 5 6 7 8 5. Create the following patterned matrices. In each case, your solution should make use of the special form of the matrix—this means that the solution should easily generalise to creating a larger matrix with the same structure and should not involve typing in all the entries in the matrix.

(a)



10 *−*10 10 10 *−*10 10

*··· ··· ···* 10 *−*10 10

0 1 2 3 4 5 6 7 8 9 1 ... 2 ... 3 ... 4 ... 5 ... 6 ... 7 ... 8 ... 9 ... 0 1 2 3 4

1 2 3 4 0 2 3 4 0 1

0 ... 3 4 0 1 2

8 9 0 1 2 3 4 5 6 7 9 0 1 2 3 4 5 6 7 8 4 0 1 2 3(c)

 (b)

0 8 7 6 5 4 3 2 1 1 0 8 7 6 5 4 3 2 2 1 0 8 7 6 5 4 3 3 2 1 0 8 7 6 5 4 4 3 2 1 0 8 7 6 5 5 4 3 2 1 0 8 7 6 6 5 4 3 2 1 0 8 7 7 6 5 4 3 2 1 0 8 8 7 6 5 4 3 2 1 0

*Programming Exercises for R Jun 7, 2013(21:45) Page 3*

*Page 4 Programming Exercises for R Jun 7, 2013(21:45)*

6. Solve the following system of linear equations in five unknowns

*x*1 + 2*x*2 + 3*x*3 + 4*x*4 + 5*x*5 = 7 2*x*1 + *x*2 + 2*x*3 + 3*x*4 + 4*x*5 = *−*1 3*x*1 + 2*x*2 + *x*3 + 2*x*4 + 3*x*5 = *−*3 4*x*1 + 3*x*2 + 2*x*3 + *x*4 + 2*x*5 = 5 5*x*1 + 4*x*2 + 3*x*3 + 2*x*4 + *x*5 = 17 by considering an appropriate matrix equation Ax = y. Make use of the special form of the matrix A. The method used for the solution should easily generalise to a larger set of equations where the matrix A has the same structure; hence the solution should not involve typing in every number of A.

7. Create a 6 *×* 10 matrix of random integers chosen from 1, 2,*...*, 10 by executing the following two lines

of code:

set.seed(75) aMat <- matrix( sample(10, size=60, replace=T), nr=6) (a) Find the number of entries in each row which are greater than 4. (b) Which rows contain exactly two occurrences of the number seven? (c) Find those pairs of columns whose total (over both columns) is greater than 75. The answer should be a matrix with two columns; so, for example, the row (1*,*2) in the output matrix means that the sum of columns 1 and 2 in the original matrix is greater than 75. Repeating a column is permitted; so, for example, the final output matrix could contain the rows (1*,*2), (2*,*1) and (2*,*2). What if repetitions are not permitted? Then, only (1*,*2) from (1*,*2), (2*,*1) and (2*,*2) would be permit- ted.

8. Calculate

(a)

∑20*i*=1

∑5*j*=1

*i*4 (3 + *j*) (b) (Hard)

∑20∑5*i*4

*i*=1

*j*=1

(3 + *ij*) (c) (Even harder!)

∑10∑*ii*4

*i*=1

*j*=1

(3 + *ij*)

**Exercises 3. Simple Functions**

1. (a) Write functions tmpFn1 and tmpFn2 returns the vector (*x*1*,x*22*,...,xnn*such that if xVec is the vector (*x*1*,x*2*,...,xn*), then tmpFn1(xVec)

) and tmpFn2(xVec) returns the vector

(*x*1*, x*2 22*,..., xn*

*nn*).

(b) Now write a function tmpFn3 which takes 2 arguments x and n where x is a single number and n is a

strictly positive integer. The function should return the value of

1 + *x*1 + *x*2 2+ *x*3 3+ *···* + *xn n*2. Write a function tmpFn(xVec) such that if xVec is the vector x = (*x*1*,...,xn*) then tmpFn(xVec) returns

the vector of moving averages: Try out your function; *x*1 for + *x*example, 3 2 + *x*3

*,* try *x*2 tmpFn( + *x*3 3 + c(1:5,6:1) *x*4

*, ..., xn−*2 + *xn−*1 + *xn*

). 3 3. Consider the continuous function*f*(*x*) =

*x*2 + 2*x* + 3 if *x <* 0

*x* + 3 if 0 *≤ x <* 2 *x*2 + 4*x −* 7 if 2 *≤ x*. Write a function tmpFn which takes a single argument xVec. The function should return the vector of values of the function *f*(*x*) evaluated at the values in xVec. Hence plot the function *f*(*x*) for *−*3 *<x<* 3.

4. Write a function which takes a single argument which is a matrix. The function should return a matrix

which is the same as the function argument but every odd number is doubled. Hence the result of using the function on the matrix

[ 1 1 3 5 2 6 *−*2 *−*1 *−*3] should be: [ 2 2 6 10 2 6 *−*2 *−*2 *−*6] *Hint:* First try this for a specific matrix on the Command Line.

5. Write a function which takes 2 arguments n and k which are positive integers. It should return the *n × n*

matrix: 

*k* 1 0 0 *···* 0 0 1 0 0 *···* 0 *k* 1 0 *···* 0 1 *k* 1 *···* 0 0 1 *k ···* 0 *···* 0 *···* 0 *···* 0 *··· ··· ··· k* 0

0 0

*···* 1



0 0 0 0 *···* 1 *k Hint:* First try to do it for a specific case such as *n* = 5 and *k* = 2 on the Command Line.

6. Suppose an angle *α* is given as a positive real number of degrees.

If 0 *≤ α <* 90 then it is quadrant 1. If 90 *≤ α <* 180 then it is quadrant 2. If 180 *≤ α <* 270 then it is quadrant 3. If 270 *≤ α <* 360 then it is quadrant 4. If 360 *≤ α <* 450 then it is quadrant 1. And so on. Write a function quadrant(alpha) which returns the quadrant of the angle *α*.

*Programming Exercises for R Jun 7, 2013(21:45) Page 5*

*Page 6 Programming Exercises for R Jun 7, 2013(21:45)*

7. (a) Zeller’s congruence *f* is = the ([2*.*6*m* formula:

*−* 0*.*2] + *k* + *y* + [*y/*4] + [*c/*4] *−* 2*c*) mod 7 where [*x*] denotes the integer part of *x*; for example [7*.*5] = 7. Zeller’s congruence returns the day of the week *f* given:

*k* = the day of the month, *y* = the year in the century *c* = the first 2 digits of the year (the century number) *m* = the month number (where January is month 11 of the preceding year, February is month 12 of the preceding year, March is month 1, etc.) For example, the date 21*/*07*/*1963 has *m* = 5, *k* = 21, *c* = 19, *y* = 63; whilst the date 21*/*2*/*1963 has *m* = 12, *k* = 21, *c* = 19 and *y* = 62. Write a function weekday(day,month,year) which returns the day of the week when given the nu- merical inputs of the day, month and year. Note that the value of 1 for *f* denotes Sunday, 2 denotes Monday, etc. (b) Does your function work if the input parameters day, month and year are vectors with the same length

and with valid entries?

8. (a) Suppose *x*0 = 1 and *x*1 = 2 and*xj* = *xj−*1 + *xj−*1 2for *j* = 1, 2, *...*.

Write a function testLoop which takes the single argument *n* and returns the first *n −*1 values of the sequence *{xj}j≥*0: that means the values of *x*0*,x*1*,x*2*,...,xn−*2. (b) Now write a function testLoop2 which takes a single argument yVec which is a vector. The function

should return ∑*nj*=1

*ej*

where *n* is the length of yVec.

9. *Solution of the difference equation xn* = *rxn−*1(1 *− xn−*1)*, with starting value x*1*.*

(a) Write a function quadmap( start, rho, niter ) which returns the vector (*x*1*,...,xn*) where *xk* =

*rxk−*1(1 *− xk−*1) and niter denotes *n*, start denotes *x*1, and rho denotes *r*. Try out the function you have written:

*•* for *r* = 2 and 0 *< x*1 *<* 1 you should get *xn →* 0*.*5 as *n → ∞*.

*•* try tmp <- quadmap(start=0.95, rho=2.99, niter=500) Now switch back to the Commands window and type:

plot(tmp, type="l") Also try the plot plot(tmp[300:500], type="l") (b) Now write a function which determines the number of iterations needed to get *| xn − xn−*1 *|<* 0*.*02. So this function has only 2 arguments: start and rho. (For start=0.95 and rho=2.99, the answer is 84.)

10. (a) Given a vector (*x*1*,...,xn*), the sample autocorrelation of lag *k* is defined to be

*rk* = ∑Thus

*ni*=*k*+1∑(*xni i*=1*−*  ̄*x*)(*xi−k* (*xi −*  ̄*x*)2 *−*  ̄*x*) *r*1 =

∑*ni*=2∑(*xni*=1*i −* (*x* ̄*x*)(*xi − i−*1 ̄*x*)2 *−*  ̄*x*) = (*x*2 *−*  ̄*x*)(*x*1 *−*  ̄*x*) ∑+ *ni*=1*···* (*xi* + *−* (*x* ̄*x*)*n* 2 *−*  ̄*x*)(*xn−*1 *−*  ̄*x*) Write a function tmpFn(xVec) which takes a single argument xVec which is a vector and returns a

*Exercises 3 Jun 7, 2013(21:45) Page 7*

list of two values: *r*1 and *r*2. In particular, find *r*1 and *r*2 for the vector (2*,*5*,*8*,...,*53*,*56). (b) (Harder.) Generalise the function so that it takes two arguments: the vector xVec and an integer k

which lies between 1 and *n −* 1 where *n* is the length of xVec. The function should return a vector of the values (*r*0 = 1*,r*1*,...,rk*). If you used a loop to answer part (b), then you need to be aware that much, much better solutions are possible—see exercises 4. (Hint: sapply.)

*Page 8 Programming Exercises for R Jun 7, 2013(21:45)*

**Exercises 4. Harder functions**

1. Suppose we are given xVec which represents the vector (*x*1*,...,xn*) and yVec which represents the vector

(*y*1*,...,ym*). Suppose further that zVec represents the vector (*z*1*,...,zn*) where

*z*1 = number(*yj < x*1) *z*2 = number(*yj < x*2) *... zn* = number(*yj < xn*) Formally, if *I* denotes the indicator function, then

*zk* =

∑*mj*=1

*I*(*yj < xk*) for *k* = 1*,* 2*, ..., n*

(a) By using the function outer, write a function which takes the arguments xVec and yVec and returns

the vector zVec. (b) Repeat part (a) but use sapply instead of outer., (b) Now repeat part (a) but use vapply instead of outer or sapply. (d) Investigate how the functions when one or both of the arguments is a vector with length 0. What if ei- ther or both arguments are matrices? *Always check your functions return sensible values whatever the values of the input parameters. Inserting checks on the values of input parameters is often necessary.* (e) Investigate the relative speed of your solutions by using system.time.

2. (a) Suppose matA is a matrix containing some occurrences of NA. Pick out the submatrix which consists of all columns which contain no occurrence of NA. So the objective is to write a function which takes a single argument which can be assumed to be a matrix and returns a matrix. (b) Now write a function which takes a single argument which can be assumed to be a matrix and returns the submatrix which is obtained by deleting every row and column from the input matrix which contains an NA.

3. *The empirical copula.*

Suppose we are given two data vectors (*x*1*,...,xn*) and (*y*1*,...,yn*). Then the empirical copula is the function *C*: [0*,*1] *×* [0*,*1] *→* [0*,*1] defined by

*C*(*u, v*) = *n*

1∑*j*=1

*n*( *ri*

*n* + 1 *≤ u, n si*

+ 1 ) *≤ v*where (*r*1*,...,rn*) denotes the vector of ranks of (*x*1*,...,xn*) and (*s*1*,...,sn*) denotes the vector of ranks of (*y*1*,...,yn*). For example, if (*x*1*,x*2*,x*3*,x*4) = (7*,*3*,*1*,*4) then (*r*1*,r*2*,r*3*,r*4) = (4*,*2*,*1*,*3), because *x*1 = 7 is the largest and hence *r*1 = 4; *x*2 = 3 which is the second largest when the *x*-values are ranked in increasing size and hence *r*2 = 2, etc. The supplied function rank returns the vector of ranks of the input vector. (a) Write a function called empCopula which takes four arguments: u, v xVec and yVec. You can assume that the values of u and v lie in [0*,*1] and xVec and yVec are numeric vectors with equal non-zero lengths. (b) Of course, users of R legitimately expect that all functions will work on vectors. In particular, users will wish to plot the empirical copula and this involves calculating its value at many points (*u, v*). Does the function you gave as the answer to part (a) work if u and v are numeric vectors with the same length and with all values lying in [0*,*1]? If not, can you write a function which does cope with that situation?

4. Experiment with different ways of defining a function which calculates the following double sum for any

value of *n*.

*f*(*n*) =

*I*

∑*ni*=1

∑*rs*=1

*s*2 10 + 4*r*3 For each function you create, time how quickly it executes by using the function system.time.

*Programming Exercises for R Jun 7, 2013(21:45) Page 9*

*Page 10 Programming Exercises for R Jun 7, 2013(21:45)*

(a) First use a loop within a loop. (b) Write a function funB which uses the functions row and col to construct a matrix with suitable entries

so that the sum of the matrix gives the required answer. (c) Write a function funC which uses the function outer to construct a matrix with suitable entries so that

the sum of the matrix gives the required answer. (d) Create a function which takes a single argument r and calculates

∑*rs*=1

*s*2 10 + 4*r*3 Then write a function funD which uses sapply to calculate the double sum. Note that sapply is just a combination of unlist and lapply. Is there any increase in speed gained by using this information (funE)? (e) Write a function which takes two arguments r and s and calculates

*I*(*s ≤ r*)*s*2 10 + 4*r*3 where *I* denotes the indicator function. Then write a function funF which calculates the double sum by using mapply to calculate all the individual terms.

Which is the fastest function?

5. *The waiting time of the n*th *customer in a single server queue.* Suppose customers labelled *C*0, *C*1, *...*,*Cn* arrive at times *τ* = 0, *τ*1, *...*, *τn* for service by a single server. The interarrival times *A*1 = *τ*1 *− τ*0, *...*, *An* = *τn −τn−*1 are independent and identically distributed random variables with the exponential density

*λae−λax* for *x ≥* 0. The service times *S*0, *S*1, *...*, *Sn* are independent and identically distributed random variables which are also independent of the interarrival times with the exponential density

*λse−λsx* for *x ≥* 0. Let *Wj* denote the waiting time of customer *Cj*. Hence customer *Cj* leaves at time *τj* + *Wj* + *Sj*. If this time is greater than *τj*+1 then the next customer, *Cj*+1 must wait for the time *τj* + *Wj* + *Sj − τj*+1. Hence we have the recurrent relation

*W*0 = 0 *Wj*+1 = max*{*0*,Wj* + *Sj − Aj*+1*}* for *j* = 0, 1, *...*, *n −* 1 (a) Write a function queue(n, aRate, sRate) which simulates one outcome of *Wn* where aRate denotes

*λa* and sRate denotes *λs*. Try out your function on an example such as queue(50,2,2) (b) Now suppose we wish to simulate many outcomes of *Wn* in order to estimate some feature of the distribution of *Wn*. Write a function which uses a loop to repeatedly call the function in part (a) to calculate *Wn*. Then write another function which uses sapply (or replicate) to call the function created in part (a). Compare the speed of the two functions by using system.time. (c) Can we do any better? Try writing a vectorised form of the basic recurrence relation *Wj*+1 = max*{*0*,Wj* + *Sj − Aj*+1*}* where *Wj* is treated as a vector. Compare the speed of this new function with the two answers to the previous part.

6. *A random walk.* A symmetric simple random walk starting at the origin is defined as follows. Suppose

*X*1, *X*2, *...* are independent and identically {+1 distributed random variables with the distribution with probability 1*/*2

*−*1 with probability 1*/*2 Define the sequence *{Sn}n≥*0 by*S*0 = 0

*Sn* = *Sn−*1 + *Xn* for *n* = 1, 2, *...* Then *{Sn}n≥*0 is a symmetric simple random walk starting at the origin. Note that the position of the walk at time *n* is just the sum of the previous *n* steps: *Sn* = *X*1 + *···* + *Xn*.

*Exercises 4 Jun 7, 2013(21:45) Page 11*

(a) Write a function rwalk(n) which takes a single argument n and returns a vector which is a realisation

of (*S*0*,S*1*,...,Sn*), the first n positions of a symmetric random walk starting at the origin. Hint: the code sample( c(-1,1), n, replace=TRUE, prob=c(0.5,0.5) ) simulates *n* steps. (b) Now write a function rwalkPos(n) which simulates one occurrence of the walk which lasts for a

length of time *n* and then returns the length of time the walk spends above the *x*-axis. (Note that a walk with length 6 and vertices at 0, 1, 0, -1, 0, 1, 0 spends 4 units of time above the axis and 2 units of time below the axis.) (c) Now suppose we wish to investigate the distribution of the time the walk spends above the *x*-axis.

This means we need a large number of replications of rwalkPos(n). Write two functions: rwalkPos1(nReps,n) which uses a loop and rwalkPos2(nReps,n) which uses replicate or sapply. Compare the execution times of these two functions. (d) In the previous question on the waiting time in a queue, a very substantial increase was obtained by

using a vector approach. Is that possible in this case?

*Page 12 Programming Exercises for R Jun 7, 2013(21:45)*

**Exercises 5. Data frame, list, array and time series**

1. *Time Series.* The code

ts(datVec, start=c(1960,3), frequency=12) creates a time series with monthly observations (frequency=12), with first observation in March 1960 (start=c(1960,3)) and with values specified in the vector datVec.

Suppose *z*1, *z*2, *...*, *zn* is a time series. Then we define the exponentially weighted moving average of this time series as follows: select a starting value *m*0 and select a discount factor *δ*. Then calculate *m*1, *m*2, *...*, *mn* recursively as follows: for *t* = 1, 2, *...*, *n*

*et* = *zt − mt−*1 *mt* = *mt−*1 + (1 *− δ*)*et* (a) Write a function tsEwma(tsDat, m0=0, delta=0.7) where tsDat is a time series, m0 is the starting

value *m*0 and delta is *δ*. The function should return *m*1, *m*2, *...*, *mn* in the form of a time series. (b) In general, *looping over named objects is much slower than looping over objects which do not have names*. This principle also applies to time series: looping over a vector is much quicker than looping over a time series. Use this observation to improve the execution speed of your function which should still return a time series. Investigate the difference in speed between the functions in parts (a) and (b) by using the function system.time.

2. (a) Write a function, called myListFn, which takes a single argument *n* and implements the following

algorithm:

1. Simulate *n* independent numbers, denoted x = (*x*1, *x*2, *...*, *xn*), from the *N*(0*,*1) distribution. 2. Calculate the mean x = ∑*nj*=1 *xj*/*n*. 3. If x *≥* 0, then simulate *n* independent numbers, denoted y = (*y*1, *y*2, *...*, *yn*), from the exponential

density with mean x. If x *<* 0, then simulate *n* independent numbers, denoted z = (*z*1, *z*2, *...*, *zn*), from the exponential density with mean *−*x. Set y = (*y*1, *y*2, *...*, *yn*) = *−*z. 4. Calculate *k* which is the number of *j* with *|yj| > |xj|*. 5. Return the list of x, y and *k* with names xVec, yVec and count respectively. (b) Execute the following lines and check the format of the answers:

lapply( rep(10,4), myListFn ) sapply( rep(10,4), myListFn ) Note that sapply is effectively lapply followed by simplify2array. If myListFn has no arguments, then similar results can be obtained with replicate(4, myListFn()) and replicate(4, myListFn(), simplify=F).

Now for a simulation study. Use lapply to call the function myListFn with *n* = 10 for 1,000 times. So the output consists of 10,000 values for *x* denoted *{xi,j*:*i* = 1*,*2*,...,*1*,*000;*j* = 1*,*2*,...,*10*}*; 10,000 values for *y* denoted *{yi,j*:*i* = 1*,*2*,...,*1*,*000;*j* = 1*,*2*,...,*10*}*; and 1,000 values for *n* denoted *n*1*,n*2*,...,n*1000. Denote the output by myList. This output is used in the remaining parts of this question.

(c) Extract all the vectors with the name yVec. The result should be a list of 1000 vectors. (d) Extract all the vectors with the name yVec. The result should be a 10 *×*1000 matrix with one column

for each of the vectors yVec. (e) Create a list which is identical to myList but all the components called count have been removed. (f) Pick out those lists in myList which are such that count is greater than 2.

*Programming Exercises for R Jun 7, 2013(21:45) Page 13*

*Page 14 Programming Exercises for R Jun 7, 2013(21:45)*

3. *This question uses the list* myList *created in the previous question.*

(a) Calculate the vector which consists of the values of *xi*1 + 2*xi*2 + *···* + 10*xi,*10 *yi*1 + 2*yi*2 + *···* + 10*yi,*10 for *i* = 1, 2, *...*, 1,000. (b) Calculate the 1*,*000 *×* 10 matrix with entries *xij − yij* for *i* = 1, 2, *...*, 1,000 and *j* = 1, 2, *...*, 10. (c) Find the value of ∑∑1000 *i*=1 1000 *i*=1 *nixiyi,*2 *i,*2 = *n*1*yx*12 12 + + 2*xn*2*y*22 22 + + *··· ···* + + 1000*xn*1000*y*1000*,*2

1000*,*2

4. *Arrays.* In order to test the functions in this question, you will need an array. We can create a three-

dimensional test array as follows:

testArray <- array( sample( 1:60, 60, replace=F), dim=c(5,4,3) ) The above line creates a 5 *×* 4 *×* 3 array of integers which can be represented in mathematics by:

*{xi,j,k* : *i* = 1*,*2*,...,*5; *j* = 1*,*2*,*3*,*4; *k* = 1*,*2*,*3*}*

Note that apply(testArray, 3, tmpFn) means that the index *k* is retained in the answer and the function tmpFn is applied to the 3 matrices: *{xi,j,*1 : 1 *≤ i ≤* 5; 1 *≤ j ≤* 4*}*, *{xi,j,*2 : 1 *≤ i ≤* 5; 1 *≤ j ≤* 4*}* and *{xi,j,*3 : 1 *≤ i ≤* 5; 1 *≤ j ≤* 4*}*. Similarly apply(testArray, c(1,3), tmpFn) means that indices *i* and *k* are retained in the answer and the function tmpFn is applied to 15 vectors: *{x*1*,j,*1 : 1 *≤ j ≤* 4*}*, *{x*1*,j,*2 : 1 *≤ j ≤* 4*}*, etc. The expression apply(testArray, c(3,1), tmpFn) does the same calculation but the format of the an- swer is different: when using apply in this manner, it is always worth writing a small example in order to check that the format of the output of apply is as you expect.

(a) Write a function testFn which takes a single argument which is a 3-dimensional array. If this array is denoted *{xi,j,k* : *i* = 1*,*2*,...,d*1; *j* = 1*,*2*,...,d*2; *k* = 1*,*2*,...,d*3*}*, then the function testFn returns a list of the *d*1 *× d*2 *× d*3 matrix *{wi,j,k}* and the *d*2 *× d*3 matrix *{zj,k}* where

*wi,j,k* = *xi,j,k −*

min *i*=1 *d*1*xi,j,k* and *zj,k* =

∑*d*1*i*=1

*xi,j,k −* max *i*=1 *d*1*xi,j,k*

(b) Now suppose we want a function testFn2 which returns the *d*2 *× d*3 matrix *{zj,k}* where

*zj,k* =

∑*d*1*i*=1

*xki,j,k*

5. In this question you will study the matrix A given by

A =

*x*1 *y*1 *x*2 *y*2 *x*3 *y*3 *x*4 *y*4 *x*5 *y*5

 = 

0 0 1 3 2 0 4*/*9 4*/*3 14*/*9 4*/*3 This matrix consists of 5 coordinates which make up the letter ‘A’ in the two-dimensional Euclidean plane. The function

drawA <- function(X) {

lines(X[1:3,1], X[1:3,2]) lines(X[4:5,1], X[4:5,2]) } adds a graph of ‘A’ to an existing plot, when provided with a correct matrix of coordinates. Use plot(c(-10,10), c(-10,10), ann=F, type=’n’) to create an empty graph space of appropriate size.

*Exercises 5 Jun 7, 2013(21:45) Page 15*

(a) Given an *n ×* 2 matrix X, we can move the shape represented by the coordinates in X by *a* in the

*x*-direction and by *b* in the *y*-direction by adding to X the *n ×* 2 matrix

S*a,b* =

*a* ... *b* ...

*a b* Write a function shift(X,a,b) which, given an *n ×* 2 matrix X of coordinates, returns X + S*a,b*. Try it out on A together with drawA to check how your function is working. (b) Given an *n ×* 2 matrix X of coordinates we can rotate the shape represented by the coordinates in X

anticlockwise about the origin by *r* radians by multiplying it by the matrix

R*r* =

( cos*r* sin*r −*sin*r* cos*r*

)

Write a function rotate(X,r) which takes an *n ×*2 matrix X as an argument and returns XR*r*. Try it out on A together with drawA to check how your function is working. (c) Create a 5 *×* 2 *×* 25 array arrayA, such that arrayA[,,1] is equal to A = AR0 and arrayA[,,i] is

equal to

AR*i−*1

2*π*24 = AR 2*π*24 (*i−*1) for *i* = 2, 3, *...*, 25 i.e. this We the *ith* layer is the same can think of of arrayA is equal to A rotated as rotating A anti-clockwise each matrix arrayA[,,i] as by the anti-clockwise 2*π*24position (*i −* 1) of radians.)

letter by 2*π*24 ‘A’ radians at time (*i i*.

*−* 1) times. (Note that

(1) Now plot the resulting 25 instances of letter ‘A’ all on one graph. (2) Plot all 25 positions of the vertex of ‘A’ on one plot. (Remember that the coordinates of the vertex

are given by the second row of each 5 *×* 2 ‘position’ matrix.) (3) Plot the *x*-coordinate of the vertex of ‘A’ against time. Now, for something a little different, let us create an animation of our rotating ‘A’. For that you will need the ‘animation’ package; on Windows, download it by clicking on ‘Packages’, then on ‘Install package(s)’ and choosing ‘animation’ in the window that appears on the screen. Then to install the library click on ‘Packages’, ‘Load package’, and choose ‘animation’.1 Once the package is installed and the library loaded, enter

oopt = ani.options(interval = 0.2, nmax = 25) for(i in 1:ani.options("nmax")) {

plot(c(-10,10), c(-10,10), ann=F, type=’n’) drawA(arrayA[,,i]) ani.pause() } (d) Multiplying any *n ×* 2 matrix X of coordinates by a matrix

T*a,b* =

(*a* 0 0 *b*

)

stretches the shape represented by X by *a* in the *x*-direction and by *b* in the *y*-direction. If *a* = *b >* 1, then X is enlarged by *a*; if *a* = *b <* 1, then X is shrunk by *a*. Write a function scale(X,a,b) which, given an *n×*2 matrix X of coordinates, returns XT*a,b*. Hence, or otherwise, transform all the instances of ‘A’ in arrayA by T2*,*3. Plot the results on the same graph as results from (c) part (1) and/or create an appropriate animation. (e) (Harder) Create a 5 *×* 2 *×* 25 array arArandom, where arArandom[,,1] is equal to A and for each *i* = 2*, ...,*25 the array slice arArandom[,,i] is obtained by first scaling, then rotating and moving arArandom[,,i-1] by random amounts. Plot the results and/or create an appropriate animation. Hint: runif(1,a,b) generates a random number uniformly distributed in the interval (*a, b*).

1 The method for installing packages is system dependent and you will need to consult your local documentation.

*Page 16 Programming Exercises for R Jun 7, 2013(21:45)*

**ANSWERS**

**Answers to Exercises 1**

1. (a) 1:20 (b) 20:1 (c) c(1:20,19:1) (d) tmp <- c(4,6,3) It is good style to use <- for assignment and to leave a space on both sides of the assignment operator <-. (e) rep(tmp,10) (f) rep(tmp,l=31) (g) rep(tmp,times=c(10,20,30))

2. tmp <- seq(3,6,by=0.1)

exp(tmp)\*cos(tmp)

3. (a) (0.1^seq(3,36,by=3))\*(0.2^seq(1,34,by=3))

(b) (2^(1:25))/(1:25) 4. (a)tmp <- 10:100

sum(tmp^3+4\*tmp^2) (b)tmp <- 1:25

sum((2^tmp)/tmp + 3^tmp/(tmp^2))

5. (a) paste("label", 1:30)

(b) paste("fn", 1:30,sep="") 6. (a) yVec[-1] - xVec[-length(xVec)]

(b) sin(yVec[-length(yVec)]) / cos(xVec[-1]) (c) xVec[-c(249,250)] + 2\*xVec[-c(1,250)]-xVec[-c(1,2)] or, for an answer which works whatever the length of xVec,

xVecLen <- length(xVec) xVec[-c(xVecLen-1,xVecLen)] + 2\*xVec[-c(1,xVecLen)] - xVec[-c(1,2)] (d) sum(exp(-xVec[-1])/(xVec[-length(xVec)]+10)) 7. (a) yVec[yVec>600]

(b) (1:length(yVec))[yVec>600] *or* which(yVec>600) (c) xVec[yVec>600] (d) sqrt(abs(xVec-mean(xVec))) (e) sum( yVec>max(yVec)-200 ) (f) sum(xVec%%2==0) (g) xVec[order(yVec)] (h) yVec[c(T,F,F)] 8. 1+sum(cumprod(seq(2,38,b=2)/seq(3,39,b=2)))

*Page 18 Jun 7, 2013(21:45) Programming Exercises for R*

**Answers to Exercises 2**

1. (a)( tmp <- matrix( c(1,5,-2,1,2,-1,3,6,-3),nr=3) )

tmp%\*%tmp%\*%tmp The brackets round the first line ensure the matrix tmp is displayed so that we can check that it has been entered correctly. (b) tmp[,3] <- tmp[,2]+tmp[,3]

2. tmp <- matrix(c(10,-10,10), b=T, nc=3, nr=15)

t(tmp)%\*%tmp *or* crossprod(tmp)

3. matE <- matrix(0,nr=6,nc=6)

matE[ abs(col(matE)-row(matE))==1 ] <- 1

4. outer(0:4,0:4,"+") 5. (a) outer(0:4,0:4,"+")%%5

(b) outer(0:9,0:9,"+")%%10 (c) outer(0:8,0:8,"-")%%9 Other solutions are possible: for example matrix(0:4+rep(0:4,times=rep(5,5)),nc=5) also solves part (a). 6. We have

x =

*x*1*x*2*x*3*x*4*x*5

*,* y = 

71 2 3 4 5 *−*1*−*35 and A = 2 3 4 1 2 3 2 3 4 1 2 3 2 1 2 5 4 3 2 1 17 Appropriate *R* code is

yVec <- c(7,-1,-3,5,17) AMat <- matrix(0,nr=5, nc=5) AMat <- abs(col(AMat)-row(AMat))+1 To solve for x, calculate A*−*1y, by using the function solve to find the inverse of A. Either solve(AMat)%\*%yVec which returns the values in x as a matrix with one column; or solve(AMat,yVec) which returns the values in x as a vector or solve(AMat,matrix(yVec,nc=1) ) which returns the values in x as a matrix with one column. If the result of any of these three expressions is saved as xVec, then we can check the solution is correct by evaluating AMat%\*%xVec which returns the values in y as a matrix with one column in all three cases.

7. (a) apply(aMat, 1, function(x){sum(x>4)})

(b) which( apply(aMat,1,function(x){sum(x==7)==2}) ) (c) Here are two solutions:

aMatColSums <- colSums(aMat) cbind( rep(1:10,rep(10,10)), rep(1:10,10) ) [outer(aMatColSums,aMatColSums,"+")>75,] *or*aMatColSums <- colSums(aMat)

which( outer(aMatColSums,aMatColSums,"+")>75, arr.ind=T ) If we wish to exclude repeats, we can use code such as

aMatColSums <- colSums(aMat) logicalMat <- outer(aMatColSums,aMatColSums,"+")>75 logicalMat[lower.tri(logicalMat,diag=T)] <- F which(logicalMat, arr.ind=T)

*Programming Exercises for R Jun 7, 2013(21:45) Page 19*

*Page 20 Programming Exercises for R Jun 7, 2013(21:45)*

8. (a) sum( (1:20)^4 ) \* sum( 1/(4:8) ) *or* sum(outer((1:20)^4,4:8,"/"))

The answer is 639,215. (b) sum( (1:20)^4 / (3 + outer(1:20,1:5,"\*"))) The answer is 89,912.021. (c) sum( outer(1:10,1:10,function(i,j){ (i>=j)\*i^4/(3+i\*j) }) ) The answer is 6,944.7434.

**Answers to Exercises 3**

1. (a)tmpFn1 <- function(xVec)

{

xVec^(1:length(xVec)) }tmpFn2 <- function(xVec) {

n <- length(xVec) (xVec^(1:n))/(1:n) } (b)tmpFn3 <- function(x, n)

{

1 + sum((x^(1:n))/(1:n)) }Always try out your functions on simple examples where you know the answer: for example tmpFn1(1:3) should return the vector (1*,*4*,*27). Also, check extreme cases: what happens if xVec has length 0? Many functions require initial if statements which check that the values of the function arguments satisfy the design requirements of the function—for example checking that the value of nis strictly positive in tmpFn3. We have not included such code in our answers.

2. tmpFn <- function(xVec)

{

n <- length(xVec) ( xVec[ -c(n-1,n) ] + xVec[ -c(1,n) ] + xVec[ -c(1,2) ] )/3 }*or*tmpFn <- function(xVec)

{

n <- length(xVec) ( x[1:(n-2)] + x[2:(n-1)] + x[3:n] )/3 }Note that tmpFn( c(1:5,6:1) ) should return the vector (2*,*3*,*4*,*5*,*5*.*333*,*5*,*4*,*3*,*2).

3. tmpFn <- function(x)

{

ifelse(x < 0, x^2 + 2\*x + 3, ifelse(x < 2, x+3, x^2 + 4\*x - 7)) }tmp <- seq(-3, 3, len=100) plot(tmp, tmpFn(tmp), type="l")

4. tmpFn <- function(mat)

{

mat[mat%%2 == 1] <- 2 \* mat[mat%%2 == 1] mat } 5. For the specific case of *n* = 5 and *k* = 2:

tmp <- diag(2, nr = 5) tmp[abs(row(tmp) - col(tmp)) == 1] <- 1 tmp Now for the function for the general case:

tmpFn <- function(n, k)

*Programming Exercises for R Jun 7, 2013(21:45) Page 21*

*Page 22 Programming Exercises for R Jun 7, 2013(21:45)*

{

tmp <- diag(k, nr = n) tmp[abs(row(tmp) - col(tmp)) == 1] <- 1 tmp }

6. quadrant <- function(alpha)

{

1 + (alpha%%360)%/%90 }*or*quadrant2 <- function(alpha)

{

floor(alpha/90)%%4 + 1 }Both functions work on vectors, as any answer should!!

7. weekday <- function(day, month, year)

{

month <- month - 2 if(month <= 0) {

month <- month + 12 year <- year - 1 }cc <- year %/% 100 year <- year %% 100 tmp <- floor(2.6\*month - 0.2) + day + year + year %/% 4 + cc %/% 4 - 2 \* cc c("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")[1+tmp%%7] }The output of executing c( weekday(27,2,1997), weekday(18,2,1940), weekday(21,1,1963) ) is the vector "Thursday", "Sunday", "Monday".

Using if in the definition of weekday means that this function does *not* work on vectors. However, we can eliminate the if statement as follows.

weekday2 <- function(day, month, year) {

flag <- month <= 2 month <- month - 2 + 12\*flag year <- year - flag cc <- year %/% 100 year <- year %% 100 tmp <- floor(2.6\*month - 0.2) + day + year + year %/% 4 + cc %/% 4 - 2 \* cc c("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")[1+tmp%%7] }The output of executing weekday2( c(27,18,21), c(2,2,1), c(1997,1940,1963) ) where all three input parameters are vectors is the vector "Thursday", "Sunday", "Monday". Clearly both weekday and weekday2 need extra lines of code which check that the values given for day, month and year are valid.

8. (a)testLoop <- function(n)

{

xVec <- rep(NA, n-1) xVec[1] <- 1 xVec[2] <- 2 for( j in 3:(n-1) )

*Answers to Exercises 3 Jun 7, 2013(21:45) Page 23*

xVec[j] <- xVec[j-1] + 2/xVec[j-1] xVec } *Important.* The colon operator has a higher precedence than the arithmetic operators such as + or \* but lower precedence than ^. So always use brackets for constructs like 1:(n-1) or 1:(20^k) so that the meaning is obvious even to those whose memory is faulty. *Important.* The above function gives the wrong answer if called with n=3. Why? A line such as the following must be inserted:

if( n <4 ) stop("The argument n must be an integer which is at least 4.\n") (b) The following code is wrong. Why?

testLoop2 <- function(yVec) {

n <- length(yVec) sum( exp(1:n) ) }The function testLoop2 returns the value of *e*0 + *e*1 if the vector yVec has length 0. A correct function is

testLoop2 <- function(yVec) {

n <- length(yVec) sum( exp(seq(along=yVec)) ) }This function now returns the correct value of 0 if yVec has length 0. *Important.* Always use seq(along=x) rather than 1:length(x). 9. (a) For a question like this where the value of *xn* must be known before the value of *xn*+1 can be calculated,

it is necessary to use a loop. First create the space for the answer with the code xVec <- rep(NA, niter) and then fill in the values. Growing a vector inside a loop is very slow and inefficient. Initialising the vector xVec to NA rather than to 0 makes it easier to spot certain errors: for example, the error that the loop stops too early.

quadmap <- function(start, rho, niter) {

xVec <- rep(NA,niter) xVec[1] <- start for(i in 1:(niter-1)) {

xVec[i + 1] <- rho \* xVec[i] \* (1 - xVec[i]) }x } (b)quad2 <- function(start, rho, eps = 0.02)

{

x1 <- start x2 <- rho\*x1\*(1 - x1) niter <- 1 while(abs(x1 - x2) >= eps) {

x1 <- x2 x2 <- rho\*x1\*(1 - x1) niter <- niter + 1 }niter } 10. Values from the vector (*x*1 *−*  ̄*x, x*2 *−*  ̄*x,...,xn −*  ̄*x*) are used *three* times in the expression for *rk*: twice in the numerator and once in the denominator.. Therefore it is important to calculate this vector once and save its value for use in all three situations. A function definition should not, for example, contain more

*Page 24 Programming Exercises for R Jun 7, 2013(21:45)*

than one occurrence of the expression mean(xVec). Writing mean(xVec) more than once means that you are asking the programme to spend time calculating it more than once. (a)tmpAcf <- function(xVec)

{

xc <- xVec - mean(xVec) denom <- sum(xc^2) n <- length(x) r1 <- sum( xc[2:n] \* xc[1:(n-1)] )/denom r2 <- sum( xc[3:n] \* xc[1:(n-2)] )/denom list(r1 = r1, r2 = r2) } (b)tmpAcf <- function(x, k)

{

xc <- x - mean(x) denom <- sum(xc^2) n <- length(x) tmpFn <- function(j){ sum( xc[(j+1):n] \* xc[1:(n-j)] )/denom } c(1, sapply(1:k, tmpFn)) }

**Answers to Exercises 4**

1. (a)fun4q1a <- function(xVec, yVec){

colSums( outer(yVec, xVec, "<") ) } (b)fun4q1b <- function(xVec, yVec){

rowSums( sapply(yVec, FUN=function(y){y < xVec}) ) } (c)fun4q1c <- function(xVec, yVec){

rowSums( vapply(yVec, FUN=function(y){y<xVec}, FUN.VALUE=seq(along=xVec)) ) }And here is yet a fourth possible solution:

fun4q1d <- function(xVec,yVec) {

leny <- length(yVec) mat <- matrix(rep(xVec,leny), byrow=T, nrow=leny) apply( yVec<mat, 2, sum ) } (d) Both fun4q1b and fun4q1d fail if either xVec or yVec has length 0; but at least they do not give incorrect answers which would be far worse. Both fun4q1a and fun4q1d fail if xVec and yVec are matrices.

(e) We can perform a timing by lines such as the following:

rjr1 <- rnorm(10000) rjr2 <- rnorm(12000) system.time(fun4q1a(rjr1,rjr2)) system.time(fun4q1b(rjr1,rjr2)) system.time(fun4q1c(rjr1,rjr2)) system.time(fun4q1d(rjr1,rjr2)) The answer using vapply is the fastest.

2. (a)tmpFn <- function(mat){

mat[, !apply(is.na(mat), 2, any), drop = F] } (b)tmpFn2 <- function(mat){

mat[!apply(is.na(mat), 1, any), !apply(is.na(mat), 2, any), drop = F] } 3. (a) First attempt:

empCopula <- function( u, v, xVec, yVec ) {

n <- length(xVec) rVecN <- rank(xVec)/(n+1) sVecN <- rank(yVec)/(n+1) sum( (rVecN <= u) & (sVecN <= v) ) /n } (b) The answer to part (a) does not work if u and v are vectors, but here are three solutions which do. All three solutions are instructive. Suppose *u* = (*u*1*,u*2*,...,uk*) and *v* = (*v*1*,v*2*,...,vk*) and set *r′i* = *ri/*(*n* + 1) and *s′i* = *si/*(*n* + 1) for *i* = 1, 2, *...*, *n*.

*Programming Exercises for R Jun 7, 2013(21:45) Page 25*

*Page 26 Programming Exercises for R Jun 7, 2013(21:45)*

*First solution using* outer*.*

empCopula2 <- function( u, v, xVec, yVec ) {

n <- length(xVec) rVecN <- rank(xVec)/(n+1) sVecN <- rank(yVec)/(n+1) valuesN <- colSums( outer(rVecN, u, "<=")&outer(sVecN, v, "<=") ) cbind( uCoord = u, vCoord = v, empCop=valuesN/n ) } In the above solution, outer(rVecN, u, "<=") gives the *n × k* logical matrix 



Hence the code outer(rVecN, u, "<=")&outer( sVecN, v, "<=") gives the *n × k* logical matrix



*r*1 *′≤* ... *u*1 *···* ... *r*1 *′≤* ... *uk rn ′≤ u*1 *··· rn ′≤ uk*

(*r*1 *′≤ u*1)&(*s*... *′*1 *≤ v*1) *···* ... (*r*1 *′≤ uk*)&(*s*... *′*1 *≤ vk*) (*rn ′≤ u*1)&(*s′n ≤ v*1) *···* (*rn ′≤ uk*)&(*s′n ≤ vk*) and then we take the sums of the columns. *Second solution using* apply*.*

empCopula3 <- function( u, v, xVec, yVec ) {

n <- length(xVec) rVecN <- rank(xVec)/(n+1) sVecN <- rank(yVec)/(n+1) tempf <- function(uv){

sum( (rVecN <= uv[1]) \* (sVecN <= uv[2]) ) }valuesN <- apply( cbind(u,v), 1, tempf ) cbind( uCoord = u, vCoord = v, empCop=valuesN/n ) } In the above solution, the function tempf is applied to each row of the following *k ×* 2 matrix: 



*Third solution using* mapply*.*

empCopula4 <- function( u, v, xVec, yVec ) {

n <- length(xVec) rVecN <- rank(xVec)/(n+1) sVecN <- rank(yVec)/(n+1) valuesN <- mapply( FUN=function(u1,v1){ sum((rVecN<=u1)\*(sVecN<=v1)) }, u, v ) cbind( uCoord = u, vCoord = v, empCop=valuesN/n ) }The function mapply is a multivariate form of sapply. Thus

mapply( FUN=f, xVec, yVec ) returns a vector of (*f*(*x*1*,y*1)*,f*(*x*2*,y*2)*,...,f*(*xn,yn*)) where xVec is the vector (*x*1*,x*2*,...,xn*) and yVec is the vector (*y*1*,y*2*,...,yn*). The recycling rule is applied if necessary. There can be any number of vectors. Our experience is that mapply is slow. The output from all three functions is a matrix and looks like this

uCoord vCoord empCop [1,] 0.602 0.687 0.433

*u*... 1 *v*... 1 *uk vk*

*Answers to Exercises 4 Jun 7, 2013(21:45) Page 27*

[2,] 0.338 0.255 0.067 [3,] 0.738 0.794 0.600 4. (a)funA <- function (n)

{

su <- 0 for(r in 1:n) {

for(s in 1:r)

su <- su+s^2/(10+4\*r^3) }su } (b)funB <- function (n)

{

mat <- matrix(0, ncol=n, nrow=n) sum( (col(mat)^2)/(10+4\*row(mat)^3)\*(col(mat)<=row(mat)) ) } (c)funC <- function (n)

{

sum( outer(1:n,1:n,FUN=function(r,s){ (s<=r)\*(s^2)/(10+4\*r^3) }) ) } (d)funD <- function (n)

{

tmpfn <- function(r){sum(((1:r)^2)/(10+4\*r^3))} sum(sapply(1:n, FUN=tmpfn)) }funE <- function (n) {

tmpfn <- function(r){sum(((1:r)^2)/(10+4\*r^3))} sum(unlist(lapply(1:n, FUN=tmpfn))) } (e)funF <- function (n)

{

tmpf <- function(s,r){(s^2)/(10+4\*r^3)\*(s<=r)} sum(mapply(tmpf, rep(1:n, times=rep(n,n)), 1:n)) }The fastest are funE and funD, but funB and funC are also quite fast. The function funA is much slower and funF is even slower! 5. (a) Here are two possible solutions

queue1 <- function(n, aRate, sRate) {

w <- 0 for(i in 1:n){

w <- max(0, w+rexp(1,sRate)-rexp(1,aRate)) }w }queue2 <- function(n, aRate, sRate)

*Page 28 Programming Exercises for R Jun 7, 2013(21:45)*

{

w <- 0 s <- rexp(n, sRate) a <- rexp(n, aRate) for(i in 1:n){

w <- max(0, w+s[i]-a[i]) }w }Note that the second solution queue2 is considerably faster then the first.

(b)queueRep1 <- function (nReps, n, aRate, sRate)

{

wVec <- rep(NA, nReps) for(j in 1:nReps)

wVec[j] <- queue2(n, aRate, sRate) wVec }queueRep2 <- function (nReps, n, aRate, sRate) {

sapply( rep(n,nReps), queue2, aRate, sRate ) }or replicate(nReps, queue2(n,aRate,sRate)).

(c)queueRep3 <- function (nReps, n, aRate, sRate)

{

w <- rep(0, nReps) s <- matrix(rexp(n\*nReps, sRate), ncol=nReps) a <- matrix(rexp(n\*nReps, aRate), ncol=nReps) for(i in 1:n){

w <- pmax(0, w+s[i,]-a[i,]) }w }There is very little difference between the times of queueRep1 and queueRep2. However, the third method, queueRep3, is *considerably* quicker then the other two methods—over 10 times quicker! 6. (a)rwalk <- function(n)

{

c( 0, cumsum(sample( c(-1,1), n, replace=TRUE, prob=c(0.5,0.5))) ) } (b) This amount of time the walk spends above the *x*-axis is the same as the number of points in the vector (*s*0 + *s*1*,s*1 + *s*2*,...,sn−*1 + *sn*) which are greater than 0. Hence we get the function:

rwalkPos <- function(n) {

rw <- cumsum(c(0, sample( c(-1,1), n, replace=TRUE, prob=c(0.5,0.5)))) sum( (rw[-(n+1)] + rw[-1]) > 0 ) } (c)rwalkPos1 <- function(nReps, n)

{

results <- rep(NA, nReps)

*Answers to Exercises 4 Jun 7, 2013(21:45) Page 29*

for(i in 1:nReps)

results[i]<-rwalkPos(n) results }rwalkPos2 <- function(nReps, n) {

replicate( nReps, rwalkPos(n) ) } (d)rwalkPos3 <- function(nReps, n)

{

stepWalks <- matrix( sample( c(-1,1), n, replace=TRUE, prob=c(0.5,0.5)), nr=nReps ) for(j in 2:n)

stepWalks[,j] <- stepWalks[,j] + stepWalks[,j-1] stepWalks <- cbind(0, stepWalks) rowSums( stepWalks[,1:n] + stepWalks[,2:(n+1)]>0 ) }

In this case, there is very little difference between the speed of the 3 functions when nReps and n have small values—presumably because there is no vector form of cumsum which has been replaced by an explicit loop in the function rwalkPos3 above. For large values of nReps and n, the third version uses a lot of memory and will slow down when paging occurs; the replicate version uses less memory and so is generally preferable.

**Answers to Exercises 5**

1. (a)tsEwma <- function( tsDat, m0=0, delta=0.7)

{

n <- length(tsDat) mVec <- rep(NA,n+1) mVec[1] <- m0 for(j in 2:(n+1)){

mVec[j] <- (1-delta)\*tsDat[j-1] + delta\*mVec[j-1] }ts(mVec[-1], start=start(tsDat), frequency=frequency(tsDat)) } (b)tsEwma2 <- function( tsDat, m0=0, delta=0.7)

{

tsPars <- tsp(tsDat) tsDat <- c(tsDat) n <- length(tsDat) mVec <- rep(NA,n+1) mVec[1] <- m0 for(j in 2:(n+1)){

mVec[j] <- (1-delta)\*tsDat[j-1] + delta\*mVec[j-1] }ts(mVec[-1], start=tsPars[1], frequency=tsPars[3]) }For testing, we could try something like

tmp <- ts(rnorm(400000), start=c(1960,3), frequency=12) system.time(tsEwma2(tmp)) system.time(tsEwma(tmp)) On one of our computers, tsEwma2 took about one third the time of tsEwma. 2. (a)myListFn <- function(n)

{

xVec <- rnorm(n) xBar <- mean(xVec) yVec <- sign(xBar)\*rexp(n, rate=abs(1/xBar)) count <- sum( abs(yVec) > abs(xVec) ) list(xVec=xVec, yVec=yVec, count=count) } (b) The line myList <- lapply( rep(10,4), myListFn ) returns a list of 4 lists—one list for each call to myListFn. The line myMatrix <- sapply( rep(10,4), myListFn ) returns a 3 *×* 4 matrix—one row for xVec, one row for yVec and one row for count. Thus myMatrix[1,1] is a vector of length 10 consisting of the 10 values in xVec from the first call of myListFn.

(c) We first call myList <- lapply( rep(10,1000), myListFn ). Here are three equivalent answers:

lapply(myList, FUN=function(x){x[[2]]}) lapply(myList, FUN="[[", 2) lapply(myList, FUN="[[", "yVec")

(d) Here are six equivalent answers: sapply(myList, FUN="[[", 2) vapply(myList, FUN="[[", FUN.VALUE=rep(0,10), 2)

*Page 30 Jun 7, 2013(21:45) Programming Exercises for R*

*Answers to Exercises 5 Jun 7, 2013(21:45) Page 31*

sapply(myList, FUN=function(x){x[[2]]}) vapply(myList, FUN=function(x){x[[2]]}, FUN.VALUE=rep(0,10)) sapply(mList, FUN="[[", "yVec") vapply(myList, FUN="[[", FUN.VALUE=rep(0,10), "yVec") (e)myList2 <- lapply(myList, function(x){list(xVec=x$xVec, yVec=x$yVec)}) (f) This code picks out the indices of those lists which satisfy the condition:

which( unlist( lapply(myList, function(x){x[[3]]>2}) ) ) So this is an answer:

myList[which( unlist(lapply( myList, function(x){x[[3]]>2} )) )]

3. (a)partA <- sapply(myList, function(x){ sum(x$xVec\*(1:10))/sum(x$yVec\*(1:10)) })

(b) Here are 3 possible solutions:

myMat <- t(sapply( myList, function(x){x$xVec-x$yVec})) myMat2 <- matrix( unlist( lapply(myList, FUN="[[",1) ) -

unlist( lapply(myList, FUN="[[",2) ), nc=10, by=T ) myMat3 <- matrix( unlist(lapply(myList, function(x){x$xVec-x$yVec})), nc=10, by=T )

(c) Here is a quick solution using sapply:

sum(sapply(myList, function(x){x$xVec[2]})\*(1:1000)) /

sum(sapply(myList, function(x){x$yVec[2]})\*sapply(myList, function(x){x$count})) An alternative solution uses the fact that if a list has components with equal lengths then it can be converted into a data.frame—although this operation is slow. Hence data.frame(myList) is a 10*×*3000 data.frame. Columns 1, 4, 7, etc are for the vectors called xVec; columns 2, 5, 8, etc are for the vectors called yVec and columns 3, 6, 9, etc are for the values of count. Recycling is used—-hence every entry in column 3 is equal to *n*1, etc. Hence the following two lines

myDf <- data.frame(myList) myDf[2, seq(1,3000,by=3)] pick out the vector (*x*12*,x*22*,...,x*1000*,*2). Calculations are faster on matrices than data frames, so we proceed as follows:

myMat <- as.matrix(data.frame(myList)) names(myMat) <- NULL sum((1:1000) \* myMat[2,seq(1,3000,by=3)])/

sum(myMat[2, seq(3,3000,by=3)] \* myMat[2, seq(2,3000,by=3)]) The last line could be replaced by

sum( (1:1000) \* myMat[2,c(T,F,F)] )/sum( myMat[2,c(F,F,T)] \* myMat[2,c(F,T,F)] ) The intermediate step of converting to a matrix only gives a worthwhile reduction in time if a lot of such calculations are to be made—otherwise it is not sensible to include it.

4. (a) The code apply(testArray, c(2,3), min) returns a *d*2 *× d*3 matrix with entries *wj,k* where

*wj,k* =

min *i*=1 *d*1*xi,j,k* Hence the code

sweep(testArray, c(2,3), apply(testArray, c(2,3), min)) returns a *d*1 *× d*2 *× d*3 matrix with entries *wi,j,k* where

*wi,j,k* = *xi,j,k −*

min *i*=1 *d*1*xi,j,k* For the matrix with entries *{zj,k}* we just need

apply(testArray, c(2,3), sum) - apply(testArray, c(2,3), max) or, better

apply(testArray, c(2,3), FUN=function(x){ sum(x) - max(x)}) So our function is

testFn2 <- function(xArray)

*Page 32 Programming Exercises for R Jun 7, 2013(21:45)*

{

wArray <- sweep(testArray, c(2,3), apply(testArray, c(2,3), min)) zArray <- apply(testArray, c(2,3), FUN=function(x){ sum(x) - max(x)}) list(wArray=wArray, zArray=zArray) } (b) Now the code

tmp <- apply(testArray, c(1,2), FUN=function(x){ x^(1:length(x))}) returns the *d*3 *× d*1 *× d*2 array with entries *zm,n,r* = *xmn,r,m*. We now need to sum over the second coordinate of the *{zm,n,r}* and return the other two coordinates in the order (*r, m*) which is done by the following code:

apply(tmp, c(3,1), sum) Hence our function is

testFn <- function( xArray) {

apply( apply(xArray, c(1,2), FUN=function(x){x^(1:length(x))}), c(3,1), sum ) }

5. (a)shift <- function(X,a,b){

X[,1] <- X[,1] + a X[,2] <- X[,2] + b X } (b)rotate <- function(X,r){

X%\*%matrix(c(cos(r), -sin(r), sin(r), cos(r)), nrow = 2) } To try shift and rotate on matrix A create it via

A <- cbind(c(0,1,2,4/9,14/9), c(0,3,0,4/3,4/3)) We also use A thus created in the code that follows. (c) If your solution looks something like

arrayA <- array(0, dim=c(5,2,25)) for(i in 1:25){

arrayA[,,i] <- rotate(A,2\*pi\*(i-1)/24) } then look up function vapply and try to eliminate the loop from the code above. Here is an alternative:

arrayA<-vapply(1:25, FUN=function(i){

rotate(A,2\*pi\*(i-1)/24) }, matrix(0,nrow=5, ncol=2) )

(1)plot(c(-10,10), c(-10,10), ann=F, type=’n’)

for(i in 1:25)

drawA(arrayA[,,i]) orplot(c(-10,10), c(-10,10), ann=F, type=’n’)

invisible(sapply( 1:25, FUN=function(i){ drawA(arrayA[,,i]) } )) Note that the function invisible suppresses display of the output of sapply, since the output is NULL and we are only interested in the resulting plot.

(2)plot(arrayA[2,1,], arrayA[2,2,])

*Answers to Exercises 5 Jun 7, 2013(21:45) Page 33*

(3)plot(1:25, arrayA[2,1,])

(d)scale <- function(X,a,b){

X%\*%matrix(c(a,0,0,b), nrow=2) }arAscaled <- vapply(1:25,

FUN=function(i){

scale(arrayA[,,i],2,3) },matrix(0,nrow=5, ncol=2) )

plot(c(-10,10), c(-10,10), ann=F, type=’n’) invisible(sapply( 1:25, FUN=function(i){ drawA(arrayA[,,i]) } )) invisible(sapply( 1:25, FUN=function(i){ drawA(arAscaled[,,i]) } ))

(e) First, as before, create an empty array randomA of appropriate size and initiate layer 1 to A.

arArandom <- array(0, dim=c(5,2,25)) arArandom[,,1] <- A Now, since for *i* = 2*, ...,*25 each A[,,i] should depend on A[,,i-1] in a random manner, we cannot use vapply, but have to create a loop instead:

for(i in 2:25){

arArandom[,,i] <-

shift(

rotate(

scale(arArandom[,,i-1], runif(1,0.5,1.5),runif(1,0.5,1.5)), 2\*pi\*runif(1,-1,1) ), runif(1,-1,1), runif(1,-1,1) ) }Then create an animation:

oopt = ani.options(interval = 0.2, nmax = 25) for (i in 1:ani.options("nmax")){

plot(c(-10,10), c(-10,10), ann=F, type=’n’) drawA(arArandom[,,i]) ani.pause() }Limit values for scaling, rotation and shifting are arbitrary and you are encouraged to play around with them to see how they affect dynamics of ‘A’. (Make sure you create a large enough initial empty graph though!)