Homework 9

Prepare your answers as a **single PDF file**.
Group work: You may work in groups of 1-3. Include all group member names in the PDF file. You may work with students in both sections (375-01, -02). Only one person in the group should submit to Canvas.
Due: check on Canvas.

1. Load the "mystery" vector in file myvec.RData on Canvas (using `load("myvec.RData")`. Note that R allows you to store objects in its own machine-independent binary format instead of a text format such as .csv). Decompose the time series data into trend, seasonal, and random components.

Specifically, write R code to do the following:

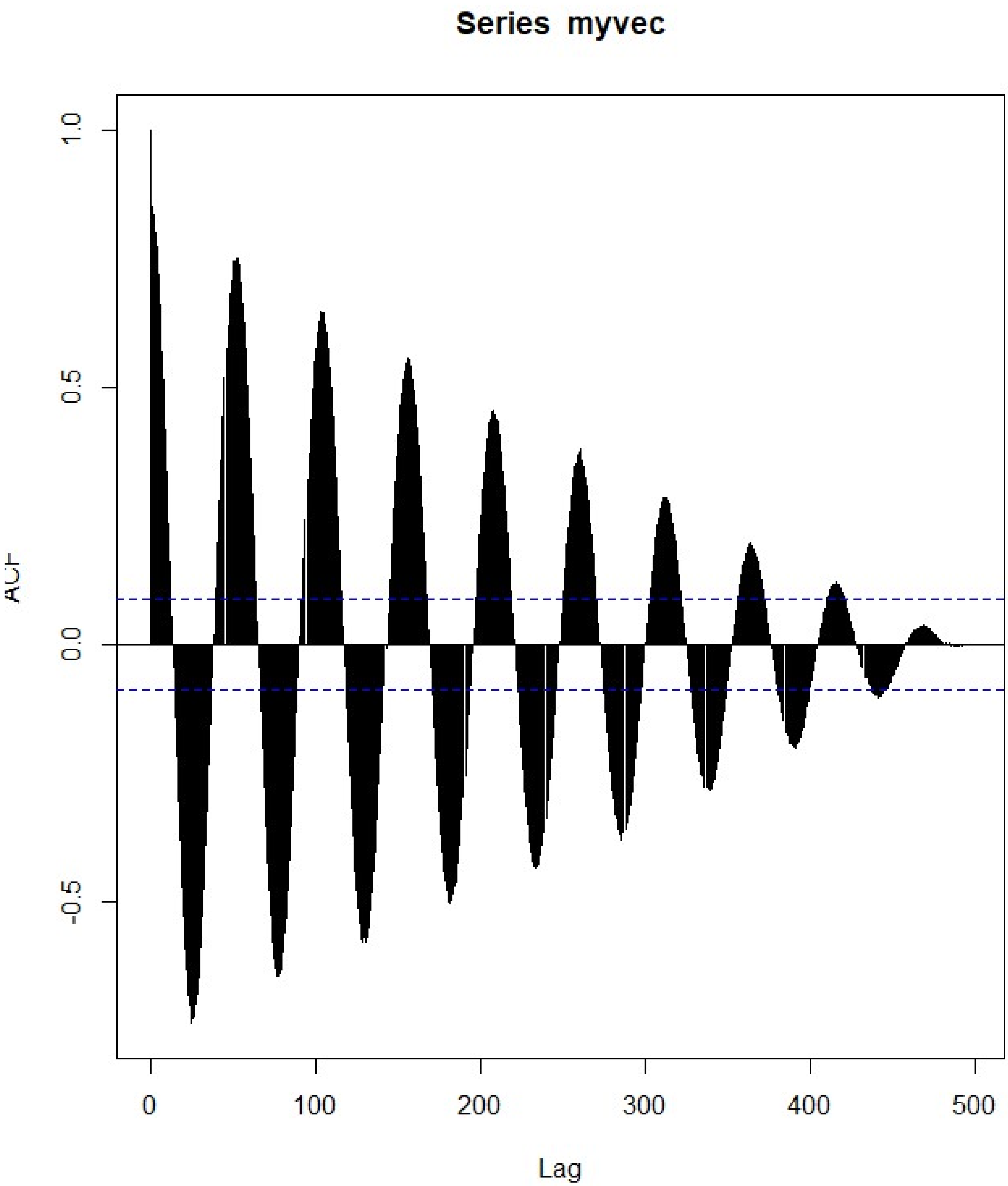
a) Load the data. [show code]

```
load("myvec.RData")

myvec
```

```
  [1] 16.065509 18.376219 18.424365 14.689026 16.744741 24.085523 16.850646
  [8] 24.455535 18.851095 27.642763 25.543738 24.318930 26.063032 21.106958
 [15] 23.142954 20.964844 26.531165 27.692150 25.193142 23.266539 26.447094
 [22] 17.017895 15.714125 16.904274 16.864563 11.818887 16.165240  9.449123
 [29] 17.114564  7.812521 15.417255  4.930847 10.641026 12.452909  7.393438
 [36]  5.593194  4.888740  1.877167  8.477422  5.681040  4.118445  2.199274
 [43]  2.632366  7.980000 10.546155  5.583414 12.062679  4.969366  7.517362
 [50] 13.679192 12.914075 12.237484 14.083577 12.106214 17.293159 23.421869
 [57] 20.441700 21.933200 19.293123 23.842111 26.368448 28.519685 29.027469
 [64] 20.469000 28.179248 21.779781 29.559210 29.204660 26.872441 21.624874
 [71] 25.807292 24.792914 25.185773 24.116557 23.950597 15.077273 22.089466
 [78] 16.112717 12.087197 13.047882 15.041172  9.697778  7.648151  6.252732
 [85]  8.575576 11.307751  6.438869  3.803366 10.234678  4.451393  4.384960
 [92]  9.739035  8.628275  3.590993  2.863157  3.909645 12.677834  9.655658
 [99] 12.687938 11.589665 16.280726 13.557195 11.270750 14.800435 18.792038
[106] 18.837602 17.290453 19.829553 24.864820 25.552150 23.796098 25.939144
[113] 28.163550 21.587487 24.219429 25.826943 21.863858 28.384402 30.186142
[120] 30.266170 21.875043 24.152810 21.816137 21.364537 22.071100 17.192373
[127] 17.403740 22.561073 14.759164 15.548721 19.099618 14.907410 15.116757
[134]  9.143809 12.286126 12.078926  7.567062  8.475139 10.479934  6.677565
[141]  3.805379  9.922783  5.913981  9.828018  2.306897  3.897910 10.037545
[148]  7.186220  7.612528 10.838023  5.571247  6.909093  8.452938 11.105287
[155] 17.982881 16.370456 19.977207 16.192634 22.284550 24.948288 19.995907
[162] 20.285085 26.183571 27.899200 23.425577 21.907150 26.938155 26.529814
[169] 24.303391 27.028121 23.529659 26.571355 29.137595 27.414806 27.323935
[176] 25.617364 25.078134 25.537725 25.232153 21.619311 20.651775 18.036035
[183] 18.743401 12.322993 13.750316 11.822992 15.540110 12.694277 10.485573
[190]  5.796546 11.980964 11.724987  8.528857  3.387515 10.955566 10.784050
[197] 11.288460  5.985189  3.112052  8.357783 13.899882  4.739434 14.689825
[204]  7.029229 10.498765 10.245177 16.275050 18.600816 21.567432 15.075506
[211] 23.993470 19.321916 25.673063 23.392479 24.737261 23.489390 22.614082
[218] 30.284938 26.442250 28.511946 24.303388 22.682458 28.545637 27.228498
[225] 23.459649 22.222114 20.937101 26.620218 24.790445 18.941421 18.604056
[232] 23.266366 19.969004 14.241842 13.752511 14.498458 12.578457 11.191873
[239]  8.838690 13.232443 14.888782  9.813826  6.641204  7.215155 12.896988
[246] 12.940763  5.425949  4.767239 11.619562  6.186215 12.192340 13.651545
[253]  9.025413 15.121366 11.390675 10.328768 15.159772 16.295600 16.715068
[260] 19.841059 16.814684 23.765800 17.435943 17.454073 22.851017 24.721435
[267] 27.890888 25.862624 28.582491 23.725339 27.323729 26.293831 31.211280
[274] 26.933265 24.556070 32.084110 23.683784 27.153040 27.004253 26.682341
[281] 26.666760 23.176622 17.869807 19.698224 24.760645 23.871825 17.493898
[288] 17.147621 12.925637 17.713466 16.420286  9.223501  8.831661 10.911190
[295]  8.206271  5.959275  4.688824  7.330967  4.165476 10.490024  6.173824
[302] 11.417906 12.502562 13.119240  9.820072 15.808370 15.554179 14.787190
[309] 10.141091 17.734484 12.863509 21.608398 17.928584 23.812418 23.830829
[316] 22.303027 21.359637 27.692651 23.194750 27.842738 30.198102 26.078712
[323] 29.985254 32.601129 28.211413 25.555013 29.723242 33.017647 25.029287
[330] 27.867115 30.461429 23.427911 26.581543 22.471649 19.461510 24.977027
[337] 19.234924 21.329546 23.084173 21.085686 21.470204 14.590053 16.484624
[344] 13.658117  9.335580 16.330964 10.113757 12.060590  9.013593 13.601696
[351]  6.305322  5.971346  6.151928  8.033209 10.678687  8.317550 10.651524
[358] 16.156546 12.715936 10.546252 11.152339 16.875885 12.746937 16.942473
[365] 14.331147 18.690962 18.673986 23.266096 22.012530 27.634141 28.962722
[372] 28.112369 23.273509 32.140501 30.971408 32.437565 27.263819 25.351537
[379] 25.397786 28.170732 28.124967 25.625141 24.073955 24.521908 22.497903
[386] 24.387382 20.051737 24.833691 21.480071 21.650368 22.902390 23.191509
[393] 18.964108 13.664401 11.505749 18.857751 14.665439  8.084751 11.083875
[400] 15.665232 15.088150  8.628176 14.817917  6.892991  6.220521 11.602147
[407]  8.317326 15.010790 13.333609 12.541940 10.263532 12.513585 12.078390
[414] 14.257609 18.491957 20.203971 20.782432 25.685149 22.160259 26.601161
[421] 26.767358 21.017028 31.600450 31.286351 31.172178 30.122969 26.187751
[428] 27.525373 28.342003 26.264863 32.826070 29.661403 26.735598 29.820250
[435] 31.199045 26.780259 31.306408 28.363966 28.098179 25.775186 18.290486
[442] 17.665038 16.918121 19.701881 20.395416 14.562328 14.055929 13.219141
[449] 13.095380 11.640385 12.827449  9.853209 10.053594 10.212968 15.606426
[456] 13.493180 14.953311 11.316630  7.521362 12.019716 15.218107  9.130669
[463] 17.721457 10.923474 13.931241 21.067879 14.995744 16.824105 24.094591
[470] 22.818305 21.171462 22.310620 24.488114 26.285130 32.094144 23.439411
[477] 29.518131 25.036817 33.932721 30.805710 32.192475 26.621050 34.998386
[484] 30.972503 31.111373 28.514509 29.950685 27.501073 28.928258 30.266318
[491] 21.257142 27.094142 18.892769 25.235055 18.599898 17.125135 20.975394
[498] 14.941908 13.006472 11.091895
```

b) Find the frequency of the seasonal component (Hint: use the autocorrelation plot. You must specify the lag.max parameter in acf() as the default is too small.) [show code and plot]

```
myvecACF <- acf(myvec, lag.max=498)

myvecACF
```



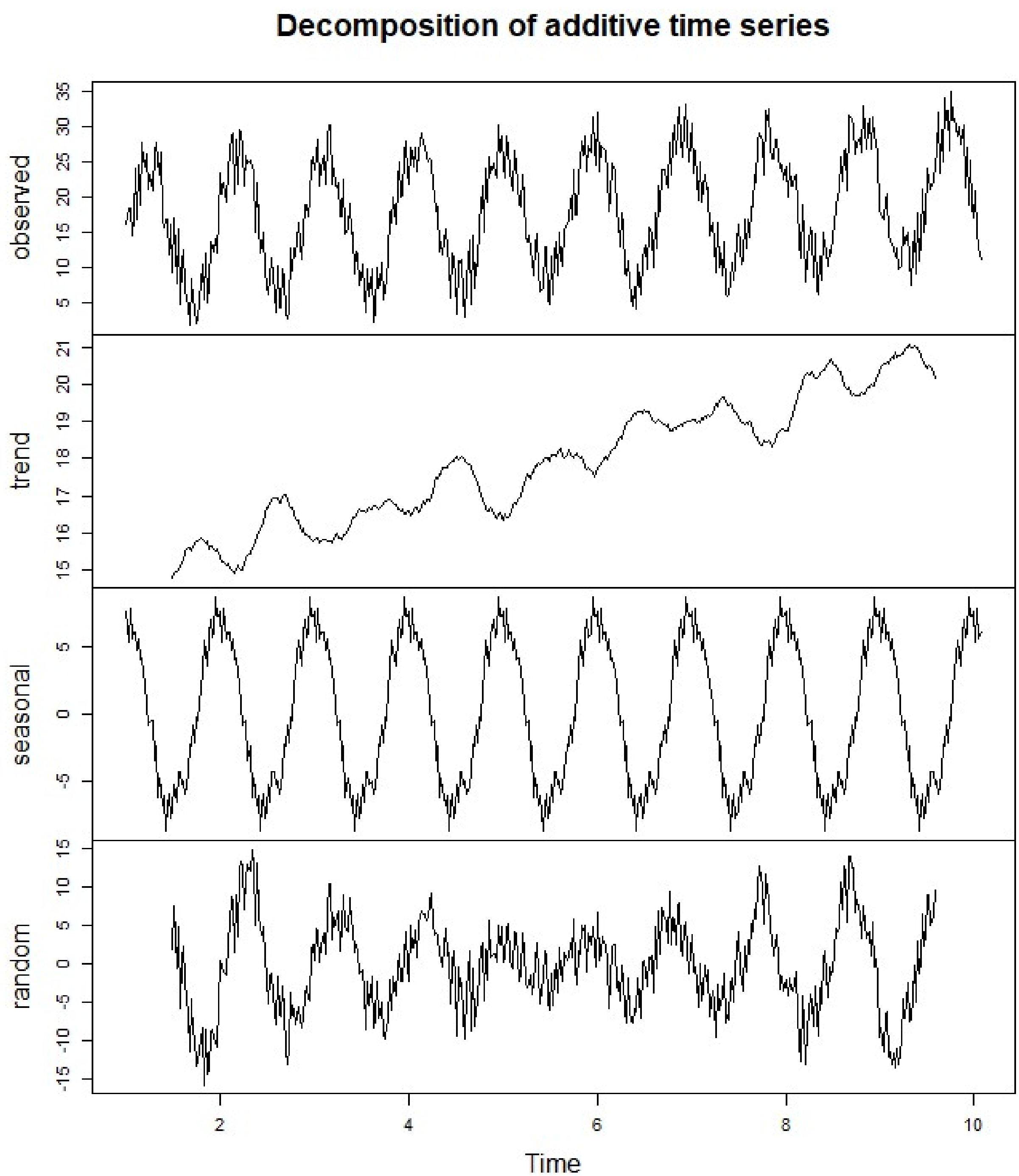
The frequency is approximated to be **55**

c) Convert to a `ts` object [show code]

```
myvects <- ts(myvec, frequency = 55)
```

d) Decompose the ts object. Plot the output showing the trend, seasonal, random components. [show code and plot]

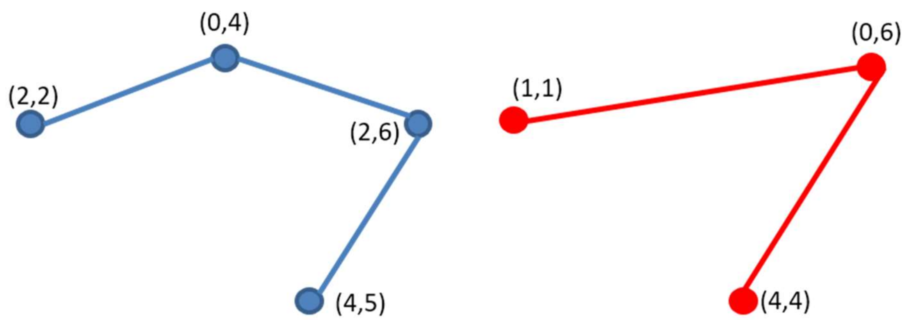
```
myvec.ts <- plot(decompose(myvects))
```



2. (Same as classwork problem) Compute the Dynamic Time Warping distance between the two time series, A and B:

A= (2,2), (0,4), (2,6), (4,5)

B= (1,1), (0,6), (4,4)



Use squared Euclidean distance as the cost function: $cost(A_i, B_j) = (A_{i,x} - B_{j,x})^2 + (A_{i,y} - B_{j,y})^2$.

a) Show the cost matrix. This is partially complete below.

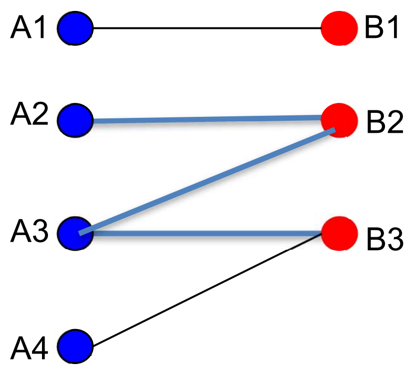
	B_1	B_2	B_3
A_1	2	20	8
A_2	10	4	16
A_3	26	4	8
A_4	25	17	1

b) Show the DTW matrix. This is partially complete below.

	B_1	B_2	B_3
A_1	2	22	30
A_2	12	6	22
A_3	38	10	14
A_4	63	27	11

c) The DTW distance between the two time-series is **11**.

d) Mark the optimal alignment between the two time-series in the diagram below.



3. a) Complete the R function below to compute the DTW distance between two time-series, v1 and v2, each containing 2D points and using the cost function as in Q2 above. So v1 and v2 will have two columns but different numbers of rows.

```
dtw <- function (A, B) {
  M <- nrow(A)
  N <- nrow(B)
  Cost <- matrix(0,M,N) # Initialize with zeros
  for (i in 1:M) {
    for (j in 1:N) {
        Cost[i,j] <- as.numeric((A[i,1] - B[j,1])^2 + (A[i,2] -
B[j,2])^2) # distance function
      }
  }
  C <- matrix(0,M,N) # Initialize with zeros
  C[1,1] <- Cost[1,1] # Initialize top left cell
  for (i in 2:M) { # Initialize first column
      C[i,1] <- C[i-1,1] + Cost[i,1]
  }
  for (j in 2:N) { # Initialize first row
      C[1,j] <- C[1,j-1] + Cost[1,j]
  }
  #
  # Complete the main loop
  #
  for(i in 2:M) {
    for(j in 2:N) {
      C[i,j] = min(C[i-1,j], C[i,j-1], C[i-1,j-1]) + Cost[i,j]
```

```
        }
      }
      return (C[M,N])
    }
```

b) Verify your answer to Q2 using the above function. [show code]

Hint: You can create the two input time-series as a two-column data.frame/tibble like so:

`A <- tibble("x" = c(2, 0, 2, 4), "y" = c(2, 4, 6, 5))`

A <- tibble("x" = c(2, 0, 2, 4), "y" = c(2, 4, 6, 5))

B <- tibble("x" = c(1, 0, 4), "y"=c(1,6,4))

dtwFunc <- dtw(A, B)

dtwFunc = **11**

4. You are given 5 time-series of 2D points (2 column tables) in CSV files: ts2.csv, ts3.csv, ts4.csv, ts5.csv, and tsX.csv. Your goal is to identify which of the time series, ts2-ts5, is most similar to the tsX time series using DTW.

a) Explain your approach in 2-3 sentences.

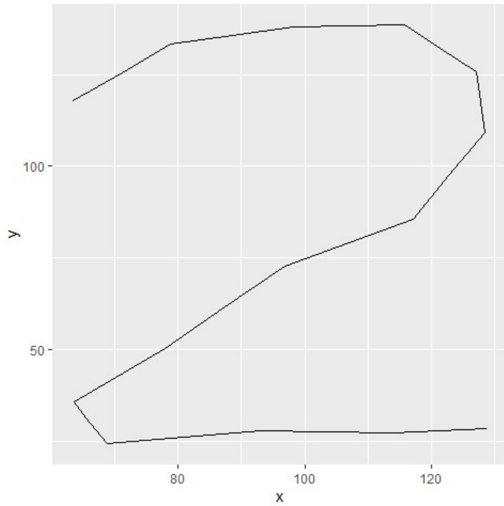
In order to find out which csv file is the most similar to tsX, we would have to create a time-series plot. With this, we would have to utilize the ggplot() and geom_path() functions.

b) Show your R code

```
ts2 <- read_csv("ts2.csv")
ts3 <- read_csv("ts3.csv")
ts4 <- read_csv("ts4.csv")
ts5 <- read_csv("ts5.csv")
tsX <- read_csv("tsX.csv")

ts2Plot <- ggplot(ts2, aes(x=x,y=y))
ts2Plot + geom_path()
ggsave("ts2.pdf")

ts3Plot <- ggplot(ts3, aes(x=x,y=y))
ts3Plot + geom_path()
ggsave("ts3.pdf")

ts4Plot <- ggplot(ts4, aes(x=x,y=y))
ts4Plot + geom_path()
ggsave("ts4.pdf")

ts5Plot <- ggplot(ts5, aes(x=x,y=y))
ts5Plot + geom_path()
ggsave("ts5.pdf")

tsXPlot <- ggplot(tsX, aes(x=x,y=y))
tsXPlot + geom_path()
ggsave("tsX.pdf")
```

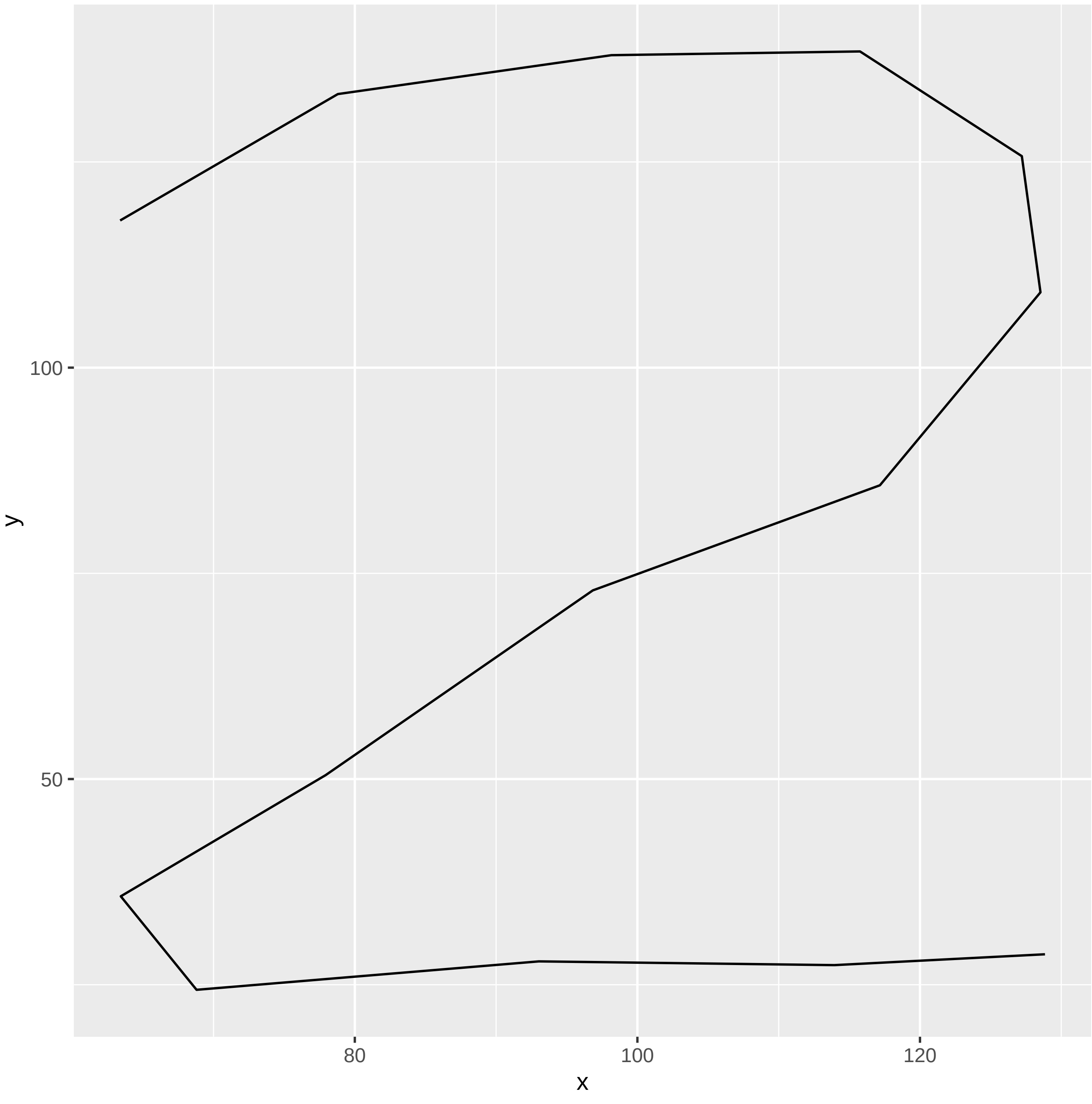
c) tsX is most similar to: ts5

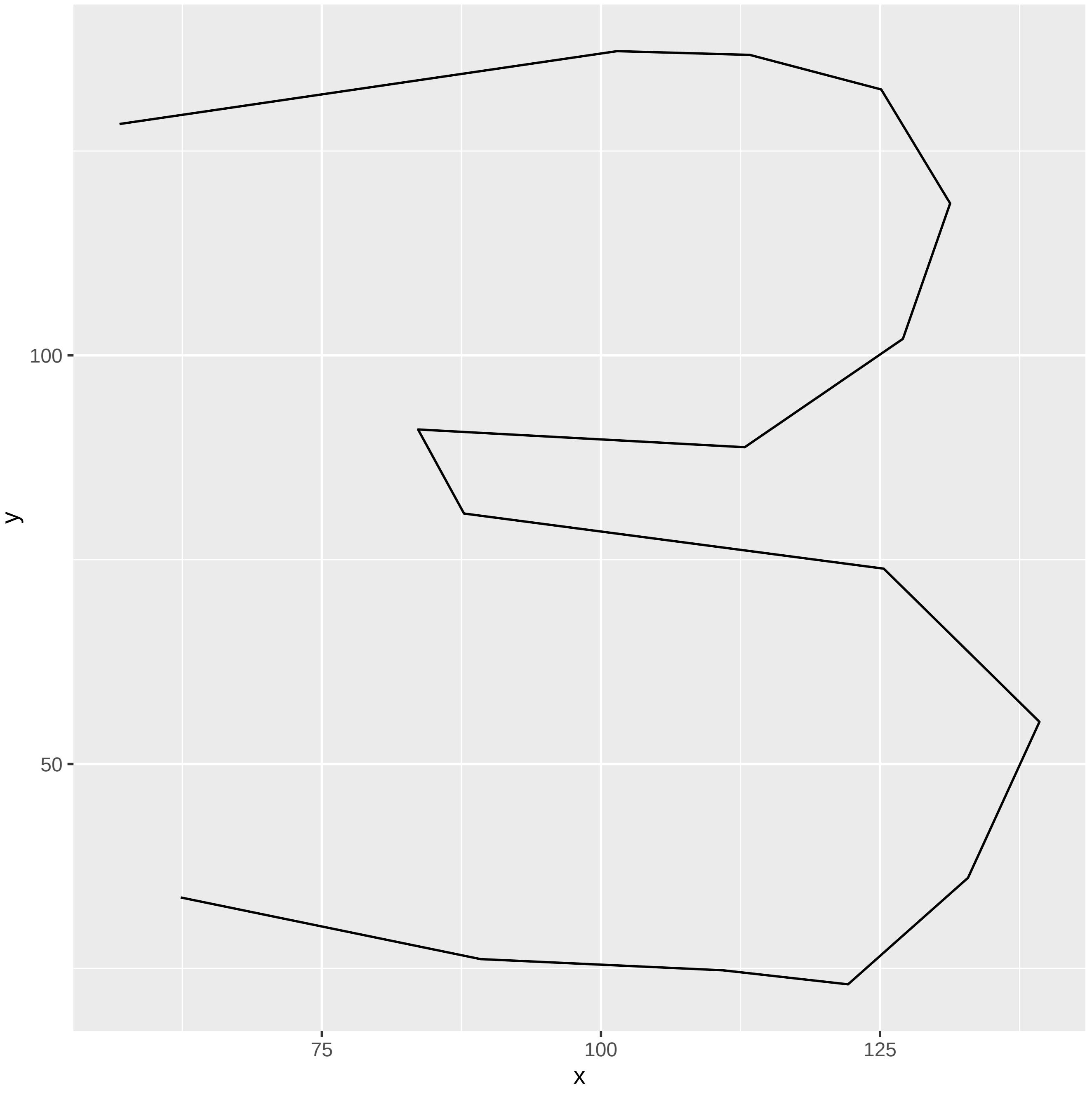
Hint: Use the DTW function from Q3. You can visualize the series of 2D points using geom_path(). For example, ts2:

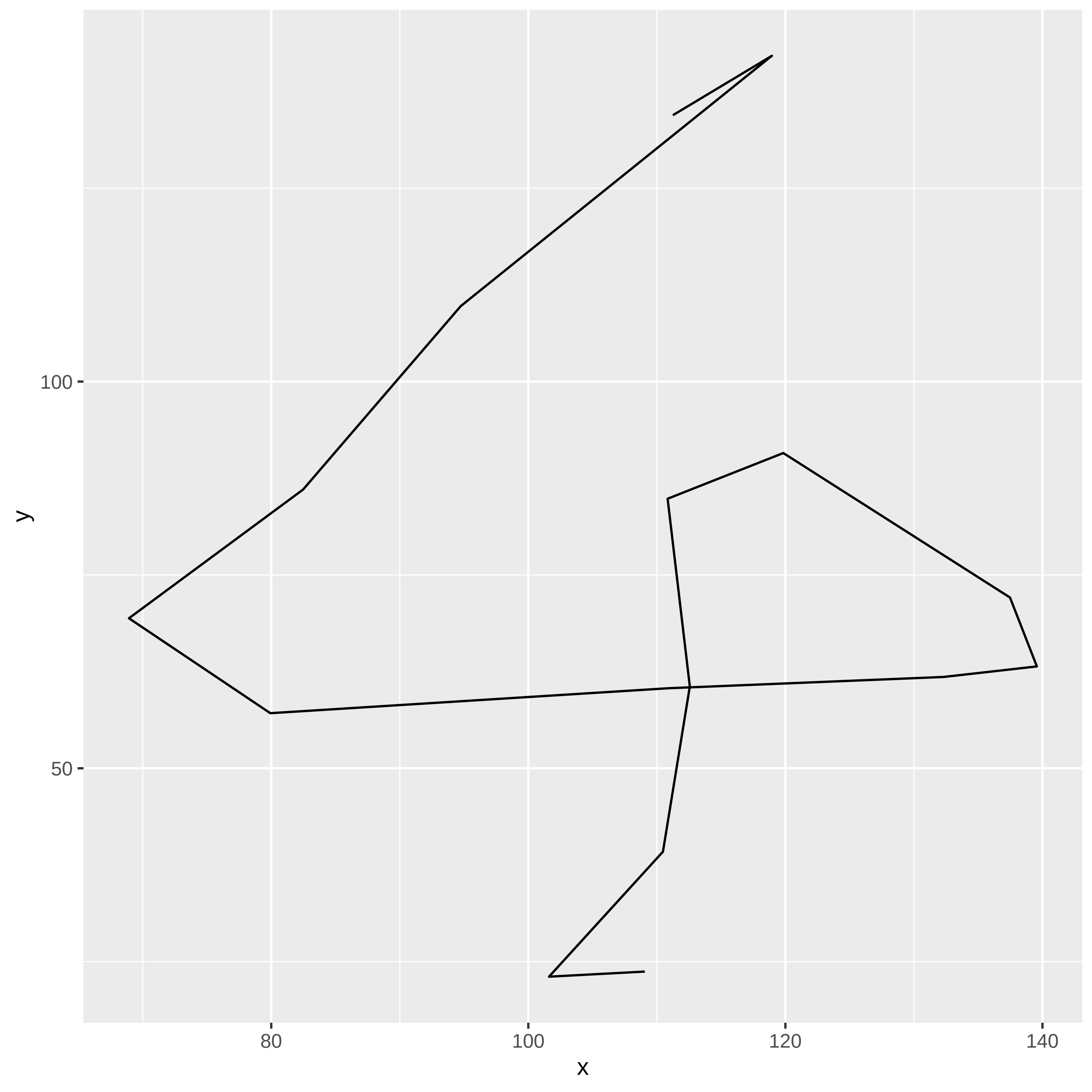


```r
library(tidyverse)

load("myvec.RData")

myvec

myvecACF <- acf(myvec, lag.max=498)

myvecACF

myvects <- ts(myvec, frequency = 55)

myvec.ts <- plot(decompose(myvects))

dtw <- function (A, B) {
  M <- nrow(A)
  N <- nrow(B)
  Cost <- matrix(0,M,N) # Initialize with zeros
  for (i in 1:M) {
    for (j in 1:N) {
      Cost[i,j] <- as.numeric((A[i,1] - B[j,1])^2 + (A[i,2] - B[j,2])^2) # distance function
    }
  }
  C <- matrix(0,M,N) # Initialize with zeros
  C[1,1] <- Cost[1,1] # Initialize top left cell
  for (i in 2:M) { # Initialize first column
    C[i,1] <- C[i-1,1] + Cost[i,1]
  }
  for (j in 2:N) { # Initialize first row
    C[1,j] <- C[1,j-1] + Cost[1,j]
  }
  #
  # Complete the main loop
  #
  for(i in 2:M) {
    for(j in 2:N) {
      C[i,j] = min(C[i-1,j], C[i,j-1], C[i-1,j-1]) + Cost[i,j]
    }
  }
  return (C[M,N])
}

A <- tibble("x" = c(2, 0, 2, 4), "y" = c(2, 4, 6, 5))
B <- tibble("x" = c(1, 0, 4), "y"=c(1,6,4))

dtwFunc <- dtw(A, B)

dtwFunc

ts2 <- read_csv("ts2.csv")
ts3 <- read_csv("ts3.csv")
ts4 <- read_csv("ts4.csv")
ts5 <- read_csv("ts5.csv")
tsX <- read_csv("tsX.csv")

ts2Plot <- ggplot(ts2, aes(x=x,y=y))
ts2Plot + geom_path()
```

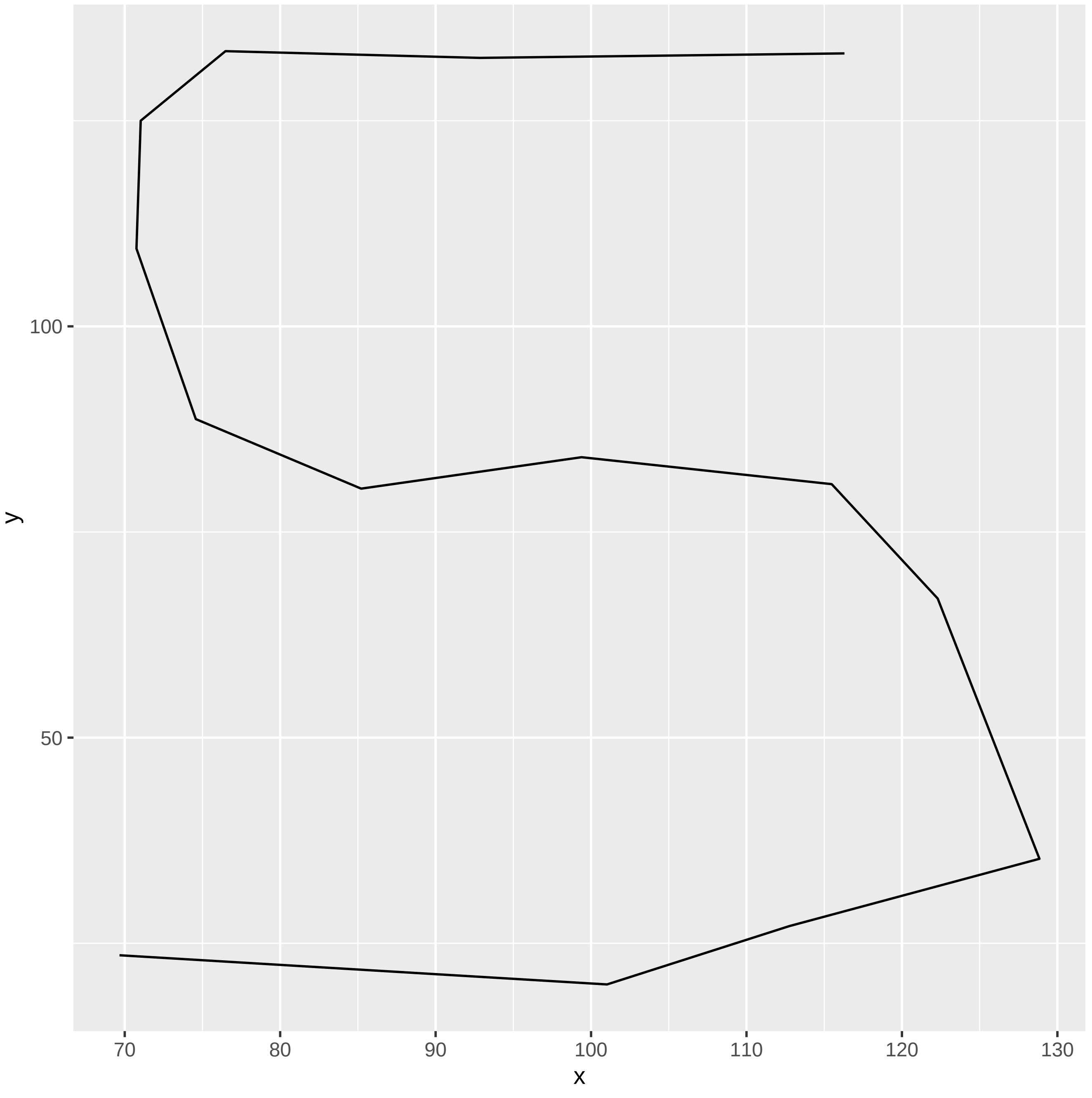
```r
ggsave("ts2.pdf")

ts3Plot <- ggplot(ts3, aes(x=x,y=y))
ts3Plot + geom_path()
ggsave("ts3.pdf")

ts4Plot <- ggplot(ts4, aes(x=x,y=y))
ts4Plot + geom_path()
ggsave("ts4.pdf")

ts5Plot <- ggplot(ts5, aes(x=x,y=y))
ts5Plot + geom_path()
ggsave("ts5.pdf")

tsXPlot <- ggplot(tsX, aes(x=x,y=y))
tsXPlot + geom_path()
ggsave("tsX.pdf")
```


100
50
y
80
100
120
x

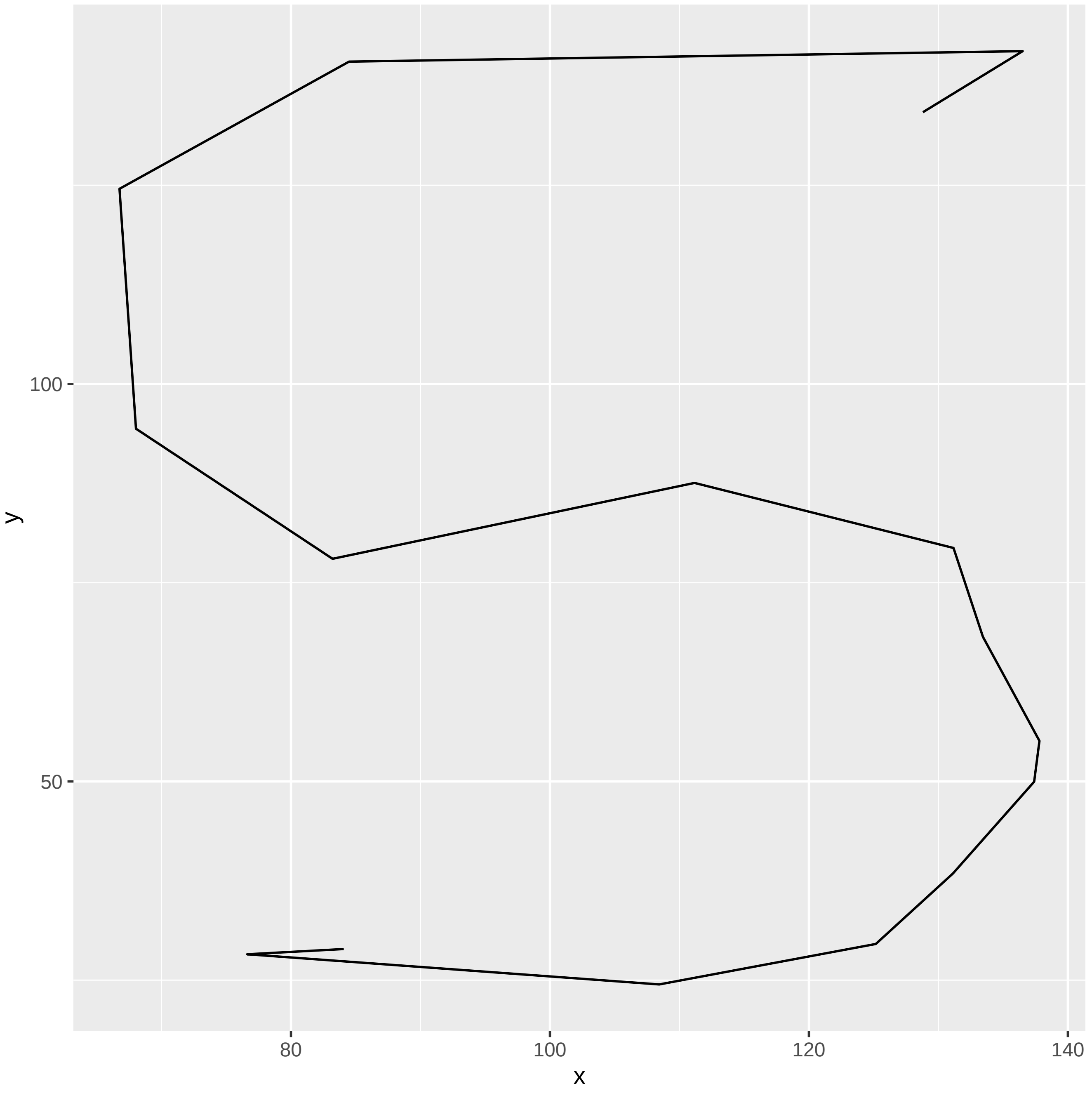

100
50
y
75
100
125
x



100
50
y
80
100
120
140
x



100
50
y
70
80
90
100
110
120
130
x


100
50
y
80
100
120
140
x