Homework I

General Homework Procedures

- Assignments must be submitted by 11:59 p.m. of the specified due date. Generally an assignment will be due in stages, so that no group accidentally falls behind.
- Submit using the **handin** program on CSIF. All registered students in the class have accounts there. SSH logins are available.
- Your writeups must be in LaTeX. See my quick tutorial.
- Package your materials into a **.tar** file, containing your **.tex** and **.pdf** files and any figure files. The PDF file will be your processed **.tex**. Your file for a CRAN project must be the CRAN source format, as a **.tar.gz** file.

  Make just one submission for the group. Name your **.tar** file under the specifications in Section 19.4, "Submitting Homework," of the ECS 132 Syllabus.
- You will be graded by me as a group, but I will ask each group member questions, both about the homework and about the course in general. Each student must be ready to answer questions about *any* part of the homework.
- For CRAN projects, see my overview. Your package must pass the "CRAN test" shown there. Make sure your documentation *exceeds* CRAN requirements -- you docs must be clear and grammatically correct, and you must have examples for every function.
- **Advice:**
  - Try to work *together* as much as possible. If you simply divide up the work among your team members, all of you will learn less and likely won't answer questions well during interactive grading.
  - In working on programming problems, take "baby steps"; don't try to do the entire project at once. Instead, work on a scaled-down version of the project first. Get it working, and **then** modify it, adding the other features.

CRAN Project

Stage I of the project, consisting of the discrete-time case, will be due Tuesday, January 19.

Here you will develop a CRAN-quality R package for the analysis of Markov chains, as follows:

- The package must handle both discrete- and continuous-time Markov chains.

- The package must handle the infinite state space case, via approximation.
- At a minimum, the package must offer the following computations:
    - Stationary distribution.
    - Submatrix of expected hitting times, with rows and columns specified by the user.
- Set up an R class '**mc**' (I personally like S3 classes, but it is up to you) containing the definition of the Markov chain, and the time category (discrete or continuous). The class will accommodate things like the stationary distribution being added later. Objects of this class will be input to your functions.
- A Markov chain definition consists of one of the following:
    - An R matrix **pijdef** specifying the transition matrix, for the discrete-time case.
    - A matrix as above, plus a vector **qidef** of holding-time rates, for the continuous-time case.
    - A function named **pijdef()**, having the call form

      ```
      pijdef(i,j,...)
      ```

      where **i** and **j** are row and column numbers (beginning at 1), and where the ellipsis is an R construct that allows the caller to throw in extra parameters, in either the discrete- or continuous-time cases. In the latter case, the user will also supply a function **qidef()** that similarly specifies the holding-time rates.

For example, suppose your function to find stationary distributions is named **stn()**, and your transition matrix is

```
0.5 0.5 0
0.5 0 0.5
0 0.5 0.5
```

You would create an instance of the "**mc**" class, say **mymc**, and then do the call **stn(mymc)**. You'd have two choices for what to put into **mymc**. One would be to do

```
> tmp <- matrix(rep(0.5,9),ncol=3)
> tmp[1,3] <- 0
> tmp[2,2] <- 0
> tmp[3,1] <- 0
> mymc$pijdef <- tmp
```

Another choice would be

```
> mymc$pijdef <- function(i,j,...) {
   if (i == 1 && j < 3 ||
       i == 2 && j != 2 ||
       i == 3 && j > 0)  return(0.5)
   0
}
```

For finite state space chains, the first method would typically be easier, but the second is needed at least in the infinite case.

- Try to use R idioms as much as possible. For example, have default values for your parameters, and try to have the code sense variable types rather than burdening the user with this. An example of the latter is R's **diag()**:

```
> m <- diag(3)  # integer argument
> m
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> m[1,3] <- 8
> m[2,2] <- 10
> m
     [,1] [,2] [,3]
[1,]    1    0    8
[2,]    0   10    0
[3,]    0    0    1
> diag(m)  # matrix argument
[1]  1 10  1
```

- You are welcome (and encouraged) to import other R packages, such as the recently released markovchain.

Math Problem A

This problem is part of Stage 1 of the homework, due January 19.

Items come in that need to be packed in boxes, in order of arrival. The maximum allowable weight per box is w_{max}; if an item would cause a box to go overweight, a new box must be started.

The weights $W_1, W_2, W_3,...$ of the items are independent and identically distributed (i.i.d.) with some distribution on $\{1,2,...,w_{max}\}$. Let X_n denote the total weight of items in the current box at "time" n, i.e. after the n^{th} item has been packed. Due to the i.i.d. assumption, the X_n obey the Markov property.

Do the following, with π denoting the stationary distribution of the chain:

- Derive an expression for the long-run mean number of items per box in terms of π.
- Derive an expression for the long-run mean weight per box in terms of π.
- Let Q denote the weight of the first item put into any box. Derive an expression for the probability mass function of Q, i.e. the values $P(Q = i)$, $i = 1,2,...,r$ in terms of π.
- Take w_{max} to be 10, and $P(W = i) = ci/10$, $i = 1,2,...,10$ for suitable c, i.e. the c that makes the probabilities sum to1. Find the values of the above expressions, and check via simulation.

Math Problem B

(Stage 2)

Recall the Bus Paradox in our book. The quantity of interest was D_t, the time until the next bus arrival, where t is the time we arrive at the bus stop. We wished to determine the long-run distribution of D_t. If we are dealing with continous random variables, that long-run density is

$f(w) = (1 - F_L(w)) / EL$

where L is the time between buses in general (not at the times you arrive). And remarkably, if we look at the time A_t since the *last* bus, we get the same distribution.

If L is a discrete random variable, the result is the same, except that instead of a density we have a probability mass function. In this problem, you will derive the above equation for this discrete case, using Markov chains, for the case of the time since the last bus.

It will be easier to think of light bulbs. As soon as one burns out, we replace it. Let Q be the length of time the current bulb has been burning, so Q = 0 at the time of a replacement. If Q = 8, say, the bulb has already lasted 8 units of time, but if it burns out at the next time step, Q immediately becomes 0 again.

Math Problem C

(Stage 2)

Here you will model queuing in a call center, with the following characterisics:

- There are s servers who take calls, and a buffer size of b calls. If a call arrives when b calls are already in the system (being served or waiting on hold), that call is declined, with a message "Try again later." From a modeling point of view, these calls are simply lost.
- A proportion q of calls are complex, and must be referred to the manager. They are queued if the manager is busy. This is determined upon call arrival to a server (not to the arrivals queue).
- Customers in the arrivals queue may get tired of waiting, and leave.
- Distributions are exponential, with the following rates:
    - Arrival rate α.
    - Call service rate σ for ordinary calls, μ for ones that need the manager's attention.
    - A customer's tolerance for waiting has mean $1/\omega$. If her wait in the arrivals queue exceeds this, she leaves.

Part A: Set this up as a continuous-time Markov chain, with states (i,j), where i is the number of ordinary calls currently in the system (being served or waiting) and j is the number of complex calls.

- Express the proportions of calls that (i) are declined and (ii) result in customers leaving due to impatience, in terms of π.

- Solve it using your CRAN project, for several sets of the parameter values, and evaluate the above two quantities. You'll have to relabel the subscripts, so that (0,0) becomes 1, (1,0) becomes and so on.

Part B: Write R simulation code for this problem, using my DES package. You need not have had any prior background in discrete-event simulation; just read the documentation and the example. Let me know if you have any questions. Write your code to be general, NOT restricted to exponential distributions. But run it with exponential distributions for the parameter values you choose in Part A, so that the two parts serve as checks on each other.