Fiche TD avec le logiciel R : METAR

Les données météorologiques des aérodromes

Pr Jean R. Lobry

Comment récupérer les archives des données météorologiques des aérodromes radio-diffusées à l'attention des aéronefs. Application au calcul de la vitesse de diffusion de pollen de chêne.

Table des matières

Figure 1 : Exemple typique d'un système automatisé d'observation des conditions au sol (ASOS [3]) localisé à proximité de la zone de toucher des roues d'un aérodrome. Copie d'écran d'une partie de la page 97 de [6].

1 Introduction

Pour les besoins de l'aviation civile, de nombreux aérodromes sont équipés à proximité de la zone de toucher des roues de stations d'enregistrement de conditions météorologiques utiles aux pilotes de lignes (*e.g.* intensité et direction du vent, température, humidité, visibilité). Il s'agit donc de données *observationnelles* et non de prédictions ou d'interpolations. Elles sont soumises à un contrôle de qualité strict (normes de la série 9000 de l'ISO) pour des raisons de sécurité bien compréhensibles. Elles sont collectées toutes les minutes par des systèmes automatisés d'observation des conditions au sol[1] (figure 1 page 2) et un résumé est radio-diffusé régulièrement à l'attention des aéronefs toutes les heures ou demi-heures[2] sous la forme de bulletins METAR (*Meteorological Aerodrome Report* voir la figure 2 page 3).

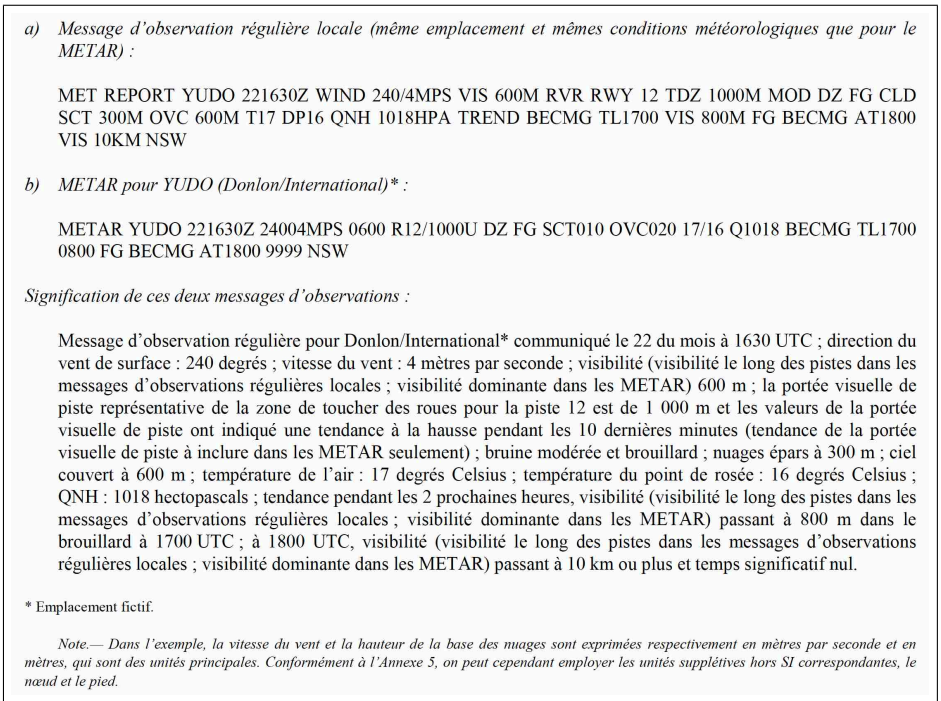
Une archive d'une partie des bulletins METAR est disponible à l'IEM (*Iowa Environment Mesonet*[3]) et décrites ici[4]. Les données de pluviométrie sont rarement disponibles car de nombreux états pensaient naguère que c'étaient des informations susceptibles d'alimenter la spéculation sur le cours en bourse des céréales. Le paquet `riem` [4] permet de récupérer facilement les données météorologiques de cette archive.

[1] ASOS : *Automated Surface Observing System* [3]

[2] Voire plus si les circonstances l'exigent, mais il ne s'agit plus alors d'un bulletin METAR *stricto sensu*, mais d'un bulletin SPECI.

[3] `https://mesonet.agron.iastate.edu/`

[4] `https://mesonet.agron.iastate.edu/info/datasets/metar.html`



a) *Message d'observation régulière locale (même emplacement et mêmes conditions météorologiques que pour le METAR) :*

MET REPORT YUDO 221630Z WIND 240/4MPS VIS 600M RVR RWY 12 TDZ 1000M MOD DZ FG CLD SCT 300M OVC 600M T17 DP16 QNH 1018HPA TREND BECMG TL1700 VIS 800M FG BECMG AT1800 VIS 10KM NSW

b) *METAR pour YUDO (Donlon/International)* :*

METAR YUDO 221630Z 24004MPS 0600 R12/1000U DZ FG SCT010 OVC020 17/16 Q1018 BECMG TL1700 0800 FG BECMG AT1800 9999 NSW

Signification de ces deux messages d'observations :

Message d'observation régulière pour Donlon/International* communiqué le 22 du mois à 1630 UTC ; direction du vent de surface : 240 degrés ; vitesse du vent : 4 mètres par seconde ; visibilité (visibilité le long des pistes dans les messages d'observations régulières locales ; visibilité dominante dans les METAR) 600 m ; la portée visuelle de piste représentative de la zone de toucher des roues pour la piste 12 est de 1 000 m et les valeurs de la portée visuelle de piste ont indiqué une tendance à la hausse pendant les 10 dernières minutes (tendance de la portée visuelle de piste à inclure dans les METAR seulement) ; bruine modérée et brouillard ; nuages épars à 300 m ; ciel couvert à 600 m ; température de l'air : 17 degrés Celsius ; température du point de rosée : 16 degrés Celsius ; QNH : 1018 hectopascals ; tendance pendant les 2 prochaines heures, visibilité (visibilité le long des pistes dans les messages d'observations régulières locales ; visibilité dominante dans les METAR) passant à 800 m dans le brouillard à 1700 UTC ; à 1800 UTC, visibilité (visibilité le long des pistes dans les messages d'observations régulières locales ; visibilité dominante dans les METAR) passant à 10 km ou plus et temps significatif nul.

* Emplacement fictif.

Note.— Dans l'exemple, la vitesse du vent et la hauteur de la base des nuages sont exprimées respectivement en mètres par seconde et en mètres, qui sont des unités principales. Conformément à l'Annexe 5, on peut cependant employer les unités supplétives hors SI correspondantes, le nœud et le pied.

Figure 2 : Copie d'écran d'une partie de l'annexe 3 « [a]ssistance météorologique à la navigation aérienne internationale » à la convention relative à l'aviation civile internationale (dite convention de Chicago de 1944) dans sa version consolidée en juillet 2013. Le bulletin METAR est un résumé dense, à caractère opérationnel, des conditions météorologiques au sol, par exemple dans l'horodatage `221630Z` seul le rang du jour dans le mois, ici 22, est donné, le mois et l'année en cours étant une information superfétatoire pour les pilotes d'aéronefs. L'heure, ici 16h30, est toujours donnée en UTC (`Z` pour « Zoulou » pour zéro : UTC+0) ce qui est bien pratique quand on ne cesse de changer de fuseau horaire. La clef de décodage des bulletins METAR est donnée début de l'annexe C du manuel technique des systèmes automatisés d'observation au sol (ASOS) [3]. Si besoin est, le paquet `pmetar` [1] permet d'extraire les informations des bulletins METAR. Par exemple, la fonction `metar_temp()` permet de récupérer la température en degré Celcius.

2 Le paquet `riem`

Le paquet `riem` [4] permet de récupérer les données météorologiques de l'archive de l'IEM. Il comporte trois fonctions qui, du plus général au plus particulier, permettent de récupérer :

1° la liste grandes régions géographiques pour les aéroports disponibles sur la planète avec `riem_networks()` ;

2° la liste des aéroports d'une région géographique donnée avec `riem_stations()` ;

3° les données pour un aéroport donné avec `riem_measures()`.

2.1 `riem_networks()`

Les aéroports sont identifiés par leur code OACI (Organisation de l'Aviation Civile Internationale), voir la figure 3 page 5, différent du code IATA (Association du Transport Aérien International) portés sur les billets d'avion. Par exemple l'aéroport de Lyon Saint-Exupéry de code IATA `LYS` est de code OACI `LFLL`. Les régions géographiques renvoyées par `riem_networks()` correspondent plus ou moins à un pays[5].

```
netw <- riem_networks()
comment(netw) <- paste("Téléchargé le :", Sys.time())
save(netw, file = "data/netw.Rda")
```

```
chmin <- "http://pbil.univ-lyon1.fr/R/donnees/METAR/"
load(url(paste0(chmin, "netw.Rda")))
comment(netw)
```

```
[1] "Téléchargé le : 2023-03-27 15:22:36"
```

```
nrow(netw)
```

```
[1] 263
```

```
head(netw)
```

```
      code                        name
1 AE__ASOS United Arab Emirates ASOS
2 AF__ASOS           Afghanistan ASOS
3 AG__ASOS  Antigua and Barbuda ASOS
4 AI__ASOS              Anguilla ASOS
5  AK_ASOS                Alaska ASOS
6 AL__ASOS               Albania ASOS
```

```
netw[netw$code == "FR__ASOS", ]
```

```
       code        name
85 FR__ASOS France ASOS
```

Le sous-ensemble qui nous intéresse porte donc le code `FR__ASOS` (avec deux blancs soulignés). On peut maintenant récupérer la liste des aéroports de cette région géographique.

[5] Si j'ai bien compris la nomenclature il y a deux blancs soulignés dans le code géographique de la région si c'est un pays *stricto sensu*, par exemple `AF__ASOS` pour l'Afghanistan, et un seul blanc souligné si c'est un sous-ensemble d'un pays, par exemple `AK_ASOS`, pour l'Alaska.

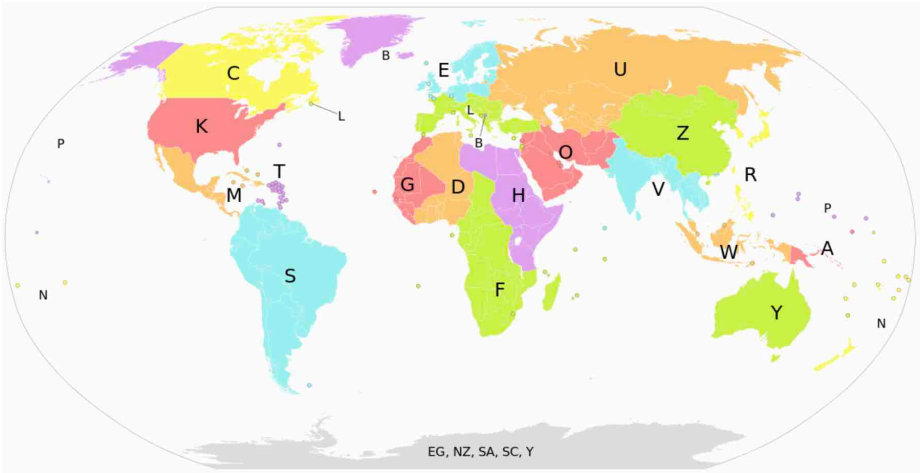


Figure 3 : La première lettre du code OACI (Organisation de l'Aviation Civile Internationale) des aéroports correspond à des grandes régions géographiques, avec quelque exceptions comme `LFVP` pour l'aéroport de Saint-Pierre à Saint-Pierre-et-Miquelon. La deuxième lettre correspond à une subdivision géographique, par exemple `F` pour France. Source : wikipedia, contribué par Hytar.

2.2 `riem_stations()`

Cette fonction nous permet de récupérer les informations sur les aéroports et la disponibilité des données. Dans le cas `FR__ASOS` cela correspond à la France métropolitaine plus l'aéroport de Saint-Pierre-et-Miquelon qui est un cas un peu particulier (voir la figure 3 page 5).

```
stations <- riem_stations(network = "FR__ASOS")
comment(stations) <- paste("Téléchargé le :", Sys.time())
save(stations, file = "data/stations.Rda")
```

```
load(url(paste0(chmin, "stations.Rda")))
comment(stations)
```

```
[1] "Téléchargé le : 2023-04-14 12:17:48"
```

```
nrow(stations)
```

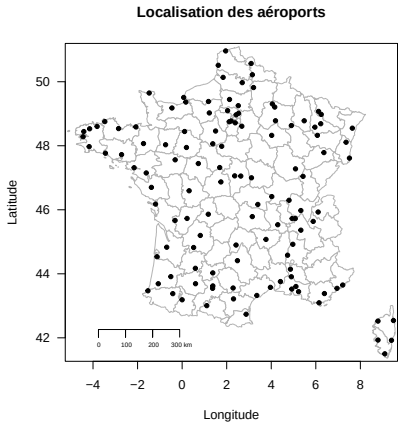
```
[1] 123
```

```
names(stations)
```

```
 [1] "index"        "id"           "synop"        "name"         "country"
 [6] "elevation"    "network"      "online"       "plot_name"    "archive_end"
[11] "modified"     "spri"         "tzname"       "iemid"        "archive_begin"
[16] "metasite"     "longitude"    "latitude"     "state"        "lon"
[21] "lat"
```

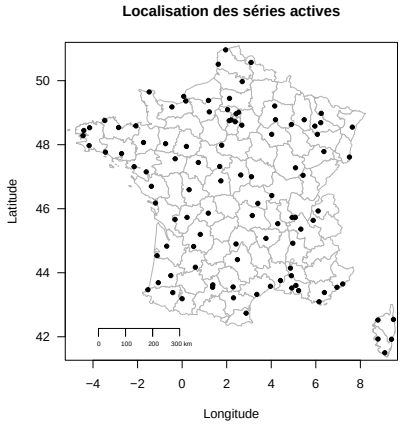
Le fuseau horaire de l'aéroport est donné dans la colonne `tzname`, ce qui nous permet de filtrer facilement pour ne conserver que les aéroports métropolitains. Les coordonnées spatiales sont données dans `longitude` et `latitude`, mais aussi de façon dupliquée dans `lon` et `lat`.

```
mapFrance <- function(){
  require(maps)
  map("france", mar = c(5, 4, 4, 2) + 0.1,
      fill = TRUE, col = "white", border = grey(0.7))
  map.axes(las = 1)
  map.scale(ratio = FALSE, cex = 0.5, relwidth = 0.2)
}
mapFrance()
title(main = "Localisation des aéroports", xlab = "Longitude", ylab = "Latitude")
with(subset(stations, tzname == "Europe/Paris"), {
  points(lon, lat, pch = 19, cex = 0.75)
})
```



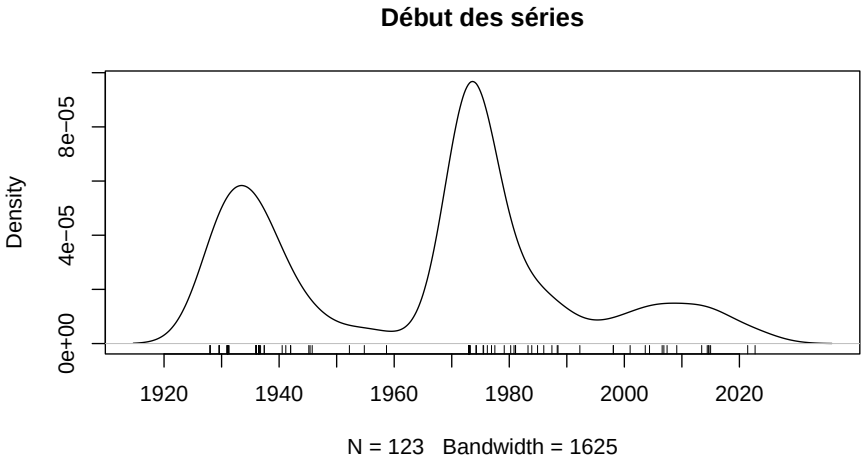
En dehors du massif alpin, on a donc une assez bonne couverture du territoire métropolitain. Voyons maintenant ce qu'il en est pour la disponibilité des données. La colonne `archive_end` porte une valeur manquante si la série est toujours activement documentée.

```
mapFrance()
title(main = "Localisation des séries actives", xlab = "Longitude", ylab = "Latitude")
with(subset(stations, tzname == "Europe/Paris" & is.na(archive_end)), {
  points(lon, lat, pch = 19, cex = 0.75)
})
```



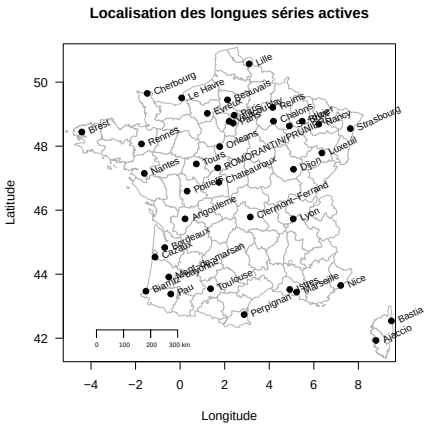
On n'a pas perdu trop de monde dans la bataille, la couverture reste assez bonne en ne conservant que les séries actives. La colonne `archive_begin` nous renseigne sur le début des séries. On peut distinguer schématiquement trois classes : celles antérieures à 1960, celles postérieures à 1995 et celles entre les deux.

```
library(lubridate, warn.conflicts = FALSE)
deb <- as.numeric(as.Date(stations$archive_begin))
plot(density(deb, adjust = 0.5), xaxt = "n", main = "Début des séries")
rug(deb)
xseq <- as.Date(paste0(seq(1920, 2020, by = 10), "-01-01"))
axis(1, year(xseq), at = as.numeric(xseq))
```



Les séries les plus longues (antérieures à 1960) toujours actives correspondent en général aux aéroports des grandes agglomérations.

```
mapFrance()
title(main = "Localisation des longues séries actives", xlab = "Longitude", ylab = "Latitude")
with(subset(stations, tzname == "Europe/Paris" & is.na(archive_end) &
    as.Date(archive_begin) < as.Date("1960-01-01")), {
  points(lon, lat, pch = 19, cex = 1)
  text(lon, lat, name, pos = 4, xpd = NA, cex = 0.75, srt = 25)
})
```



2.3 riem_measures()

Cette fonction permet de récupérer les données entre deux dates pour un aéroport donné. Illustrons ceci avec l'aéroport de Brest. Nous avons besoin de récupérer son code OACI (`LFRB`) et de la plage temporelle de disponibilité des données. La colonne `archive_end` n'est pas documentée, la série est donc toujours active. La date de dernière modification (`modified`) est le `2020-10-05` et `archive_begin` vaut `1929-08-01` :

```
(myid <- stations[stations$name == "Brest", "id"])
```

```
[1] "LFRB"
```

```
stations[stations$id == myid,
         c("id", "name", "plot_name", "archive_begin", "modified", "archive_end")]
```

```
    id  name        plot_name        archive_begin             modified archive_end
21 LFRB Brest BREST/GUIPAVAS   1929-08-01T00:00:00Z 2020-10-05T10:52:09Z        <NA>
```

Nous avons donc maintenant toutes les informations nécessaires pour télécharger les données. Comme je ne veux pas solliciter indûment le serveur de l'IEM trop souvent, je n'effectue ma requête qu'une seule fois et sauvegarde au format XDR [7] le résultat stocké dans l'objet `brest` dans le fichier `brest.Rda`. J'ajoute au passage un horodatage pour la traçabilité et le transtype en un objet de la classe `data.frame` qui est une classe de base de R.

```
brest <- riem_measures(
  station = "LFRB",
  date_start = "1929-08-01",
  date_end = "2020-10-05")
brest <- as.data.frame(brest)
comment(brest) <- paste("Téléchargé le :", Sys.time())
save(brest, file = "data/brest.Rda")
```

```
load(url(paste0(chmin, "brest.Rda")))
comment(brest)
```

```
[1] "Téléchargé le : 2023-03-27 17:37:31"
```

```
names(brest)
```

```
 [1] "station"           "valid"             "lon"               "lat"
 [5] "tmpf"              "dwpf"              "relh"              "drct"
 [9] "sknt"              "p01i"              "alti"              "mslp"
[13] "vsby"              "gust"              "skyc1"             "skyc2"
[17] "skyc3"             "skyc4"             "skyl1"             "skyl2"
[21] "skyl3"             "skyl4"             "wxcodes"           "ice_accretion_1hr"
[25] "ice_accretion_3hr" "ice_accretion_6hr" "peak_wind_gust"    "peak_wind_drct"
[29] "peak_wind_time"    "feel"              "metar"             "snowdepth"
```

```
brest[1, "metar"]
```

```
[1] "LFRB 010600Z AUTO 29018KT 6SM 16/ RMK SLP073 53004 IEM_DS3505"
```

Les données originelles, le bulletin METAR, sont préservées dans la colonne `metar` et ont été décodées dans les autres colonnes. La date et l'heure de l'observation sont données au format `POSIXct`[6] dans la colonne `valid` en UTC :

```
class(brest$valid)
```

```
[1] "POSIXct" "POSIXt"
```

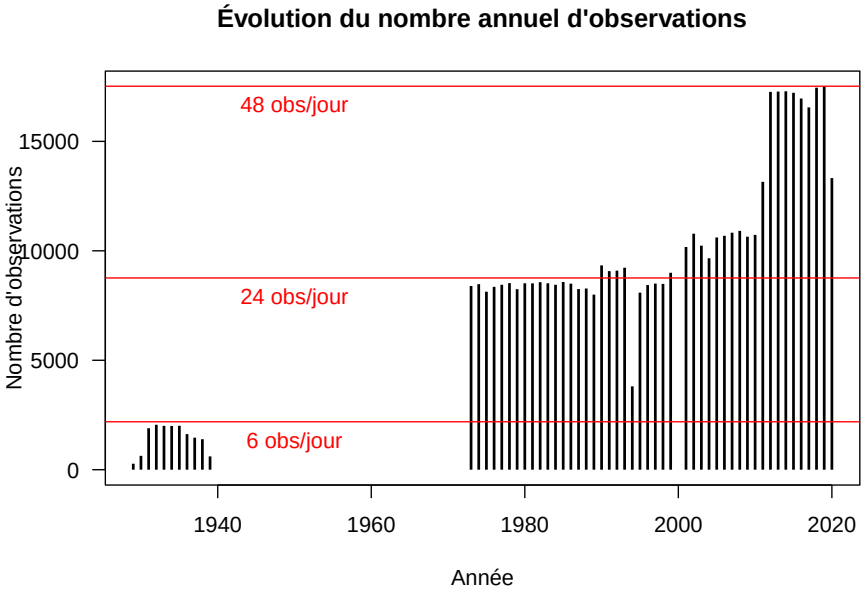
```
tz(brest$valid)
```

```
[1] "UTC"
```

[6] Pour la manipulation de ce type de données voir la fiche « [m]anipulation de données temporelles appliquée au suivi horaire de la croissance de dix chênes pendant un an » à `http://pbil.univ-lyon1.fr/R/pdf/microdendroCHS57.pdf`.

De combien d'observations dispose-t-on par année ? Ça a tendance à augmenter au cours du temps et on passe de 6, 24 puis 48 observations par jour en moyenne. La série n'est pas complète : il y a un gros trou de 34 ans après 1940[7]. On a des données de 1973 à 2020 avec l'an 2000 qui manque.

```
brest$year <- year(brest$valid)
nopy <- table(brest$year)
x <- as.integer(names(nopy)) ; y <- as.integer(nopy)
par(lend = "butt")
plot(x, y, ylim = c(0, max(y)), pch = 19, cex = 0.5, type = "h", lwd = 2, las = 1,
     xlab = "Année", ylab = "Nombre d'observations",
     main = "Évolution du nombre annuel d'observations")
npj <- c(6, 24, 48) ; h <- 365*npj
abline(h = h, col = "red")
text(rep(1950, 3), h, paste(npj, "obs/jour"), pos = 1, col = "red")
```



3 Application au calcul de la vitesse de diffusion pollinique

Le point de départ est celui des équations d'un article [5] qui cherche à modéliser la dispersion des grains de pollen de chêne.

3.1 Pression de vapeur à saturation e_{sat}

Elle est donnée par l'équation A2 page 192 de [5]. C'est la pression de vapeur[8] en hPa à saturation en fonction de la température en degrés Celcius. C'est donc la pression maximum que l'on puisse obtenir dans l'air, elle augmente

[7] Les données antérieures à 1940 me semblent anachroniques, peut-être est-ce une reconstitution de l'IEM à partir d'autres sources, d'ailleurs les bulletins METAR correspondants portent à la fin la mention non-standard `IEM_DS3505`

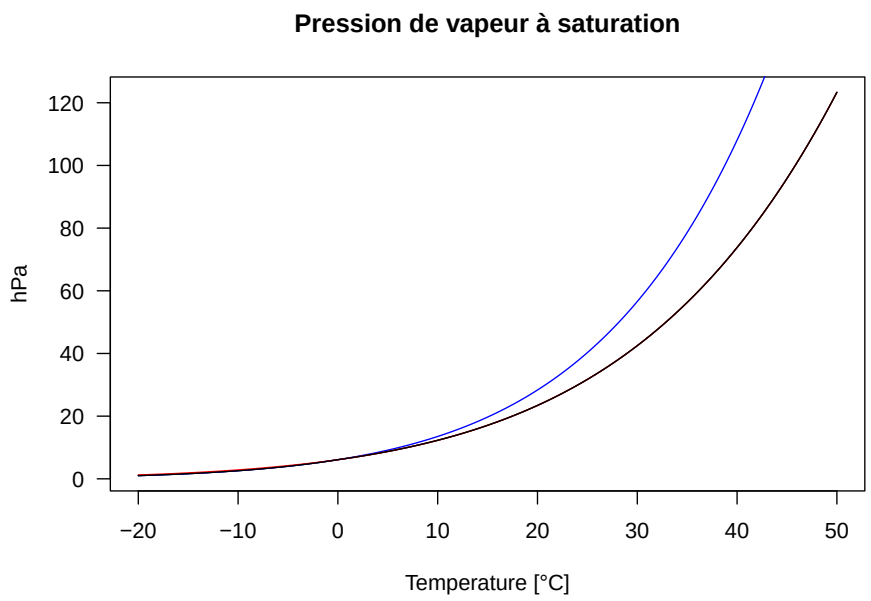
[8] Il est sous-entendu dans l'ensemble de ce document qu'il s'agit de vapeur d'eau.

avec la température. Les auteurs font référence à Magnus mais sans donner de référence bibliographique. J'ai trouvé un article de 1844 [2] sur « les forces de tension de la vapeur d'eau[9] » où on trouve page 246 une expression qui ressemble fort à celle utilisée :

$$e = 4{,}525 . 10^{\frac{7{,}4475\, t}{234{,}69 + t}} \text{ mm}$$

La principale différence dans [5] est que La formule utilisée distingue les températures négatives et positives, je n'ai pas trouvé la référence donnant les valeurs numériques utilisées. La correction à basse température est très faible, de l'ordre de l'épaisseur du trait.

```
# Ta is the temperature in Celcius
esat <- function(Ta){
  ifelse(Ta < 0,
    6.1078*exp(22.44294*Ta/(272.44 + Ta)),
    6.1078*exp(17.08085*Ta/(234.175 + Ta)))
}
x <- seq(-20, 50, length = 255) ; y <- esat(x)
plot(x, y, type = "l", main = "Pression de vapeur à saturation",
     xlab = "Temperature [řC]",
     ylab = "hPa", las = 1)
lines(x, 6.1078*exp(22.44294*x/(272.44 + x)), col = "blue")
lines(x, 6.1078*exp(17.08085*x/(234.175 + x)), col = "red")
lines(x, y)
```



La pression de vapeur à saturation gambade entre 0 et 100 hPa, à comparer avec la pression atmosphérique qui est de l'ordre de 1000 hPa au niveau de la mer.

[9] die Spannkräfte des Wasserdampfs

3.2 La pression de vapeur $e_{\mathbf{curr}}$

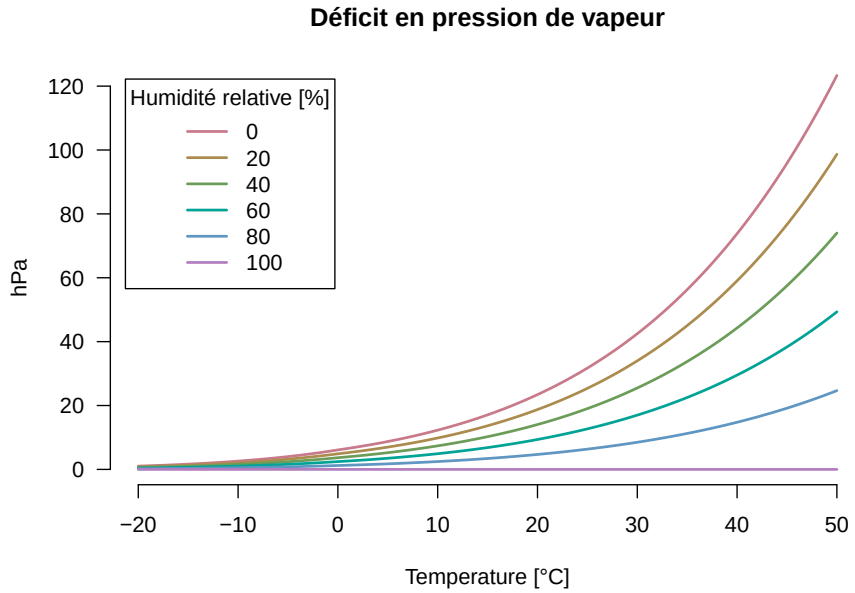
La pression « courante » de vapeur est donnée par l'équation A3 page 192 de [5], c'est simplement l'humidité relative multipliée par la pression à saturation. Une humidité relative de 100 % signifie que la pression « courante » est la pression à saturation.

```
# current water vapor pressure (ecurr, hPa):
ecurr <- function(Ta, RH) esat(Ta)*RH/100
```

3.3 Déficit en pression de vapeur VPD

Le VPD pour *Vapor Pressure Deficit*, soit le déficit de pression de vapeur, est donné par l'équation A1 page 192 de [5]. C'est simplement l'écart entre la pression à saturation et la pression courante. Plus il est grand, plus il est facile de se refroidir en transpirant. Pour une humidité relative de 0 %, VPD est égal à e_{sat}.

```
VPD <- function(Ta, RH) esat(Ta) - ecurr(Ta, RH)
plot.new() ; plot.window(xlim = c(-20, 50), ylim = c(0, 120))
x <- seq(-20, 50, length = 255)
RHseq <- seq(0, 100, by = 20)
n <- length(RHseq)
cols <- hcl.colors(n, "Dark 2")
for(i in seq_len(n)){
  y <- VPD(x, RHseq[i])
  points(x, y, type = "l", col = cols[i], lwd = 2)
}
title(main = "Déficit en pression de vapeur") ; axis(1) ; axis(2, las = 1)
title(xlab = "Temperature [ř C]", ylab = "hPa")
legend("topleft", inset = 0.02, legend = RHseq, lty = 1,
       col = cols, title = "Humidité relative [%]", lwd = 2)
```



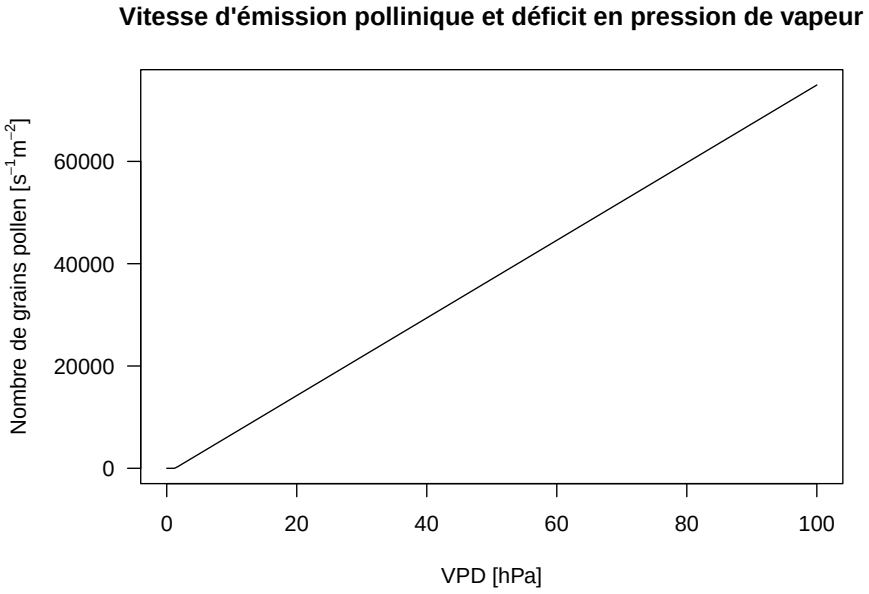
3.4 Vitesse d'émission pollinique P_A

L'émission de pollen P_A est donnée par l'équation 4 page 183 de [5] :

$$P_A = \begin{cases} \Delta t P_{Ts}(33VPD - 42) \text{ for } VPD > 1.273, \\ 0 \text{ else.} \end{cases} \quad (4)$$

VOICI ce que j'ai compris de l'explication originelle[10] : le nombre de grains de pollen émis par mètre carré de forêt de chênes, P_A, à chaque intervalle de temps, Δt, est calculé en fonction de la *production* totale de 1 m^2 de forêt de chênes exprimée comme un nombre potentiel de grains de pollen par seconde, P_{Ts}, et de VPD. Je n'ai pas trouvé la valeur de P_{Ts} utilisée, j'ai posé de façon arbitraire $\Delta t P_{Ts} = 23$ pour me caler sur les conditions du 2 mai de la figure 3 de [5], donc pas loin du pic de production pollinique. Il y a une valeur seuil de 1.273 hPa pour VPD en dessous de laquelle il n'y a aucune émission de pollen, au dessus de ce seuil elle croit linéairement avec VPD. Ce seuil correspond en fait à la valeur en deçà de laquelle le modèle est négatif et sert donc à forcer la positivité.

```
VPDseq <- seq(0, 100, le = 255)
par(mar = c(5, 6, 4, 2) + 0.1)
ylab <- expression(paste("Nombre de grains pollen [",
                         s^-1*m^-2, "]"))
plot(VPDseq, pmax(0, 23*(33*VPDseq - 42)), type = "l", las = 1,
     xlab = "VPD [hPa]", ylab = "",
     main = "Vitesse d'émission pollinique et déficit en pression de vapeur")
title(ylab = ylab, line = 4)
```

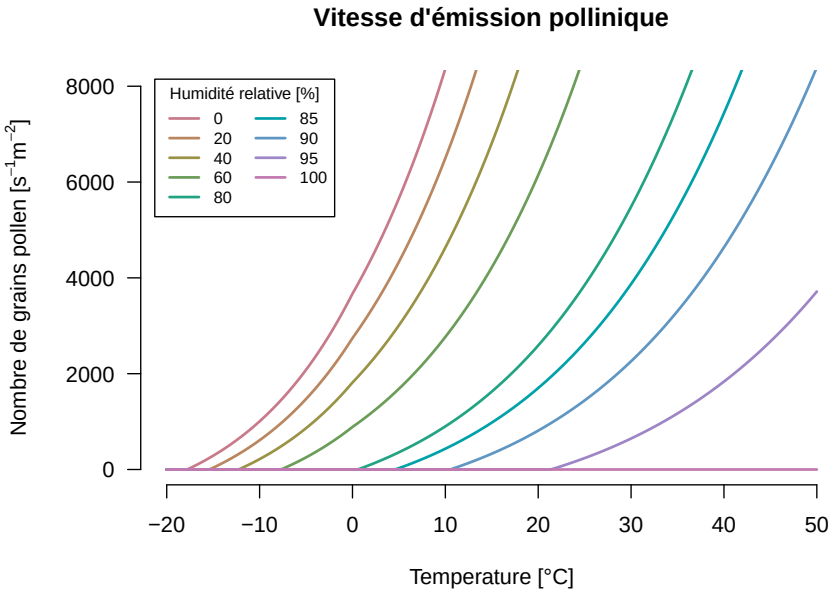


ON voit que sur la gamme des valeurs physiquement vraisemblables pour VPD, la vitesse d'émission pollinique est quasi-synonyme (à une constante

[10] The relative number of emitted pollen grains per square meter oak forest (P_A) in each model time step Δt is calculated dependent on the total pollen production of 1 m^2 oak forest, expressed as potential number of pollen grains per second (P_{Ts}), and on VPD.

près) de déficit en pression de vapeur. On aurait peut-être intérêt à travailler directement avec VPD pour avoir des unité (hPa) moins exotiques que des grains de pollen par seconde et par mètre carré de forêt de chêne. Il suffirait de forcer les valeurs de VPD inférieures à 1.273 à zéro pour rester cohérent avec le modèle de vitesse d'émission pollinique. Cela éviterait d'avoir à poser de façon un peu arbitraire $\Delta tP_{Ts} = 23$.

```
PA <- function(Ta, RH) pmax(0, 23*(33*VPD(Ta, RH) - 42))
par(mar = c(5, 6, 4, 2) + 0.1)
plot.new() ; plot.window(xlim = c(-20, 50), ylim = c(0, 8000))
x <- seq(-20, 50, length = 255)
RHseq <- c(seq(0, 80, by = 20), 85, 90, 95, 100)
n <- length(RHseq)
cols <- hcl.colors(n, "Dark 2")
for(i in seq_len(n)){
  y <- PA(x, RHseq[i])
  points(x, y, type = "l", col = cols[i], lwd = 2)
}
title(main = "Vitesse d'émission pollinique") ; axis(1) ; axis(2, las = 1)
title(xlab = "Temperature [ř C]")
title(ylab = ylab, line = 4)
legend("topleft", inset = 0.02, legend = RHseq, lty = 1,
       col = cols, title = "Humidité relative [%]", lwd = 2, ncol = 2, cex = 0.75)
```



La vitesse d'émission pollinique croit très rapidement avec la température avec un effet de « veto » de l'humidité relative. À 100 % d'humidité il n'y a aucune émission de pollen, à 95 % il n'y a aucune émission en dessous de 20 degrés et à 90 % en dessous de 10 degrés. Quand l'air est sec la vitesse de diffusion pollinique est loin d'être négligeable même à très basse température, la fécondation anémophile reste alors possible tandis que l'entomogamie ne le serait plus à cause de la poïkilothermie des insectes.

3.5 Aéroport de Brest

On veut calculer la vitesse d'émission pollinique avec les données météorologiques de Brest. Dans les archives de l'IEM les températures sont données en degrés Farenheit dans la colonne `tmpf`, la première chose à faire est donc des les convertir en degrés `Celcius`.

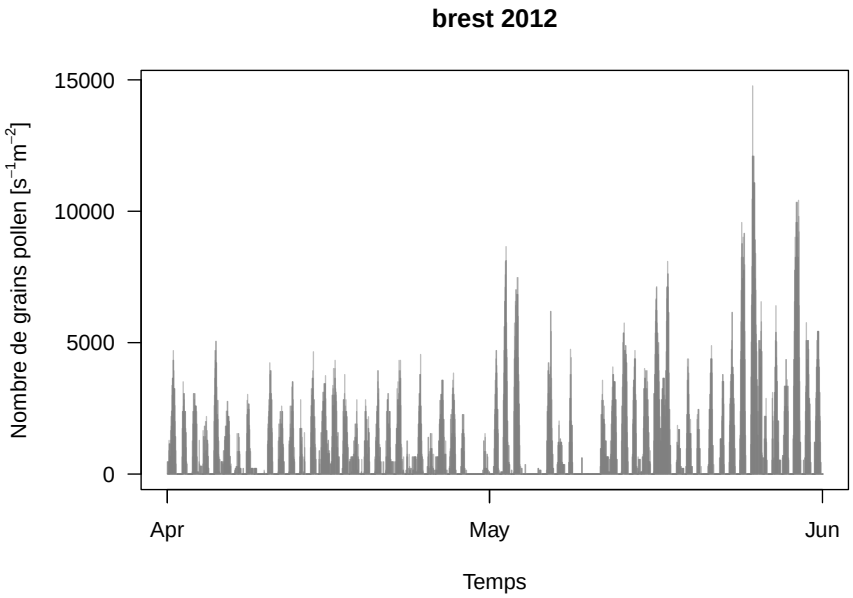
```
F2C <- function(x) (x - 32)/1.8 # Farenheit -> Celcius
brest$tmpc <- F2C(brest$tmpf)
```

Toutes les fonctions définies étant vectorisées, la vitesse d'émission pollinique se calcule directement avec la fonction `PA()` en lui passant la température en degrés Celcius de la colonne `tmpc` et l'humidité relative en pourcent de la colonne `relh` :

```
brest$PA <- with(brest, PA(tmpc, relh))
```

On peut maintenant définir la fonction `plotPA()` pour représenter la vitesse de diffusion pollinique au cours du temps, on regardera par défaut les mois d'avril et de mai correspondant à la fenêtre de diffusion du pollen de chêne.

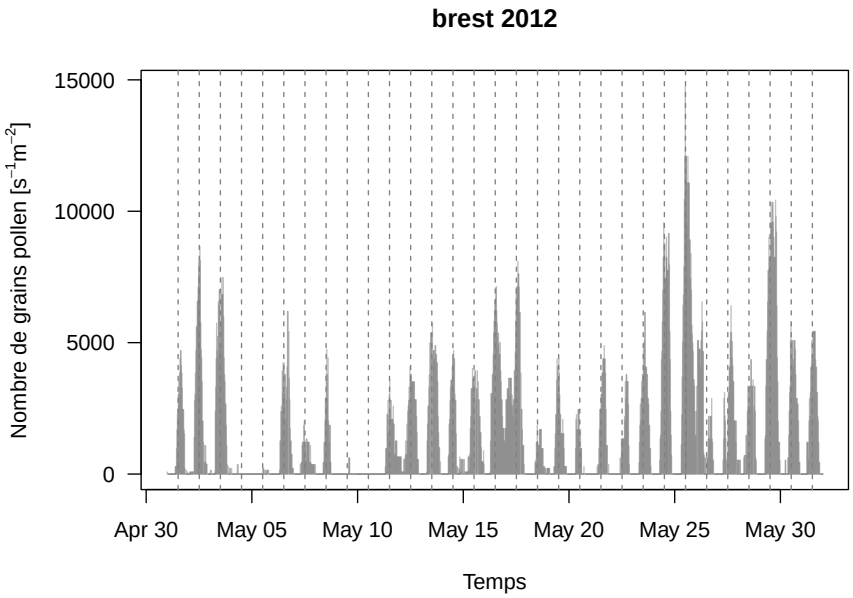
```
library(suncalc)
plotPA <- function(dta, the_year, minm = 4, maxm = 5, addM = FALSE){
  tmp <- subset(dta, as.integer(year(valid)) == the_year &
                month(valid) <= maxm &
                month(valid) >= minm &
                !is.na(PA))
  ylab <- expression(paste("Nombre de grains pollen [", s^-1 * m^-2, "]"))
  main <- paste(substitute(dta), the_year)
  par(mar = c(5, 6, 4, 2))
  plot(tmp$valid, tmp$PA, type = "h", col = rgb(0.5, 0.5, 0.5, 0.5),
       las = 1, main = main, xlab = "Temps", ylab = "")
  title(ylab = ylab, line = 4)
  if(addM){
    jours <- unique(as.Date(tmp$valid))
    midi <- getSunlightTimes(jours,
                 lat = tmp[1, "lat"],
                 lon = tmp[1, "lon"],
                 tz = "UTC")["solarNoon"]
    abline(v = midi[, 1], lty = 2, col = grey(0.5))
  }
}
plotPA(brest, 2012)
```



On voit que le pollen n'est pas toujours émis, et quand il l'est c'est sous la forme d'une pulsation journalière. On a ici des pics d'émission à 10000 grains de pollen par seconde et par mètre carré de forêt de chêne (avec une production correspondant à celle du pic de pollinisation de [5] en 2000).

Pour faire le lien avec les heures de la journée, on zoome sur un seul mois et on représente le midi solaire avec des lignes verticales. La valeur du midi solaire pour un lieu de latitude et de longitude données se calcule facilement avec le paquet `suncalc` [8].

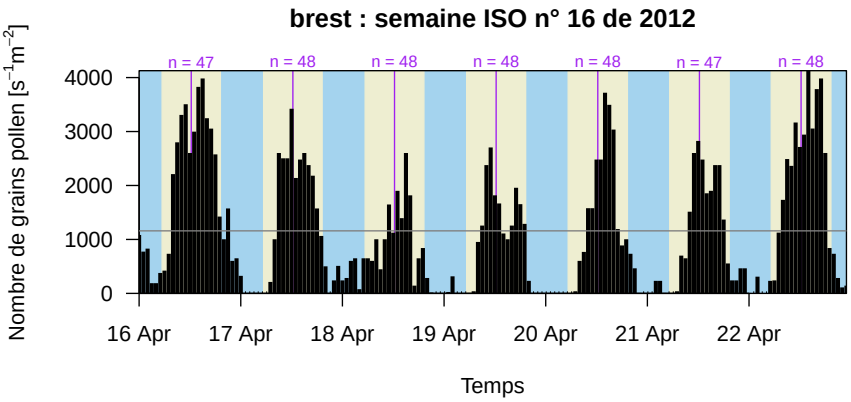
```
plotPA(brest, 2012, 5, 5, addM = TRUE)
```



Comme on peut le constater, le pic de la vitesse de diffusion pollinique, quand il existe, est centré autour du midi solaire. Cela n'a rien d'étonnant, à une latitude de 48.4 ° le maximum de température journalier est en général voisin d'icelui. Comme la vitesse de diffusion pollinique croit fortement avec la température, on se retrouve avec un pic étroit centré autour du midi solaire. Pour regarder les choses de plus près on définit une fonction pour représenter les données moyennées par heure à l'échelle d'une semaine.

```
plotPAw <- function(dta, the_year, the_week,
    nb = "lightskyblue2", yd = "lightyellow2", lwd = 3){
  nom <- deparse1(substitute(dta))
  # Sélection des données de la semaine à représenter
  dta$tmpc <- F2C(dta$tmpf)
  dta$PA <- with(dta, PA(tmpc, relh))
  tmp <- subset(dta, as.integer(year(valid)) == the_year &
                as.integer(isoweek(valid)) == the_week &
                !is.na(PA))
  # Calcul des moyennes horaires de PA
  # Algorithme de la nounou : toute heure entamée est due
  tmp$myfac <- tmp$valid
  minute(tmp$myfac) <- 0 ; second(tmp$myfac) <- 0
  tmp$myfac <- as.factor(tmp$myfac)
  y <- with(tmp, tapply(PA, myfac, mean))
  x <- as.POSIXct(names(y), tz = "UTC")
  # Calcul du nombre d'observations par jour
  tmp$facj <- tmp$valid
  hour(tmp$facj) <- 0 # Algorithme de l'intérimaire
  minute(tmp$facj) <- 0 ; second(tmp$facj) <- 0
  nobs <- with(tmp, tapply(PA, facj, length))
  # Graphique : options générales du périphérique
  par(mar = c(5, 6, 4, 2), lend = "butt", yaxs = "i", xaxs = "i")
  plot.new() ; plot.window(xlim = range(x), ylim = range(y))
  # Graphique : apprêt jour/nuit et midi solaire
  soleil <- getSunlightTimes(unique(as.Date(x, tz = "UTC")),
                lat = tmp[1, "lat"],
                lon = tmp[1, "lon"],
                tz = "UTC")[c("sunrise", "sunset", "solarNoon")]
  pu <- par("usr") ; xleft <- pu[1] ; ybottom <- pu[3]
  xright <- pu[2] ; ytop <- pu[4]
  rect(xleft, ybottom, xright, ytop, col = nb, border = nb)
```

```
  for(i in seq_len(nrow(soleil))){
    rect(soleil[i, "sunrise"], ybottom,
         soleil[i, "sunset"], ytop, col = yd, border = yd)
  }
  abline(v = soleil[ , "solarNoon"], col = "purple")
  # Graphique : nombre de données par jour
  msg <- paste("n =", nobs)
  text(soleil[ , "solarNoon"], 1.04*ytop, msg, xpd = NA, col = "purple", cex = 0.75)
  # Graphique : les données
  points(x, y, type = "h", lwd = lwd)
  # Graphique : les cosmétiques
  ylab <- expression(paste("Nombre de grains pollen [", s^-1 * m^-2, "]"))
  main <- paste(nom, ": semaine ISO nř", the_week, "de", the_year)
  title(main = main, xlab = "Temps")
  title(ylab = ylab, line = 4)
  xct <- as.POSIXct(x, origin = origin)
  xct <- xct[!duplicated(day(xct))]
  axis(1, xct, paste(day(xct), month(xct, TRUE)))
  axis(2, las = 1) ; box()
  abline(h = mean(y), col = grey(0.5))
}
plotPAw(brest, 2012, 16)
```



Grace aux données météorologiques des aérodromes nous avons une vision très fine de la vitesse de diffusion pollinique. Dans le cas présent nous avons en général 48 points par jour, soit une donnée toutes les demi-heures. Le pic de diffusion a lieu de jour (fond jaune) avec un maximum voisin du midi solaire (ligne verticale pourpre). En moyenne on a une vitesse de l'ordre de 10^3 $s^{-1}m^{-2}$, comme une journée comporte 86400 secondes, cela correspond à vitesse de l'ordre de 10^8 grains de pollen par jour et par mètre carré de chêne. D'après la figure 2 de [5], le gros de l'émission pollinique se fait en 10 jours, on retombe donc sur l'ordre de grandeur de 10^9 grains de pollen produits en une saison par un mètre carré de chêne [5].

Comme le pollen est émis de façon pulsée et journalière, on se pose la question de savoir si on peut arriver à estimer sa vitesse d'émission lorsque l'on ne possède que de données journalières dégradées : la température moyenne, l'humidité relative moyenne, la température minimale et la température maximale. Pour faire cette étude, je ne vais conserver que les années avec au moins 15000 données, soit une quarantaine de valeurs par jour en moyenne, soit de 2012 à 2019.

```
selectyears <- function(dta, thres = 15000){
  nobsperyear <- table(year(dta$valid))
  target <- names(nobsperyear[nobsperyear > thres])
```

```
  res <- subset(dta, year(valid) %in% target)
  res$tmpc <- F2C(res$tmpf)
  res$year <- as.integer(year(res$valid))
  res$PA <- PA(res$tmpc, res$relh)
  res$month <- month(res$valid)
  return(res)
}
brst <- selectyears(brest, thres = 15000)
unique(year(brst$valid))
```

```
[1] 2012 2013 2014 2015 2016 2017 2018 2019
```

La fonction suivante calcule la « vraie » valeur de la vitesse d'émission pollinique en faisant la moyenne de toutes les valeurs observées pour une journée. Elle calcule ensuite les données dégradées et les valeurs de la vitesse d'émission pollinique que l'on peut en déduire.

```
getcomps <- function(dta){
  options(warn = -1) ; on.exit(options(warn = 0))
  # Calcul d'un facteur ayant une modalité pour chaque jour
  dta$Yd <- paste(year(dta$valid), format(dta$valid, "%j"), sep = "-")
  dta$Yd <- as.factor(dta$Yd)
  # Calcul du "vrai" PA en faisant la moyenne par jour
  vraiPA <- tapply(dta$PA, dta$Yd, \(x) mean(x, na.rm = TRUE))
  # Calcul des valeurs journalières dégradées
  meanT <- tapply(dta$tmpc, dta$Yd, \(x) mean(x, na.rm = TRUE))
  minT <- tapply(dta$tmpc, dta$Yd, \(x) min(x, na.rm = TRUE))
  maxT <- tapply(dta$tmpc, dta$Yd, \(x) max(x, na.rm = TRUE))
  meanRH <- tapply(dta$relh, dta$Yd, \(x) mean(x, na.rm = TRUE))
  # Création du tableau des résultats
  comps <- as.data.frame(cbind(vraiPA, meanT, minT, maxT, meanRH))
  # Nettoyage des données manquantes
  comps <- subset(comps, is.finite(vraiPA) & is.finite(meanT) &
    is.finite(minT) & is.finite(maxT) & is.finite(meanRH))
  # Calcul des PA sur données journalières
  comps$JPAmean <- PA(comps$meanT, comps$meanRH)
  comps$JPAmax <- PA(comps$maxT, comps$meanRH)
  comps$JPAmin <- PA(comps$minT, comps$meanRH)
  return(comps)
}
brst.comps <- getcomps(brst)
```

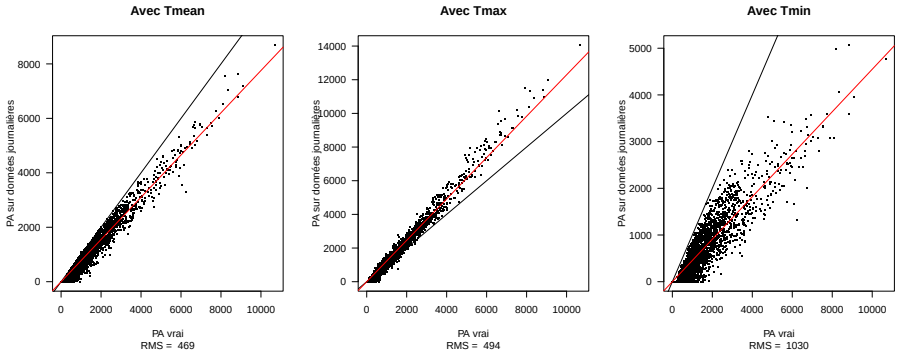
Dans les graphiques suivants la « vraie » valeur de la vitesse de diffusion pollinique est toujours en abscisse et celle reconstituée avec les données journalières en ordonnée. La ligne noire est la première bissectrice et la ligne rouge le modèle de régression linéaire par l'origine. La valeur de `RMS` donne une idée de la qualité de la reconstitution, elle est calculée comme,

$$\mathrm{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - x_i^r)^2}$$

où x représente la vraie valeur, x^r la valeur reconstituée et n le nombre total de points.

```
plotcomps <- function(comps){
  par(mfrow = c(1, 3))
  plotone <- function(qui, ...){
    x <- comps$vraiPA ; y <- comps[ ,qui]
    plot(x, y, las = 1, pch = ".",
         xlab = "PA vrai",
         ylab = "PA sur données journalières", ...)
    abline(c(0, 1))
    abline(lm1 <- lm(y~x - 1), col = "red")
    rms <- sqrt(mean((x-y)^2))
    title(sub = paste("RMS = ", signif(rms,3)))
  }
  plotone("JPAmean", main = "Avec Tmean")
```

```
  plotone("JPAmax", main = "Avec Tmax")
  plotone("JPAmin", main = "Avec Tmin")
}
plotcomps(brst.comps)
```



QUAND on travaille avec la température moyenne on a tendance à sous-estimer la vitesse d'émission pollinique. Il y a de nombreux cas où la valeur reconstituée est nulle alors que la valeur réelle ne l'est pas. Ceci correspond aux cas où la température moyenne est trop basse pour permettre l'émission de pollen mais le pic de température à midi le permet. Le RMS est de l'ordre de 500, ce qui n'est pas négligeable puisque l'écart inter-quartiles va de 435 à 1781. Avec la température maximale on a tendance à sur-estimer la valeur réelle, mais ce biais n'est pas systématique (il ne faut pas oublier que l'information sur l'humidité relative à été dégradée également). Avec la température minimale on retrouve, en pire, la même chose qu'avec la température moyenne.

ON cherche maintenant à ruser en essayant de prédire la valeur réelle à partir des trois valeurs reconstituées à partir des températures journalières moyennes, minimales et maximales. On introduit les variables dans l'ordre présumé d'importance et on force à passer par l'origine.

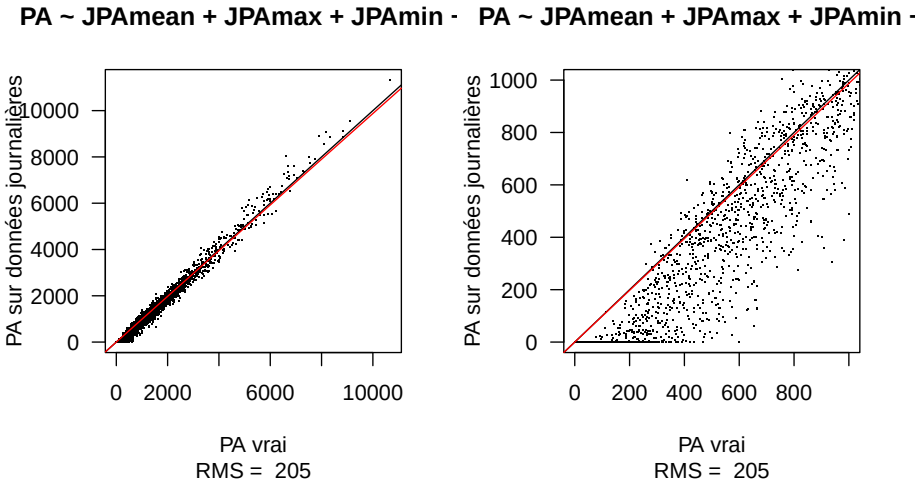
```
summary(lm3 <- lm(vraiPA ~ JPAmean + JPAmax + JPAmin - 1, brst.comps))
Call:
lm(formula = vraiPA ~ JPAmean + JPAmax + JPAmin - 1, data = brst.comps)
Residuals:
     Min       1Q   Median       3Q      Max
-1436.42    -4.90    79.58   189.44   899.77

Coefficients:
        Estimate Std. Error t value Pr(>|t|)
JPAmean  0.58780    0.04250   13.83   <2e-16 ***
JPAmax   0.55233    0.01778   31.07   <2e-16 ***
JPAmin  -0.32385    0.03037  -10.66   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 205 on 2916 degrees of freedom
Multiple R-squared:  0.9874,        Adjusted R-squared:  0.9874
F-statistic: 7.634e+04 on 3 and 2916 DF,  p-value: < 2.2e-16
lm3$coefficients
   JPAmean     JPAmax     JPAmin
 0.5878039  0.5523271 -0.3238526
```

LES trois variables reconstituées sont toutes très significatives dans la régression, mais ce n'est pas très étonnant car la table `brst.comps` comporte 2919 lignes. Les coefficients nous disent qu'il faut faire en gros la moyenne entre les valeurs reconstituées avec la température moyenne et maximale et retrancher un facteur correctif porté par la valeur reconstituée avec la température minimale. Voyons maintenant quelle est la qualité de la reconstitution.

```
plotlm3 <- function(comps, zoom = 1000){
  x <- comps$vraiPA
  lm3 <- lm(vraiPA ~ JPAmean + JPAmax + JPAmin - 1, comps)
  y <- lm3$fitted.values
  myplot <- function(...){
  plot(x, y, las = 1, pch = ".", xlab = "PA vrai",
                ylab = "PA sur données journalières",
     main = "PA ~ JPAmean + JPAmax + JPAmin - 1", ...)
  abline(c(0, 1))
  rms <- sqrt(mean((x-y)^2))
  title(sub = paste("RMS = ", signif(rms,3)))
  abline(lm(y~x - 1), col = "red")
  }
  par(mfrow = c(1, 2))
  myplot() ; myplot(xlim = c(0, zoom), ylim = c(0, zoom))
}
plotlm3(brst.comps)
```



GLOBALEMENT on n'a plus de biais systématique, on a bien amélioré la qualité de la prédiction avec un RMS de l'ordre de 200. Le zoom à droite montre qu'il reste toujours quelques cas rétifs où la valeur reconstituée est nulle quand la vraie valeur peut aller jusqu'à 600. On a tendance à sous-estimer les faibles valeurs et sur-estimer les fortes valeurs, mais dans l'ensemble ce n'est pas si mal.

3.6 Aéroport Strasbourg

```
strasbourg <- riem_measures(
  station = "LFST",
  date_start = "1931-01-01",
  date_end = "2020-10-05")
strasbourg <- as.data.frame(strasbourg)
comment(strasbourg) <- paste("Téléchargé le :", Sys.time())
save(strasbourg, file = "data/strasbourg.Rda")
```

```
load(url(paste0(chmin, "strasbourg.Rda")))
comment(strasbourg)
```

```
[1] "Téléchargé le : 2023-04-14 11:38:54"
```

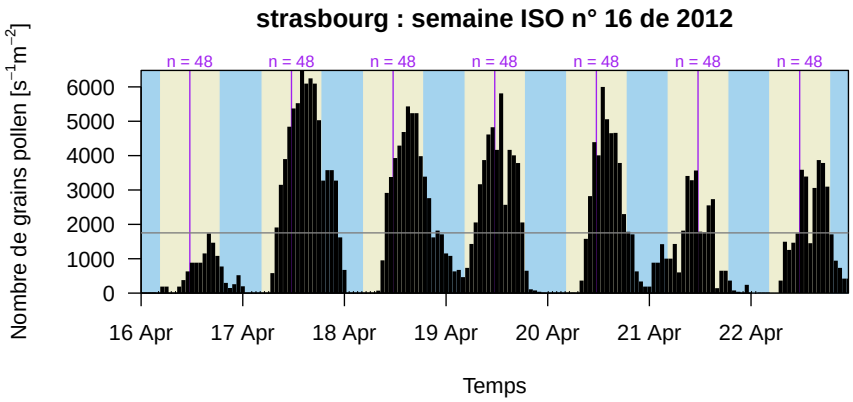
```
stras <- selectyears(strasbourg)
unique(year(stras$valid))
```

```
 [1] 1977 1978 1979 1981 1982 1983 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
[17] 2015 2016 2017 2018 2019
```

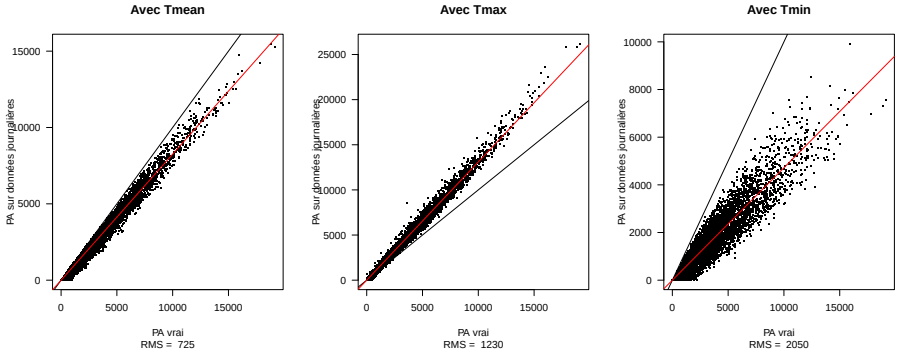
```
stras.comps <- getcomps(stras)
lm3 <- lm(vraiPA ~ JPAmean + JPAmax + JPAmin - 1, stras.comps)
lm3$coefficients
```

```
  JPAmean     JPAmax     JPAmin
0.6990230  0.4075661 -0.2427087
```

```
plotPAw(strasbourg, 2012, 16)
```

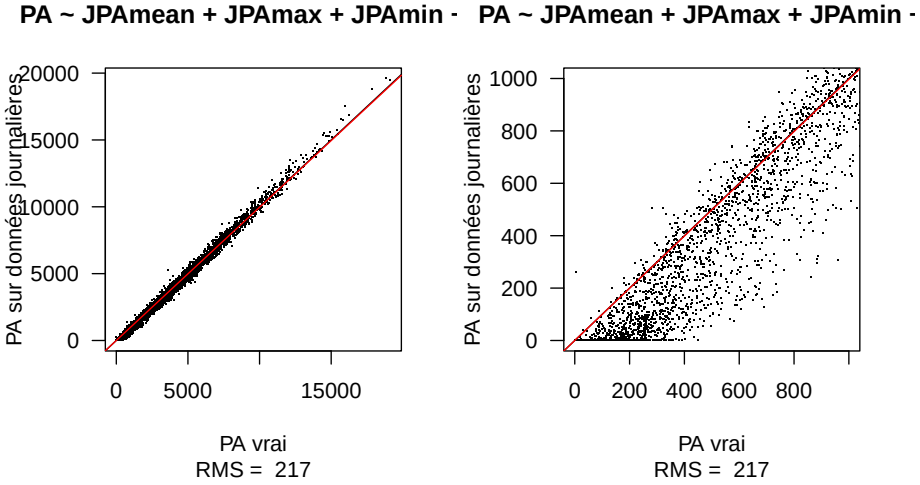



```
plotcomps(stras.comps)
```



PAR rapport à Brest, on a le même résultat mais avec des RMS plus importants. On avait raison de se méfier puisque l'on voit que les coefficients de la régression ne sont pas stables.

```
plotlm3(stras.comps)
```



On a toujours un RMS de l'ordre de 200, donc du point de vue de la qualité de la reconstitution c'est assez stable.

3.7 Aéroport de Lyon

```
lyon <- riem_measures(
  station = "LFLL",
  date_start = "1931-01-01",
  date_end = "2020-10-05")
lyon <- as.data.frame(lyon)
comment(lyon) <- paste("Téléchargé le :", Sys.time())
save(lyon, file = "data/lyon.Rda")
```

```
load(url(paste0(chmin, "lyon.Rda")))
comment(lyon)
```

```
[1] "Téléchargé le : 2023-04-14 12:21:57"
```

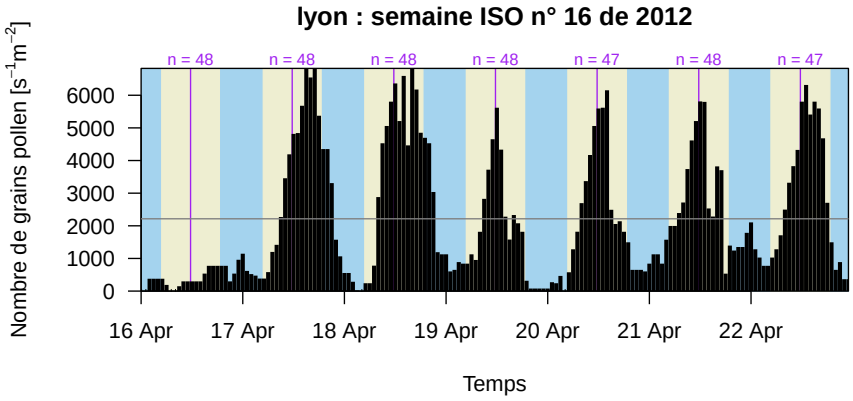
```
ly <- selectyears(lyon)
unique(year(ly$valid))
```

```
 [1] 1976 1977 1978 1979 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992
[17] 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016
[33] 2017 2018 2019
```

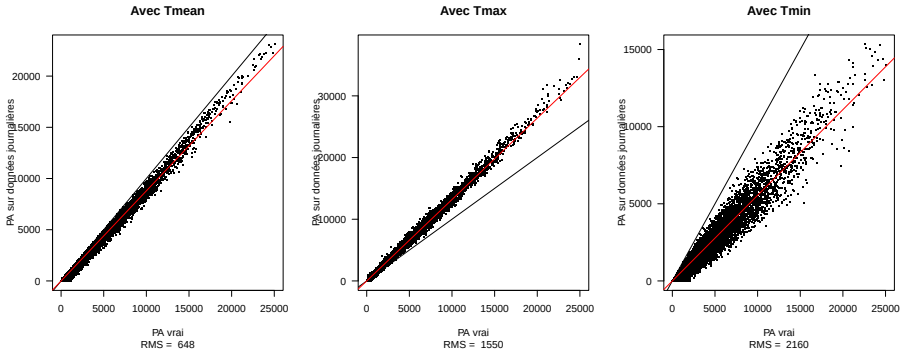
```
ly.comps <- getcomps(ly)
lm3 <- lm(vraiPA ~ JPAmean + JPAmax + JPAmin - 1, ly.comps)
lm3$coefficients
```

```
  JPAmean     JPAmax     JPAmin
0.6829311  0.3857539 -0.2017926
```

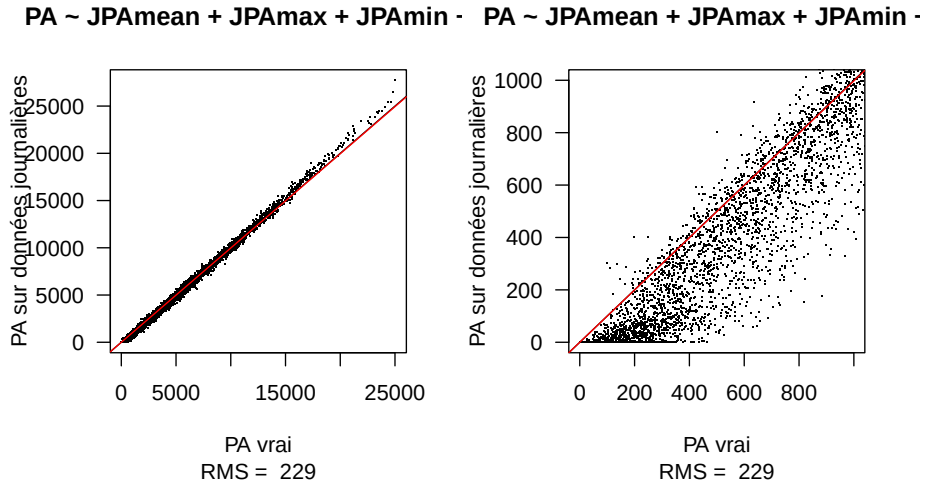
```
plotPAw(lyon, 2012, 16)
```




```
plotcomps(ly.comps)
```




```
plotlm3(ly.comps)
```



3.8 Aéroport de Lille

```
lille <- riem_measures(
  station = "LFQQ",
  date_start = "1945-02-28",
  date_end = "2020-10-05")
lille <- as.data.frame(lille)
comment(lille) <- paste("Téléchargé le :", Sys.time())
save(lille, file = "data/lille.Rda")
```

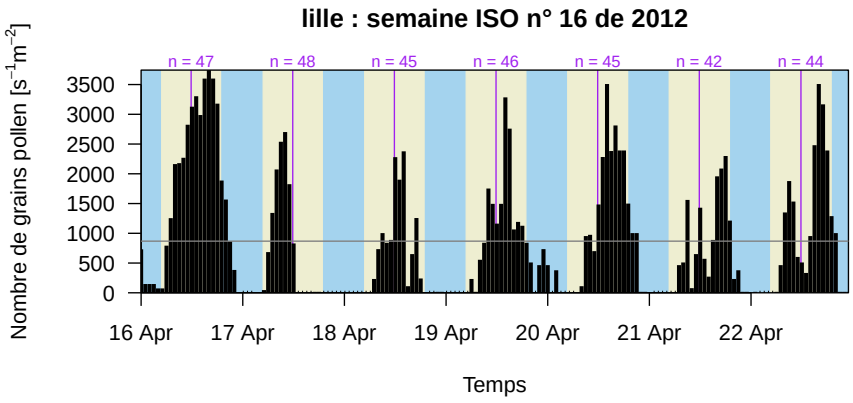
```
load(url(paste0(chmin, "lille.Rda")))
li <- selectyears(lille)
unique(year(li$valid))
```

```
 [1] 1979 1981 1982 1983 1984 1985 1986 2007 2008 2009 2010 2011 2012 2013 2014 2015
[17] 2016 2017 2018 2019
```

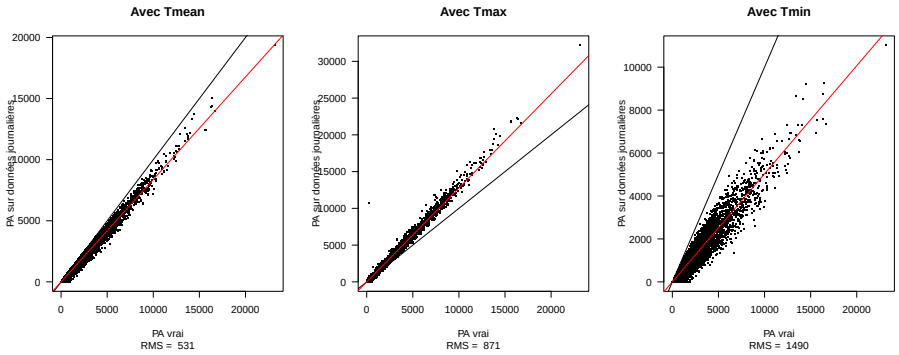
```
li.comps <- getcomps(li)
lm3 <- lm(vraiPA ~ JPAmean + JPAmax + JPAmin - 1, li.comps)
lm3$coefficients
```

```
  JPAmean     JPAmax     JPAmin
0.8655452  0.3565112 -0.3703407
```

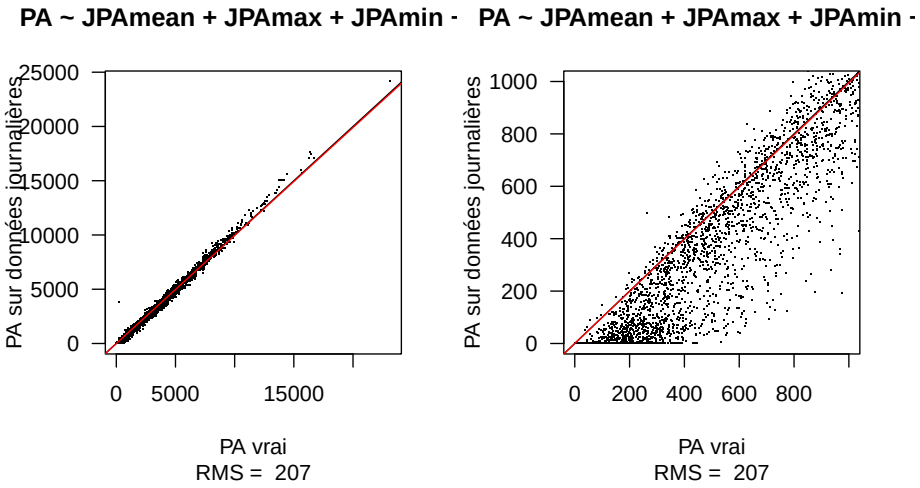
```
plotPAw(lille, 2012, 16)
```




```
plotcomps(li.comps)
```




```
plotlm3(li.comps)
```



3.9 Aéroport de Bordeaux

```
bordeaux <- riem_measures(
  station = "LFBD",
  date_start = "1931-01-01",
  date_end = "2020-10-05")
bordeaux <- as.data.frame(bordeaux)
comment(bordeaux) <- paste("Téléchargé le :", Sys.time())
save(bordeaux, file = "data/bordeaux.Rda")
```

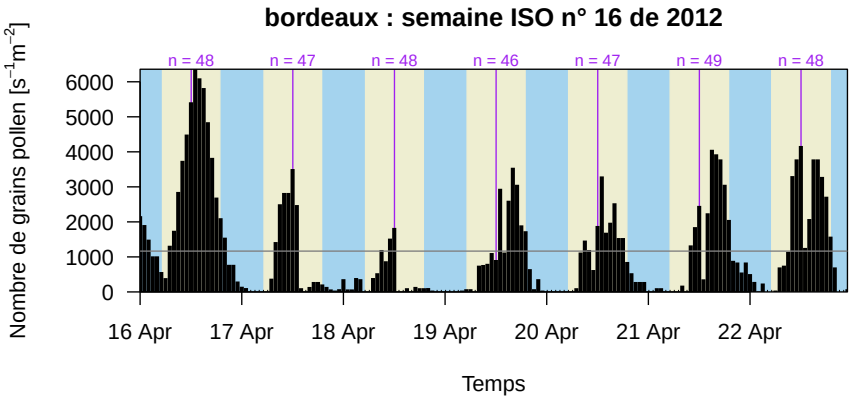
```
load(url(paste0(chmin, "bordeaux.Rda")))
bor <- selectyears(bordeaux)
unique(year(bor$valid))
```

```
 [1] 1975 1977 1978 1979 1981 1982 1983 1984 1985 1986 1987 1988 1990 1991 1992 2001
[17] 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
[33] 2018 2019
```

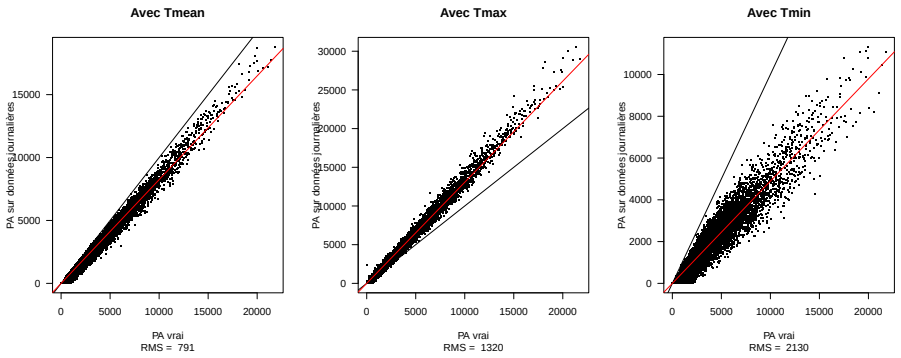
```
bor.comps <- getcomps(bor)
lm3 <- lm(vraiPA ~ JPAmean + JPAmax + JPAmin - 1, bor.comps)
lm3$coefficients
```

```
  JPAmean     JPAmax     JPAmin
0.7210921  0.4153611 -0.2904088
```

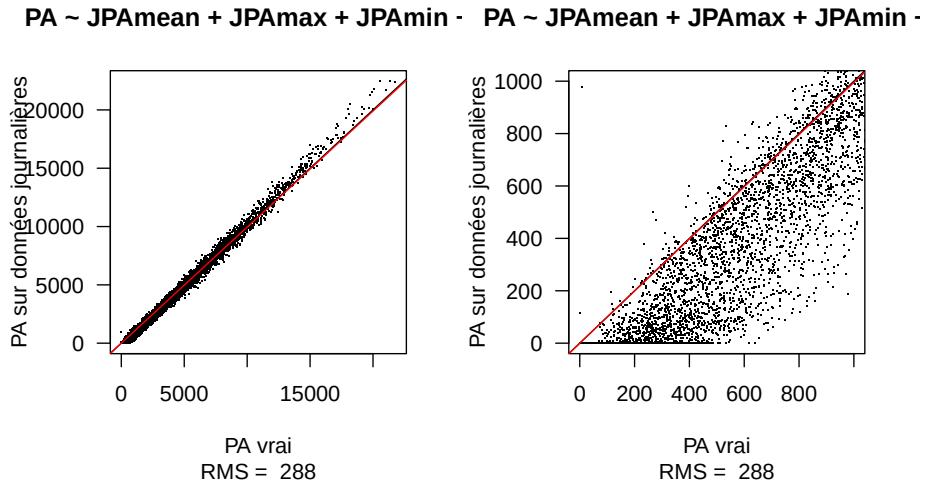
```
plotPAw(bordeaux, 2012, 16)
```



```
plotcomps(bor.comps)
```




```
plotlm3(bor.comps)
```



3.10 Aéroport de Nice

```
nice <- riem_measures(
  station = "LFMN",
  date_start = "1945-09-30",
  date_end = "2020-10-05")
nice <- as.data.frame(nice)
comment(nice) <- paste("Téléchargé le :", Sys.time())
save(nice, file = "data/nice.Rda")
```

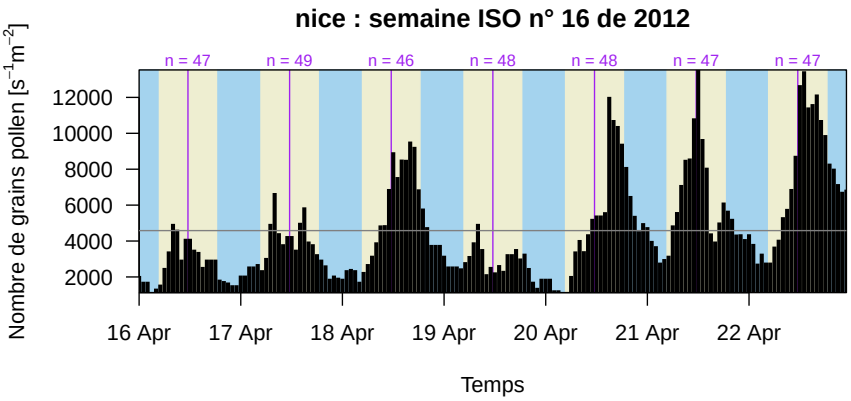
```
load(url(paste0(chmin, "nice.Rda")))
nic <- selectyears(nice)
unique(year(nic$valid))
```

```
 [1] 1977 1978 1979 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 2001
[17] 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
[33] 2018 2019
```

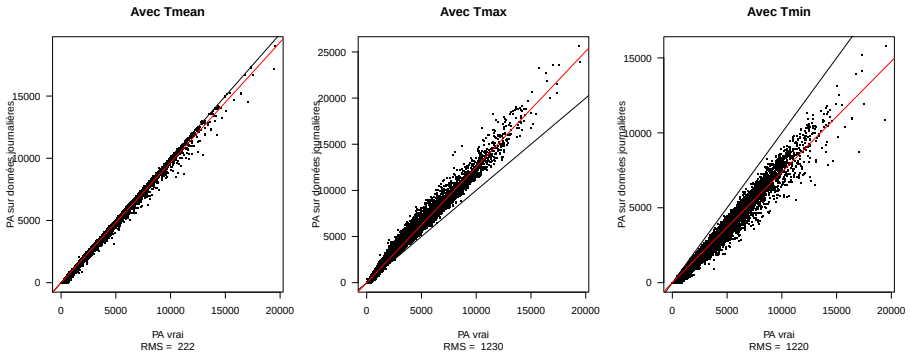
```
nic.comps <- getcomps(nic)
lm3 <- lm(vraiPA ~ JPAmean + JPAmax + JPAmin - 1, nic.comps)
lm3$coefficients
```

```
  JPAmean     JPAmax     JPAmin
1.0141135  0.1249190 -0.1861578
```

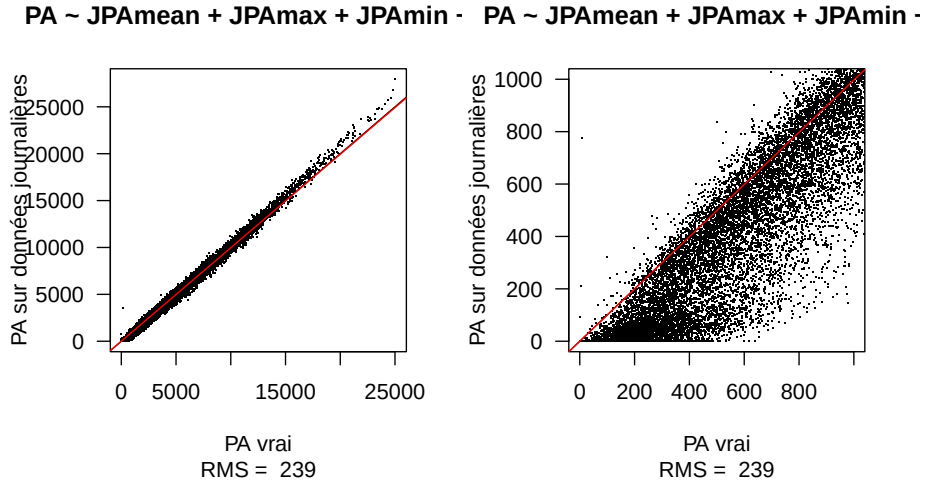
```
plotPAw(nice, 2012, 16)
```




```
plotcomps(nic.comps)
```




```
plotlm3(nic.comps)
```



3.11 Concaténation des aéroports

L'IDÉE ici est de combiner les données des aéroports en espérant avoir une bonne couverture des conditions météorologiques possibles en France métropolitaine.

```
all.tabs <- ls(pattern ="\\.comps")
all.tabs
```

```
[1] "bor.comps"   "brst.comps"  "li.comps"    "ly.comps"    "nic.comps"
[6] "stras.comps"
```

```
conc.comps <- do.call(rbind, lapply(all.tabs, get))
nrow(conc.comps)
```

```
[1] 55313
```

```
(coefs <- lm(vraiPA ~ JPAmean + JPAmax + JPAmin - 1, conc.comps)$coefficients)
```

```
   JPAmean     JPAmax     JPAmin
0.8590268  0.3294001 -0.3229303
```

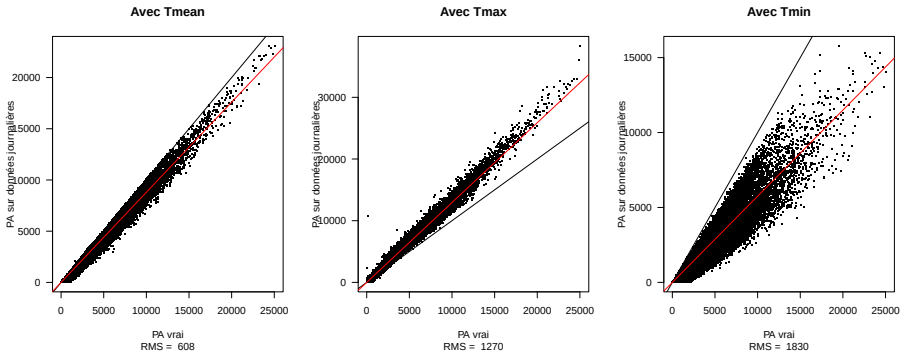
```
dput(coefs)
```

```
c(JPAmean = 0.859026811227271, JPAmax = 0.329400114478291, JPAmin = -0.322930252697188
)
```

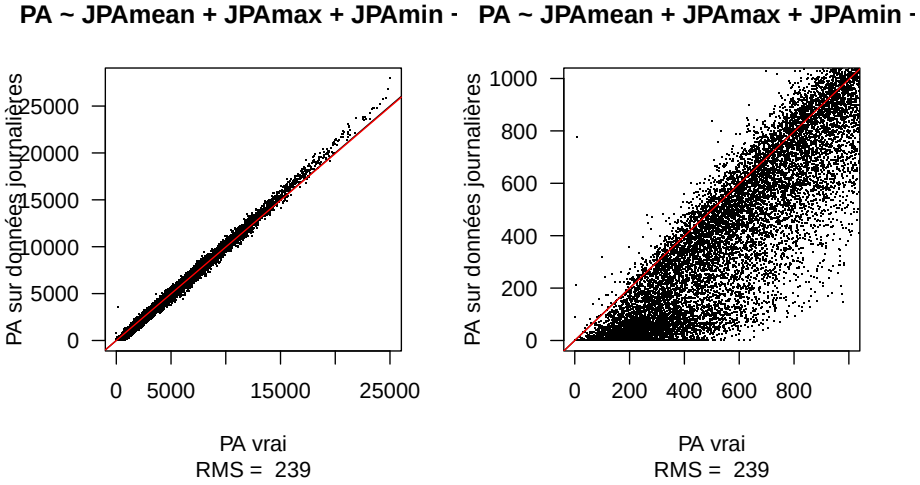
```
comment(coefs) <- paste("Calculé le :", Sys.time())
# Sauvegarde des coefficients et des fonctions utilitaires
# utiles pour calculer les données reconstituées
save(coefs, esat, ecurr, VPD, PA, F2C, file = "data/coefs.Rda")
```

ON pourra noter ici que les valeurs absolue des coefficients devant `JPAmax` et `JPAmin` sont très proches, notre modèle s'écrit quasiment `0.86*JPAmean + 0.32*(JPAmax - JPAmin)`.

```
plotcomps(conc.comps)
```




```
plotlm3(conc.comps)
```



Nous avons donc un RMS de l'ordre de 200, ce qui n'est pas mauvais, même s'il restera toujours des valeurs difficiles à reconstituer pour les valeurs les plus faibles.

Références

[1] P. Cwiek. *pmetar : Processing METAR Weather Reports*, 2022. R package version 0.4.0.

[2] G. Magnus. Versuche über die Spannkräfte des Wasserdampfs. *Annalen der Physik*, 137(2) :225–247, 1844.

[3] V.L. Nadolski. Automated Surface Observing System (ASOS) User's guide. Technical report, National Oceanic and Atmospheric Administration and Department of Defense and Federal Aviation Administration and United States Navy from the United States of America, 1998.

[4] M. Salmon. *riem : Accesses Weather Data from the Iowa Environment Mesonet*, 2022. R package version 0.3.0.

[5] S. Schueler and K.H. Schlünzen. Modeling of oak pollen dispersal on the landscape level with a mesoscale atmospheric model. *Environmental Modeling & Assessment*, 11(3) :179–194, 2006.

[6] C.M. Shun, I. Lisk, C. McLeod, and K.L. Johnston. Meteorological services to aviation. *WMO Bulletin*, 58(2) :94–103, 2009.

[7] Sun Microsystems. XDR : external data representation standard. RFC 1014. Technical report, Network Working Group, 1987.

[8] B. Thieurmel and A. Elmarhraoui. *suncalc : Compute Sun Position, Sunlight Phases, Moon Position and Lunar Phase*, 2022. R package version 0.5.1.