

# Correction des coordonnées

Eric Marcon

Vincyane Badouard

30 avril 2025

## Résumé

Repositionnement des arbres du projet Alt.

## 1 Problème à traiter

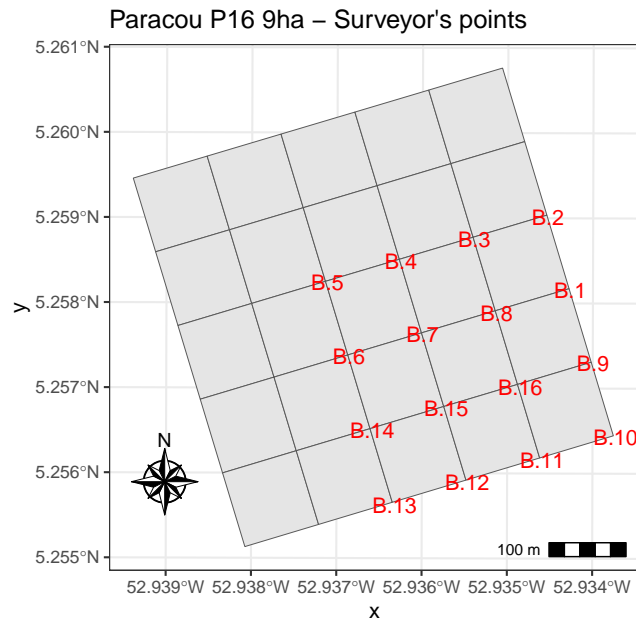
Les petits arbres de 9 hectares de la parcelle 16 de Paracou sont localisés sur le terrain dans des quadrats de 10m sur 10m, eux-mêmes regroupés dans des sous-parcelles d'un hectare dont la position des sommets (nommés B.1 à B.13) est connue très précisément grâce à un relevé de géomètre.

```
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")
paracou_16 %>%
  ggplot() +
    geom_sf() +
    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, 1:4]
```

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

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 précises, 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. les écarts éventuels entre les positions des points en bordure droite et haute de chaque sous-parcelle et gauche et basse de la sous-parcelle voisine sont doivent être corrigés parce que les piquets sont uniques. Les bordures gauche et basse sont considérées comme fiables.
5. enfin, les arbres, dont les coordonnées ont été mesurées dans chaque quadrat, sont projetés dans le quadrat corrigé par la matrice de transition qui projette les limites basses et gauche du quadrat à partir du repère local.

### 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 Changement de repère

La matrice de changement de base (des coordonnées locales aux coordonnées UTM) est constituée des coordonnées des vecteurs unitaires de la nouvelle base dans le repère de l'ancienne.

la fonction `local2utm()` retourne cette matrice pour la sous-parcelle choisie.

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

```

## 3.2 Coordonnées locales

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

```

# Next column number, i.e. UTM coordinates
col_utm_first <- ncol(subplots) + 1
col_utm_last <- col_utm_first + 7
# 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 according 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
  # subtract 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, col_utm_first:col_utm_last] -
      rep(subplots[i, (col_utm_last - 1):col_utm_last], 4)
  ) %>%
  # Make a matrix, columns are relative 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, (col_utm_last + 1):(col_utm_last + 8)]
}

```

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(
  scalar_product = up_left_x_field * down_left_x_field +
    up_left_y_field * down_right_y_field
) %>%
select(subplot, scalar_product)

```

```

##  subplot scalar_product
## 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

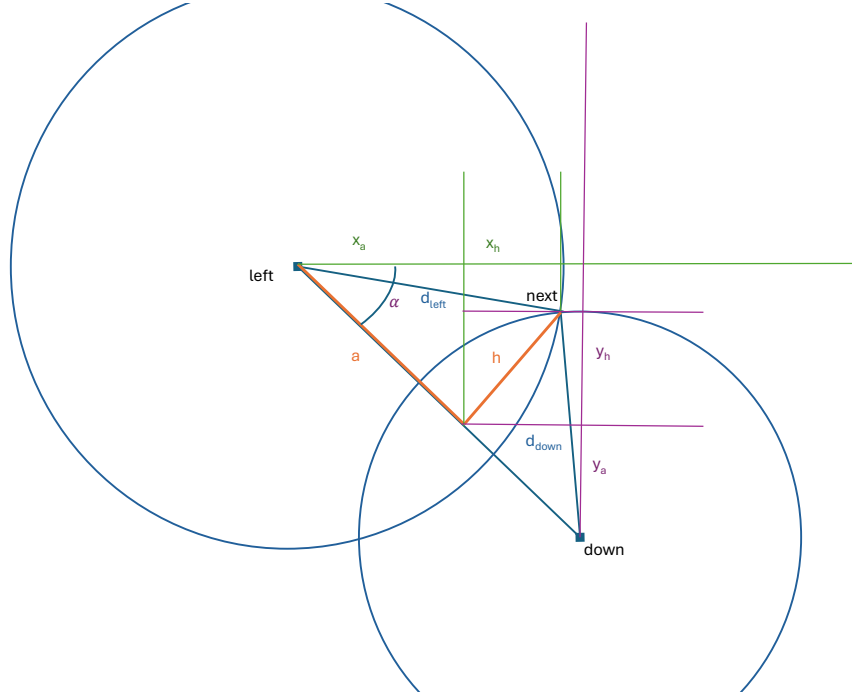
```

Le dataframe `subplots` contient maintenant les coordonnées des sous-parcelles en UTM (`up_left_x_utm` et 7 autres) et dans le repère de terrain (`up_left_x_field` et 7 autres), qui sont proches de 0 ou 100.

## 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 (« next », 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  $h$

$$a^2 + h^2 = d_{left}^2, \quad (1)$$

$$(d - a)^2 + h^2 = d_{down}^2. \quad (2)$$

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  $\alpha$  est calculé :

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

Il reste à projeter :

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

et

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

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_down_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, `quadrat_x` et `quadrat_y` pour le numéro du point, de (0, 0) pour le coin inférieur gauche à (10, 10) pour le coin supérieur droit.

Les points situés sur les bordures basse et gauche de chaque sous-parcelle sont les plus fiables. Ils seront utilisés pour corriger les bordures haute et droite des sous-parcelles voisines, sont la position calculée accumulera les erreurs. Ces données peuvent être mesurées sur le terrain indépendamment des arbres : dans ce cas, seules les distances entre les points des bordures sont dans le tableau de données. Les sous-parcelles correspondantes sont sans intérêt pour elles-mêmes (« dummy subplots ») : elles ne contiennent pas tous les points délimitant les quadrats et aucun arbre mesuré.

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

```
# data
read_xlsx("data/plots.xlsx", sheet = "quadrats") %>%
  # Homogenize the column names
  rename(
    subplot = "Subplot",
    quadrat_x = "quadra.nb.X",
    quadrat_y = "quadra.nb.Y"
  ) %>%
  # Add columns for the correct coordinates
  mutate(x = 0, y = 0) %>%
  as.data.frame() -> quadrats
# Edge point numbers
right_edge <- up_edge <- 10
# Quadrat size
x_M_field <- y_M_field <- 10
# List subplots measured in the field, i.e. with full data
quadrats %>%
  group_by(subplot) %>%
  summarise(points_n = n()) %>%
  # 10 x 10 points to define quadrats
  filter(points_n == right_edge * up_edge) %>%
  pull(subplot) -> subplots_field
# List dummy subplots,
```



```
# used to correct right or up edges of subplots measured in the field
subplots_dummy <- setdiff(unique(quadrats$subplot), subplots_field)
```

- ajoute les piquets des bordures droites et supérieures, s'ils sont absents du tableau Excel,

```
# Prepare a new table to add rows
quadrats %>%
# Delete all rows
filter(subplot == 0) ->
quadrats_edges
# Add points (10, y) and (x, 10) to subplots
for (subplot_number in unique(quadrats$subplot)) {
  is_in_subplot <- (quadrats$subplot == subplot_number)
  for (x in unique(quadrats[is_in_subplot, "quadrat_x"])) {
    # Add the up-edge piquets
    quadrats_edges %>%
      add_row(
        subplot = subplot_number,
        quadrat_x = x,
        quadrat_y = up_edge,
        # Distance to the next piquet is in the previous piquet data
        Q_lengthX = quadrats[
          is_in_subplot &
            (quadrats$quadrat_x == x) &
            (quadrats$quadrat_y == up_edge - 1),
          "Q_lengthX1"
        ],
        x = 0,
        y = 0
      ) -> quadrats_edges
  }
  for (y in unique(quadrats[is_in_subplot, "quadrat_y"])) {
    # Add the right-edge piquet
    quadrats_edges %>%
      add_row(
        subplot = subplot_number,
        quadrat_x = right_edge,
        quadrat_y = y,
        # Distance to the next piquet is in the previous piquet data
        Q_lengthY = quadrats[
          is_in_subplot &
            (quadrats$quadrat_x == right_edge - 1) &
            (quadrats$quadrat_y == y),
          "Q_lengthY1"
        ],
        x = 0,
        y = 0
      ) -> quadrats_edges
  }
  quadrats_edges %>%
    # Add the top right corner
    add_row(
      subplot = subplot_number,
      quadrat_x = right_edge,
      quadrat_y = up_edge,
      x = 0,
      y = 0
    ) -> quadrats_edges
}
# Add the new rows to quadrats and sort for readability
quadrats %>%
  bind_rows(quadrats_edges) %>%
  arrange(subplot, quadrat_x, quadrat_y) -> quadrats
```

- calcule les coordonnées de tous les points par triangulation.

```

# Compute the positions of the points
for (i in seq_len(nrow(quadrats))) {
  # Compute x. Ignore edges
  if (quadrats[i, "quadrat_x"] > 0) {
    # Previous piquet on the left
    is_piquet_left <-
      # Same subplot
      (quadrats$subplot == quadrats$subplot[i]) &
      # Left column
      (quadrats$quadrat_x == quadrats$quadrat_x[i] - 1) &
      # Same row
      (quadrats$quadrat_y == quadrats$quadrat_y[i])
    # Cumulate
    quadrats[i, "x"] <-
      quadrats[is_piquet_left, "x"] + quadrats[is_piquet_left, "Q_lengthX"]
  }
  # Compute y. Ignore edges
  if (quadrats[i, "quadrat_y"] > 0) {
    # Previous piquet down
    is_piquet_below <-
      # Same subplot
      (quadrats$subplot == quadrats$subplot[i]) &
      # Same column
      (quadrats$quadrat_x == quadrats$quadrat_x[i]) &
      # Row below
      (quadrats$quadrat_y == quadrats$quadrat_y[i] - 1)
    # Cumulate
    quadrats[i, "y"] <-
      quadrats[is_piquet_below, "y"] + quadrats[is_piquet_below, "Q_lengthY"]
  }
}

```

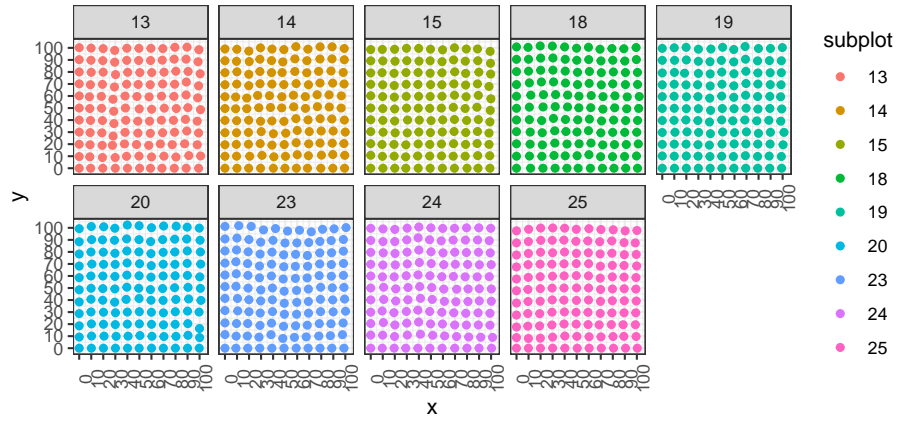
Le dataframe `quadrats` contient maintenant la position des quadrats, en coordonnées locales (x et y, comprises entre 0 et 100) et UTM (`x_utm` et `y_utm`).

Carte des quadrats, en coordonnées locales :

```

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)

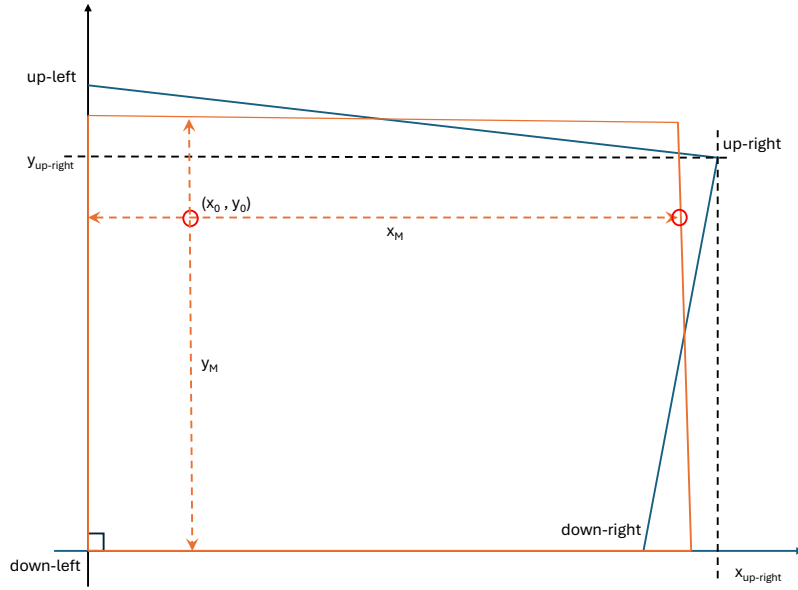
```



### 4.3 Interpolation

Les piquets des quadrats doivent être repositionnés pour que la forme de chaque sous-parcelle issue du terrain (en rouge sur la figure suivante) corresponde à sa forme réelle (en bleu), 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

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

et

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

où  $(x_0, y_0)$  sont les coordonnées du point avant interpolation et  $x_M$  et  $y_M$  les valeurs maximales des mesures de terrain, théoriquement 100 mètres, mais connues plus précisément. Pour chaque piquet,  $x_M$  est l'abscisse du piquet le plus à droite de la rangée de même ordonnée, et de même pour  $y_M$ .

Application aux quadrats :

```

quadrats %>%
  # Rename raw x and y columns
  rename(x_0 = x, y_0 = y) %>%
  # Join the subplot coordinates
  left_join(subplots) %>%
  # Prepare columns
  mutate(x_M = NA, y_M = NA) -> quadrats

# Loop to calculate x_M and y_M (can't be vectorized)
for (i in seq_len(nrow(quadrats))) {
  is_in_subplot <- quadrats$subplot == quadrats[i, "subplot"]
  is_same_x <- quadrats$quadrat_x == quadrats[i, "quadrat_x"]
  is_same_y <- quadrats$quadrat_y == quadrats[i, "quadrat_y"]
  quadrats[i, "x_M"] <- max(quadrats$x_0[is_in_subplot & is_same_y])
  quadrats[i, "y_M"] <- max(quadrats$y_0[is_in_subplot & is_same_x])
}

# Add dummy x_M and y_M for dummy subplots
is_dummy <- quadrats$subplot %in% subplots_dummy
is_x_M_missing <- quadrats$x_M == 0
is_y_M_missing <- quadrats$y_M == 0
quadrats[is_dummy & is_x_M_missing, "x_M"] <- 100
quadrats[is_dummy & is_y_M_missing, "y_M"] <- 100

# Interpolate
quadrats %>%
  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 +
      x_0 * (up_right_y_field - up_left_y_field) /
      (up_right_x_field - up_left_x_field)
    ) / y_M,
  ) -> quadrats

```

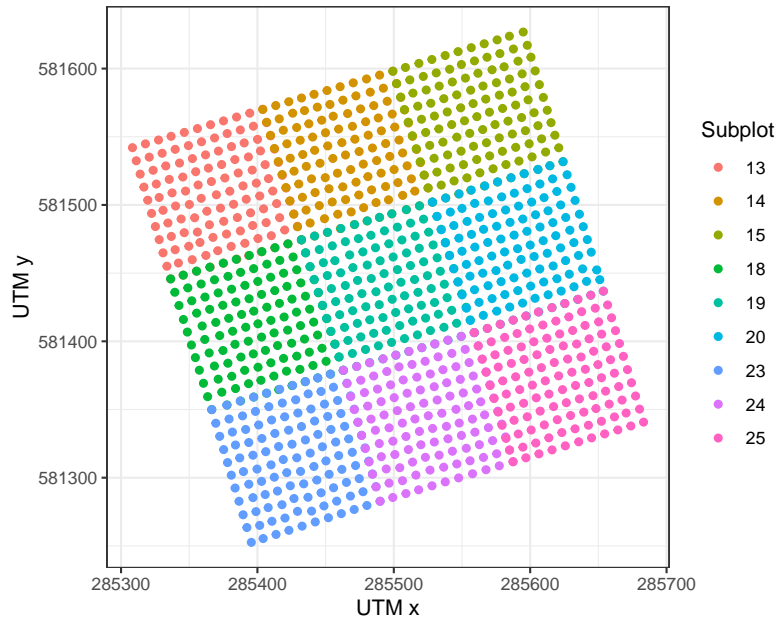
Les coordonnées UTM des quadrats sont recalculées :

```

# Prepare the columns
quadrats$x_utm <- quadrats$y_utm <- NA
# Project quadrats to UTM
for (subplot in unique(quadrats$subplot)) {
  # Points of the subplot. Rows are x and y
  is_in_subplot <- quadrats$subplot == subplot
  xy_local <- as.matrix(quadrats[is_in_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)) %*% t(xy_local) +
    quadrats[is_in_subplot, c("down_left_x_utm", "down_left_y_utm")]
  # Save the coordinates
  quadrats[is_in_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")

```



#### 4.4 Réconciliation des sous-parcelles

Quand la position des piquets de la sous-parcelle à droite ou en haut de chaque sous-parcelle est connue, elle est plus fiable que celle des piquets en bordures droite et haute. La position des piquets est réconciliée par interpolation :

- les piquets de la bordure droite sont déplacés à la position des piquets de la sous-parcelle suivante à droite, si l'information est disponible (ex. : les piquets de la sous-parcelle 23 sont remplacés à la position de ceux de la sous-parcelle 24).
- les piquets de la bordure gauche ne sont pas modifiés,
- les piquets intermédiaires subissent une fraction du déplacement du piquet de droite : 1/10 pour le deuxième piquet, ..., 9/10 pour le dixième.

Les mêmes ajustements sont appliqués aux piquets en limite haute.

Ces corrections ont lieu en coordonnées UTM.

```
# Which subplots can be corrected
subplots %>%
  filter(!is.na(next_right)) %>%
  pull(subplot) -> to_correct_right
subplots %>%
  filter(!is.na(next_up)) %>%
  pull(subplot) -> to_correct_up

# Right edge correction
quadrats %>%
  filter(
    subplot %in% to_correct_right &
    quadrat_x == right_edge
  ) %>%
  select(subplot, quadrat_y, x_utm, y_utm) %>%
```

```

# Get the next plot to the right
left_join(
  subplots %>%
    select(subplot, next_right)
) %>%
# Get the coordinates of its left edge
left_join(
  quadrats %>%
    rename(x_ref_utm = x_utm, y_ref_utm = y_utm) %>%
    filter(quadrat_x == 0) %>%
    select(subplot, quadrat_y, x_ref_utm, y_ref_utm) %>%
    rename(next_right = subplot)
) %>%
# Correction to apply
mutate(dx_utm = x_ref_utm - x_utm, dy_utm = y_ref_utm - y_utm) %>%
select(subplot, quadrat_y, dx_utm, dy_utm) ->
reconcile_right
# Apply the right-edge correction
quadrats %>%
# Get the correction
left_join(
  quadrats %>%
    inner_join(reconcile_right) %>%
    # Corrected coordinates
    mutate(
      x_utm_right = x_utm + dx_utm * quadrat_x / right_edge,
      y_utm_right = y_utm + dy_utm * quadrat_x / right_edge,
    ) %>%
    select(subplot, quadrat_x, quadrat_y, x_utm_right, y_utm_right)
) %>%
# Save the coordinates
mutate(
  x_utm = ifelse(is.na(x_utm_right), x_utm, x_utm_right),
  y_utm = ifelse(is.na(y_utm_right), y_utm, y_utm_right),
) ->
quadrats

# Top edge correction
quadrats %>%
filter(
  subplot %in% to_correct_up &
  quadrat_y == up_edge
) %>%
select(subplot, quadrat_x, x_utm, y_utm) %>%
# Get the next plot up
left_join(
  subplots %>%
    select(subplot, next_up)
) %>%
# Get the coordinates of its down edge
left_join(
  quadrats %>%
    rename(x_ref_utm = x_utm, y_ref_utm = y_utm) %>%
    filter(quadrat_y == 0) %>%
    select(subplot, quadrat_x, x_ref_utm, y_ref_utm) %>%
    rename(next_up = subplot)
) %>%
# Correction to apply
mutate(dx_utm = x_ref_utm - x_utm, dy_utm = y_ref_utm - y_utm) %>%
select(subplot, quadrat_x, dx_utm, dy_utm) ->
reconcile_up
# Apply the right-edge correction
quadrats %>%
# Get the correction
left_join(
  quadrats %>%
    inner_join(reconcile_up) %>%

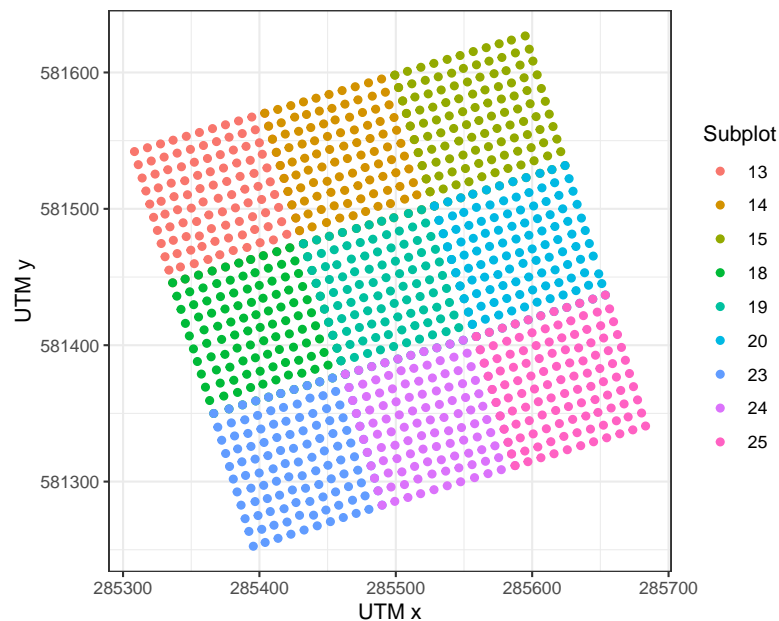
```

```

# Corrected coordinates
mutate(
  x_utm_up = x_utm + dx_utm * quadrat_y / up_edge,
  y_utm_up = y_utm + dy_utm * quadrat_y / up_edge,
) %>%
select(subplot, quadrat_x, quadrat_y, x_utm_up, y_utm_up)
) %>%
# Save the coordinates
mutate(
  x_utm = ifelse(is.na(x_utm_up), x_utm, x_utm_up),
  y_utm = ifelse(is.na(y_utm_up), y_utm, y_utm_up),
) -> quadrats

# 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")

```



Les coordonnées locales des quadrats doivent être mises à jour :

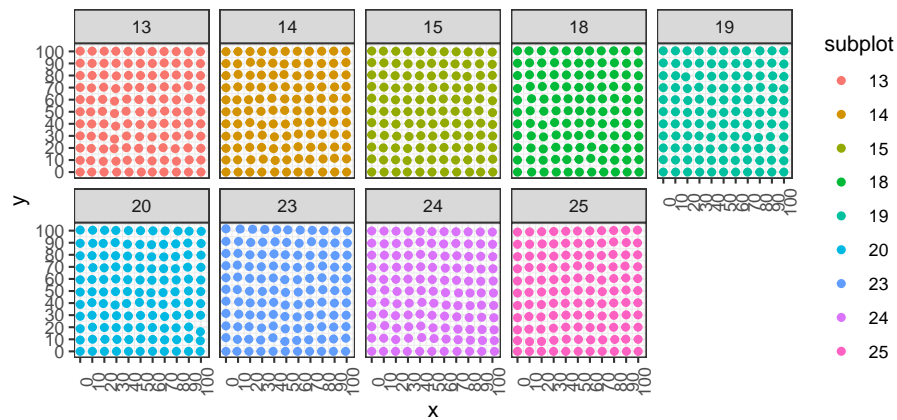
```

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

```



```
# Map
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)
```



## 5 Position des arbres

Les coordonnées de terrain des arbres sont mesurées dans chaque quadrat (valeurs comprises théoriquement entre 0 et 10m).

```
read.csv2("data/trees.csv") %>%
  # Homogenize the column names
  rename(
    subplot = "Subplot",
    quadrat_x = "Quadra.nb.X",
    quadrat_y = "Quadra.nb.Y",
    x_field = "Dist.X",
    y_field = "Dist.Y",
    diameter = "DBH"
  ) %>%
  drop_na() ->
  trees
```

## 5.1 Projection dans les quadrats réels

La position des arbres a été estimée relativement aux bords de chaque quadrat, en supposant que ce sont des carrés de 10m de côté. Leur forme réelle a été calculée à l'étape précédente. Les arbres doivent être projetés dans chacun des quadrats réels.

La matrice de projection de chaque quadrat de ses coordonnées de terrain (repère orthonormé, côtés de 10m) dans le quadrat réel est calculée par la fonction `ortho2real()`. Ses arguments sont les coordonnées des coins du quadrats calculées précédemment, à l'exception du coin supérieur droit, inutile.

```
ortho2real <- function(x_ul, x_dr, x_dl, y_ul, y_dr, y_dl) {  
  # Down edge  
  i_dx <- x_dr - x_dl  
  i_dy <- y_dr - y_dl  
  i_length <- sqrt(i_dx^2 + i_dy^2)  
  # Unit vector  
  i <- c(i_dx, i_dy) / i_length  
  # Left edge  
  j_dx <- x_ul - x_dl  
  j_dy <- y_ul - y_dl  
  j_length <- sqrt(j_dx^2 + j_dy^2)  
  # Unit vector  
  j <- c(j_dx, j_dy) / j_length  
  # Projection matrix  
  return(cbind(i, j))  
}
```

Il n'y a pas de moyen rigoureux d'interpoler ensuite la position des arbres dans les quadrats. La simple projection fait l'économie d'hypothèses supplémentaires sur la possibilité que les arbres dont une coordonnée locale est supérieure à 10m se trouve ou non dans le quadrat voisin : si une coordonnée excède celle des piquets, l'arbre est dans le quadrat voisin.

La fonction est appliquée à chaque quadrat :

```
# Prepare columns  
trees[, c("x_quadrat", "y_quadrat")] <- NA  
# Loop in the subplots  
for (subplot in subplots_field) {  
  # Loop in the quadrats  
  for (x_left in 0:(right_edge - 1)) {  
    for (y_down in 0:(up_edge - 1)) {  
      # Indicators to simplify the code later  
      is_the_subplot <- quadrats$subplot == subplot  
      is_left <- quadrats$quadrat_x == x_left  
      is_right <- quadrats$quadrat_x == x_left + 1  
      is_down <- quadrats$quadrat_y == y_down  
      is_up <- quadrats$quadrat_y == y_down + 1  
      # Corners of the quadrat  
      x_ul <- quadrats[is_the_subplot & is_up & is_left, "x"]  
      x_ur <- quadrats[is_the_subplot & is_up & is_right, "x"]  
      x_dr <- quadrats[is_the_subplot & is_down & is_right, "x"]  
      x_dl <- quadrats[is_the_subplot & is_down & is_left, "x"]  
      y_ul <- quadrats[is_the_subplot & is_up & is_left, "y"]  
      y_ur <- quadrats[is_the_subplot & is_up & is_right, "y"]  
      y_dr <- quadrats[is_the_subplot & is_down & is_right, "y"]  
      y_dl <- quadrats[is_the_subplot & is_down & is_left, "y"]  
      # Which trees?  
      is_in_quadrat <- trees$subplot == subplot &  
        trees$quadrat_x == x_left & trees$quadrat_y == y_down
```

```

# Coordinates of the trees in the quadrat
trees[is_in_quadrat, c("x_field", "y_field")] %>%
  # Matrix is necessary for the projection
  as.matrix() %>%
  # Coordinates must be rows
  t() -> trees_in_quadrat
# Project the trees into the real quadrat coordinate system
ortho2real(x_ul, x_dr, x_dl, y_ul, y_dr, y_dl) %*%
  trees_in_quadrat %>%
  # Transpose to have 2 columns rather than 2 rows
  t() ->
  # Save the coordinates
  trees[is_in_quadrat, c("x_quadrat", "y_quadrat")]
}
}
}

```

Les coordonnées des arbres sont relatives aux quadrats à ce stade. Il reste à les calculer dans le repère de chaque sous-parcelle.

```

trees %>%
  left_join(
    quadrats %>%
      # Get the coordinates of the origin of the quadrats and
      # the coordinates of the corners of the quadrats
      select(subplot, quadrat_x, quadrat_y, down_left_x_utm, down_left_y_utm, x, y),
      by = c("subplot", "quadrat_x", "quadrat_y")
  ) %>%
  # Coordinates of the trees in the quadrat coordinates
  mutate(x = x_quadrat + x, y = y_quadrat + y) -> trees

```

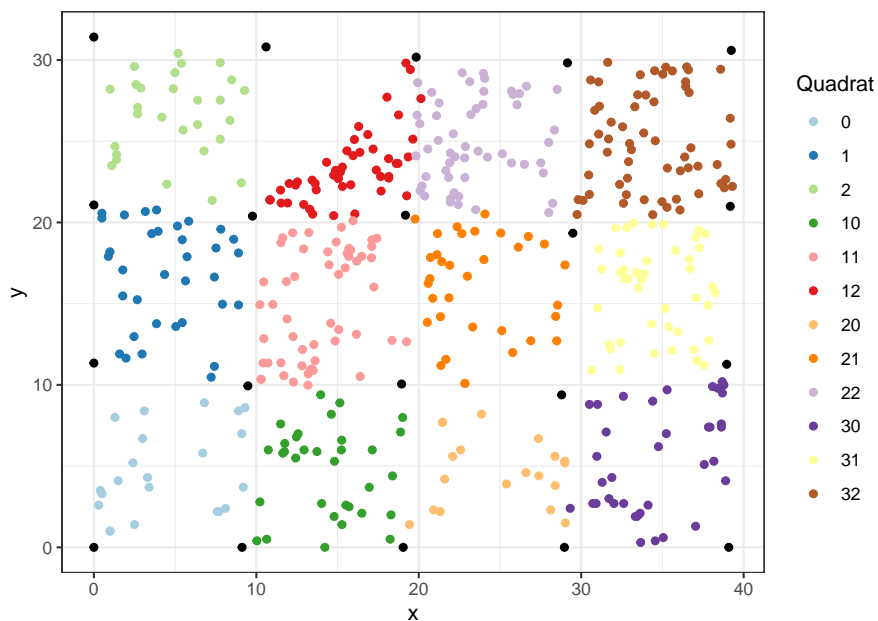
Des cartes locales peuvent être produites, par exemple pour les quadrats du coin du bas à gauche de la sous-parcelle 21 :

```

map_quadrats <- function(trees, the_subplot, x_min, x_max, y_min, y_max) {
  trees %>%
    filter(
      subplot == the_subplot &
      quadrat_x %in% (x_min %/% 10):(x_max - 1) %/% 10 &
      quadrat_y %in% (y_min %/% 10):(y_max - 1) %/% 10
    ) %>%
    mutate(quadrat = 10 * quadrat_x + quadrat_y) %>%
    ggplot() +
      geom_point(aes(x = x, y = y, color = as.factor(quadrat))) +
      geom_point(
        data = quadrats %>%
          filter(
            subplot == the_subplot &
            quadrat_x %in% (x_min %/% 10):(x_max %/% 10) &
            quadrat_y %in% (y_min %/% 10):(y_max %/% 10)
          ),
        aes(x = x, y = y)
      ) +
      coord_fixed() +
      labs(color = "Quadrat") +
      scale_colour_brewer(palette = "Paired", type = "div")
}

map_quadrats(trees, the_subplot = 23, x_min = 0, x_max = 40, y_min = 0, y_max = 30)

```

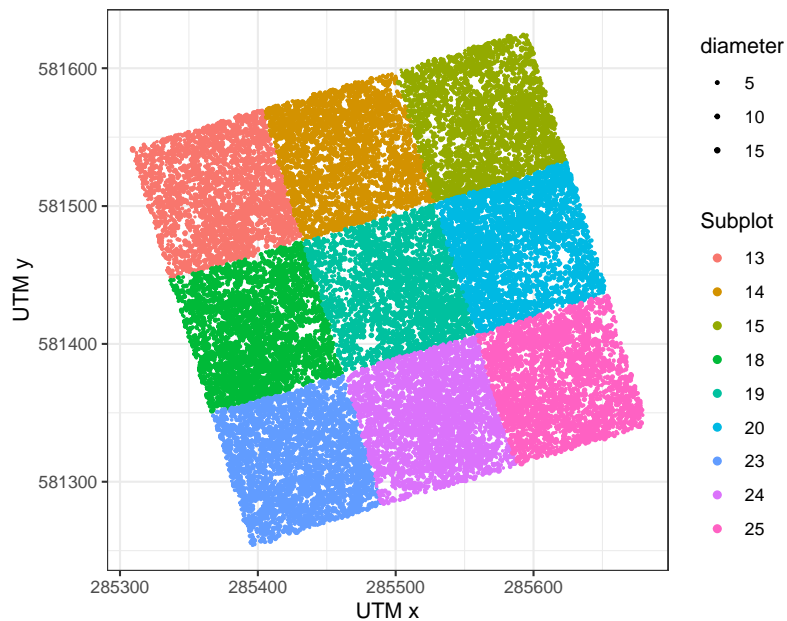


## 5.2 Coordonnées UTM

Finalement, les arbres doivent être projetés dans le repère UTM.

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

# Map
ggplot(
  trees,
  aes(x = x_utm, y = y_utm, color = as.factor(subplot), size = diameter)
) +
  geom_point() +
  coord_fixed() +
  scale_size_continuous(range = c(0.1, 1)) +
  labs(color = "Subplot", x = "UTM x", y = "UTM y")
```



Les coordonnées corrigées des arbres sont dans les colonnes `x` et `y` (repère local, valeurs comprises entre 0 et 100) et `x_utm` et `y_utm`.

## 6 Effet des corrections

Le déplacement des arbres relativement aux mesures brutes de terrain (dans chaque sous-parcelle, 10 fois le numéro du quadrat plus la position de l'arbre dans le quadrat) est de l'ordre d'un mètre. La figure suivante montre la densité de probabilité du déplacement en fonction de la sous-parcelle. La courbe en pointillés rassemble tous les arbres.

```
trees %>%
  mutate(
    x_field_subplot = x_field + 10 * quadrat_x,
    y_field_subplot = y_field + 10 * quadrat_y,
    correction_dx = x - x_field_subplot,
    correction_dy = y - y_field_subplot,
    correction_d = sqrt(correction_dx^2 + correction_dy^2)
  ) -> trees

# Distribution
trees %>%
  ggplot(aes(correction_d)) +
  geom_density(aes(x = correction_d), bounds = c(0, Inf), lty = 2) +
  geom_density(
    aes(
      x = correction_d,
      color = as.factor(subplot),
      fill = as.factor(subplot)
    ),
    bounds = c(0, Inf),
```

```

    alpha = 0.2
  ) +
  labs(
    color = "Sous-parcelle",
    fill = "Sous-parcelle",
    x = "Déplacement",
    y = "Densité"
  )

```

