Explanation

Moritz Mennenga

15. August 2017

A small explanation of how it works:

Aim

The aim is to rotate the profile fotogrammetric control points (fgc) so that it is possible to rectify profile images of archaeological excavations.

The georeferencer of QGIS provides the projective transformation for georeferencing, that is used to rectify the profiles.

Requirement

- Minimum 5 fotogrammetric control points of a profile
- These points have to be on one plane
- The best results are if the plane is perfect straight and vertical

Execution

Loading the data.

The data basis can be a spatial data frame with 3D points or a table with x,y and z coordinates. The process is only for cartesian coordinates.

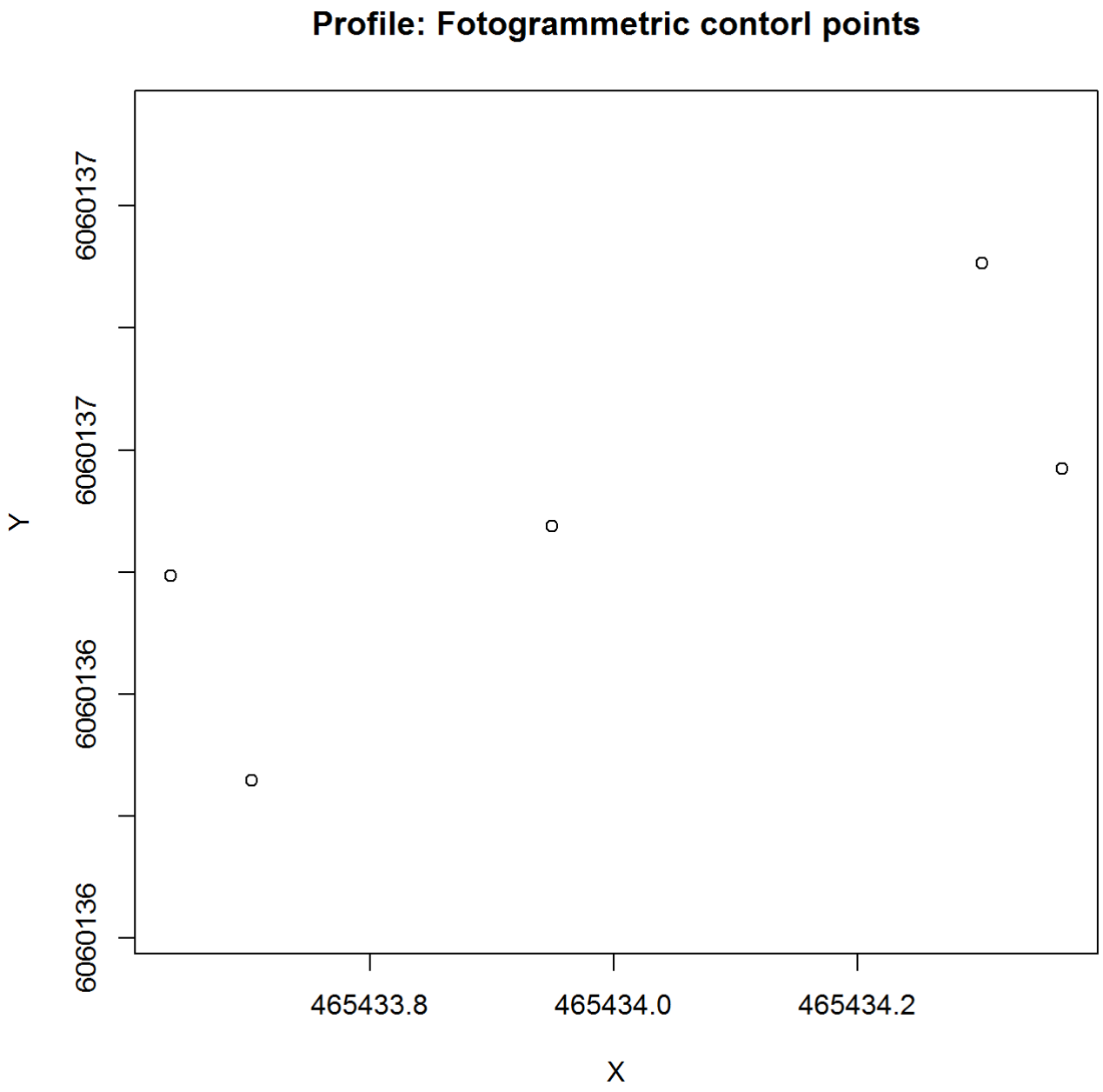
These are the input options

- *(Spatial)dataframe*: Input data
- *view_col*: The column with the information of the direction of view (the position of the observer towards the profile -> “N, E, S, W”). This information influences the direction of rotation.
- *profile_col*: The column with the information of the profilenames.
- *view*: There are two possibilities for the transformation of data. In the *surface* option, the points are processed for an orthogonal view on the profile. In the *projected* case, the data is transformed for a view on a vertical sectional pane (like a traditional drwaing on an excavation). Standart: *surface*
- *direction*: The position of the fgc in relation to the profile. The *original* option exports the data (nearly) parallel to the cutting line. *horizontal* will show the profile parallel to the x-axis (internal differences in height are still present). Standart: *horizontal*

```
devtools::load_all()
library(sp)
fotogram_sdf <- sp::SpatialPointsDataFrame(coords = fotogram_pts[ ,c(1,2,3)],
                                           data <- fotogram_pts,
                                           proj4string = sp::CRS('+proj=utm +zone=32 +ellps=G
RS80 +units=m +no_defs'))

#As an example only one profile
fotogram_sdf <- fotogram_sdf[fotogram_sdf@data$pr == 3,]
```

If plotted in QGIS the fgc points would look like the following. The profile is tilt by 20° backwards. I.e. the points in south-east are below.

```
plot(fotogram_sdf@data$x, fotogram_sdf@data$y, asp = 1, main = "Profile: Fotogrammetric conto
rl points", xlab = "X", ylab =  "Y")
```



Step one: Preparing the data

To bring the data into the 2.5D space and to fold them forward, they must first be rotated. This step is also more practical when the data is to be exported for printing in the GIS.

At first we have to define the options. in this case we use the standart options for the profiles.

```
fotogram_pts <- NULL
fotogram_pts <- fotogram_sdf

#The fuction for a sdf is called by
#profile <- archprofile(
#   fotogram_pts = fotogram_sdf,
#   profile_col = "pr",
#   view_col = "view"
# )

#In this example everything will be done step by step
profile_col <- "pr"
view_col <- "view"
view <- "projected"
direction <- "horizontal"
```

Now we are checking if the data is a spatialdataframe (in this case it is important, that the sdf has a @coords [, 3] column, if not it is a 2D sdf and not to use) or a dataframe (containing x, y, z). The coordinates and the relevant data is stored in a dataframe.

```
#If input is a spatialdataframe
  if(typeof(fotogram_pts)=="S4"){
    coord <- data.frame(
    x = fotogram_pts@coords[, 1],
    y = fotogram_pts@coords[, 2],
    z = fotogram_pts@coords[, 3],
    pr = fotogram_pts@data[, profile_col],
    view = fotogram_pts@data[, view_col]
  )
  }

# If input is a dataframe
  if(typeof(fotogram_pts) == "list" && !missing(z) ||
     typeof(fotogram_pts) == "list" && !missing(y) ||
     typeof(fotogram_pts) == "list" && !missing(x)){

      coord <- data.frame(
      x = fotogram_pts[,x],
      y = fotogram_pts[, y],
      z = fotogram_pts[, z],
      pr = fotogram_pts[, profile_col],
      view = fotogram_pts[, view_col]
    )
  } else if(typeof(fotogram_pts) == "list" && missing(z) ||
            typeof(fotogram_pts) == "list" && missing(y) ||
            typeof(fotogram_pts) == "list" && missing(x)) {
    stop('Coordinates missing')
  }

colnames(coord) <- c("x", "y", "z", "pr","view")

fotogram_pts@data
```

```
##          x       y       z view pr
## 1 465433.7 6060136 4.36277    N  3
## 2 465434.4 6060137 4.36426    N  3
## 3 465433.9 6060137 4.65011    N  3
## 4 465433.6 6060136 4.83866    N  3
## 5 465434.3 6060137 4.84038    N  3
```

For performance reasons, some additional steps will be done:

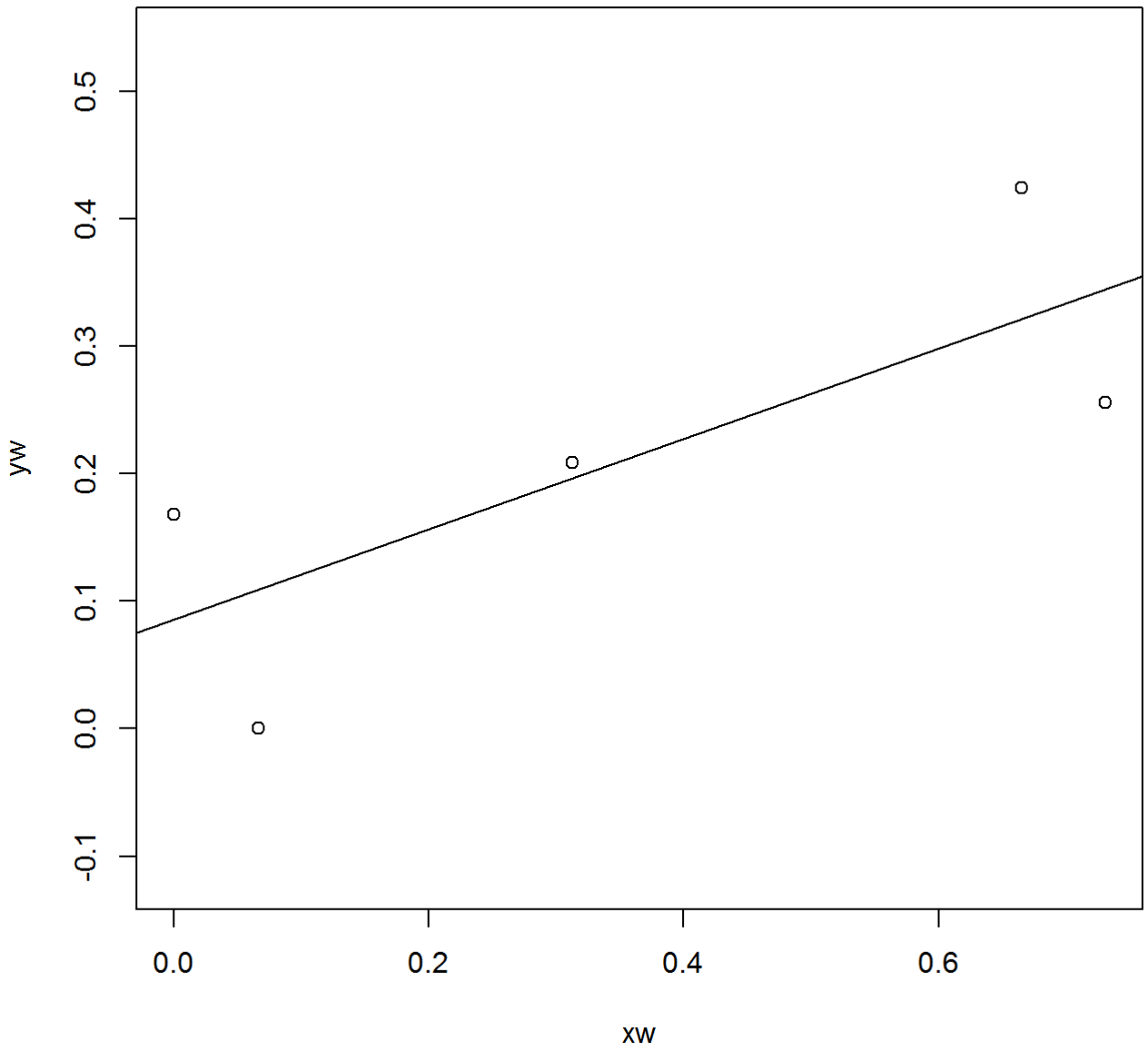
```
  #Now starting with each profile individual
  #possible nas has to be omitted
  coord <- na.omit(coord)
  #Getting all names of the profiles, to use the amoun for the n of iterations of the loop
  prnames <- levels(as.factor(coord$pr))

  #A dataframe with the same length as the import is used for the export,
  #A copy of the import df is used. All colums have 0 values for seeing errors

  coord_export <- coord
  coord_export$view <- NULL
  coord_export$pr <- 0
  coord_export$x <- 0
  coord_export$y <- 0
  coord_export$z <- 0
```

Step two: roating on the z-axis

For each profile we start with rotating the profile parallel to the x-axis:

Therefore we use a dataframe containig only the data of one (the actual) profile.
For roating the fgcs, the slope of the profile in relation th the x-axis is determined. For this step a linear regression is used.

```
i<-1 #Countingvariable (in this case 1, normally the amount of profiles)

    #Writing all data of the actual profile in a teporary dataframe
    coord_proc <- coord[which(coord$pr == prnames[i]),]

    #A linear regression is used to get the gradient of the profile,
    #the regression balances the askew profile
    #First step is to rotate the profile control points around z-axis
    #therefore the angle of the profile to the x axis is necessary
    yw <- c(coord_proc$y) - min(coord_proc$y)
    xw <- c(coord_proc$x) - min(coord_proc$x)
    fm <- lm(yw ~ xw)

    #extrakte the solpe of the profile
    slope <- coef(fm)[2]

plot(xw, yw, asp=1, main = "Profile: Fotogrammetric control points and regression line")
abline(fm)
```



The slope of the straight and the point of view determines the angle of rotation. It will be distinguished between positive and negative slopes and the four main cardinal directions.The view is defined by the position of the observer (view from east)

There are four cases (+ special case m = 0):

1. m = - view = N/E -> clockwise
2. m = - view = S/W -> counterclockwise
3. m = + view = S/E -> clockwise
4. m = + view = N/W -> counterclockwise

Now we have to find the angle of rotation for this cases (rad -> deg)

```
#Um die Fotogrammetrienägel korrekt anzeigen zu können, sollen diese gedreht werden.
#Dazu muss der Winkel zwischen der Regressionsgerade und der x-Achse berechnet werden
#Steigung der Gerade
slope <- coef(fm)[2]

    view_proc <- coord_proc$view[1]

if (slope < 0 && view_proc %in% c("N", "E")) {
  slope_deg <- 180 - abs((atan(slope) * 180) / pi) * -1
} else if (slope < 0 && view_proc %in% c("S", "W")) {
  slope_deg <- abs((atan(slope) * 180) / pi)
} else if (slope > 0 && view_proc %in% c("S", "E")) {
  slope_deg <- ((atan(slope) * 180) / pi) * -1
} else if (slope > 0 && view_proc %in% c("N", "W")) {
  slope_deg <- 180 - ((atan(slope) * 180) / pi)
} else if (slope == 0 && view_proc == "N") {
  slope_deg <- 180
} else if (slope == 0 && view_proc == "N") {
  slope_deg <- 0
}

slope_deg
```

```
##       xw
## 160.4551
```

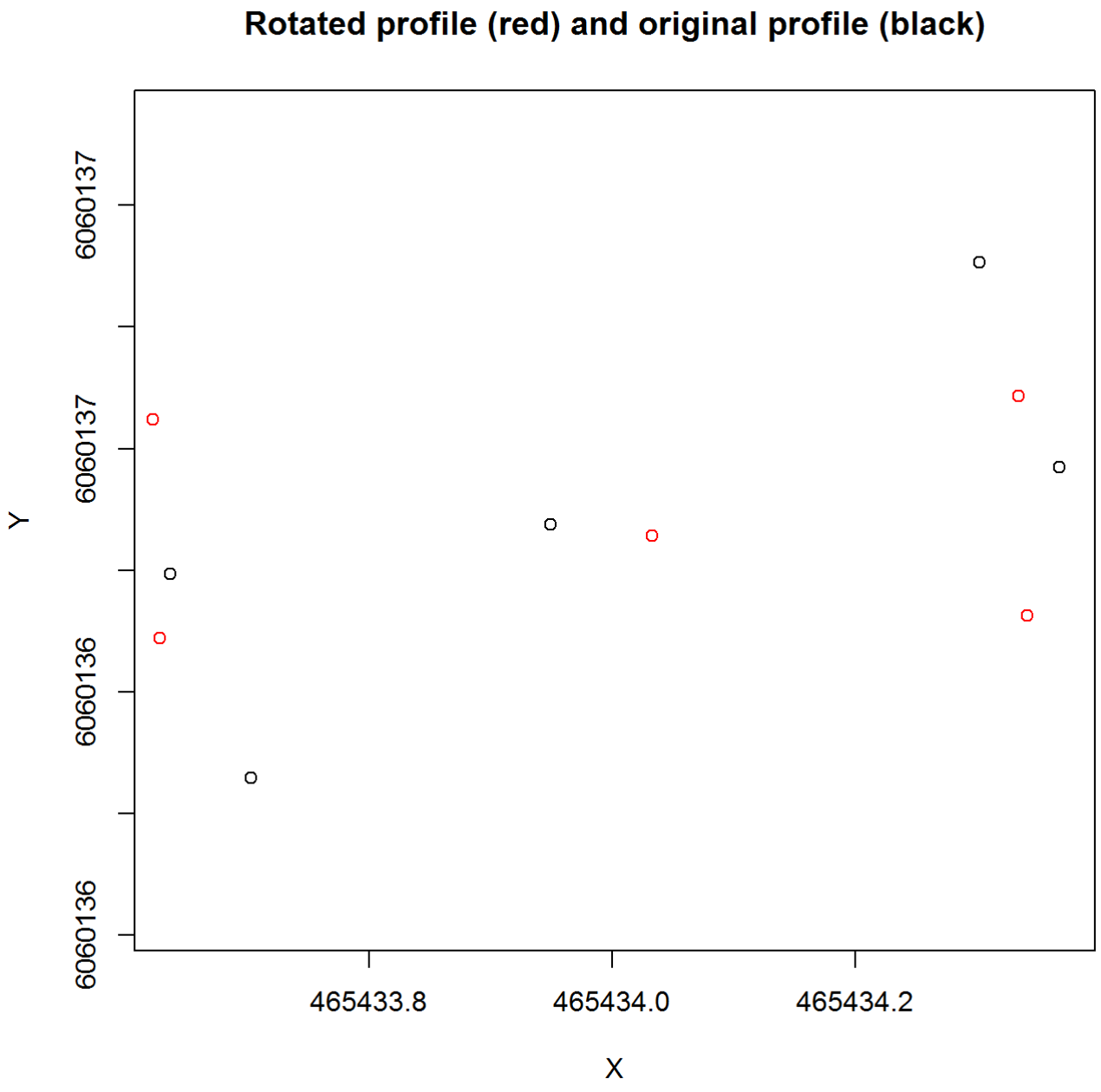
Now it is neccessary to determine the rotation point.

```
#Next step is to find the rotation point
#This is in the middle of the profile
center_x <- sum(coord_proc$x) / nrow(coord_proc)
center_y <- sum(coord_proc$y) / nrow(coord_proc)

#Rotate around the point and use a temp dataframe
coord_trans <- coord_proc
#df without the view column
coord_trans$view <- NULL
```

Next part is the rotation of the points. (Example view from North) Therfore a temporary dataframe is used, which will include the transformed and rotated values. Source:
http://www.matheboard.de/archive/460078/thread.html
(http://www.matheboard.de/archive/460078/thread.html*)

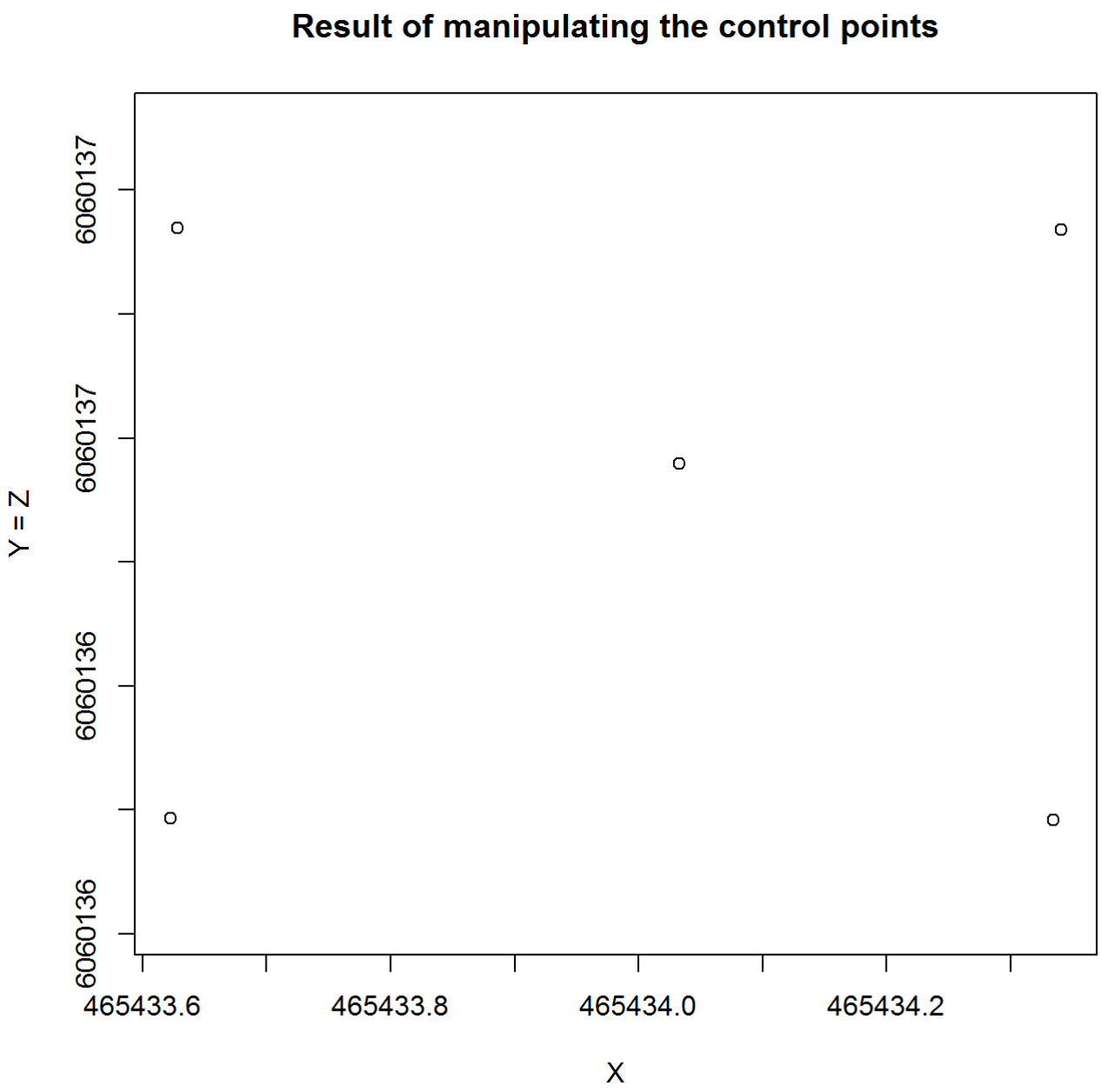
An important point is *coord_proc*$z[z] + center_y - mean(coord_procz)$. The z-Values will be fitted to the y-Values, thereby the fgc are next to the feature in GIS.

```
#We use translation and rotation to find the new points
#The mean y-Value will be added to the z-Value, therefore the control points are on the profi
le in the end
for (z in 1:nrow(coord_proc)) {
  coord_trans[z,] <- c(
center_x + (coord_proc$x[z] - center_x) *
  cos(slope_deg / 180 * pi) - sin(slope_deg / 180 * pi) *
  (coord_proc$y[z] - center_y),
center_y + (coord_proc$x[z] - center_x) *
  sin(slope_deg / 180 * pi) + (coord_proc$y[z] - center_y) *
  cos(slope_deg / 180 * pi),
coord_proc$z[z] + center_y - mean(coord_proc$z),
as.numeric(as.character(coord_trans$pr[z]))
  )
}
plot(coord$x, coord$y, asp = 1, main = "Rotated profile (red) and original profile (black)",
xlab = "X", ylab =  "Y")
points(coord_trans$x,coord_trans$y, col="red")
```



Now the fgcs are preparated with the standart values. For exporting the data it the Y and Z values have to be changed.

```
plot(coord_trans$x,coord_trans$z,asp=1, main = "Result of manipulating the control points", x
lab = "X", ylab =  "Y = Z")
```

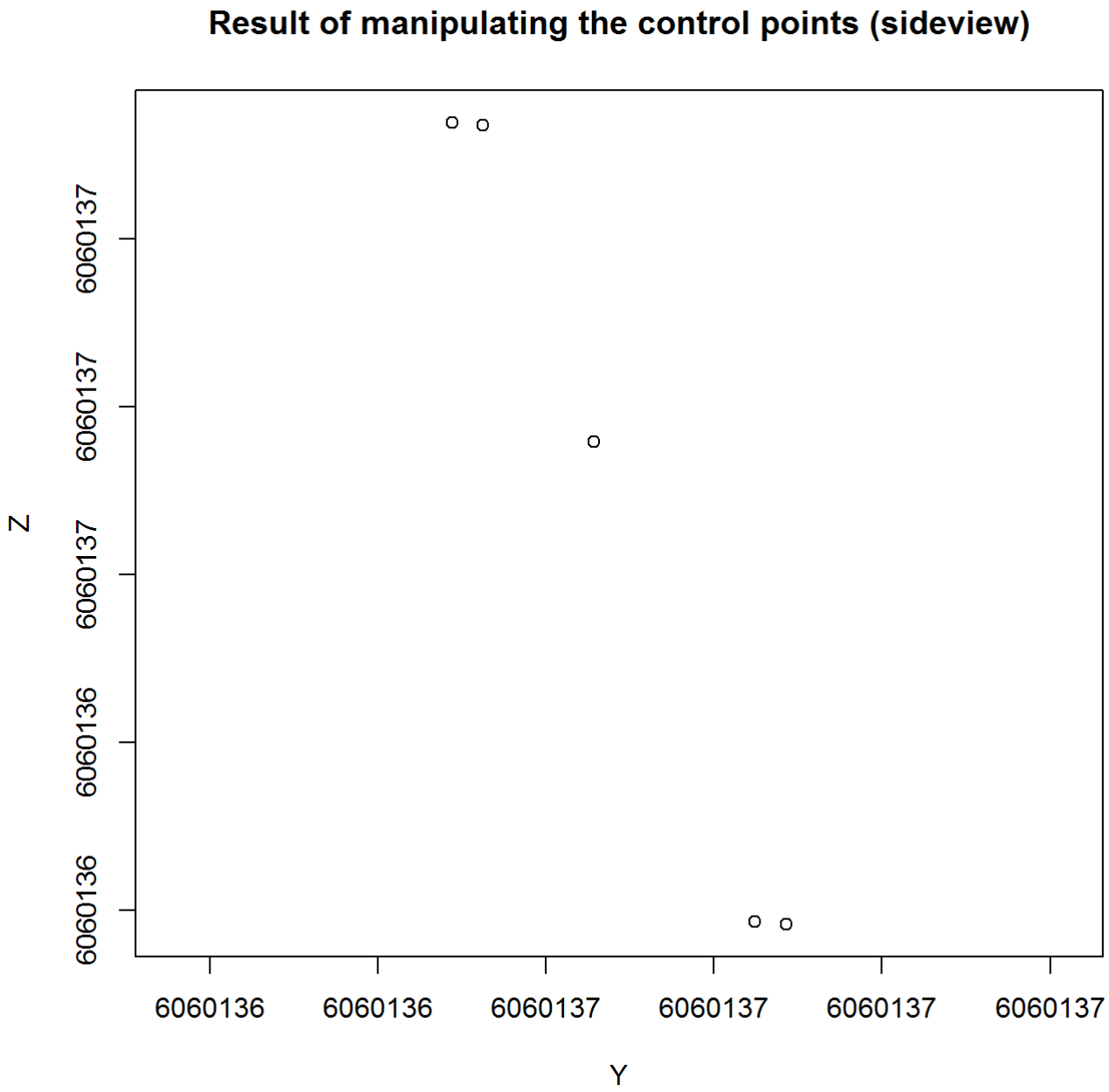



```
#Export Data
```

Option 1: Looking at the surface

The shown result is a projection of the profile on a virtual vertikal plane (like an archaeologists would do it while painting a profile on an excavation). But in some cases (e.g. processing the photos for a 3D presentation), it is important to have an orthogonal view on the profile. If it is a sideview, we see the profile was badly excavated, it tilts backwards

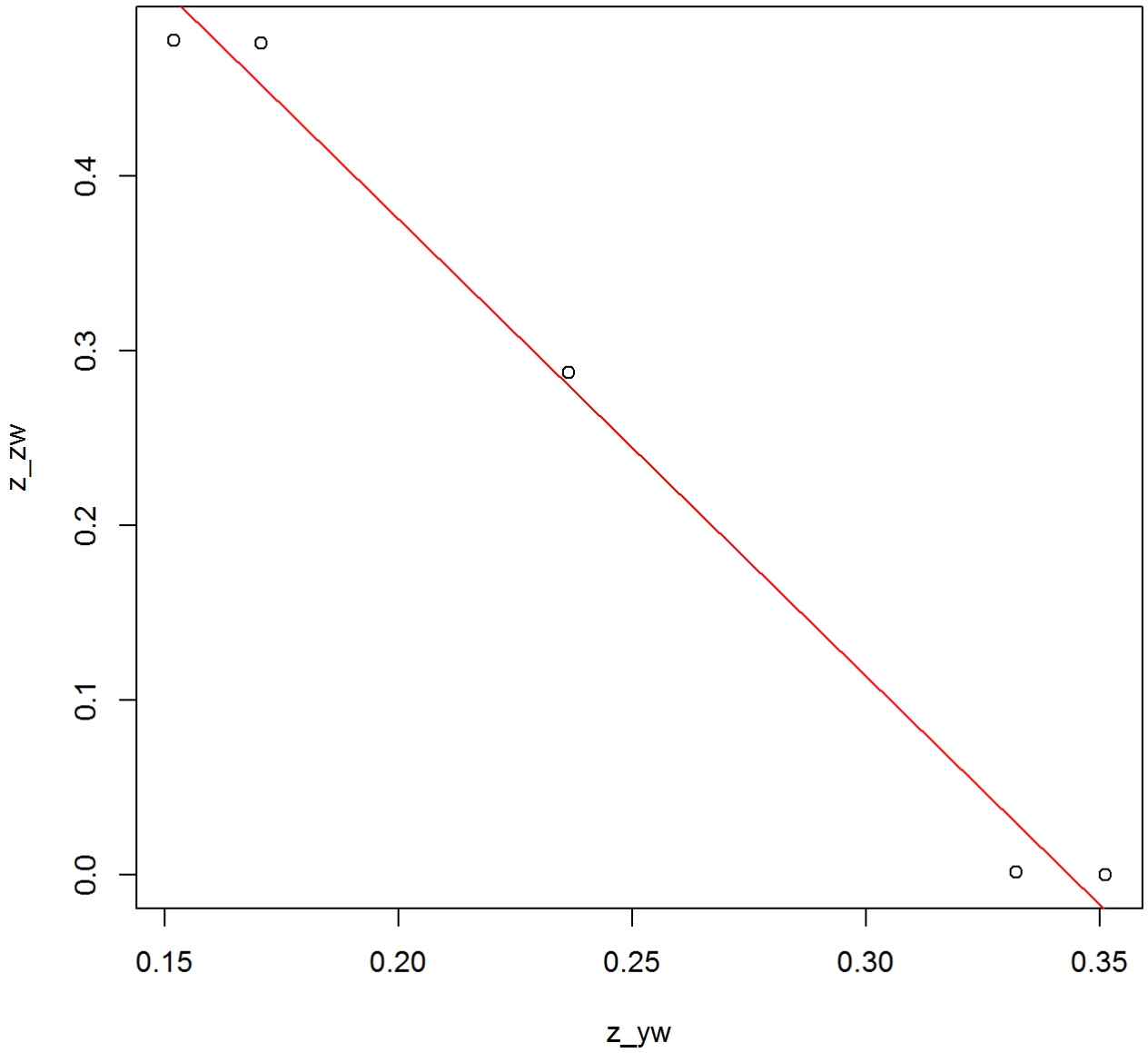
```
plot(coord_trans$y,coord_trans$z,asp=1, main = "Result of manipulating the control points (si
deview)", xlab = "Y", ylab =  "Z")
```




```
#Export Data
```

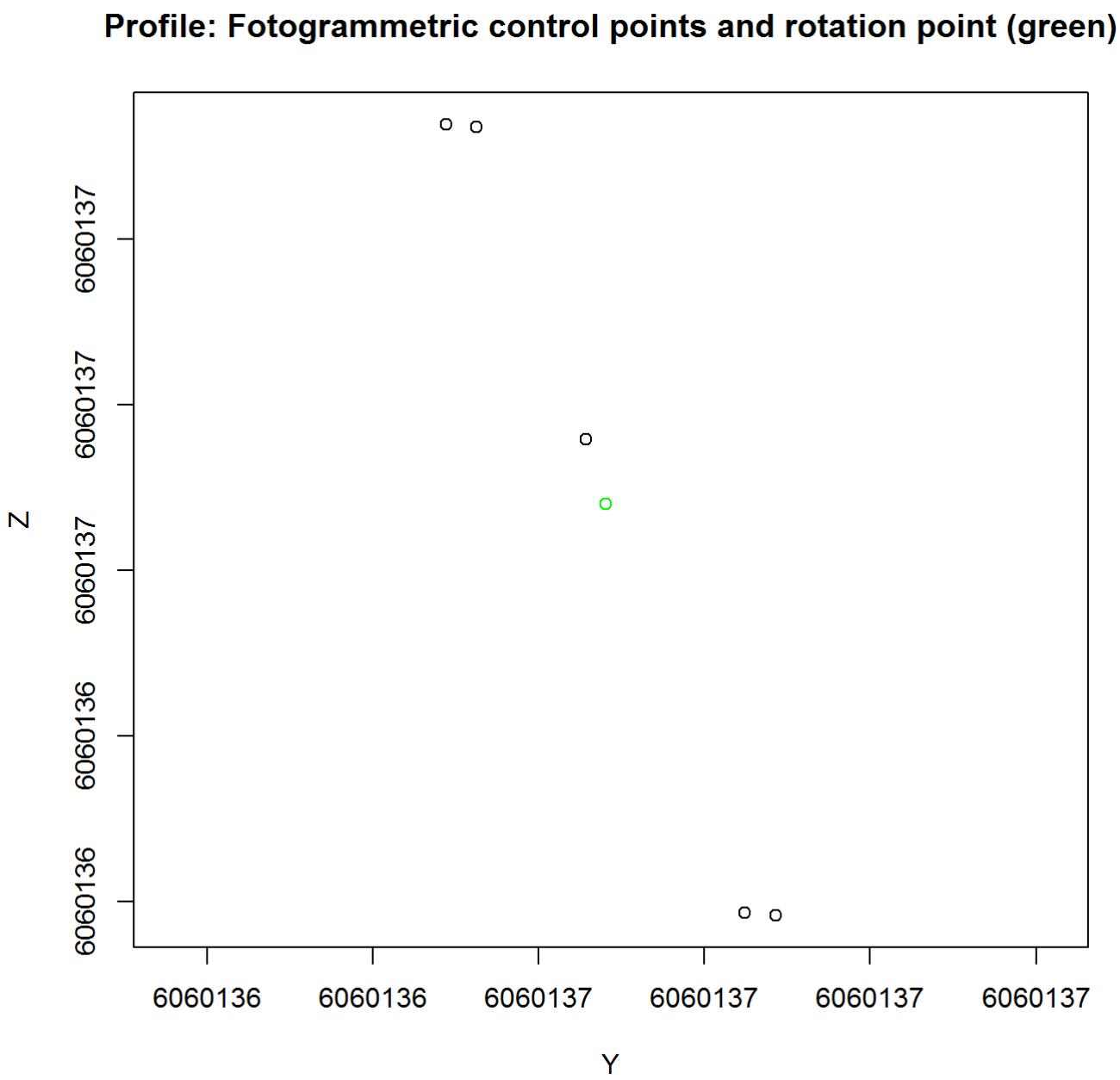
To fix this, the fgcs have to be rotated again. The profile have to be perfect vertical. The way of doing this, is exactly the same like before, but the rotation is not around the z-axis, but the x-axis.

```
#If the aim is to get the view of the surface, we have to do the same with a rotation on the
 x-axis

z_yw <- c(coord_trans$y - min(c(coord_trans$y, coord_trans$z)))
z_zw <- c(coord_trans$z - min(c(coord_trans$y, coord_trans$z)))
z_fm <- lm(z_zw ~ z_yw)


plot(z_yw, z_zw)
abline(z_fm, col="red", main = "Profile: Fotogrammetric control points and regression line",
xlab = "Y", ylab =  "Z")
```



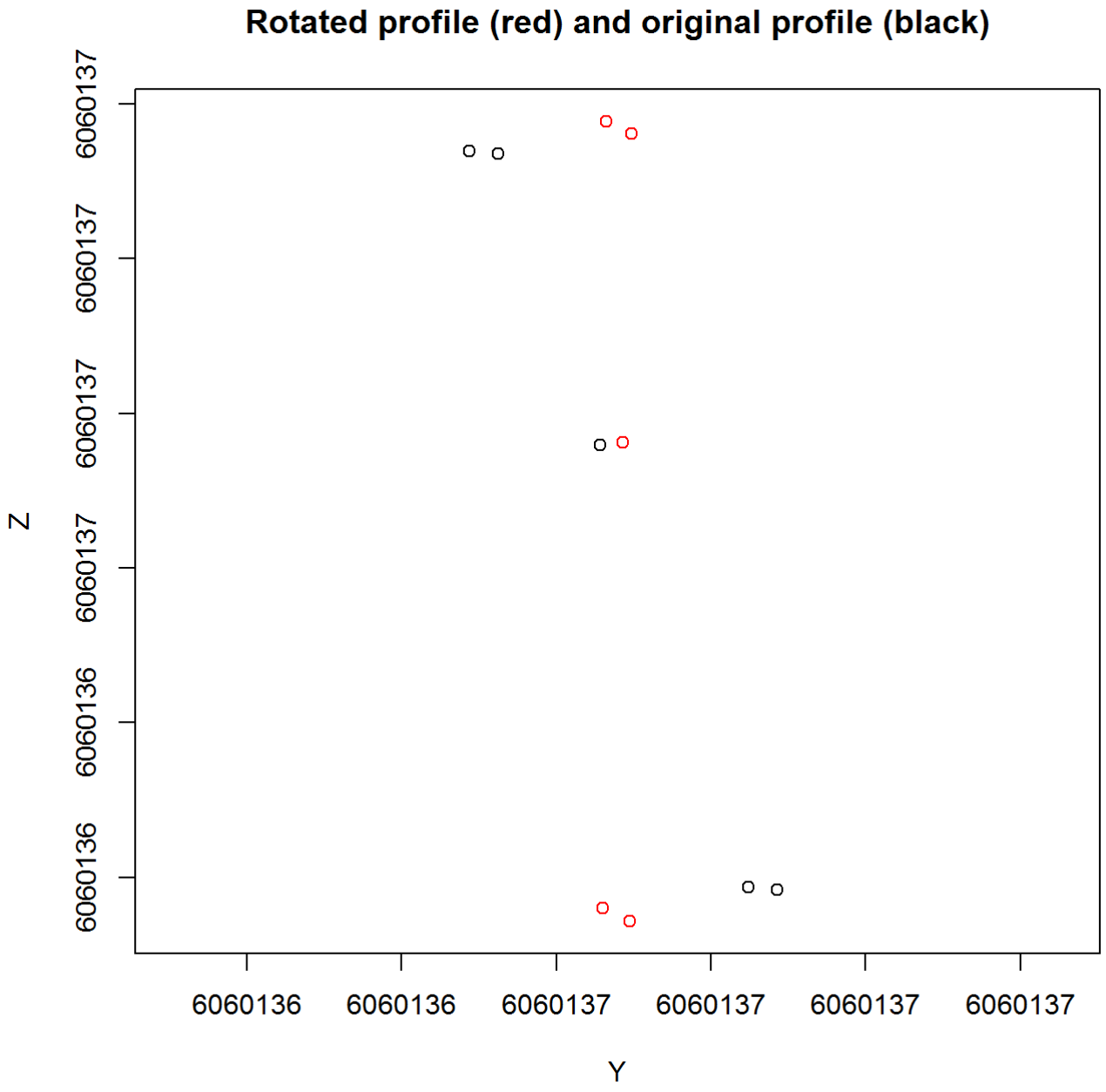
In this case the determination of the slope is much more easy, because the fgcs have to be straightend up. Therefore the rotation angle is postivie or negative. Afterwards the rotation point have to be found.

```
z_slope <- coef(z_fm)[2]

if (z_slope < 0) {
  z_slope_deg <- -(90 - abs((atan(z_slope) * 180) / pi))
} else if (z_slope > 0) {
  z_slope_deg <- 90 - ((atan(z_slope) * 180) / pi)
} else if (z_slope == 0) {
  z_slope_deg <- 0
}

z_center_y <- sum(coord_trans$y) / nrow(coord_trans)
z_center_z <- sum(coord_trans$z) / nrow(coord_trans)

plot(coord_trans$y, coord_trans$z, asp=1, main = "Profile: Fotogrammetric control points and
 rotation point (green)", xlab = "Y", ylab =  "Z")
points(z_center_y,z_center_z,col="green")
```



And now rotating everything around the x-axis.

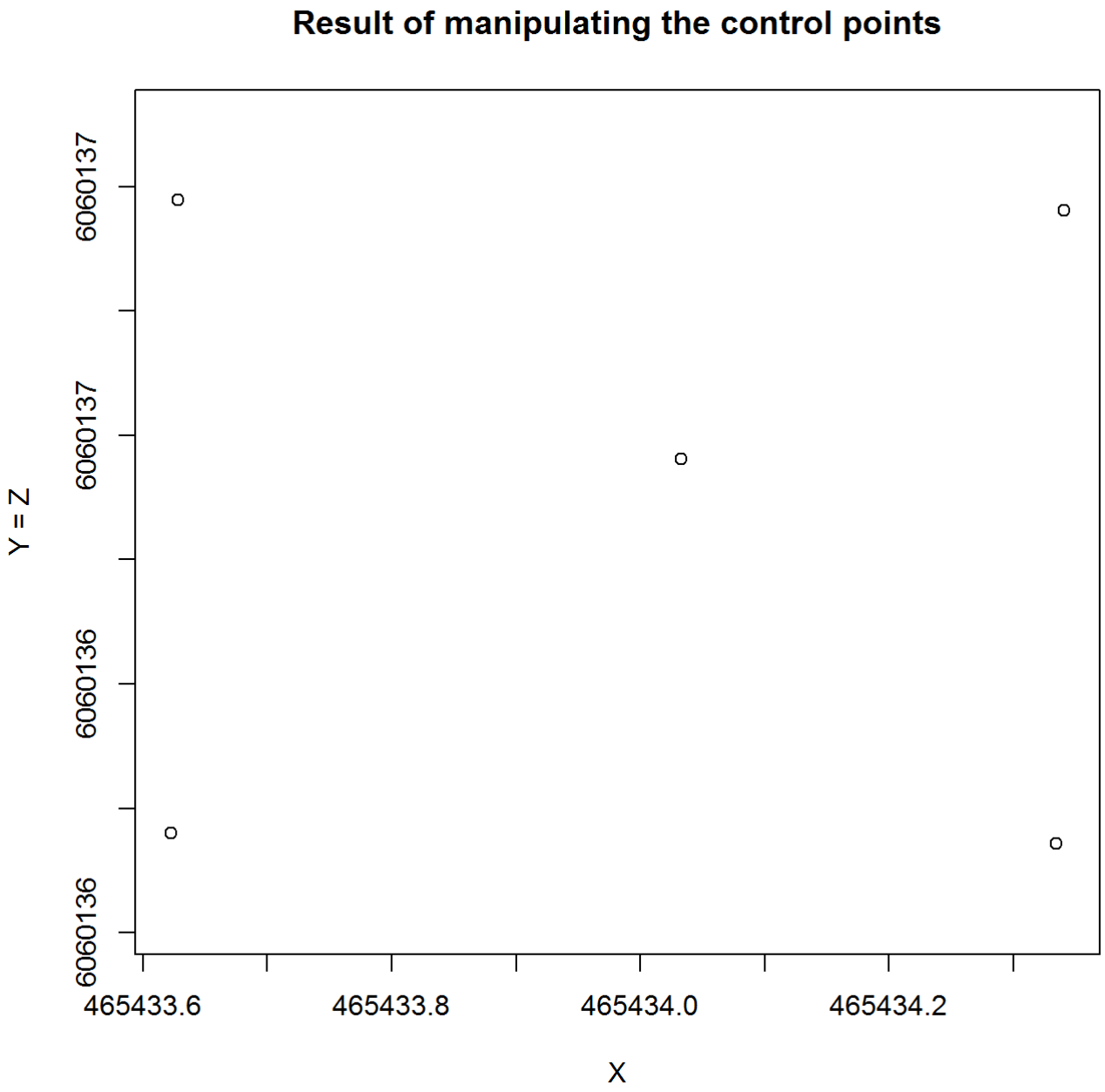
```
z_coord <- coord_trans

for (z in 1:nrow(z_coord)) {
  z_coord[z,] <- c(
    coord_trans$x[z],
    z_center_y + (coord_trans$y[z] - z_center_y) * cos(z_slope_deg / 180 * pi) -
(coord_trans$z[z] - z_center_z) * sin(z_slope_deg / 180 * pi),
    z_center_z + (coord_trans$y[z] - z_center_y) * sin(z_slope_deg / 180 * pi) +
(coord_trans$z[z] - z_center_z) * cos(z_slope_deg / 180 * pi),
    as.numeric(as.character(coord_trans$pr[z]))
  )
}

plot(z_coord$y,z_coord$z,asp=1, col = "red", main = "Rotated profile (red) and original profi
le (black)", xlab = "Y", ylab =  "Z")
points(coord_trans$y,coord_trans$z, col = "black")
```




```
coord_trans <- z_coord
```

The result (after changing y and z axis, are the fgc on the surface)

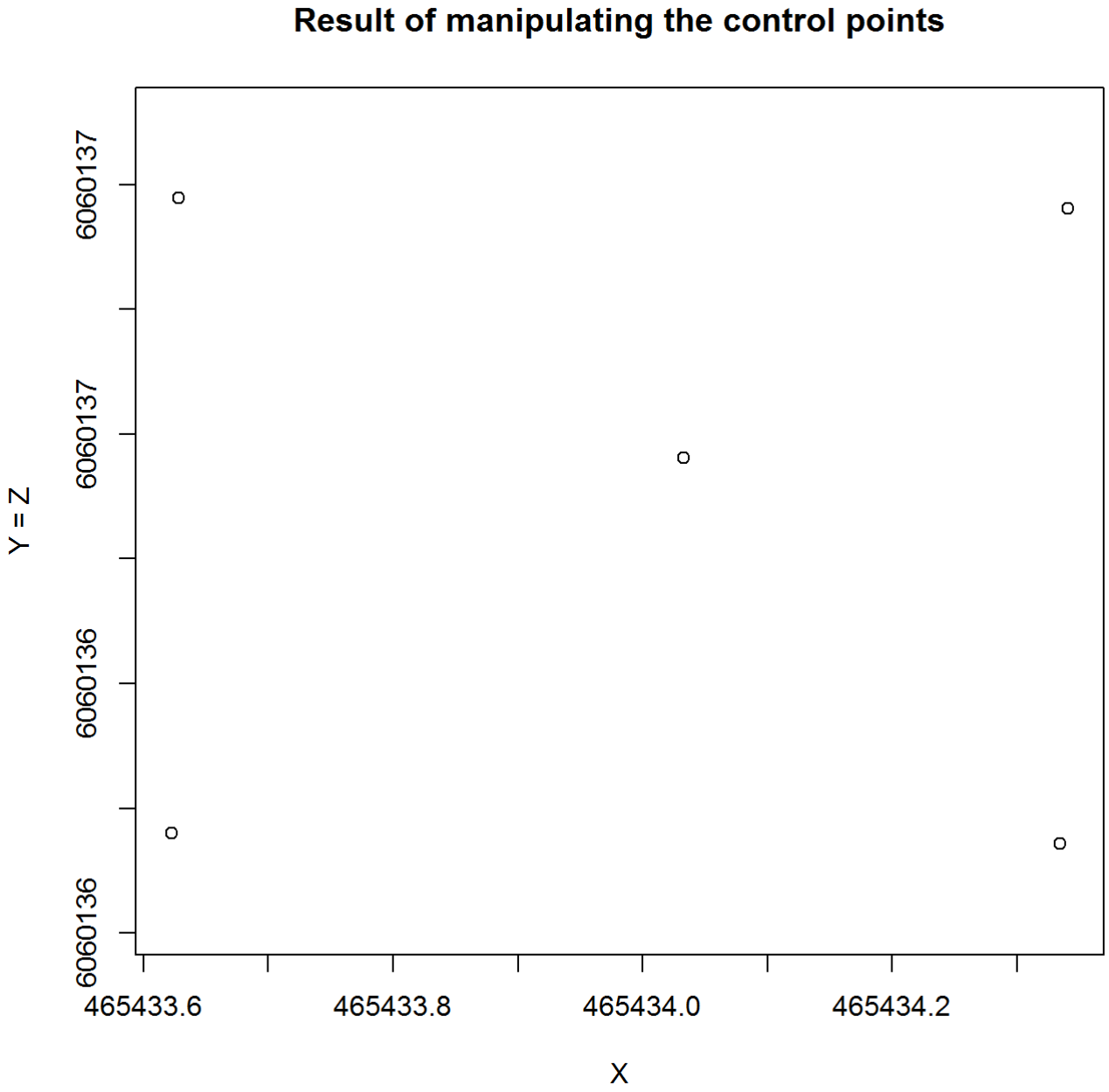
```
plot(coord_trans$x,coord_trans$z,asp=1, main = "Result of manipulating the control points", x
lab = "X", ylab =  "Y = Z")
```



Option 2: direction = original

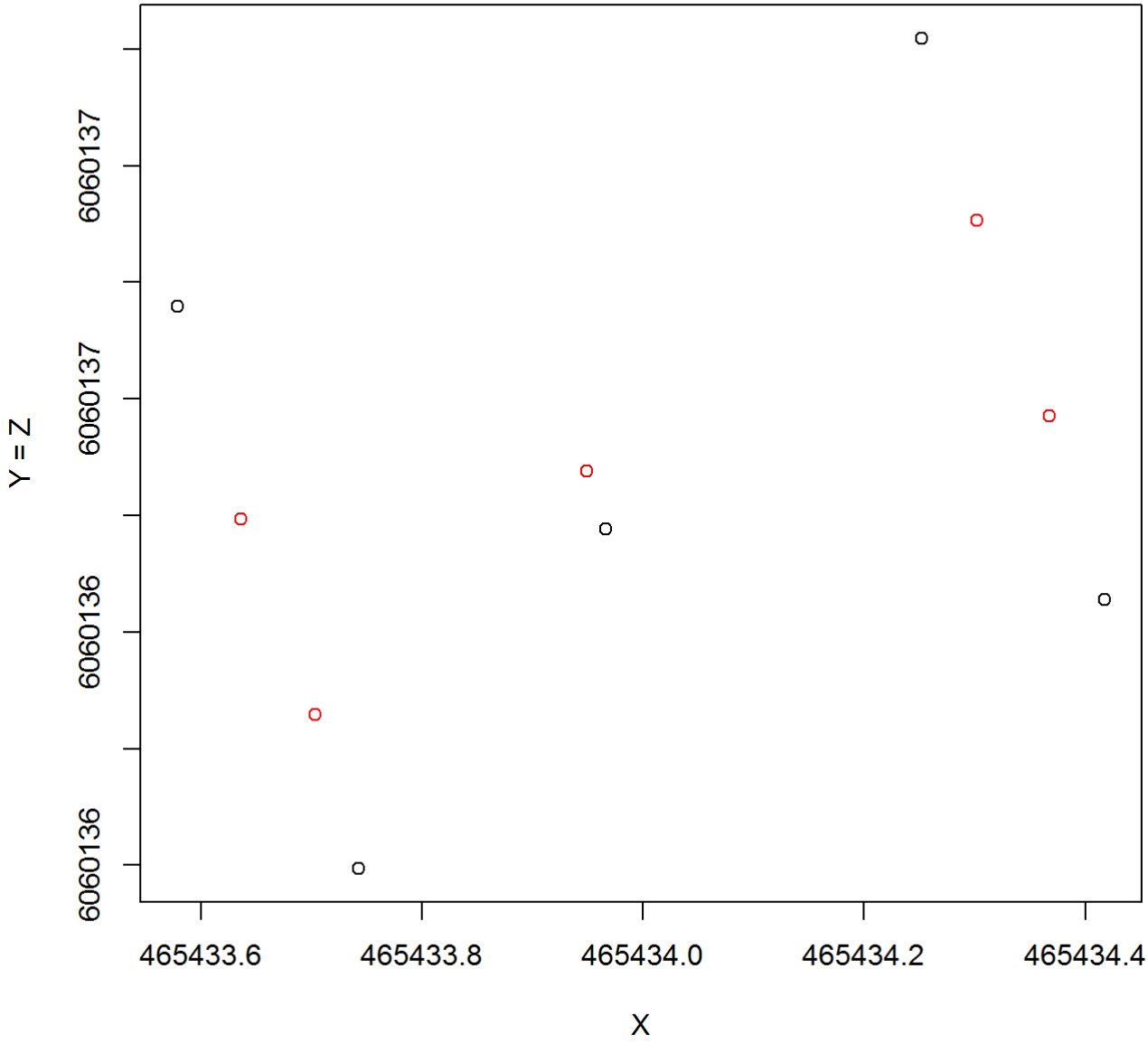
In the cases shown before, the fgcs of the profile are displayed parallel to the x-axis after rotating and transforming. But in some cases it is handy to show the profile parallel to the cutting line of the profile. In this case the first rotation have to be redone. The rotation angle is the negative angle of slope_deg

```
y_xw <- c(coord_trans$x -  min(c(coord_trans$x, coord_trans$z)))
y_zw <- c(coord_trans$z -  min(c(coord_trans$x, coord_trans$z)))
y_fm <- lm(y_zw ~ y_xw)


y_slope_deg <- -slope_deg

y_center_x <- sum(coord_trans$x) / nrow(coord_trans)
y_center_z <- sum(coord_trans$z) / nrow(coord_trans)

y_coord <- coord_trans


for (z in 1:nrow(y_coord)) {
  y_coord[z,] <- c(
    y_center_x + (coord_trans$x[z] - y_center_x) * cos(y_slope_deg / 180 * pi) -
(coord_trans$z[z] - y_center_z) * sin(y_slope_deg / 180 * pi),
    coord_trans$y[z],
    y_center_z + (coord_trans$x[z] - y_center_x) * sin(y_slope_deg / 180 * pi) +
(coord_trans$z[z] - y_center_z) * cos(y_slope_deg / 180 * pi),
    as.numeric(as.character(coord_trans$pr[z]))
  )

}

plot(coord_trans$x,coord_trans$z,asp=1, main = "Result of manipulating the control points", x
lab = "X", ylab =  "Y = Z")
```




```
plot(y_coord$x, y_coord$z, main = "Result of manipulating the control points including option
 2 (black) and the original fgcs (red)", xlab = "X", ylab =  "Y = Z")
points(fotogram_sdf@data$x, fotogram_sdf@data$y, col = "red")
```

of manipulating the control points including option 2 (black) and the origin




```
coord_trans <- y_coord
```

Export

The points will be returned as a SpatialDataframe or a Dataframe based on the input data