Stats 506: Problem Set #1: Question 3

Diana Liang

9/23/2019

"Mouse-Tracking" Data

Many experiments want to measure the degree of delays when making decisions. A simple way is to track the movement of the mouse using Euclidean (x,y) coordinates and time t. The data is tabulated in a nx3 matrix of [x,y,t] values. To ease calculation, many normalize this matrix.

Normalize Function

This function must be able to take the input matrix of [x,y,t] so that the matrix can: (i) be translated all the points such that the first point is the origin; (ii) have the angle between the secant, of the first and final points, and the x axis; (iii) and be rotated by the forementioned angle so the secant can lie on the x axis.

(i) Translate function 3a

```
# Shifts all points so that (x1,y1) is at the origin
translate <- function (matrix_og){

  #Input: a nx3 matrix of [x,y,t]
  #Output: shifted nx3 matrix of [x,y,t]

    #Takes first row vector
    vector_og <- c(matrix_og[1, 1], matrix_og[1, 2], matrix_og[1, 3])

    #Subtract first row from all the rows
    counter <- 1
    while (counter <= nrow(matrix_og)){
        matrix_og[counter, 1] <- matrix_og[counter, 1] - vector_og[1]
        matrix_og[counter, 2] <- matrix_og[counter, 2] - vector_og[2]
        matrix_og[counter, 3] <- matrix_og[counter, 3] - vector_og[3]

        counter <- counter + 1
    }
    return(matrix_og)
}
```

(ii) Secant Angle function 3b

```
# Finds angle between x axis and secant connecting final point to origin
sec_angle <- function (matrix_og){
```

```
  #Input: a matrix of at least 2 colums detailing [x,y,...];
  #       assumes that input matrix starts at the origin
  #Output: angle between secant and x axis

    theta <- atan2(matrix_og[nrow(matrix_og),2],
matrix_og[nrow(matrix_og),1])

    return(theta)
}
```

(iii) Rotate function 3c

```
# Rotates (x,y) coordinates so secant falls on positive x axis
rotate <- function(matrix_og,theta){

  ##Input: a nx3 matrix of [x,y,t] with starting point at origin and time
zero
  ##Output: nx3 matrix of [x,y,t] with secant along x axis

    vector_og <- matrix(nrow=2, ncol=1)

    #Create rotation matrix
    trans_matrix <- matrix(nrow=2, ncol=2)
    trans_matrix[1,1] <- cos(theta*-1)
    trans_matrix[1,2] <- -sin(theta*-1)
    trans_matrix[2,1] <- sin(theta *-1)
    trans_matrix[2,2] <- cos(theta*-1)

    #Transform each row by the rotation matrix
    counter <- 1
    while(counter <= nrow(matrix_og)){
        vector_og[1,1] <- matrix_og[counter,1]
        vector_og[2,1] <- matrix_og[counter,2]

        vector_rot <- trans_matrix %*% vector_og

        matrix_og[counter,1] <- vector_rot [1,1]
        matrix_og[counter,2] <- vector_rot [2,1]

        counter <- counter + 1

    }
    return(matrix_og)

}
```

(iv) Normalize Function 3d

```
# Shift and rotate matrix so secant is on x axis
normalize <- function (matrix_og){
```

```
  ##Input: a nx3 matrix of [x,y,t]
  ##Output: a nx3 matrix whose secant is on the positive x axis

  #Translate to start at origin at t=0
  vector_og <- c(matrix_og[1, 1], matrix_og[1, 2], matrix_og[1, 3])
  counter <- 1
  while (counter <= nrow(matrix_og)){
    matrix_og[counter, 1] <- matrix_og[counter, 1] - vector_og[1]
    matrix_og[counter, 2] <- matrix_og[counter, 2] - vector_og[2]
    matrix_og[counter, 3] <- matrix_og[counter, 3] - vector_og[3]
    counter <- counter + 1
  }

  #Find angle between origin->final and x axis
  theta <- atan2(matrix_og[nrow(matrix_og),2], matrix_og[nrow(matrix_og),1])

  #Rotate about the origin so that final is on the positive x axis
  vector_og <- matrix(nrow=2, ncol=1)

  trans_matrix <- matrix(nrow=2, ncol=2)
  trans_matrix[1,1] <- cos(theta*-1)
  trans_matrix[1,2] <- -sin(theta*-1)
  trans_matrix[2,1] <- sin(theta *-1)
  trans_matrix[2,2] <- cos(theta*-1)

  counter <- 1
  while(counter <= nrow(matrix_og)){
    vector_og[1,1] <- matrix_og[counter,1]
    vector_og[2,1] <- matrix_og[counter,2]

    vector_rot <- trans_matrix %*% vector_og

    matrix_og[counter,1] <- vector_rot [1,1]
    matrix_og[counter,2] <- vector_rot [2,1]

    counter <- counter + 1
  }
  return(matrix_og)
}
```

Normalized Metrics

Once the data matrix has been normalized, it's easier to calculate metrics that represent delays. While looking at the time delay provides insight, the path of the mouse may provide deeper understanding. For example, a more direct path may represent lesser hesitancy while a more circuitous path may represent greater hesitancy. The metrics below are only

some of the possible measurements: the total distance travelled by the mouse, the maximum deviation from the most direct path, the average deviation from the most direct path, and the area under the trajectory relative to the secant line.

Normalized Metrics function 3e

```
# Calculates measurements based on normalized matrix
norm_metrics <- function(norm_matrix){

  #Input: normalized nx3 matrix of [x,y,t]
  #Output: vector of total distance, max deviation,
  #         average deviation, and area

  #Finds total distance travelled (i)
  counter <- 2
  tot_dist <- 0

  while (counter <= nrow(norm_matrix)) {
    new_dist <- sqrt( ( (norm_matrix[counter,1]-norm_matrix[(counter - 1),
1])^2
                      + (norm_matrix[counter,2]-norm_matrix[(counter - 1),
2])^2 ) )
    counter <- counter + 1
    tot_dist <- tot_dist + new_dist
  }

  #Max deviation from secant (ii) is greatest absolute y value
  #Avg deviation from secant (iii) is mean value of y's
  ycol <- matrix(data = norm_matrix[,2], nrow = nrow(norm_matrix), ncol = 1)
  ycol <- abs(ycol)

  #Finds area under curve (iv)
  counter <- 2
  tot_area <- 0

  while (counter <= nrow(norm_matrix)) {
    new_area <- 0.5*(norm_matrix[counter,1]-norm_matrix[(counter-
1),1])*(norm_matrix[counter,2]+norm_matrix[(counter-1),2])
    new_area <- abs(new_area)
    counter <- counter + 1
    tot_area <- tot_area + new_area
  }

  return(c(tot_dist, max(ycol), mean(ycol), tot_area))
}
```

Application 3f

Below are test trajectories and the calculated normalized metrics to demonstrate how the matrix of points can be transformed into useful measurements.

Here are the original test trajectories:

```
print(test_trajectories)
```

```
## # A tibble: 1,007 x 5
##    subject_nr count_trial  xpos  ypos    tm
##         <dbl>       <dbl> <dbl> <dbl> <dbl>
##  1          6           1   -91   374 43584
##  2          6           1   -91   374 43594
##  3          6           1   -91   374 43604
##  4          6           1   -91   374 43614
##  5          6           1   -91   374 43624
##  6          6           1   -91   374 43634
##  7          6           1   -91   374 43644
##  8          6           1   -91   374 43654
##  9          6           1   -91   374 43664
## 10          6           1   -91   374 43674
## # ... with 997 more rows
```

Here are the metrics after the trajectories were normalized:

```
print(test_measurements)
```

```
## # A tibble: 5 x 6
##    subject trial total_dist max_dev avg_dev total_area
##      <dbl> <dbl>      <dbl>   <dbl>   <dbl>      <dbl>
##  1       6     1      1651.   465.    90.4     275070.
##  2       7     1      1253.    35.5    4.72     19795.
##  3       8     1      1069.    18.4    1.76     10116.
##  4       9     1      1092.    74.2    7.30     35963.
##  5      10     1      1087.    85.3   12.5      51405.
```

While the original test trajectories provide a map of the mouse path, the table of metrics provide a mathematical representation that is more useful in data interpretation.