Assignment 5 – Worksheet 1
R functions

1. Warm-up

(a) Write a function which takes a numeric vector *x*, and returns a named list containing the mean, median and variance of the values in *x*.

(b) Write a function with arguments *x* and *n*, which evaluates $\sum_{i=0}^{n} \frac{e^{-x} x^i}{i!}$. (Use factorial().)

(c) Write a function which goes through every entry in a list, checks whether it is a character vector (is.character()), and if so prints it (print() or cat()).

(d) Write a function with an argument *k* which simulates a symmetric random walk on the integers, stopping when the walk reaches k (or $-k$). A random walk on the integers is a sequence $X_1, X_2, X_3, \ldots$ with $X_0=0$ and $X_i=X_{i+1}+D_i$ where the D_i are independent with $P(D_i=+1)=P(D_i=-1)=1/2$.

2. Moving Averages

a) Write a function to calculate the moving averages of length 3 of a vector $(x_1, \cdots, x_n)^T$.

(The function returns $(z_1, \cdots, z_{n-2})^T$, where

$$z_i = \frac{1}{3}(x_i + x_{i+1} + x_{i+2}), \qquad i = 1, \cdots, n-2 \ .$$

Call this function `ma3()`.

(b) Write a function which takes two arguments, x and k, and calculates the moving average of x of length k. You can use a `for()` loop, but there has to be a better way, right?

(c) How does your function behave if k is larger than (or equal to) the length of x?

(d) You can (and should) return an error in this case. Use the stop() function. Are there other choices?

(e) How does your function behave if k = 1? What should it do? Fix it if necessary.

3. Optional Plot

Take the continuous functions

$$f(x)=\{ \begin{array}{ll} x^2+2x+3 & if\ x<0 \\ x+3 & if\ 0\leq x<2 \\ x^2+4x-7 & if\ 2\leq x \end{array}$$

Write a function which takes a vector and returns a vector of the values of $f(x)$. The function should be valid for inputs where $-4<x<4$.

Your function should check the input for validity and should offer the user the option of plotting the values the function returns – something like `plot=TRUE.`

4. Matrix Input

Write a function which takes a single argument – a matrix or an argument that can be coerces into a matrix. the function should return a matrix which is the same as the function argument, but ever odd number is doubled.

So, if the input is $\begin{bmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{bmatrix}$ the output should be $\begin{bmatrix} 2 & 2 & 6 \\ 10 & 2 & 6 \\ -2 & -2 & -6 \end{bmatrix}$.

5. Poisson process

A Poisson process of rate λ is a random vector of times $(T_1, T_2, T_3, \ldots)$ where the interarrival times $T_1, T_2 - T_1, T_3 - T_2, \ldots$ are independent exponential random variables with paramerter λ. Note that this implies $T_{i+1} > T_i$.

a) Write a function with arguments λ and M which generates a Poisson lprocess up until the time reaches M. Using the `rexp()` family in R will be helpful.

b) Generate 10,000 of these series with $\lambda = 5$ and $M = 1$. Record the lengths of each of the 10,000 vectors returned. Plot the vector lengths as a histogram. Calculate their mean and variance. How do you think the lengths are distributed? Explain.

Git interlude #1

This is an individual assignment. Pick any one of the functions you are working on and to the following in git. Turn in the final git log.

1. Run the status command. Notice how it tells you what branch you are in.
2. Use the branch command to create a new branch.
3. Use the checkout command to switch to it.
4. Make a couple of commits in the branch – perhaps adding a new file and/or editing existing ones.
5. Use the log command to see the latest commits. The two you just made should be at the top of the list.
6. Use the checkout command to switch back to the master branch. Run log again. Notice that your commits don't show up now. Check the files. They should have their original contents.
7. Use the checkout command to switch back to your branch. Use gitk to take a look at the commit graph; notice it's linear.
8. Now checkout the master branch again. Use the merge command to merge your branch in to it. Look for information about it having been a fast-forward merge. Look at git log, and see that there is no merge commit. Take a look in gitk and see how the DAG is linear.
9. Switch back to your branch. Make a couple more commits.
10. Switch back to master. Make a commit there, which should edit a different file from the ones you touched in your branch – to be sure there is no conflict.
11. Now merge your branch again. Note that you don't need to do anything to inform Git that you only want to merge things added since your previous merge.
12. Look at git log. Notice that there is a merge commit. Also look in gitk. Notice the DAG now shows how things forked, and then were joined up again by a merge commit.

Git interlude #2

This is an group assignment. Pick any one of the functions you are working on and do the following in git. Turn in the final git log.

Each of you will need to have established a GitHub account to do this part of the assignment.

1. First, one person in the group should create a public repository using their GitHub account.

2. This same person should then follow the instructions from GitHub to add a remote, and

then push their repository. Do not forget the –u flag, as suggested by GitHub!

3. All of the other members of the group should then be added as collaborators, so they can also commit to the repository.

4. Next, everyone else in the group should clone the repository from GitHub. Verify that the context of the repository is what is expected.

5. One of the group members who just cloned should now make a local commit, then push it. Everyone should verify that when they pull, that commit is added to their local repository Use git log to check for it.

6. Look at each other's git log output. Notice how the SHA-1 is the same for a given commit across every copy of the repository. Why is this important?

7. Two members of the group should now make a commit locally, and race to push it. To keep things simple, be sure to edit different files. What happens to the runner-up?8. The runner-up should now pull. As a group, look at the output of the command. Additionally, look at the git log, and notice that there is a merge commit. You may also wish to view the DAG in gitk.

9. Repeat the last two steps a couple of times, to practice.