ps1_q3

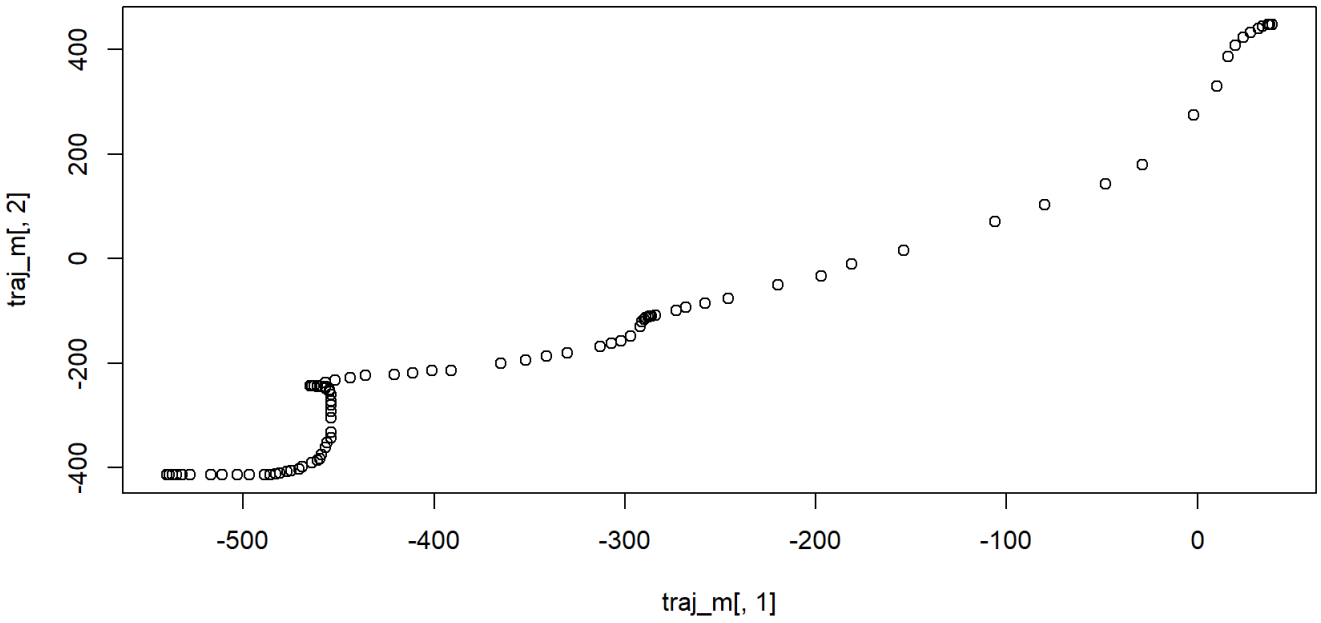
Sijun Zhang

2019/9/23

Problem 3

Import Data

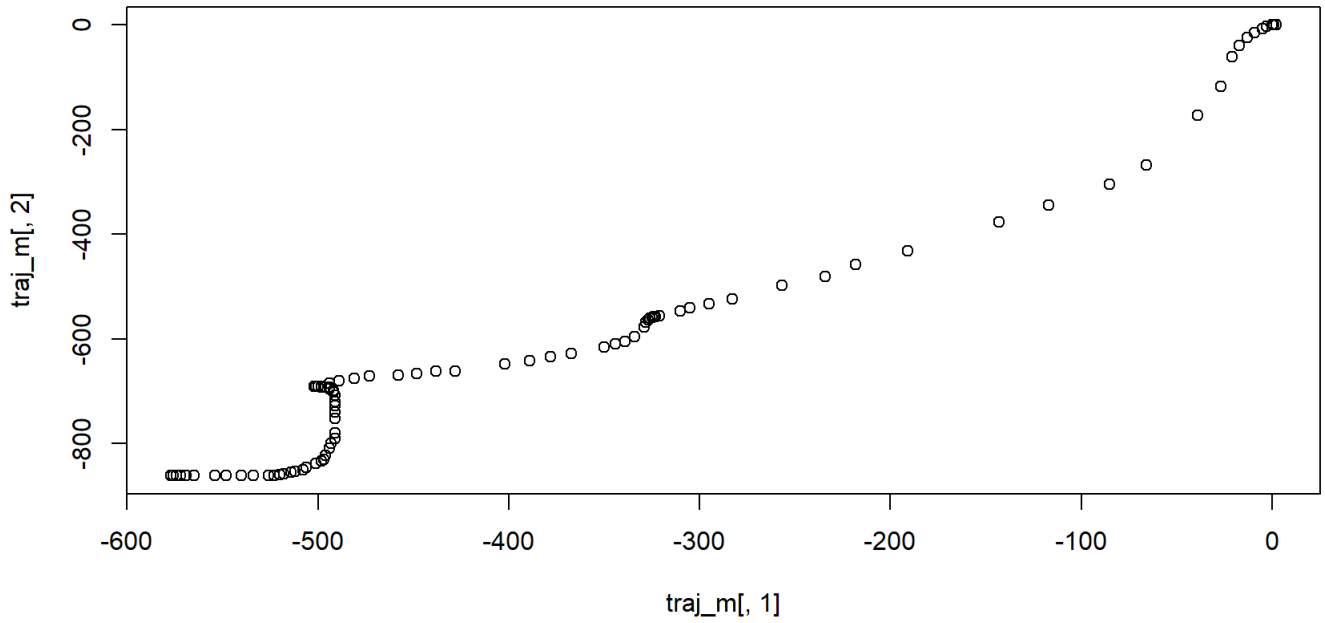
```
traj <- read.table('train_trajectories.csv', sep = ',', stringsAsFactors = FALSE, header = TRUE)
traj_01_04 <- traj[(traj$subject_nr == 1) & (traj$count_trial == 4), 3:5]
traj_m <- as.matrix(traj_01_04)
plot(traj_m[, 1], traj_m[, 2])
```



Set Origin Zero

[5pts] Write a function that accepts a n n×3 matrix representing the trajectory (x, y, t) and translates it to begin with time zero at the origin.

```r
# INPUT: n×3 matrix representing the trajectory (x, y, t)
# OUTPUT: nx3 matrix begining with time zero at the origin
set_origin_zero = function(x) {
  n <- dim(x)[1]
  origin <- x[1,]
  y <- x
  for (i in 1:n) {
    y[i,] <- y[i,] - origin
  }
  return(y)
}
traj_m <- set_origin_zero(traj_m)
plot(traj_m[, 1], traj_m[, 2])
```



In this part, I sequentially minused each point with the origin coordination, so that the curve was translated to the one beginning at origin and time zero.

Compute Angle

[5pts] Write a function that computes the angle θ formed by the secant line connecting the origin and the final position in the trajectory. Your answer should be an angle between [−π,π]. Be sure your your solution works for a trajectory ending in any of the four quadrants.

```r
# INPUT: xops and yops value of a point
# OUTPUT: the angle θ formed by the line connecting the origin and the point
compute_angle = function(x, y) {
  if (x == 0) {
    if (y == 0) {theta <- 0}
    else {
      theta <- sign(y) * (1 / 2) * pi
    }
  }
  else if (x > 0) {
    theta <- atan(y / x)
  }
  else {
    tmp <- atan(y / x)
    theta <- tmp - sign(tmp) * pi
    if (y == 0) {
      theta <- -pi
    }
  }
  names(theta) <- NULL
  return(theta)
}

# INPUT: n×3 matrix representing the trajectory (x, y, t) beginning at zero
# OUTPUT: the angle θ formed by the secant line connecting the origin and the final position in the traje
ctory
compute_secant_angle = function(x) {
  n <- dim(x)[1]
  final <- x[n,]
  final_x <- final[1]
  final_y <- final[2]
  theta <- compute_angle(final_x, final_y)
  return(theta)
}
compute_secant_angle(traj_m)
```

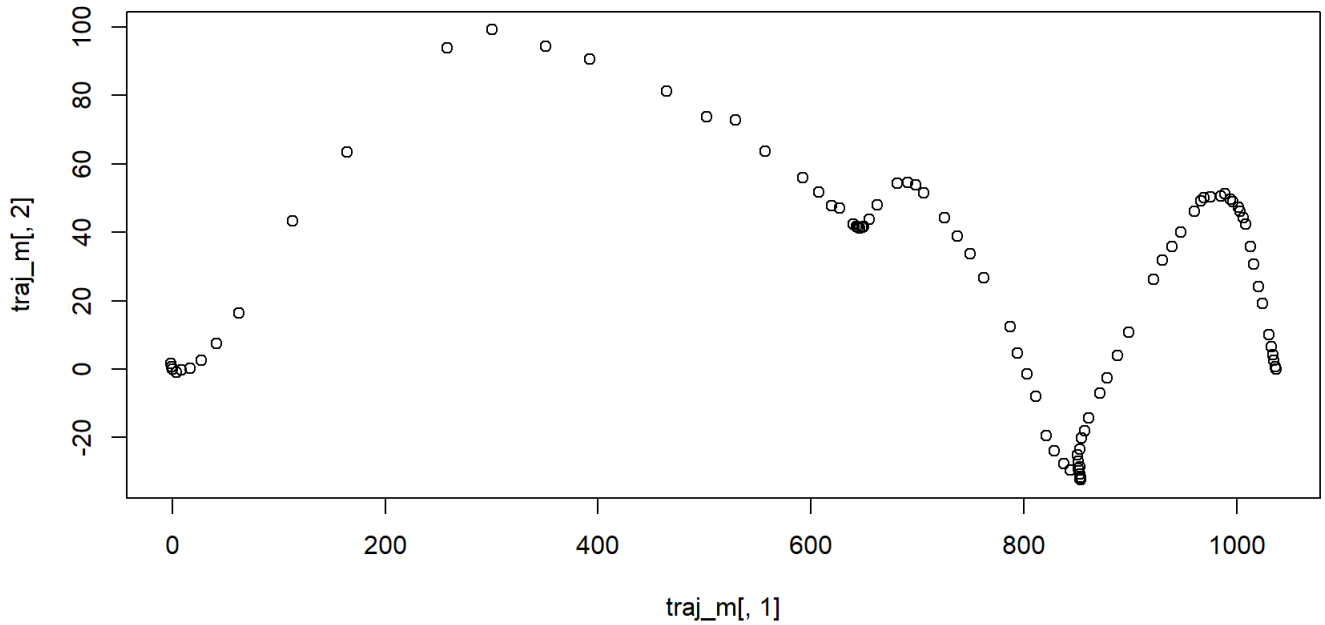
```
## [1] -2.161207
```

In this part, I wrote a universal function to compute the angle between the origin and the selected pint. I found that the period of tan function is only π, there were two special cases need to aware, one is the points at 3rd and 2nd quadrants and the other one is the points lying on the negative axis. I set the the points lying on the negative axis have -π angle to avoid runtime error.

Rotate End to X-axis

[5pts] Write a function to rotate the (x, y) coordinates of a trajectory so that the final point lies along the positive x-axis.

```r
# INPUT: n×3 matrix representing the trajectory (x, y, t) beginning at zero
# OUTPUT: rotate the (x, y) coordinates of a trajectory so that the final point lies along the positive x
-axis.
rotate_end_to_xaxis = function(x) {
  n <- dim(x)[1]
  y <- x
  theta <- compute_secant_angle(x)
  for (i in 1:n) {
    theta_i <- compute_angle(x[i, 1], x[i, 2])
    theta_i <- theta_i - theta
    secant_distance <- sqrt(x[i, 1]^2 + x[i, 2]^2)
    y[i,1] <- secant_distance * cos(theta_i)
    y[i,2] <- secant_distance * sin(theta_i)
  }
  return(y)
}
traj_m <- set_origin_zero(traj_m)
traj_m <- rotate_end_to_xaxis(traj_m)
plot(traj_m[, 1], traj_m[, 2])
```

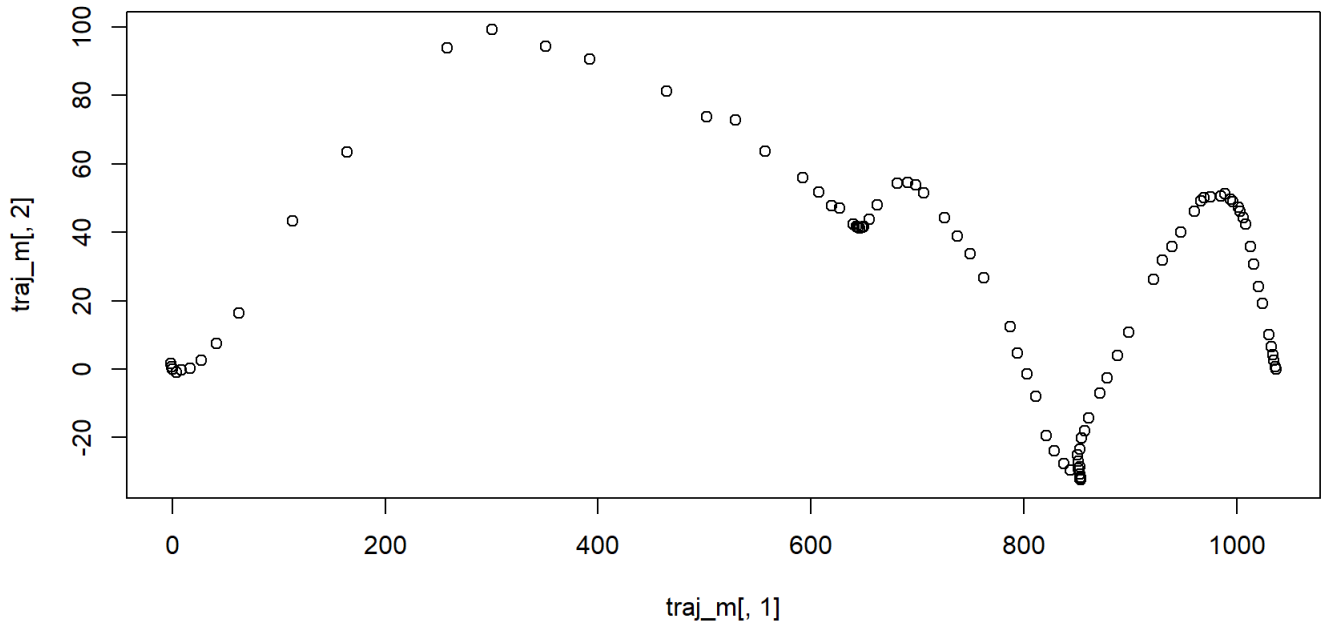


In this part, I computed each point's angle to the origin and minus the angle of the last points, then maintain the secant distance unchanged to rotate each point to the new angle, so that the curve was rotated to have the ends on x-axis.

Normalize

[2pts] Combine the three parts above into a single function that normalizes an n×3 trajectory matrix to begin at the origin and end on the positive x-axis.

```
# INPUT: n×3 matrix representing the trajectory (x, y, t)
# OUTPUT: rotate the (x, y) coordinates of a trajectory so that the final point lies along the positive x
-axis.
normalize = function(x) {
  y <- set_origin_zero(x)
  y <- rotate_end_to_xaxis(y)
  return(y)
}
traj_m <- normalize(traj_m)
plot(traj_m[, 1], traj_m[, 2])
```



This part combine the previous functions into a pipeline.

Measure the Curvature

[8pts] Write a function that accepts a normalized trajectory and computes the following metrics describing its curvature:

```
# INPUT: a normalized trajectory
# OUTPUT:  the metrics describing its curvature
measure_curvature = function(x) {
  # Calc the total (Euclidean) distance traveled
  n <- dim(x)[1]
  tot_dist <- 0
  for (i in 1:(n - 1)) {
    dist_i <- sqrt((x[i, 1] - x[i + 1, 1])^2 + (x[i, 2] - x[i + 1, 2])^2)
    tot_dist <- dist_i + tot_dist
  }

  # Calc the maximum absolute deviation from the secant connecting the starting and final positions
  max_abs_dev <- max(abs(x[, 2]))

  # Calc the average absolute deviation of the observed trajectory from the direct path
  avg_abs_dev <- mean(abs(x[, 2]))

  # Calc the (absolute) area under the curve for the trajectory relative to the secant line using the tra
pezoidal rule to integrate
  AUC <- 0
  for (i in 1:(n - 1)) {
    x_dist <- x[i + 1, 1] - x[i, 1]
    AUC_i <- (1 / 2) * x_dist * (abs(x[i + 1, 2]) + abs(x[i, 2]))
    AUC <- AUC + AUC_i
  }
  re <- c(tot_dist, max_abs_dev, avg_abs_dev, AUC)
  return(re)
}
traj_m <- normalize(traj_m)
re <- measure_curvature(traj_m)
print(re)
```

```
## [1]  1153.46372    99.18373    18.17234 54516.01033
```

In this part, the total (Euclidean) distance traveled, and AUC need to traverse the curve and as the curve has been normalized, the deviation from secant is actually the y value of each points. Finally I combined the four results into a list to output.

Benchmarks

[5pts] Apply your function to the sample trajectories at the Stats506_F19 repo on GitHub and check your solutions against the sample measures. Then, compute the metrics above for the test trajectories and report your results in a nicely formatted table.

```
options(digits = 15)

traj = read.table('train_trajectories.csv', sep = ',', stringsAsFactors = FALSE, header = TRUE)
re_train <- matrix(ncol = 6, nrow = 0)
for (i in min(traj$subject_nr):max(traj$subject_nr)) {
  traj_i <- traj[traj$subject_nr == i, ]
  for (j in as.numeric(names(summary(factor(traj_i$count_trial))))) {
    traj_i_j <- traj[(traj$subject_nr == i) & (traj$count_trial == j), 3:5]
    traj_m <- as.matrix(traj_i_j)
    traj_m <- normalize(traj_m)
    curvature <- measure_curvature(traj_m)
    re_i_j <- c(as.integer(i), as.integer(j), curvature)
    re_train <- rbind(re_train, re_i_j)
  }
}
mea_train <- read.table('train_measures.csv', sep = ',', stringsAsFactors = FALSE, header = TRUE)
all(abs(mea_train - re_train) < 1e-7)
```

```
## [1] TRUE
```

Firstly, I use the the train_trajectories.csv to test the accuracy of the curvature measurements function and normalizing part. Then I load the validation measures and compare the value for each trajectory. The result shows that the computaion are very close to the validation results.

```
traj = read.table('test_trajectories.csv', sep = ',', stringsAsFactors = FALSE, header = TRUE)
re_test <- matrix(ncol = 6, nrow = 0)
for (i in min(traj$subject_nr):max(traj$subject_nr)) {
  traj_i <- traj[traj$subject_nr == i, ]
  for (j in as.numeric(names(summary(factor(traj_i$count_trial))))) {
    traj_i_j <- traj[(traj$subject_nr == i) & (traj$count_trial == j), 3:5]
    traj_m <- as.matrix(traj_i_j)
    traj_m <- normalize(traj_m)
    curvature <- measure_curvature(traj_m)
    re_i_j <- c(as.integer(i), as.integer(j), curvature)
    re_test <- rbind(re_test, re_i_j)
  }
}
options(digits = 9)
rownames(re_test) <- NULL
re_test <- data.frame(re_test)
names(re_test) <- c("subject_nr","count_trial","tot_dist","max_abs_dev","avg_abs_dev","AUC")
print(re_test)
```

```
##   subject_nr count_trial   tot_dist max_abs_dev avg_abs_dev         AUC
## 1          6           1 1650.76910 464.8991018 90.38782548 275254.3537
## 2          7           1 1252.55027  35.4682341  4.72356215  19981.1991
## 3          8           1 1069.15769  18.4112977  1.75701518  10133.9932
## 4          9           1 1092.07644  74.2055008  7.30294464  36134.4045
## 5         10           1 1086.83468  85.3393311 12.48771481  51446.3234
```

The test_trajectories.csv contains 5 different trajectories and using the previous functions, I obtained the above metrics for the curvature.