Correction des coordonnées

Eric Marcon

Vincyane Badouard

6 décembre 2024

Résumé

Repositionnement des arbres du projet Alt.

1 Problème à traiter

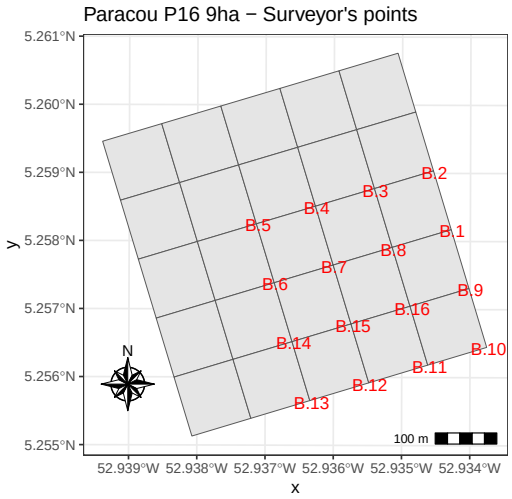
Les petits arbres de 9 hectares de la parcelle 16 de Paracou sont localisés sur le terrain dans des quadrats de 10m sur 10m, eux-mêmes regroupés dans des sous-parcelles d'un hectare dont la position des sommets (nommés B.1 à B.13) est connue très précisément grâce à un relevé de géomètre.

```
library("tidyverse")

# Paracou 16 shapefile
library("terra")
library("sf")
vect("data/Plot16.shp") %>%
  st_as_sf() -> paracou_16

# Surveyor's points
library("readxl")
read_xlsx("data/plots.xlsx", sheet = "surveyor") %>%
  as.data.frame() -> surveyor
surveyor %>%
  st_as_sf(coords = c("x_utm","y_utm")) %>%
  st_set_crs(crs(paracou_16)) -> surveyor.sf

# Map
library("ggspatial")
ggplot() +
  geom_sf(data = st_cast(paracou_16)) +
  geom_sf_text(data = surveyor.sf, aes(label = point), col = "red") +
  ggtitle("Paracou P16 9ha - Surveyor's points") +
  annotation_scale(location = "br") +
  annotation_north_arrow(
    pad_y = unit(1, "cm"),
    style = north_arrow_nautical()
  )
```



Les sous-parcelles (« subplots ») sont définies par leurs quatre coins (haut-gauche, haut-droit, bas-droit, et bas-gauche qui est l'origine du repère local) et numérotées.

```
read_xlsx("data/plots.xlsx", sheet = "subplots") %>%
  as.data.frame() -> subplots
# Example
subplots[1:3, ]
```

```
##   subplot up_left up_right down_right down_left
## 1      13     B.5      B.4        B.7       B.6
## 2      14     B.4      B.3        B.8       B.7
## 3      15     B.3      B.2        B.1       B.8
```

Les quadrats de 10m sont marqués par des piquets. Leur position est approximative (la progression et les mesures d'angle et de distance en sous-bois sont difficiles) mais les distances entre chaque piquet et ses voisins ont été remesurées précisément au laser.

Enfin, les arbres ont été positionnés dans les quadrats à partir de leur distance aux bords, avec une certaine incertitude.

L'objectif est de replacer le plus précisément possible les arbres dans les sous-parcelles pour obtenir leurs coordonnées dans le référentiel standard local, UTM zone 26N.

2 Méthode

Dans un premier temps, les quadrats seront repositionnés à l'intérieur des sous-parcelles, les arbres seront ensuite repositionnés dans chaque quadrat. Pour cela :

1. Les coordonnées des sous-parcelles doivent être transformées en coordonnées locales, dont l'origine est le coin inférieur gauche de chacune d'elles.

2. Les quadrats doivent être repositionnés dans chaque sous-parcelle, sur la base des hypothèses suivantes :
   - les limites basse et gauche des sous-parcelles parcourues sur le terrain pour y placer les quadrats sont rectilignes,
   - elles ne sont pas forcément orthogonales : l'angle précis de la base du repère est donné par les mesures du géomètre,
   - les distances mesurées au laser sont exactes, ce qui permet de calculer de proche en proche la position des points constituant les quadrats par triangulation, à partir de l'origine du repère de la sous-parcelle,
3. les points obtenus sont replacés dans les limites précises de chaque sous-parcelle par interpolation,
4. enfin, les arbres sont repositionnés dans les limites précises de chaque quadrat par interpolation.

3 Coordonnées locales des sous-parcelles

Les sous-parcelles doivent être projetées dans leur système de coordonnées locales : le point en bas à gauche est l'origine du repère.

Le passage des coordonnées locales aux coordonnées UTM est un changement de base (selon les coordonnées des vecteurs unitaires en UTM) suivie d'une translation (selon la position du point d'origine de la sous-parcelle).

3.1 Rotation

La matrice de changement de base (des coordonnées locales aux coordonnées UTM) est calculée par la fonction suivante :

```
local2utm <- function(subplot, surveyor, subplots) {
  # Find the points
  is_origin <- (surveyor$point == subplots[subplots$subplot == subplot, "down_left"])
  is_down_right <- (surveyor$point == subplots[subplots$subplot == subplot, "down_right"])
  is_up_left <- (surveyor$point == subplots[subplots$subplot == subplot, "up_left"])

  # X-axis vector
  i_dx_utm <- surveyor[is_down_right, "x_utm"] - surveyor[is_origin, "x_utm"]
  i_dy_utm <- surveyor[is_down_right, "y_utm"] - surveyor[is_origin, "y_utm"]
  i_length <- sqrt(i_dx_utm^2 + i_dy_utm^2)
  # Unit vector
  i <- c(i_dx_utm, i_dy_utm) / i_length

  # Y-axis vector
  j_dx_utm <- surveyor[is_up_left, "x_utm"] - surveyor[is_origin, "x_utm"]
  j_dy_utm <- surveyor[is_up_left, "y_utm"] - surveyor[is_origin, "y_utm"]
  j_length <- sqrt(j_dx_utm^2 + j_dy_utm^2)
  # Unit vector
  j <- c(j_dx_utm, j_dy_utm) / j_length

  return(cbind(i, j))
}

# Test the function: transition matrix of subplot 13
local2utm(13, surveyor, subplots)
```

```
##             i         j
```

```
## [1,] 0.9581356 -0.2792892
## [2,] 0.2863147  0.9602070
```

Le passage des coordonnées UTM aux coordonnées locales utilise la matrice inverse :

```
# local coordinates of subplot 13, expected to be (100, 0)
local2utm(13, surveyor, subplots) %>%
  # UTM to local
  solve() %*%
  # UTM vector X
  c(
    surveyor[surveyor$point == "B.7", "x_utm"] -
      surveyor[surveyor$point == "B.6", "x_utm"],
    surveyor[surveyor$point == "B.7", "y_utm"] -
      surveyor[surveyor$point == "B.6", "y_utm"]
  )
```

```
##           [,1]
## i 1.000892e+02
## j 3.136823e-15
```

Coordonnées locales

Les coordonnées locales des sommets des sous-parcelles sont calculées :

```
# Get the UTM coordinates of the points
subplots %>%
  # add coordinates of point up_left
  left_join(surveyor, by = join_by("up_left" == "point")) %>%
  # delete the altitude
  select(-z_utm) %>%
  # rename the columns accoding to the chosen point
  rename(up_left_x_utm = x_utm, up_left_y_utm = y_utm) %>%
  # repeat all three steps for up_right
  left_join(surveyor, by = join_by("up_right" == "point")) %>%
  select(-z_utm) %>%
  rename(up_right_x_utm = x_utm, up_right_y_utm = y_utm) %>%
  # repeat all three steps for down_right
  left_join(surveyor, by = join_by("down_right" == "point")) %>%
  select(-z_utm) %>%
  rename(down_right_x_utm = x_utm, down_right_y_utm = y_utm) %>%
  # repeat all three steps for down_left
  left_join(surveyor, by = join_by("down_left" == "point")) %>%
  select(-z_utm) %>%
  rename(down_left_x_utm = x_utm, down_left_y_utm = y_utm) ->
  subplots

# Prepare the columns
subplots %>%
  mutate(
    # Local coordinates before interpolation
    up_left_x_field = NA, up_left_y_field = NA,
    up_right_x_field = NA, up_right_y_field = NA,
    down_right_x_field = NA, down_right_y_field = NA,
    down_left_x_field = NA, down_left_y_field = NA
  ) -> subplots

# Get the coordinates
for (i in seq_len(nrow(subplots))) {
  subplot <- subplots$subplot[i]
  # Transition matrix
  utm2local <- solve(local2utm(subplot, surveyor, subplots))
  # Relative UTM coordinates
  # substract the coordinates of the origin to that of all points
  # to get relative coordinates
```

```
  # do.call transforms the obtained list into a vector
  do.call('c', subplots[i, 6:13] - rep(subplots[i, 12:13], 4)) %>%
    # Make a matrix, columns are realtive X and Y
    matrix(nrow = 4, ncol = 2, byrow = TRUE) -> utm_relative
  # Multiply by the transition matrix
  utm2local %*% t(utm_relative) %>%
    # Make a vector and save it into subplots
    as.vector() ->
    subplots[i, 14:21]
}
```

L'orthogonalité entre abscisse et ordonnée des sous-parcelles est vérifiée par la nullité du produit scalaire des vecteurs constitués par les bordures bas et gauche des sous-parcelles.

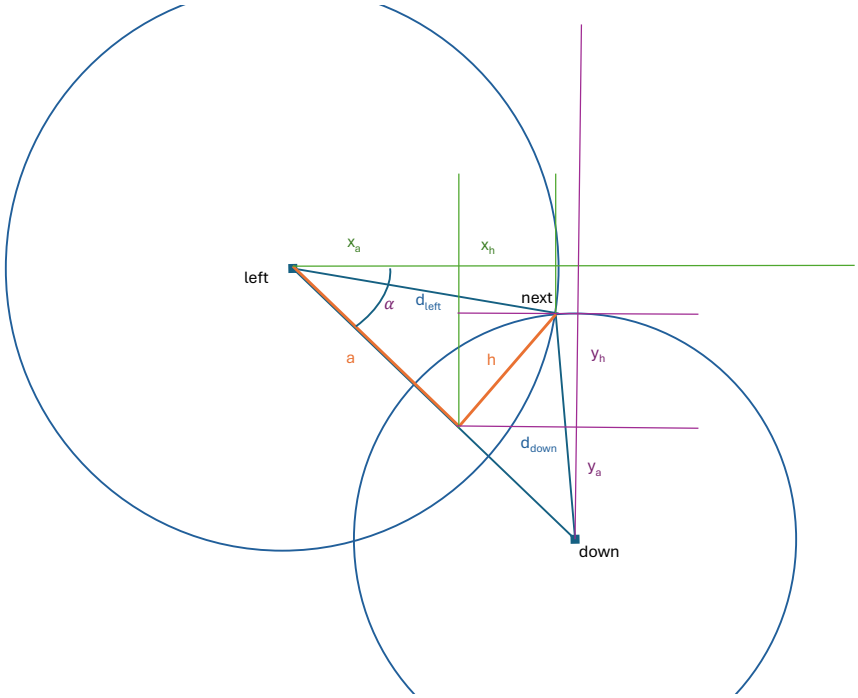
```
subplots %>%
  mutate(scal_prod = up_left_x_field * down_left_x_field + up_left_y_field * down_right_y_field) %>%
  select(subplot, scal_prod)
```

```
##   subplot     scal_prod
## 1      13  3.144020e-13
## 2      14 -3.414544e-13
## 3      15  4.726148e-13
## 4      18  8.368951e-15
## 5      19  3.707301e-13
## 6      20  3.469311e-13
## 7      23  5.171196e-14
## 8      24 -5.759519e-13
## 9      25 -1.905517e-13
```

4 Position des quadrats

4.1 Fonction de triangulation

On connaît la position des points à gauche (x_{left}, y_{left}) et en dessous (x_{down}, y_{down}) du point à placer (à l'intersection des deux cercles), ainsi que les distances entre eux et le point à placer : d_{left} et d_{down}. Par construction de l'algorithme, le point à placer est situé au dessus et à droite des points précédents.



Par le théorème de Pythagore, on connaît :

- la distance entre les points de gauche et du bas :

$$d = \sqrt{(x_{left} - x_{down})^2 + (y_{left} - y_{down})^2},$$

- et deux équations reliant a et h

$$a^2 + h^2 = d_{left}^2, \tag{1}$$
$$(d-a)^2 + h^2 = d_{down}^2. \tag{2}$$

La première équation du système permet d'isoler h :

$$h^2 = d_{left}^2 - a^2.$$

En substituant h^2 dans la deuxième équation

$$(d-a)^2 + d_{left}^2 - a^2 = d_{down}^2,$$

d'où

$$a = \frac{d^2 + d_{left}^2 - d_{down}^2}{2d}$$

et

$$h = \sqrt{d_{left}^2 - a^2}.$$

Les distances entre les points précédents et le point suivant sont calculées en projetant a et h sur les axes du repère. Pour cela, l'angle α est calculé :

$$\alpha = \arctan \frac{x_{down} - x_{left}}{y_{left} - y_{down}}.$$

Il reste à projeter :

$$\begin{aligned} d_{left} &= x_a + x_h \\ &= a \sin \alpha + h \cos \alpha \end{aligned} \tag{3}$$

et

$$\begin{aligned} d_{down} &= y_a + y_h \\ &= (d - a) \sin (\frac{\pi}{2} - \alpha) + h \sin \alpha. \end{aligned} \tag{4}$$

La fonction `next_point()` calcule les coordonnées du point suivant :

```
# Triangulation
next_point <- function(
    x_left,
    y_left,
    x_down,
    y_down,
    d_left,
    d_down) {
  # distance left-down
  d_squared <- (x_left - x_down)^2 + (y_left - y_down)^2
  d <- sqrt(d_squared)
  # distance left-height
  a <- (d_squared + d_left^2 - d_down^2) / 2 / d
  # height
  h <- sqrt(d_left^2 - a^2)
  # angle
  alpha <- atan((x_down - x_left) / (y_left - y_down))
  # next point
  d_left_a <- a * sin(alpha)
  d_left_h <- h * cos(alpha)
  d_down_a <- (d - a) * sin(pi / 2 - alpha)
  d_down_h <- h * sin(alpha)
  return(c(d_left_a + d_left_h, d_down_a + d_left_h))
}
```

Test de la fonction :

```
# Test the function
x_left <- 0
y_left <- 10
x_down <- 10
y_down <- 0
d_left <- 11
d_down <- 10
next_point(x_left, y_left, x_down, y_down, d_left, d_down)
```

```
## [1] 10.999886  9.949886
```

4.2 Placement des quadrats

Le tableau des mesures contient trois colonnes pour décrire la position des angles des quadrats : `plot` pour l'hectare, `point_x` et `point_y` pour le numéro du point, de (0, 0) pour le coin inférieur gauche à (10, 10) pour le coin supérieur droit.

Pour chaque point, la distance à son voisin de gauche (`y` identique) et du bas (`x` identique), mesurées sur le terrain, sont dans les colonnes `d_left` et `d_down`.

Le code suivant :

- lit le tableau des mesures et prépare deux colonnes supplémentaires, `x` et `y`, pour y placer les coordonnées à calculer,

```
# data
read_xlsx("data/plots.xlsx", sheet = "quadrats") %>%
  # Add columns for the correct coordinates
  mutate(x = 0, y = 0) -> quadrats
```

- ajoute les coins inférieurs gauche (0, 0),

```
# Add points (0,0) to plots
for (subplot_number in unique(quadrats$subplot)) {
  quadrats %>%
    add_row(
      subplot = subplot_number,
      point_x = 0,
      point_y = 0,
      x = 0,
      y = 0
    ) -> quadrats
}
```

- trie les données et transforme le tibble en dataframe pour que les extractions futures, comme `quadrats[i, "x"]`, soient des scalaires et non des tibbles,

```
# Sort so that the next point can rely on previous ones
quadrats %>%
  arrange(subplot, point_x, point_y) %>%
  as.data.frame() -> quadrats
```

- calcule les coordonnées des points situés sur les bords gauche et bas des hectares par simple cumul des distances mesurées depuis l'origine,

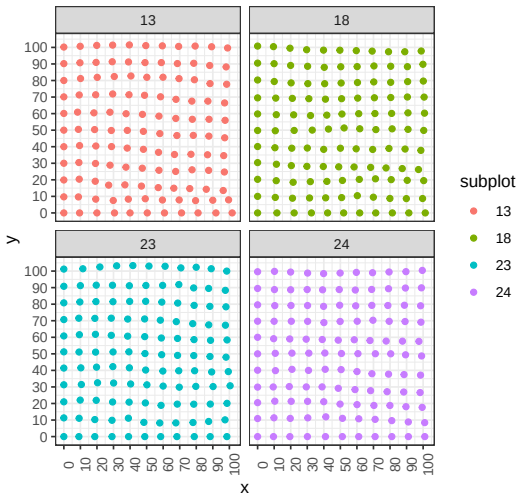
```
# Deal with edges: cumulative sum of distances from the origin
for (plot_number in unique(quadrats$subplot)) {
  is_left_edge <-
    quadrats$subplot == plot_number &
    quadrats$point_x == 0 &
    quadrats$point_y != 0
  quadrats[is_left_edge, "y"] <- cumsum(
    quadrats[is_left_edge, "d_down"]
  )
  is_down_edge <-
    quadrats$subplot == plot_number &
    quadrats$point_x != 0 &
    quadrats$point_y == 0
  quadrats[is_down_edge, "x"] <- cumsum(
    quadrats[is_down_edge, "d_left"]
  )
}
```

- calcule les coordonnées de tous les autres points par triangulation.

```
# Compute the positions of the points
for (i in seq_len(nrow(quadrats))) {
  # Ignore the edges
  if (quadrats[i, "point_x"] != 0 & quadrats[i, "point_y"] != 0) {
    x_left = quadrats[
      quadrats$subplot == quadrats[i, "subplot"] &
        quadrats$point_x == max(quadrats[i, "point_x"] - 1, 0) &
        quadrats$point_y == quadrats[i, "point_y"],
      "x"
    ]
    y_left = quadrats[
      quadrats$subplot == quadrats[i, "subplot"] &
        quadrats$point_x == max(quadrats[i, "point_x"] - 1, 0) &
        quadrats$point_y == quadrats[i, "point_y"],
      "y"
    ]
    x_down = quadrats[
      quadrats$subplot == quadrats[i, "subplot"] &
        quadrats$point_x == quadrats[i, "point_x"] &
        quadrats$point_y == max(quadrats[i, "point_y"] - 1, 0),
      "x"
    ]
    y_down = quadrats[
      quadrats$subplot == quadrats[i, "subplot"] &
        quadrats$point_x == quadrats[i, "point_x"] &
        quadrats$point_y == max(quadrats[i, "point_y"] - 1, 0),
      "y"
    ]
    quadrats[i, c("x", "y")] <-
      c(x_left, y_down) +
      next_point(
        x_left = x_left,
        y_left = y_left,
        x_down = x_down,
        y_down = y_down,
        d_left = quadrats[i, "d_left"],
        d_down = quadrats[i, "d_down"]
      )
  }
}
```

Le dataframe `quadrats` contient maintenant la position des quadrats. Figure :

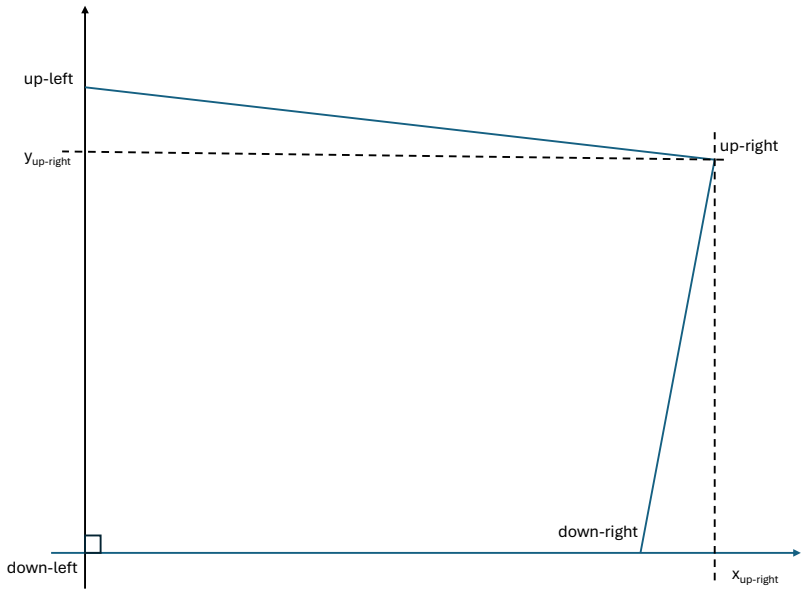
```
quadrats %>%
  ggplot(aes(x = x, y = y, color = as.factor(subplot))) +
  geom_point() +
  scale_color_discrete() +
  scale_x_continuous(breaks = (0:10) * 10) +
  scale_y_continuous(breaks = (0:10) * 10) +
  coord_fixed() +
  theme(axis.text.x = element_text(angle = 90)) +
  labs(color = "subplot") +
  facet_wrap(~ subplot, nrow = 2)
```



4.3 Interpolation

Les piquets des quadrats doivent être repositionnés pour que la forme de chaque sous-parcelle issue du terrain corresponde à sa forme réelle, issue des mesures du géomètre, qui n'est pas exactement un carré. Cette opération ne sert qu'à réconcilier les limites de la sous-parcelle sur le terrain (considérée comme un carré) et les données du géomètre.

Les coordonnées des points doivent être interpolées pour que la valeur maximale de l'abscisse, pour une ordonnée donnée, corresponde aux mesures du géomètre. Le raisonnement est le même pour les ordonnées.



A condition que les limites du bas et de gauche de la sous-parcelle soient orthogonales, la valeur maximale de l'abscisse x, qui dépend de y, est donnée par

$$x_{max} = x_{down,right} + y \times \frac{x_{up,right} - x_{down,right}}{y_{up,right} - y_{down,right}}.$$

De même,

$$y_{max} = y_{up,left} + x \times \frac{y_{up,right} - y_{up,left}}{x_{up,right} - x_{up,left}}.$$

Les coordonnées interpolées sont

$$x = x_0 + \frac{x_{max}}{x_M}$$

et

$$y = y_0 + \frac{y_{max}}{y_M}$$

où (x_0, y_0) sont les coordonnées du point avant interpolation et x_M et y_M les valeurs maximales assumées pendant les mesures de terrain, ici 100 mètres.

Application aux quadrats :

```r
# Theoretical subplot size
x_M <- 100
y_M <- 100
quadrats %>%
  # Rename raw x and y columns
  rename(x_0 = x, y_0 = y) %>%
  # Join the subplot coordinates
  left_join(subplots) %>%
  mutate(
    x = x_0 * (
      down_right_x_field +
        y_0 * (up_right_x_field - down_right_x_field) /
        (up_right_y_field - down_right_y_field)
      ) / x_M,
    y = y_0 * (
      up_left_y_field +
        y_0 * (up_right_y_field - up_left_y_field) /
        (up_right_x_field - up_left_x_field)
      ) / y_M,
  ) -> quadrats
```

Les coordonnées UTM des quadrats sont recalculées :

```r
# Prepare the columns
quadrats$x_utm <- quadrats$y_utm <- NA
# Project quadrats to UTM
for (subplot in unique(quadrats$subplot)) {
  # Points of the subplot. Rows are x and y
  xy_local <- t(as.matrix(quadrats[quadrats$subplot == subplot, c("x", "y")]))
  # Apply the rotation matrix and add the coordinates of the origin of the plot
  xy_utm <- t(local2utm(subplot, surveyor, subplots) %*% xy_local) +
    quadrats[quadrats$subplot == subplot, c("down_left_x_utm", "down_left_y_utm")]
  # Save the coordinates
  quadrats[quadrats$subplot == subplot, c("x_utm", "y_utm")] <- xy_utm
}

# Map
ggplot(quadrats, aes(x = x_utm, y = y_utm, color = as.factor(subplot))) +
  geom_point() +
  coord_fixed() +
  labs(color = "Subplot", x = "UTM x", y = "UTM y")
```