Projet d'Informatique III : CY-Météo

Filières : PréING2 • 2022-2023

Auteurs : Mehdi TORJMEN – Moutie ZAHREZ

Encadrant : M. Romuald GRIGNON

Remerciements

Nous tenons tout d'abord à remercier et à faire preuve de reconnaissance envers ces personnes, qui nous ont non seulement permis de prendre connaissance de ce projet, mais qui nous ont aussi encadré et aidé pour sa réalisation :

- M. Romuald GRIGNON, responsable et chargé de Travaux Dirigés du groupe 2 dont nous faisons partie en deuxième année du cycle pré-ingénieur (PréING2) en filière Mathématiques et Informatique (MI) à CY Tech en 2022-2023. Ses conseils précis, sa sympathie et sa patience lors des TD nous ont été d'une grande aide quant à la résolution de plusieurs difficultés conceptuelles et techniques.
- Mme. Eva ANSERMIN et M. Renaud VERIN, qui par leur expertise et leur travail nous ont permis avec M. Romuald GRIGNON de prendre connaissance et de réaliser le projet CY-Météo.
- CY Tech et la totalité des enseignants et travailleurs au sein de l'école qui par leur coopération, rigueur et assiduité, nous ont permis tout au long de la réalisation de ce projet d'avoir accès à un environnement académique riche, plein de savoir-faire et d'expertise.

0 – Introduction

Dans le cadre de notre seconde année du cycle pré-ingénieur (PréING2) en filière Mathématiques et Informatique (MI) à CY Tech, il nous est suggéré de réaliser un projet dans la matière Informatique III entre le 9 décembre 2022 (09/12/2022) et le 3 février 2023 (03/02/2023). Ses intérêts pédagogiques cardinaux sont :

- D'encourager et fortifier notre capacité à travailler en équipe, à répartir les tâches au sein d'un groupe et à établir un planning
- D'affermir notre compétence à gérer un projet sur une durée limitée avec différentes contraintes
- De développer notre aptitude à analyser et à prévoir conceptuellement un problème
- D'améliorer notre technique et notre maitrise des différents types d'algorithmes de tri en langage C, en passant par des structures telle que des tableaux, des listes chaînées, des arbres binaires de recherche (ABR) et des AVL
- De nous inciter à approfondir en binôme notre maitrise des commandes UNIX, du Script Shell et du Bourne-Again shell (Bash) via de la recherche et de la pratique
- De nous introduire au logiciel interactif en ligne de commande Gnuplot

Ayants une optique de vie professionnelle dans le monde de l'ingénierie informatique, c'est avec joie que notre groupe composé de Mehdi TORJMEN et de Moutie ZAHREZ, s'est appuyé sur une telle occasion pleine d'intérêts pédagogiques afin de réaliser un projet collectif et le soumettre avec ce rapport à notre responsable d'Informatique III M. Romuald GRIGNON.

Ce rapport de projet a pour but de synthétiser le déroulement de la réalisation de CY-Météo en incluant l'essentiel de l'ensemble des éléments de notre travail.

Dans une première partie, nous aborderons les différents points qui ont permis d'établir une problématique du sujet. Nous étudierons d'abord son contexte, ses motivations, ses enjeux, puis notre analyse conceptuelle et technique du sujet par un découpage en modules, une description des workflows et un choix d'implémentation et de modularité, en déduisant les contraintes et délais qu'il pose.

Dans une seconde partie, nous décrirons la gestion conceptuelle du projet que nous avons suivi, en allant de la mise en place de la méthode adoptée en Agile Scrum, puis en passant par l'établissement du planning avec une répartition des tâches via un backlog.

Dans une troisième partie, nous exposerons la résolution technique du projet, à partir de sa stratégie, ses premiers pas, les difficultés et obstacles conceptuels, techniques ou logistiques rencontrés jusqu'aux différentes solutions apportées.

Dans une quatrième partie, nous expliquerons l'originalité de notre rendu final, que ça soit du point de vue conceptuel au niveau d'un Script Shell générique et modulaire ou du point de vue technique avec différents paramètres optionnels qui enrichissent le rendu du programme (barre mouvante, multiplot, debug etc…).

Dans une cinquième partie, nous afficherons plusieurs tests et résultats de démonstration sur Gnuplot à partir de certaines données dans le Script Shell.

I – Problématique du sujet

1 – Contexte

Comment être apte à exploiter un fichier Excel au format .csv contenant tant de lignes et de données qu'il est impossible de l'ouvrir sur son logiciel même ? Dans un contexte météorologique où une très grande quantité d'informations sont récoltées sur des caractéristiques diverses telles que la température, la pression, le vent, l'altitude ou encore l'humidité, sur une soixantaine de stations dans le monde au cours d'une plage temporelle au voisinage de la décennie, une telle question vient à être posée.

En effet, l'humanité dispose aujourd'hui d'une grande quantité de stations météorologiques autour du monde. La France par exemple, en possède dans six limitations géographiques : en France Métropolitaine + Corse (caractérisée par le filtre -F), en Guyane française (caractérisée par le filtre -G), à Saint-Pierre et Miquelon (caractérisée par le filtre -S), aux Antilles (caractérisées par le filtre -A), dans l'Océan Indien (caractérisé par le filtre -O) et en Antarctique (caractérisé par le filtre -Q). Ces stations disposent d'équipements et de dispositifs de mesure tels qu'ils peuvent atteindre une précision problématique à stocker dans la plage de valeurs acceptée de certains types (les entiers en langage C par exemple).

C'est ainsi qu'en regroupant de manière désordonnée l'ensemble des données qui ont été recueillies par toutes les stations de ces six limitations géographiques sur plus d'une décennie (2010-2020), nous nous retrouvons avec le fichier meteo_filtered_data_v1.csv qui a été fourni pour la réalisation de CY-Météo.

2 – Motivation

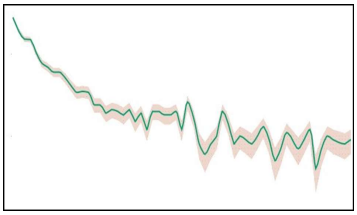
À partir du problème posé par le contexte du sujet, la motivation principale est de créer un programme capable de traiter le fichier meteo_filtered_data_v1.csv afin d'en créer plusieurs graphiques via Gnuplot selon les arguments entrés par l'utilisateur :

i. Température et pression :
   - Version 1 : barres d'erreurs
   - Version 2 : ligne simple
   - Version 3 : multi-lignes

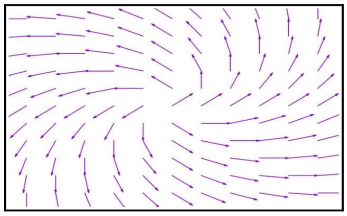
ii. Vent : vecteurs

iii. Altitude et humidité : carte interpolée et colorée

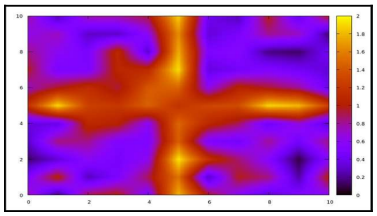
Exemples illustratifs tirés du sujet de CY-Météo :



Graphique de type 'barres d'erreurs'



Graphique de type 'vecteurs' (carte des vents



Graphique de type 'carte interpolée' (altitude, humidité)

Pour ce faire, il est expliqué qu'il faudra passer par une phase d'analyse/contrôle des arguments entrés, de filtrage et d'écriture des fichiers en fonction des besoins de l'utilisateur en Script Shell. Ensuite, un appel itératif d'un programme C doit se faire, qui effectuera un tri du tableau filtré selon un choix entré en argument dans le Shell parmi trois méthodes (ci-dessous listés par ordre décroissant de complexité temporelle en recherche/insertion/suppression) :

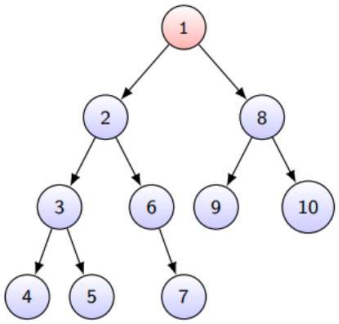
- Un tri via l'utilisation d'un tableau ou d'une liste chainée.

Tableau standard :

Liste chainée :

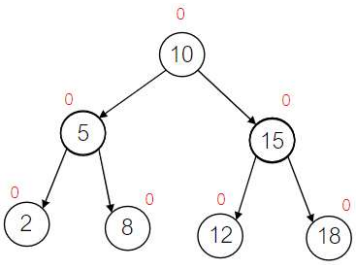
- Un tri via l'utilisation d'un arbre binaire de recherche (ABR). Un ABR est un arbre binaire dans lequel chaque nœud possède une valeur telle que tout nœud du sous-arbre gauche possède une valeur inférieure à celle du nœud visé et que tout nœud du sous-arbre droit possède une valeur supérieure à celle du nœud visé.

Arbre binaire de recherche :



- Un tri via l'utilisation d'un AVL. Un AVL, dénomination venant de ses créateurs Georgii Adelson-Velsky et Evgueni Landis, est un arbre binaire de recherche automatiquement équilibré en termes de hauteur. Le facteur d'équilibre de chaque nœud (indexé par un nombre rouge ci-dessous) est calculé comme la soustraction de la hauteur du sous-arbre droit du nœud considéré avec celle du sous arbre-gauche.

AVL :



Une fois que le programme C fournit tous les fichiers triés selon les arguments de l'utilisateur, le Script Shell récupère son résultat afin d'appeler un Gnuplot, un logiciel interactif en ligne de commande permettant d'afficher des graphiques.

3 – Enjeux

Ce projet présente des enjeux de différents types : technologiques, pédagogiques mais aussi humains.

En effet, que de permettre à des utilisateurs selon leurs besoins d'obtenir des graphiques et des fichiers filtrés puis triés en un temps cours à partir d'une grande quantité de donnée difficile à exploiter, constitue un enjeu technologique cardinal. Comme expliqué en partie I-1 (Contexte), un domaine tel que la météorologie se voit aujourd'hui à devoir mesurer puis exploiter des mesures sur des caractéristiques variées (température, pression, vent, altitude, humidité…) avec une précision croissante dans de nombreuses stations autour du monde au cours de plages temporelles dépassant la décennie. Il est donc primordial, afin d'être apte à exploiter ces données par exemple dans une optique prédictive, de pouvoir les filtrer, les trier, ainsi que de les représenter via différents graphiques. Cet enjeu est d'autant plus marqué qu'il est possible de créer un programme générique, modulaire et évolutif, capable de prendre en compte de nouveaux contextes et caractéristiques mesurées avec un minimum d'intervention sur les sources voire aucune (no code).

De plus, les enjeux pédagogiques, comme préalablement décrits en partie 0 (Introduction), sont nombreux et féconds. Premièrement, afin de créer un Shell permettant de contrôler les arguments entrés par l'utilisateur, en déduire le filtrage du fichier meteo_filtered_data_v1.csv, appeler itérativement un programme de tri en langage C et finalement en venir à un usage de Gnuplot pour afficher un graphique, il est nécessaire d'acquérir une maitrise et une capacité de jongler sur trois environnements complémentaires à la fois (Script Shell (avec la complexité de la commande awk), C et Gnuplot), au-delà de ce qui est présenté dans le cours. C'est donc notre capacité de recherche, notre autonomie et notre aptitude d'apprentissage qui sont ici mises en exercice. Deuxièmement, afin de ne pas se perdre dans la gestion conceptuelle de CY-Météo, il nous est précieux d'acquérir une vision d'ensemble permettant de mener à bien la résolution du projet avec une analyse conceptuelle et technique approfondie du sujet et un planning rigoureux.

Finalement, afin de maintenir un environnement collaboratif de travail, il est fondamental d'être apte à faire preuve de cohésion d'équipe. Même s'il est assez connu que dans chaque groupe tel qu'un binôme ou un trinôme, il y'a une probabilité non-négligeable de déséquilibre entre les taux d'investissement de chacun des membres quant à la réalisation d'un projet, il est important d'être apte à remédier à ce problème en communiquant et en motivant son prochain à l'apprentissage et au travail d'équipe. Ainsi, la réalisation de CY-Météo pose des enjeux humains essentiels pour la vie future, qu'elle soit professionnelle ou plus largement sociale.

4 – Analyse Conceptuelle et technique du sujet

a – Modules

Tel que spécifié dans le cahier des charges de CY-Météo, le projet se divise globalement en trois modules :

- Un module pour le Script Shell comme lieu de contrôle, de filtrage et d'appels
- Un module pour le C s'occupant du tri du fichier filtré envoyé par le Shell sous 3 méthodes (AVL, ABR, TAB),

- Un module pour Gnuplot, qui étant appelé par le Shell, reçoit un fichier trié pour afficher divers graphiques (barres d'erreurs, ligne simple, multi-lignes, vecteurs et carte interpolée et colorée)

Ces trois modules doivent être en interaction, avec pour cœur le Script Shell qui doit appeler le programme C ainsi que Gnuplot.

b – Description du workflow

Afin d'acquérir une vision d'ensemble du sujet et de clarifier le chemin que nous allions suivre, nous avons établi un workflow (ou flux de travaux), représentation graphique du séquencement des process réalisés afin de parvenir à un but donné.

Ayants écrit la totalité de notre projet en anglais, le workflow que nous avons construit en a de ce fait respecté cette norme.

Légende :

Start : représente le début du programme

Stop : représente la fin du programme

Get Arguments : (Shell) phase du programme qui récupère les arguments entrés par l'utilisateur en utilisant l'option getopts du Bash

Check Arguments : (Shell) phase du programme qui vérifie que les arguments entrés par l'utilisateur respectent le cahier des charges, entre autres pour vérifier les paramètres obligatoires, les paramètres exclusifs et la cohérence globale de la requête.

Scope/Context : (Shell) un type de donnée sur laquelle l'on peut trier. Typiquement, t1/p1 (Température/Pression en mode 1), t2/p2, t3/p3, w (Vent), h (altitude) ou encore m (humidité). Chaque scope est associé à une key (clé) et une value (valeur). Un programme générique et modulaire, par le biais d'une définition des scopes (ScopeDef), permet d'insérer de nouveaux scopes avec quasiment du no-code.

Process Each Context : (Shell) boucle dans laquelle un ensemble de traitements est appliqué pour chaque contexte selon la requête de l'utilisateur

Prepare Data : (Shell) phase du programme dans laquelle un ensemble de traitements est effectué pour préparer les éléments nécessaires au filtrage et au tri. Cette préparation se fait uniquement en fonction de la définition du scope/context. De ce fait, les requêtes de filtrage et de tri sont génériques et indépendantes des requêtes spécifiques telles que définies dans le cahier des charges.

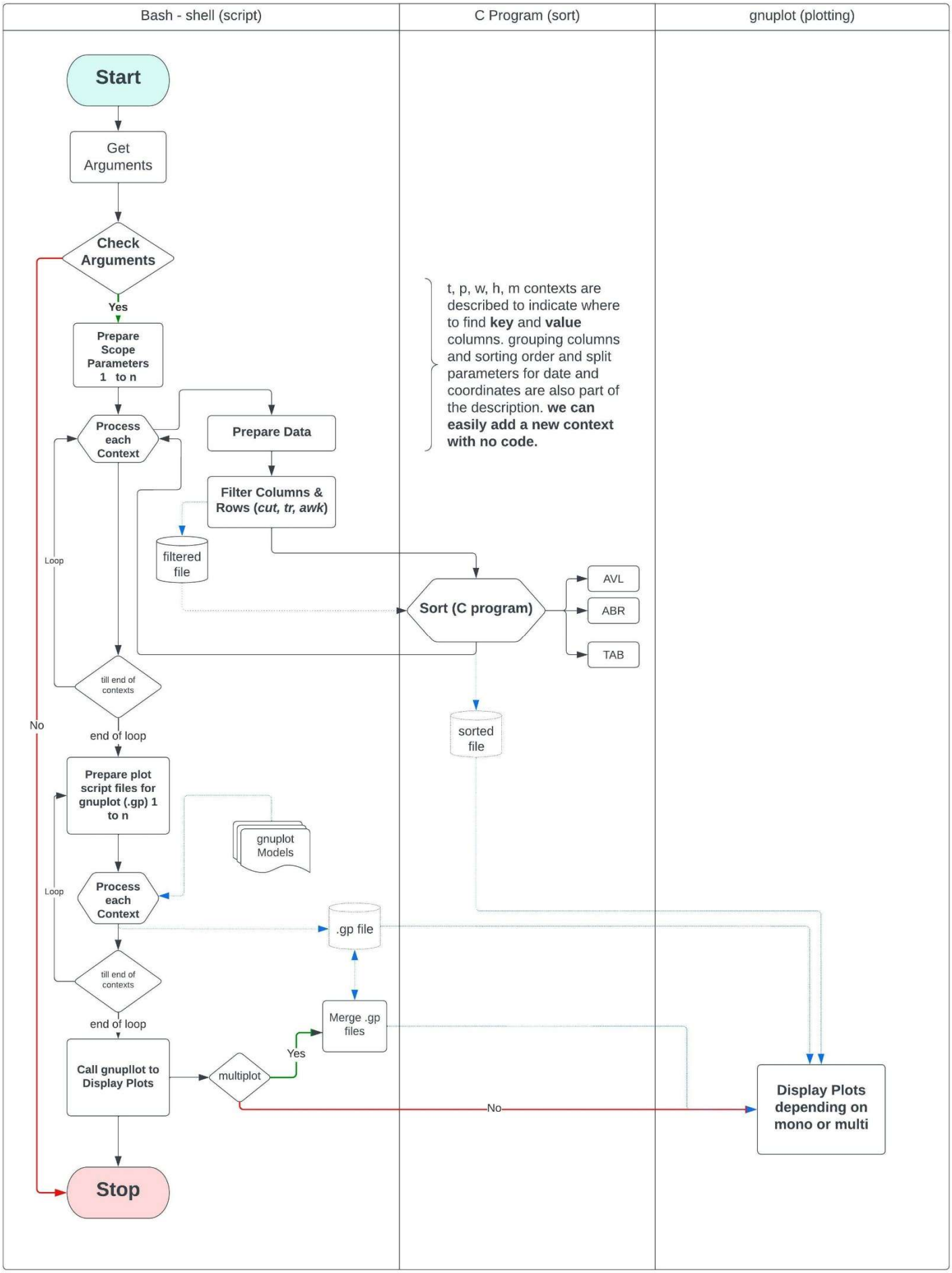
Filter Columns & Rows : (Shell) phase du filtrage dans laquelle le fichier d'entrée est réduit en termes de colonnes (cut), éventuellement transformé (conversion de la date ISO en deux colonnes date et heure, coordonnées en longitude et latitude, coordonnées de vecteurs d'angulaire à cartésien). Les filtres choisis par l'utilisateur sont pris en considération pour filtrer les Rows (lignes). Par exemple : plage de date, plage de coordonnées géographiques, lieux.

Sort : (C) phase de tri et de grouping, dans le programme C, appelé par le Shell, qui selon l'argument entré par l'utilisateur (parmi --tab, --abr ou --avl), effectue dans le main (*csort*) un tri sur les keys et values avec si nécessaires le maximum, le minimum et la moyenne, en utilisant les structures et fonctions de la méthode entrée en argument.

Monoplot : affichage des graphiques en différentes fenêtres pour chaque scope entré en argument. Il s'agit de l'affichage par défaut dans une requête.

Multiplot : option supplémentaire d'affichage « tout en un » de Display Plots, c'est-à-dire dans laquelle tous les graphiques sont affichés dans une même fenêtre (obtenable en ajoutant --multiplot dans la requête)

Display Plots : phase d'affichage des graphiques sur Gnuplot où chaque scope est associé à une PlotDef (type de graphique, titre...) dans un ensemble de définitions donnant généricité et multiplicité au programme


Bash - shell (script)
C Program (sort)
gnuplot (plotting)
Start
Get Arguments
Check Arguments
Yes
Prepare Scope Parameters 1 to n
Process each Context
Prepare Data
Filter Columns & Rows (cut, tr, awk)
filtered file
Loop
t, p, w, h, m contexts are described to indicate where to find key and value columns. grouping columns and sorting order and split parameters for date and coordinates are also part of the description. we can easily add a new context with no code.
Sort (C program)
AVL
ABR
TAB
sorted file
till end of contexts
No
end of loop
Prepare plot script files for gnuplot (.gp) 1 to n
gnuplot Models
Process each Context
Loop
.gp file
till end of contexts
end of loop
Merge .gp files
Yes
Call gnuplot to Display Plots
multiplot
No
Display Plots depending on mono or multi
Stop

c – Choix d'implémentation et modularité

Le premier choix évident et imposé dès le cahier des charges auquel nous avons été confrontés est celui de la langue : entièrement en anglais ou entièrement en français. Sachant que la programmation est un milieu très fortement imprégné de la langue anglaise et d'autant plus dans le monde professionnel qu'académique, étant compréhensible par une grande partie de l'humanité, nous avons pensé pertinent de nous éloigner de notre zone de confort et d'opter pour un projet et une documentation totalement écrits en anglais, sauf pour ce rapport de projet qui doit être écrit en français.

De plus, après avoir établi le workflow du projet, deux possibilités s'offraient à nous :

1. Une possibilité plus aisée de codage dur (hard coding) où le code source est quasiment entièrement dépendant des requêtes spécifiques telles que définies dans le cahier des charges (sujet de CY-Météo) et donc très peu adaptable à l'introduction de nouveaux contextes ou de nouveaux plots et redondant, ce qui implique une maintenance ardue
2. Une possibilité plus ambitieuse de soft coding où le programme est indépendant des requêtes spécifiques telles que définies dans le cahier des charges, c'est-à-dire générique et évolutif, ainsi facilement maintenable et adaptable (quasiment en no code) à l'introduction de nouveaux contextes ou de nouveaux plots.

Dans la continuité de notre volonté de sortir de notre zone de confort, nous avons opté pour la deuxième possibilité.

Il en va de même pour la gestion des erreurs, des messages et du mode debug. Ces techniques ont été adoptées et mises en place pour les mêmes raisons.

Les nomenclatures des variables, des fonctions, des procédures et les commentaires ont été normalisées pour une meilleure lisibilité et une compréhension et maintenance plus confortable des code-sources.

Concernant la structure de Node (Chainon) et de Tree (Arbre), nous avons décidé, par soucis de performance, de lisibilité, de généricité et de modularité d'opter non pas pour une longue structure de vecteurs gourmands en mémoire et dupliqués autant de fois qu'il y de scopes mais plutôt pour une structure contenant un pointeur vers un vecteur de champs alloué dynamiquement en fonction du nombre strictement nécessaire pour chaque scope. Les statistiques (Max, Min, Average) ont également été gérées dans un vecteur de taille dynamique selon le scope (StatsDef).

5 – Contraintes et délais

La contrainte la plus directe à laquelle nous avons été soumis est celle du temps. Nous ne disposions que d'un peu moins de 2 mois pour achever le projet dans sa totalité. Compte tenu du fait qu'il était important que nous nous préparions aux autres examens auxquels nous avons été soumis lors de la dernière semaine de janvier, le temps raisonnable que nous pouvions accorder à la résolution du projet s'en est trouvé réduit. Il fallait donc travailler efficacement, de telle manière à maximiser la productivité en un minimum de temps.

La deuxième contrainte à laquelle nous avons été confrontés concerne la performance changeante de notre programme selon les machines sur lesquelles il est compilé. Il nous arrivait par exemple de constater un rapport de 2 (ou 1/2) entre les performances temporelles de nos propres machines, du fait de la différence de puissance de leur CPU. Nous avions donc parfois du mal à nous situer en termes de performance.

La troisième contrainte à laquelle nous avons été confrontés est celle de la distance. En effet, lors des vacances d'hiver, notre binôme s'est éloigné des environs de CY Tech et ne pouvait donc se rencontrer matériellement pendant une vingtaine de jours. Il fallait donc être apte à gérer un travail d'équipe à distance, par des appels vidéo ou par messagerie instantanée.

La quatrième contrainte que nous avons rencontrée est celle de l'insuffisance de la compréhension cours et de la réalisation des TD pour venir à bout du projet. En effet, que cela soit en Shell, en C ou encore pour Gnuplot, il était nécessaire de sortir de sa zone de confort en pratiquant et en effectuant des recherches sur Internet ou dans des livres afin d'acquérir de la connaissance importante pour l'écriture du code.

II – Gestion du projet

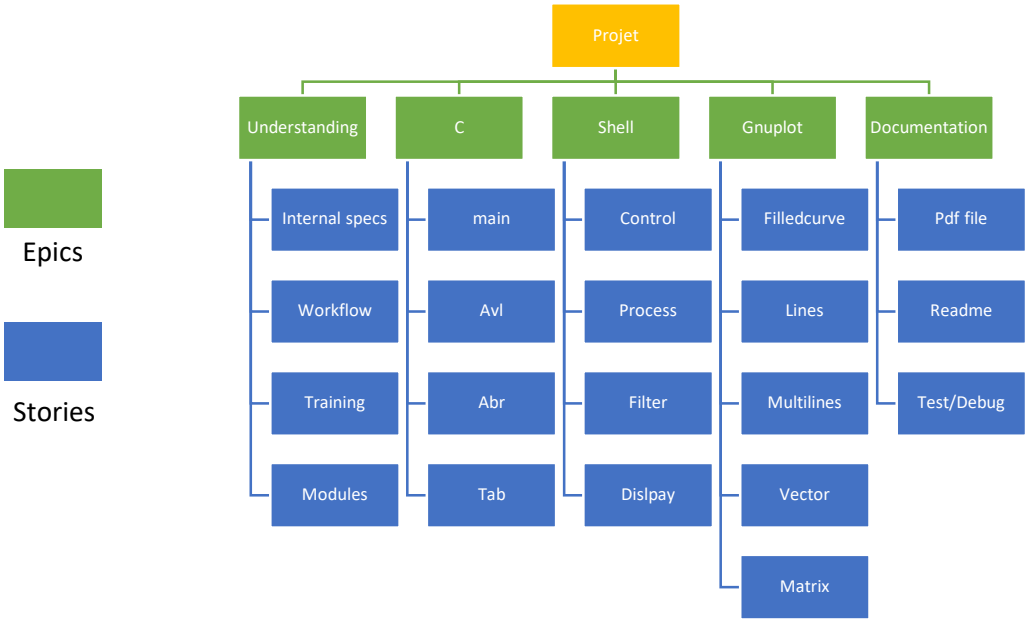
1 – Méthode adoptée

Afin de gérer conceptuellement le projet, d'acquérir une vision d'ensemble et de surmonter les contraintes, il est important de se donner une méthode à suivre. Nous avons ainsi décidé de nous inspirer du cadre de développement Agile Scrum. Les pratiques Agile ont l'avantage d'offrir un framework malléable, itératif et dynamique, incitant à la fois à la collaboration et à l'auto-organisation. Scrum, qui est une méthode Agile, est optimale pour améliorer la productivité d'un groupe séparé par la distance. C'est notamment pour cela que, compte tenu de notre troisième contrainte, liée à la distance, nous avons décidé de suivre cette méthode.

Dans la pratique Agile Scrum, les tâches sont divisées en :

- Epics : ce sont des collections de tâches importantes en un bloc qui sont composées de sous-tâches que sont les stories
- Stories : ce sont des tâches découlant de requêtes brèves, pensées du point de vue de l'utilisateur

De plus, il est essentiel en suivant la méthode Agile Scrum de découper le projet en plages temporelles nommées sprints. Les sprints permettent de poser le cadre itératif via une planification suivie d'un travail puis d'une rétrospection avant de démarrer un nouveau sprint.



Découpage du projet en epics et stories

2 – Planning

En s'aidant des idées de sprint, d'epics et de stories de la méthode Agile Scrum, nous avons pu assez tôt dégager un planning et une répartition des tâches détaillés. Nous nous sommes basés pour notre backlog sur un diagramme GANTT, outil permettant d'ordonner temporellement et structurellement les différentes tâches d'un projet sous la forme d'un graphe. Les plages temporelles, correspondantes à des sprints, sont indiquées par des barres de différentes couleurs. Les lignes, correspondant à des stories, sont organisées en blocs, correspondant à des epics.

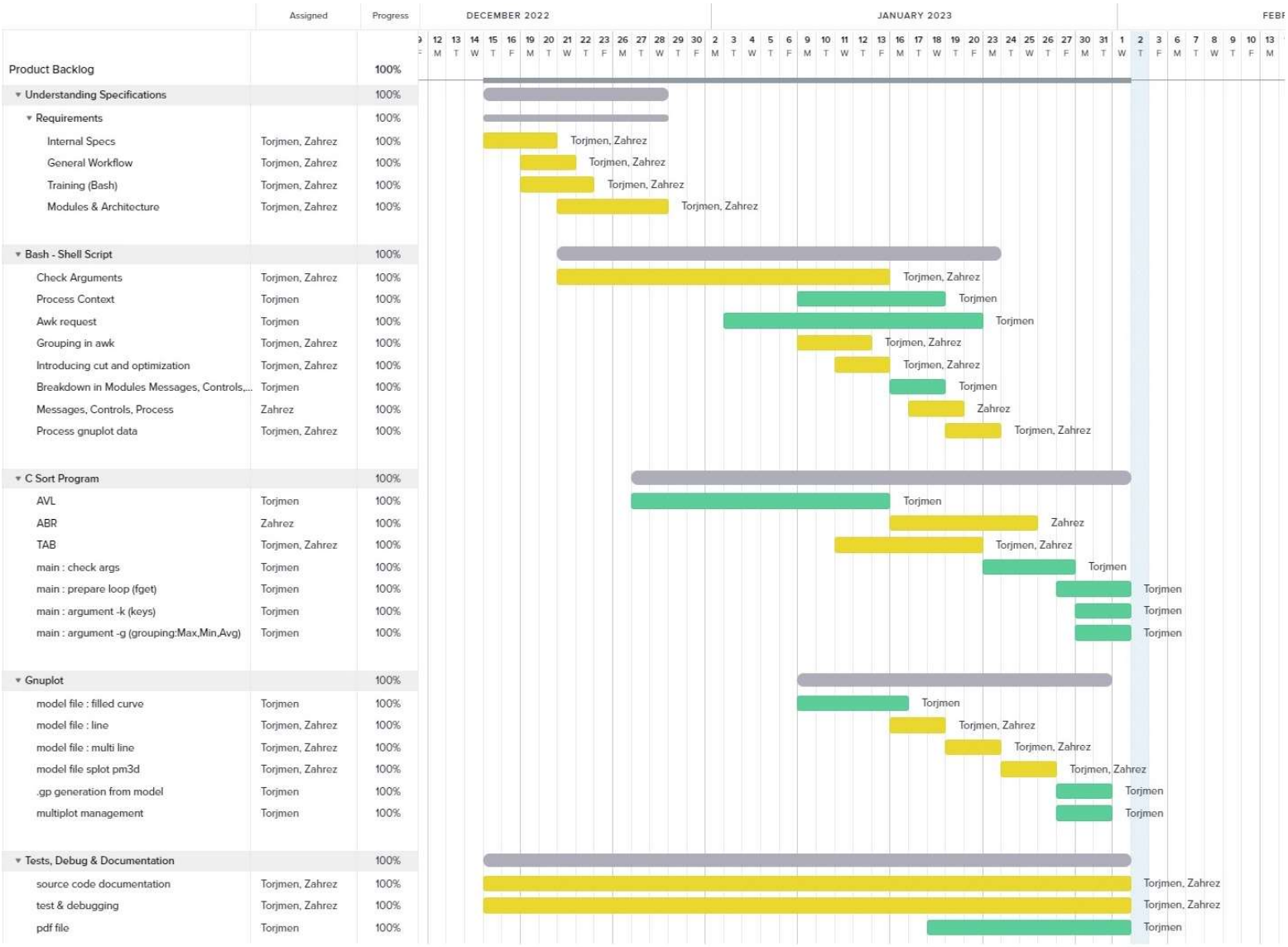


Diagramme GANTT du planning et de la répartition des tâches

III – Résolution technique du projet

1 – Les premiers pas

Les premiers pas de la résolution technique du projet ont principalement reposé sur deux activités, durant les vacances d'hiver :

- Une phase d'analyse conceptuelle et technique du sujet et de mise en place du planning, expliqués tous deux dans les parties précédentes
- Une phase d'entrainement et de recherche en Bash, en C et pour Gnuplot, nous ayant permis d'acquérir une maitrise et un niveau supérieurs à ceux conférés par la lecture du cours et la résolution des TD

Des sites tels que Stack Overflow ou Wikipedia (et bien d'autres) nous ont été d'une grande aide pour nos recherches et l'acquisition d'informations capitales pour la réalisation du projet, que ce soit dans la partie théorique/conceptuelle, de gestion ou technique.

2 - Difficultés rencontrées et solutions apportées

Étant donné que nous avons opté pour une méthodologie Agile Scrum, nous avons pu faire évoluer notre solution sprint après sprint (notre sprint durait 1 semaine) en fonction des enjeux et des difficultés rencontrés (notamment en ce qui concerne la performance et la consommation CPU et mémoire).

Comme indiqué préalablement (I-4-c), nous avons opté pour une approche générique afin de décrire les scopes, de sorte que le filtrage, le tri et les groupements se fassent indépendamment des requêtes spécifiques se trouvant dans le cahier des charges. Cette généricité a été gérée grâce à l'introduction de la variable *ScopeDef*, définissant les scopes en indiquant les keys, les values, ainsi que les consignes de traitement tels que le split de date ou des coordonnées géographiques. Le sens de tri et le groupement y sont également indiqués.

```
ScopeDef="t1:1-11g \
          t2:2a,2b-11g \
          t3:2b,1,2a-11 \
          p1:1-7g \
          p2:2a,2b-7g \
          p3:2b,1,2a-7 \
          w:1-10a,10b,4g,5g \
          h:1-10b,10a,14r \
          m:1-10b,10a,6g \
          x:1-6g"
```

Par exemple, t3:2b,1,2a-11 indique que dans le scope t3 (Température en mode 3), la key de tri est composée de l'heure (colonne 2 partie b post-split) puis du code de la station (colonne 1) et enfin du jour (colonne 2 partie a post-split). Le tiret "-" représente le séparateur entre les keys et les values. Ici, il n'y a qu'une seule value (colonne 11 : température).

Le contexte h:1-10b,10a,14r signifie que dans le scope h (altitude), la key est la station (colonne 1) et les values sont composées de la longitude (colonne 10 partie b post-split) puis de la latitude (colonne 10 partie a post-split) et enfin de l'altitude (colonne 14). Le suffixe r indique qu'il faut trier l'altitude dans le sens décroissant.

La taille du fichier meteo_filtered_data_v1.csv étant gigantesque, il était important de s'assurer que le livrable final puisse traiter les requêtes demandées en un temps acceptable et raisonnable. La première approche consistait à filtrer le fichier global par la commande awk en traitant la totalité du fichier

(filtrage des colonnes et des lignes), ainsi que pour grouper et calculer les statistiques (Max, Min, Average) à la sortie du programme de tri en C (*csort*). Force de constater que cette technique ne permettait pas d'obtenir un temps de réponse acceptable, nous avons décidé, suite à un conseil de M. Romuald GRIGNON, d'utiliser la commande cut qui permettait de notablement améliorer le temps de traitement du filtrage. Par la suite, la commande tr a été introduite afin de traiter les coordonnées et la date (format ISO) et de splitter ainsi la date ISO en date et heure et les coordonnées en longitude et latitude. Cependant, le temps cumulé du cut suivi du tr suivi du awk n'était toujours pas à la hauteur de nos attentes. C'est ainsi que les trois commandes ont été imbriquées par l'utilisation des pipes (cut | tr | awk). Ce faisant, le filtrage fût optimisé et le temps de la requête divisé par un rapport de 5.

Concernant le groupement fait après le programme C (*csort*), la première approche consistait à le traiter par awk. De nouveau, le temps de réponse était important. De ce fait, nous avons changé de stratégie et décidé de gérer le groupement et le calcul des statistiques (Max, Min, Average) dans le programme C lui-même ; si bien que les arguments passés à *csort* ainsi que l'algorithme de tri et la structure des nœuds s'est adaptée pour que ces calculs se fassent au moment même de l'insertion des records (lignes du fichier d'entrée) dans l'arbre ou la liste chaînée, selon la méthode.

Par la suite, nous avons optimisé la structure de stockage des statistiques (Max, Min, Average) en passant d'un vecteur statique à un vecteur dynamique, réduisant ainsi la consommation mémoire pour que le strict nécessaire soit alloué en fonction du scope. L'introduction de flagstats (indicateur du périmètre statistique à traiter : Max = 1, Min = 2, Average = 4), conçu en octal à l'instar des droits d'accès aux fichiers *rwx-rwx-rwx* dans Linux, nous a permis de calculer les statistiques demandées selon le contexte. Les contextes statistiques sont définis dans la variable *StatsDef*, tout comme *ScopeDef* (définition des scopes), déclarés dans *meteo.sh*.

```
StatsDef="t1:7 t2:4 t3:0 p1:7 p2:4 p3:0 w:4 h:0 m:1 x:7"
```

C'est ainsi que pour le contexte w (Vent), le 4 indique qu'on doit uniquement calculer la statistique Average (moyenne) ou que pour t1 (Température en mode 1), le 7 (1 + 2 + 4) indique qu'il faut calculer les statistiques Max, Min et Average.

Quant à l'utilisation de Gnuplot, nous initialement créé un script gnuplot par contexte, mais la ressemblance de ces scripts entre température et pression ou entre humidité et altitude nous a invité à plutôt créer des modèles contenant des variables telles que le titre, les libellés des axes ou les colonnes à utiliser. Ensuite, la commande sed nous a permis de remplacer ces variables par leurs valeurs respectives et définies dans la variable PlotDef, déclarée dans *meteo.sh*.

```
PlotDef="t1:filledcurve-Temperature.min,max,mean.by.Station-Temperature-Station|\
        t2:lines-Average.temperature-Temperature-Time|\
        t3:multilines-Temperature.by.hour.and.station-Temperature-Date|\
        p1:filledcurve-Pressure.min,max,mean.by.station-Pressure-Date|\
        p2:lines-Average.pressure-Pressure-Time|\
        p3:multilines-Pressure.by.hour.and.station-Pressure-Date|\
        h:matrix-Altitude-Latitude-Longitude-4 \
        m:matrix-maximum.moisture-Latitude-Longitude-4 \
        w:vector-average.wind-Latitude-Longitude\
        x:filledcurve-Humidité.min.max.mean.by.Station-Humidité-Station"
```

Par exemple, le contexte m est associée au modèle gnuplot *matrix* puis par le séparateur "-" au titre *maximum.moisture* puis à un axe Latitude, un axe Longitude et un axe lié à la colonne 4 (humidité).

L'affichage des plots dans des fenêtres séparées et éparpillées pour chaque scope demandé se faisait après la génération des scripts gnuplot dans une boucle, mais nous avons pensé à introduire une amélioration consistant à afficher ces plots regroupés en une seule fenêtre grâce à l'utilisation du *multiplot*. Un algorithme approprié permettait de trouver la meilleure combinaison pour la disposition des plots dans la fenêtre.

Nb Scope	Lignes	Colonnes
1	1	1
2	2	1
3	2	2
4	2	2
5	3	2
6	3	2
7	3	3
8	3	3
9	3	3
...	...	...

Tableau représentant les nombres de lignes et de colonnes optimaux pour afficher le nombre de scopes demandés

IV - Originalité du rendu final

À l'issue des difficultés rencontrées et solutions apportées, notre programme présente plusieurs avantages qui ne sont pas exigés dans le cahier des charges.

Premièrement, comme expliqué dans les précédentes sections (I-4-c et III-2), nous sommes parvenus à avoir un programme générique et modulaire indépendant des requêtes spécifiques dans le cahier des charges, par le biais d'une définition des scopes (*ScopeDef*), des statistiques (*StatsDef*) ainsi que des plots (*PlotDef).*

Ainsi, il existe dans le code-source, plus précisément dans *meteo.sh*, un scope supplémentaire, nommé x, ajouté à titre de démonstration de la généricité de notre programme. Pour l'ajouter il faut ajouter sa définition dans *ScopeDef*, dans *StatsDef* puis dans *PlotDef*. Enfin, il suffit d'ajouter deux fois son nom dans la fonction *CheckArgs* dans *controls.sh,* une fois dans cette ligne :

Définition du scope x dans *ScopeDef* : `x:1-6g`

Définition des statistiques de x dans *StatsDef* : `x:7`

Définition du plot associé à x dans *PlotDef* :

```
x:filledcurve-Humidité.min.max.mean.by.Station-Humidité-Station
```

Enfin, il suffit d'ajouter son nom dans la variable *OptDef*, ainsi que dans la place convenable dans le case de *CheckArg*s, en l'occurrence pour le scope x ici :

```
47                                  w|h|m|x)
```

En plus de la généricité de notre programme, nous avons inclus des filtrages selon la longitude et la latitude, chose qui constituait un bonus dans le cahier des charges de CY-Météo. Ainsi, il est par exemple possible de filtrer un fichier selon une longitude et une latitude min et max. En outre, afin d'effectuer le filtrage selon les lieux (ex : -F), nous les avons encadrés selon des longitude et latitude min et max. De ce fait, il est possible de réduire les différents lieux à leur équivalent en coordonnées géographiques. Il en découle que la possibilité de mettre en place une généricité pour les filtres tels que les lieux s'est aussi offerte à nous, mais puisque ces derniers ne sont gérés que dans une petite partie du Shell (*controls.sh*), nous ne l'avons finalement pas suivie.

En outre, à des fins de lisibilité et d'amélioration de l'interface utilisateur, nous avons mis en place :

- Une barre de progression, par le biais de awk, montrant le défilement des éléments traités lors du filtrage, du tri et du groupage, avec une barre dans laquelle un curseur défile. Afin de l'afficher après une requête, il faut ajouter le paramètre --p ou --progress.

```
awk: progress -> [1074967] lines [---------[]----------------------------]
```

- Un mode debug permettant d'afficher plusieurs détails de filtrage, de tri et de groupement. Afin d'y accéder lors d'une requête, il faut ajouter le paramètre --p ou --debug ou --verbose

- Un mode help, inspiré de la commande man, servant de manuel détaillé pour compiler *meteo.sh* tel qu'on le souhaite. Afin de l'afficher, il suffit d'ajouter le paramètre --help

- Un mode multiplot, permettant d'afficher les plots de manière regroupée en une seule fenêtre et d'éviter que les fenêtres soient séparées et éparpillées. Afin d'ouvrir le mode multiplot, il faut ajouter le paramètre --multiplot ou --m dans la requête. Il est important de préciser que l'affichage est optimal lorsque la machine sur laquelle il y'a compilation supporte x11. Pour des raisons de sécurité, nous avons préféré utiliser qt, malgré un affichage moins efficace. Si la machine sur laquelle il y'a compilation supporte x11, il suffit, dans *display.sh*, de commenter cette ligne :

```
113         cmd="gnuplot -p -e 'set terminal qt size 1200,800 position 50,10;' $allplots_filename"
```

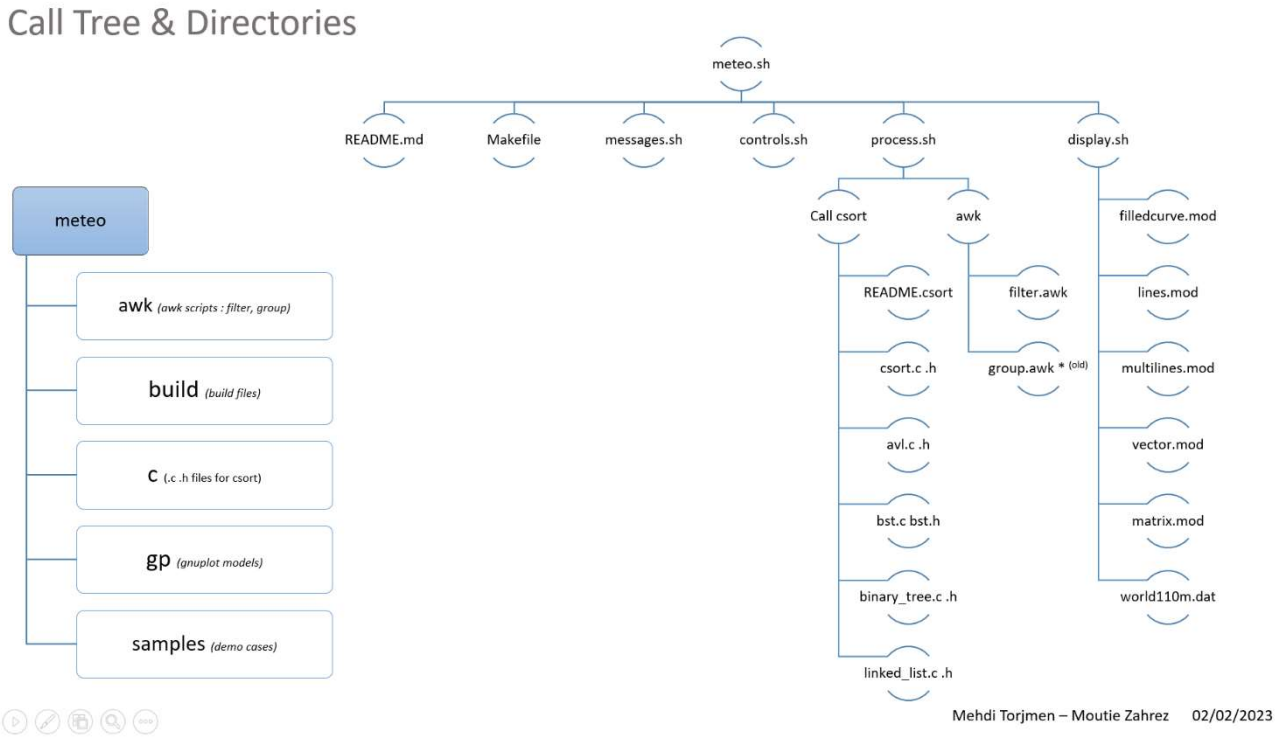
ainsi que de de décommenter ces lignes :

```
109         # Xaxis=$(echo "$(xrandr --current | grep '*' | uniq | awk '{print $1}' | cut -d 'x' -f1)*0.9"|bc)
110         # Yaxis=$(echo "$(xrandr --current | grep '*' | uniq | awk '{print $1}' | cut -d 'x' -f2)*0.9"|bc)
```

```
113         # cmd="gnuplot -p -e 'set terminal x11 size "$Xaxis,$Yaxis" position 50,10;' $allplots_filename"
```

- Un graphique avec la carte du monde, s'adaptant aux filtrages par lieux ou coordonnées géographiques, grâce au fichier *world110m.dat*. La carte du monde apparait dans le cas du scope w (Vent), dont le graphique affiche une carte de vecteurs par station, représentant des moyennes de vitesse et d'orientation des vents.

V – Arborescence des fichiers et des appels

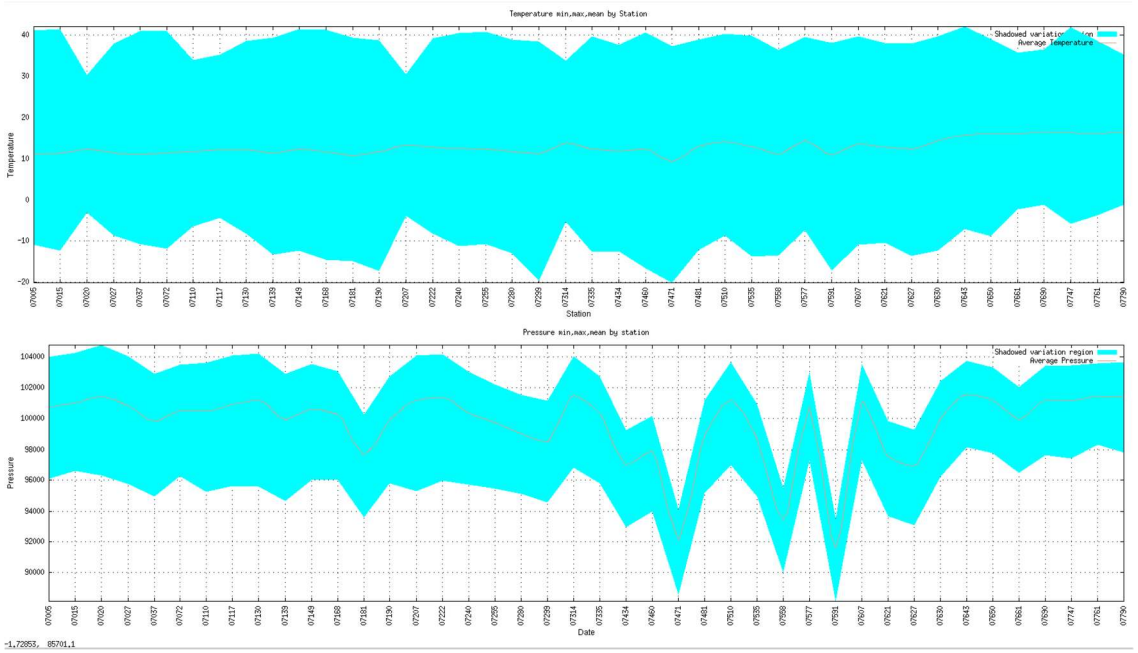


VI - Résultats/tests

Voici les différents résultats et tests que nous avons obtenus et dont nous avons posé et précisé les étapes intermédiaires dans le répertoire samples du projet :

- Le fichier après filtrage *Scope_filename*.csv,
- Le fichier après tri s_*Scope_filename*.csv,
- Le fichier après un éventuel deuxième tri sur une valeur comme le cas des scopes h et m g_s_*Scope_filename*.csv,
- Le fichier *filename*.gp qui est le script gnuplot y afférent.

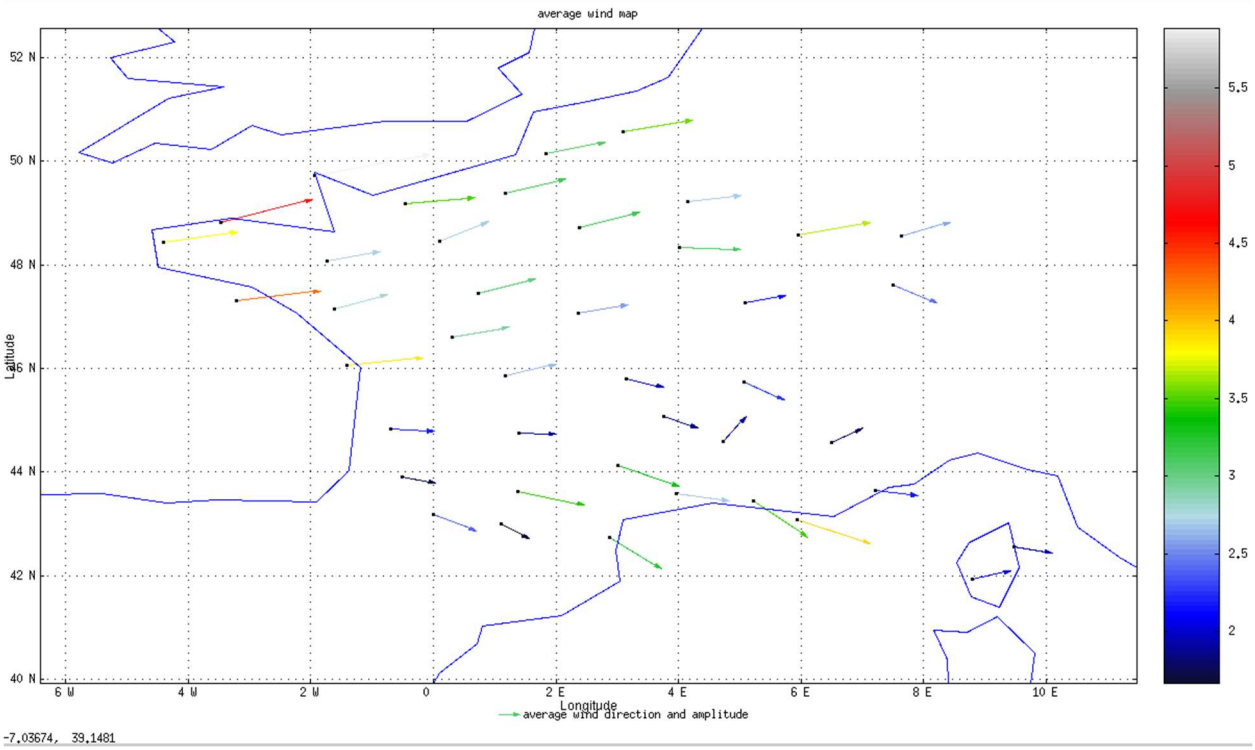
Chaque graphique est accompagné d'un titre, de la commande qui a permis de l'obtenir et du chemin des fichiers dans lesquelles ses étapes intermédiaires sont posées.



Average temperature and pressure and variation for France between 2010 and 2020

```
$ bash meteo.sh -t1  -p1 -f meteoall.csv -F -d 2010-01-01 2020-12-31 --multiplot
```

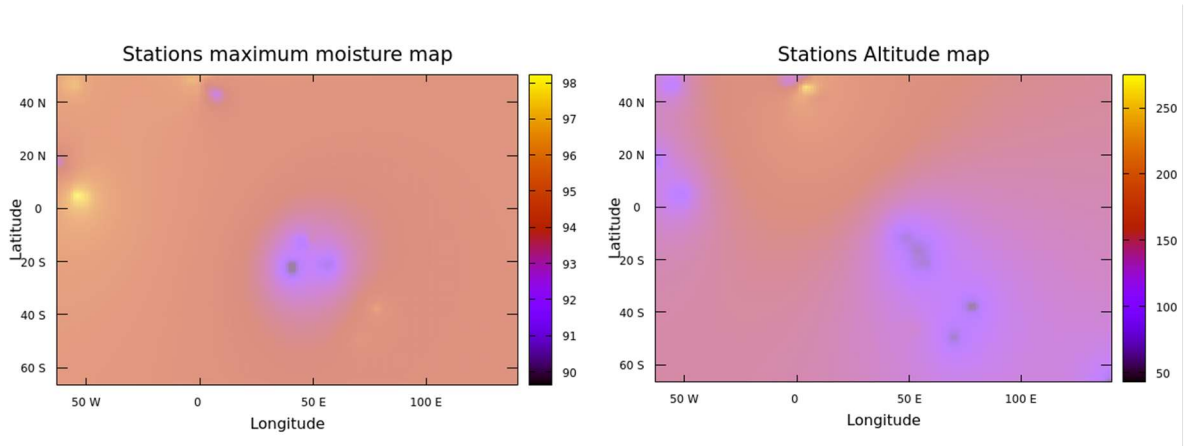
Files are in **"./sample-2010-2020-F-t1-p1"**



Average wind intensity and direction in March 2020 in France

```
$ bash meteo.sh -w -f meteoall.csv -F -d 2020-03-01 2020-04-01
```

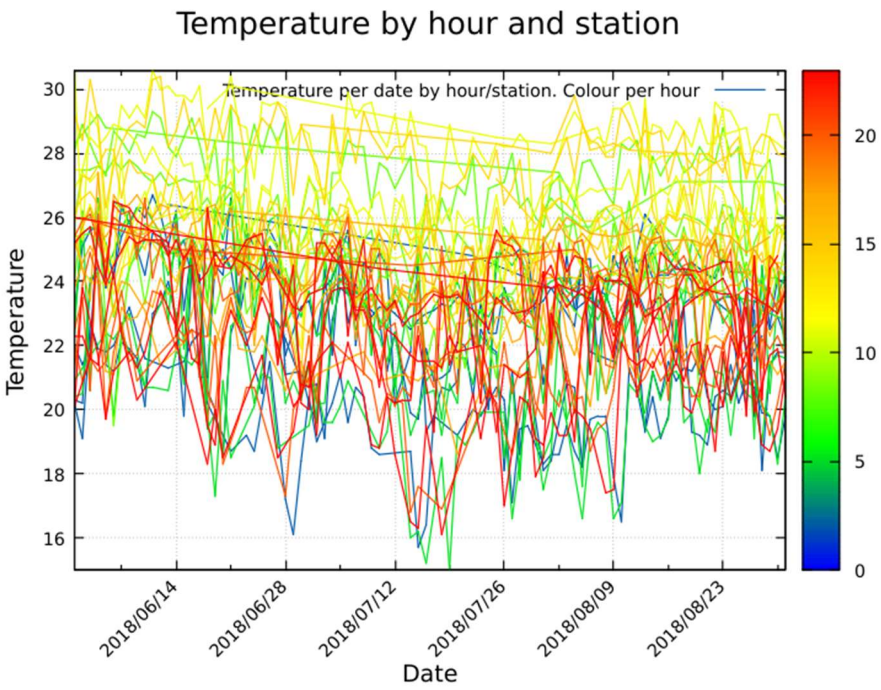
Files are in **"./sample-2010-03-F-w"**



Left : Maximum humidity first week of July 2020 all stations (sorted by TAB method)
Right : Altitude of all stations

```
$ bash meteo.sh –w -h -f meteoall.csv --tab -d 2020-07-01 2020-07-08
```

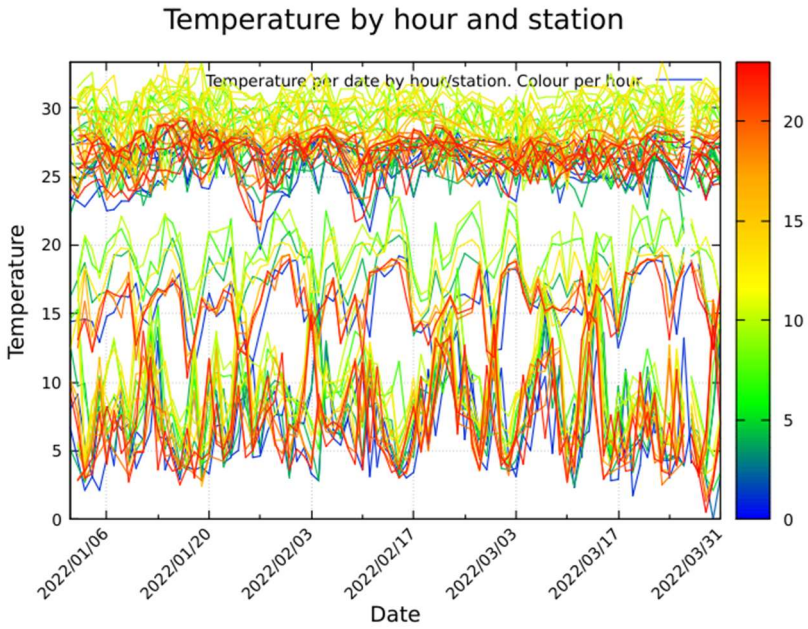
Files are in **"./sample-2020-07-01-w-h"**



Temperature variation in summer 2018 in Indian Ocean (color indicates the hour). Sorted by ABR

```
$ bash meteo.sh -t3 -f meteoall.csv --abr -d 2018-06-01 2018-08-31 -O
```

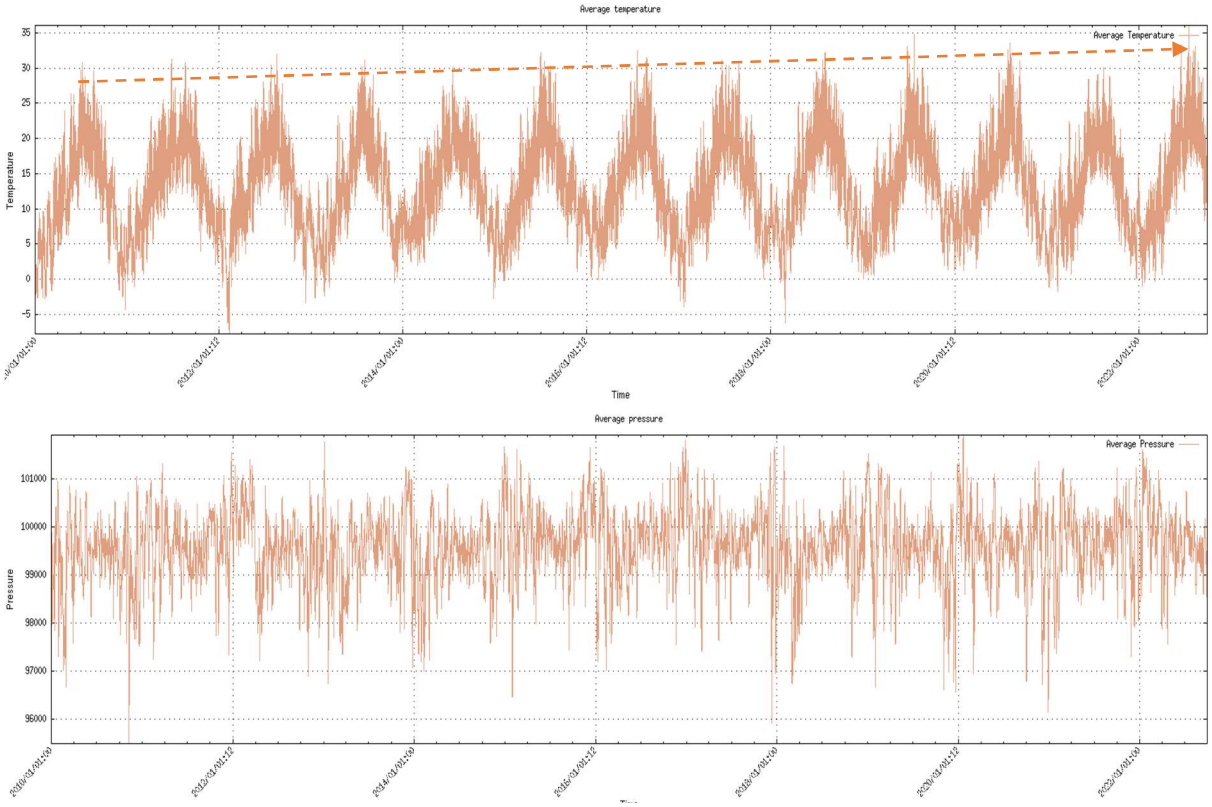
Files are in **"./sample-2010-03-F-w"**



Temperature variation Q1-17 in the frame of longitude [30 : 100] and latitude [-60 : 0]

```
$ bash meteo.sh -t3 -f meteoall.csv -d 2022-01-01 2022-03-31 -g 30 100 -a -60 0
```

Files are in **"./sample-2017-Q1-g-a-t3"**

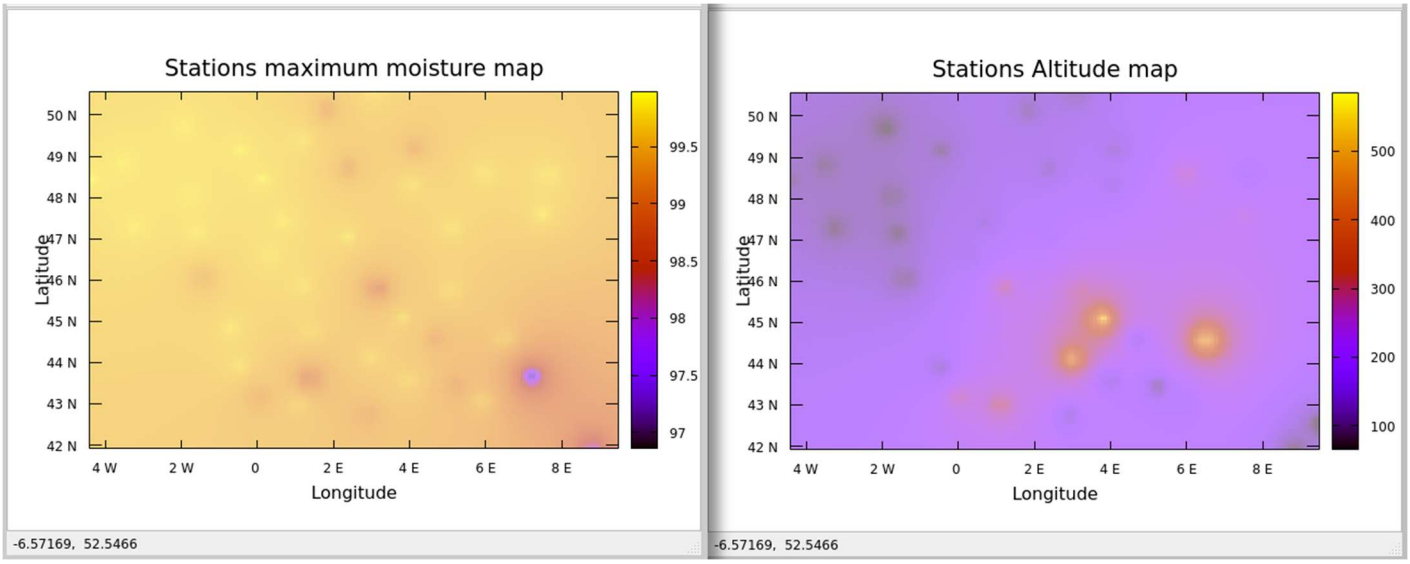


Temperature & Pressure variation over 12 years in France

We can clearly notice the increasing trend year after year (Climate change ?!)

```
$ bash meteo.sh -t2 -p2 -f meteoall.csv -d  -F --multiplot
```

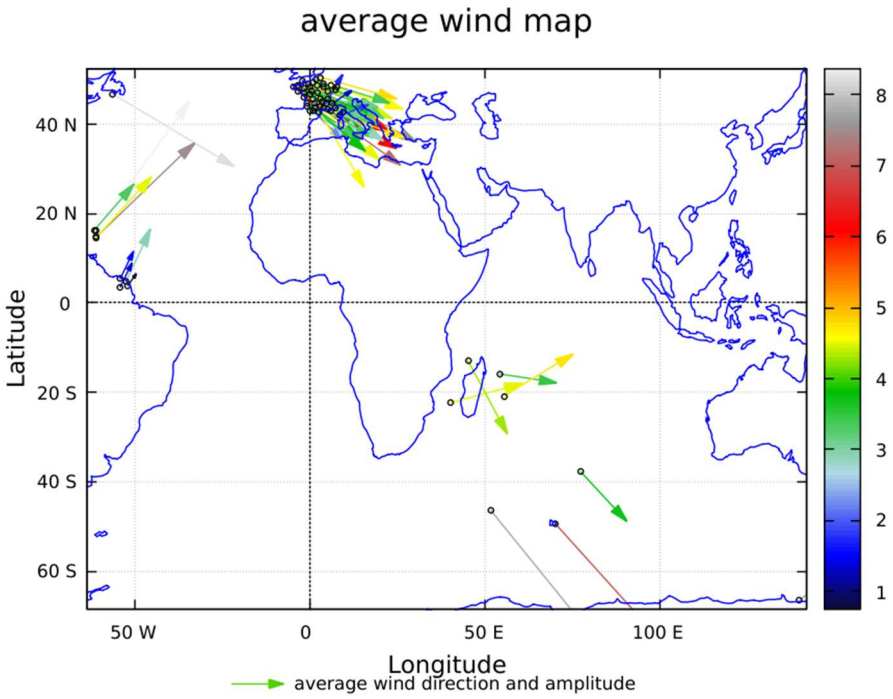
Files are in **"./sample-F-t1-p1"**



Max humidity in France in 2018 (Right: Reminder of the stations altitude)

```
$ bash meteo.sh -m -h -f meteoall.csv -F -d 2018-01-01 2018-12-31
```

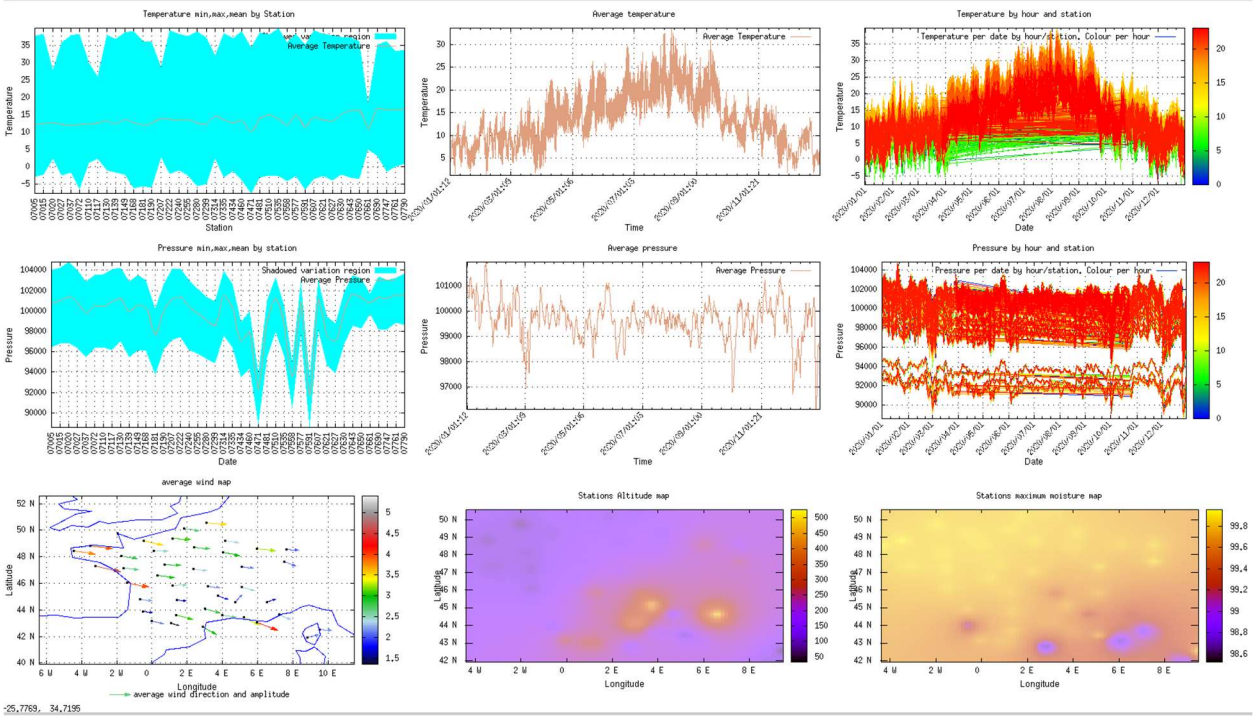
Files are in **"./sample-2018-F-m-h"**



Average wind intensity and direction in January 2018 in all french stations

```
$ bash meteo.sh -w -f meteoall.csv  -d 2018-01-01 2018-01-31
```

Files are in **"./sample-2018-01--w"**



Dashboard for France for 2020

```
$ bash meteo.sh -t1 -t2 -t3 -p1 -p2 -p3 -w -h -m -f meteoall.csv -F -d 2020-01-01 2020-12-31 --multiplot
```

Files are in **"./sample-2020-F-dashboard"**