Problem Set 1, Solutions

Stats 506, F19

September 27, 2019

Example solutions for the first problem set are below. The source file for this document and underlying scripts can all be found at the course git repo: https://github.com/jbhender/Stats506_F19/.

Problem 1

Here is the solution text for `ps1_q1.sh`. See the `stats506_F19` repo for an executable version.

```
#!/bin/env bash
# Stats 506, Fall 2019
# This script contains solutions for
# problem set 1, question 1.
#
# Author: James Henderson
# Updated: September 25, 2019
# 80: --------------------------------------------------------------------------
# a - download data if not present
file="recs2015_public_v4.csv"
url="https://www.eia.gov/consumption/residential/data/2015/csv/recs2015_public_v4.csv"

if [ ! -f "$file" ]; then
  wget $url
fi

# b - extract header row and output to a file with one name per line
new_file="recs_names.txt"

# delete the file if it is present
if [ -f "$file" ]; then
  rm "$new_file"
fi

< $file head -n1 | tr , \\n > "$new_file"

# c - get column numbers for DOEID and the brr weights
# as a comma separated string
cols=$(
  < $new_file
  grep -n -E "DOEID|BRR" |
  cut -f1 -d: |
  paste -s -d, -
)

# d - cut out the appropriate columns
<"$file" cut -f"$cols" -d, > recs_brrweights.csv
```

Problem 2

The text of the script is below. See the `stats506_F19` repo for an executable version `cutnames.sh`

```
#!/usr/bin/env bash
# Stats 506, Fall 2019
# This script contains solutions for problem set 1, question 2.

# Author: James Henderson
# Date: Sep 24, 2019

# Our solution is a command line utility for
# extracting columns from a csv file by name.

# Usage: ./cutnames.sh file expression

# Note: to make the script executable, use
# chmod +x ./cutnames.sh
# Otherwise, call as bash ./cutnames.sh

# First argument is the file we are extracting columns from.
file=$1

## The second argument is an (extended) regular
## expression for the column names we want.
expr=$2

## Here is where we do the work
if [ ! -f "$file" ]; then
  # This line echos to stderr rather than stdout
  echo "Could not find file $file." > /dev/stderr
else
  # get column numbers whose names match expr
  cols=$(
   <"$file" head -n1 |
    tr , \\n |
    grep -n -E "$expr" |
    cut -f1 -d: |
    paste -s -d, -
  )

  # cut those columns out
  <"$file" cut -f"$cols" -d,
fi
```

Problem 3

Parts a, b, c, and d

Below is the code form `ps1_q3.R` answering parts a-d.

```
# Stats 506, Fall 2019
# Problem Set 1, Question 3
#
# This script contains functions for translating a set of points in the plane
# to begin at the origin and end with the secant along the positive x-axis.
# It also contains functions to compute summary measures describing the
# efficiency of this path.
#
# Author: James Henderson (jbhender@umich.edu)
# Date: September 26, 2019
#80: --------------------------------------------------------------------------

# libraries: -------------------------------------------------------------------
library(tidyverse)

# read in the  data: -----------------------------------------------------------
## This data will be used for the final part of the question.
url_base = paste0(
  'https://raw.githubusercontent.com/jbhender/Stats506_F19/master/',
  'problem_sets/data'
)
train_file = './train_trajectories.csv'
if ( !file.exists(train_file) ) {
  train_url = sprintf('%s/train_trajectories.csv', url_base)
  train_traj = read_delim(train_url, delim = ',')
  write_delim(train_traj, path = train_file, delim = ',')
} else {
  train_traj = read_delim(train_file, delim = ',')
}

train_meas_file = './train_measures.csv'
if ( !file.exists(train_meas_file) ) {
  train_meas_url = sprintf('%s/train_measures.csv', url_base)
  train_meas = read_delim(train_meas_url, delim = ',')
  write_delim(train_meas, path = train_meas_file, delim = ',')
} else {
  train_meas = read_delim(train_meas_file, delim = ',')
}

test_file = './test_trajectories.csv'
if ( !file.exists(test_file) ) {
  test_url = sprintf('%s/test_trajectories.csv', url_base)
  test_traj = read_delim(test_url, delim = ',')
  write_delim(test_traj, path = test_file, delim = ',')
} else {
  test_traj = read_delim(test_file, delim = ',')
}

# (a) translate so t0 is at origin: --------------------------------------------
```

```r
trans_to_origin = function(xyt){
  # This function translates an n x 3 matrix of triples, xyt, so the first
  # point is at the origin. We assume the matrix is already ordered by t.
  #
  # inputs: xyt - a numeric matrix with three columns
  # outputs: the translated matrix

  cbind(  xyt[, 1] - xyt[1, 1], xyt[,2] - xyt[1, 2], xyt[ ,3] - xyt[1, 3])
}

# unit test for trans_to_origin: ------------------------------------------------
stopifnot( all( trans_to_origin( cbind(rnorm(3), rnorm(3), 1:3) )[1,] == 0 ) )

# (b) find angle between a point and the origin: --------------------------------
compute_angle = function(xy){
  # compute the angle in radians a point (x, y) forms with the origin and the
  #   x axis.
  # inputs: xyt a length 2 numeric vector c(x, y)
  # outputs: an angle, in radians, between [-pi/2, pi/2]
  # Note: an angle between [0, 2*pi] would make this and the next part easier.
  if ( xy[1] == 0 ) {
    # split special case when x = 0
    return( sign(xy[2]) * pi / 2 )
  } else if ( xy[1] > 0 ) {
    # x > 0
    return( sign(xy[2]) * atan( abs( xy[2] / xy[1] ) ) )
  } else {
    # x < 0
    return( sign(xy[2]) * {pi - atan( abs( xy[2] / xy[1] ) )} )
  }
}

# unit tests for compute_angle :------------------------------------------------
.eps = sqrt(.Machine$double.eps)
stopifnot( abs(compute_angle(c( 0,  1 )) - .5  * pi) < .eps )
stopifnot( abs(compute_angle(c( 0, -1 )) + .5  * pi) < .eps )
stopifnot( abs(compute_angle(c( 0.5,  0.5)) - .25 * pi) < .eps )
stopifnot( abs(compute_angle(c(-0.5,  0.5)) - .75 * pi) < .eps )
stopifnot( abs(compute_angle(c( 0.5, -0.5)) + .25 * pi) < .eps )
stopifnot( abs(compute_angle(c(-0.5, -0.5)) + .75 * pi) < .eps )
rm(.eps)

# (c) rotation function: --------------------------------------------------------
rotate = function(xyt, theta = 0, clockwise = FALSE) {
  # function to rotate the first 2-columns of a coordinate matrix xyt
  # by theta radians.
  # inputs: xyt - a three column numeric matrix
  #         theta - the angle to rotate xyt[,1:2]
  #         clockwise - rotate clockwise (defaults to false)

  if ( clockwise ) {
    s = 1
  } else {
```

```
    s = -1
  }
  R = matrix( c(cos(theta), -s*sin(theta), s*sin(theta), cos(theta) ), 2, 2 )

  ## Apply the rotation
  cbind( xyt[ , 1:2] %*% R, xyt[ , 3] )
}
# unit tests for the rotation: ------------------------------------------------
.test_rotate = function(){
  .eps = sqrt( .Machine$double.eps )
  x = .5 * c(1, -1, -1,  1)
  y = .5 * c(1,  1, -1, -1)

  for ( i in 1:4 ) {
    xyt = cbind( c(0, x[i]), c(0, y[i]), 1:length(x) )
    theta = compute_angle(xyt[2, 1:2])
    stopifnot( abs( rotate(xyt, theta)[2, 2] ) < .eps )
  }

}
.test_rotate()
rm(.test_rotate)

# (d) combine the above into a single function for normalizing a trajectory: ---
normalize_traj = function(xyt) {
  # This function normalizes the trajectory in xyt to start at the origin and
  # conclude along the positive x-axis.
  #
  # Inputs:
  #   xyt - an n x 3 numeric matrix with time in the final column
  # Output:
  #   A matrix with the same dimensions as xyt representing the normalized
  #   trajectory.

  # check input
  stopifnot( is.numeric(xyt) )
  stopifnot( ncol(xyt) == 3 )

  # Compute the rotation agnle
  theta = compute_angle( xyt[nrow(xyt), 1:2] - xyt[1, 1:2] )

  # Translate and rotate to normalize
  rotate( trans_to_origin(xyt), theta = theta )
}
# Unit tests for normalize_traj: ----------------------------------------------
.test_normalize_traj = function() {
  n = 10
  tm = seq(12, 24, length.out = n)
  x = rnorm(n)
  y = rnorm(n)
  xyt = cbind(cumsum(x), cumsum(y), tm)

  xyt0 = normalize_traj(xyt)
```

```r
  stopifnot( all( xyt0[1, ] == 0 ) )
  stopifnot( abs( xyt0[n, 2] ) < sqrt( .Machine$double.eps ) )
}
.test_normalize_traj()
rm(.test_normalize_traj)

# (e) compute measures: ------------------------------------------------------
## total distance function
comp_dist = function(x, y) {
  # compute the total distance between successive (x, y) pairs
  # inputs: x, y - numeric vectors of the same length
  # output: a numeric constant with the distance traveled along the trajectory
  #         (x, y)
  stopifnot( length(x) == length(y) )
  sum( sqrt( diff( x )^2 + diff( y )^2 ) )
}

## area under the curve integrated using trapezoidal rule
comp_auc = function(x, y, absy = TRUE){
  # compute the area under the curve traced out by x, y
  # allowing cancellation in x and (optionally) y.
  # inputs: x, y - numeric vectors of the same length
  #         abs - should we use the absolute value of y or allow cancellation?
  # output: .5 * sum_i (|y_i| + |y_i+1|)*(x_i+1 - x_i)
  stopifnot( length(x) == length(y) )

  dx = diff(x)
  if ( absy ) {
    return( .5 * sum( {abs( y[-length(y)] ) + abs( y[-1] )} * dx ) )
  } else {
    return( .5 * sum( {y[-length(y)] +  y[-1]} * dx ))
  }

}

comp_measures = function(xyt) {
  # This function computes the following curvature measures for a normalized
  # trajectory xyt given as an n x 3 numeric matrix. Each row  of xyt should
  # give the coordinates in the the x, y at time t (in that order).
  # If the trajectory is not normalized, this function will error with reference
  # to

  # Inputs:
  #  xyt - a numeric matrix with three columns
  # Output:
  #  A named vector with the following curvature measures:
  #    dist - the total (Euclidean / L2) distance traveled along the trajectory
  #    max_abs_dev - the maximum absolute deviation from the secant line
  #                  representing a direct path from the first to last point.
  #    avg_abs_dev - the average abosolute deviation form the secant line.
  #    auc - the absolute area under the curve, computed with the trapezoidal rule
  #          and allowing cancelation in the "x" dimension.
```

```
  # Test the input
  stopifnot( is.numeric(xyt) )
  stopifnot( ncol(xyt)  == 3 )
  if ( !all( xyt[1, ] == 0 ) ||
       {abs( xyt[nrow(xyt), 2] ) > sqrt( .Machine$double.eps )} ) {
    stop("Trajectory is not normalized. Use normalize_traj() first.\n")
  }

  # Compute the curvature measures and return them
  c(
    'tot_dist' = comp_dist(xyt[, 1], xyt[, 2]),
    'max_abs_dev' = max( abs(xyt[, 2]) ),
    'avg_abs_dev' = mean( abs(xyt[, 2]) ),
    'AUC' = comp_auc(xyt[, 1], xyt[, 2])
  )

}

# Test part (e) using the provided training data: ------------------------------

#! This is an approach using only base R, see below for alternatives.
trials = unique( train_traj[, c('subject_nr', 'count_trial')] )

## container for results
measure_matrix = matrix(NA_real_, nrow = nrow(trials), ncol = 6 )
colnames(measure_matrix) = c('subject_nr', 'count_trial',
                             'tot_dist', 'max_abs_dev', 'avg_abs_dev', 'AUC')

for( i in 1:nrow(measure_matrix) ) {
  subj = trials[['subject_nr']][i]
  trial = trials[['count_trial']][i]
  rows = with(train_traj, which(subject_nr == subj & count_trial == trial) )

  traj = as.matrix( train_traj[rows, c('xpos', 'ypos', 'tm')])

  measure_matrix[i, ] =
    c(subject_nr = subj, count_trial = trial,
       comp_measures( normalize_traj(traj) ) )
}

train_measures = as_tibble(measure_matrix)

# Check equality column by column using "near" from dplyr: ---------------------
passing = vector( length = ncol(train_meas), mode = 'logical')
names(passing) = colnames(train_meas)
for ( col in names(train_measures) ) {
  passing[col] = all( near( train_meas[[col]], train_measures[[col]] ) )
}
stopifnot( all( passing ) )

# Compute curvature metrics for the test data: ---------------------------------
#! This is an alternative "tidy" approach to computing measures by group using
#  the general purpose verb/function "do"
```

```
## Formats output of comp_measures() as a tibble
convert_vec2tibble = function(x) as_tibble( as.list(x) )

test_meas =
  test_traj %>%
  group_by(subject_nr, count_trial) %>%
  do( convert_vec2tibble(
        comp_measures( normalize_traj( cbind(.$xpos, .$ypos, .$tm ) ) )
      )
  )

## To produce a table
cn = c('Subject', 'Trial', 'Total Distance',
         'Max Abs Dev', 'Avg Abs Dev', 'AUC')
#knitr::kable(test_meas, digits = 1, col.names = cn)
```

Part e

Here are the measures computed on the test data, presented as a nicely formatted table.

```
source('./ps1_q3.R')
cap =
 '*Measures of curvature computed from the "test" trajectories.*'
knitr::kable(test_meas, digits = 1, col.names = cn, caption = cap)
```

Table 1: *Measures of curvature computed from the "test" trajectories.*

Subject	Trial	Total Distance	Max Abs Dev	Avg Abs Dev	AUC
6	1	1650.8	464.9	90.4	275254.4
7	1	1252.6	35.5	4.7	19981.2
8	1	1069.2	18.4	1.8	10134.0
9	1	1092.1	74.2	7.3	36134.4
10	1	1086.8	85.3	12.5	51446.3