# Qualification temps réel des données trajectoire des flotteurs Argo.

Herbert Gaëlle (gaelle.herbert@ecomail.fr), Mai 2020
Programme réalisé sous Matlab, version 2016b.

## I. INTRODUCTION

Ce document décrit l'algorithme des différents tests réalisés sur les variables du fichier TRAJ et META des flotteurs Argo visant à qualifier les données en temps réel. Les principales variables concernées sont les cycles (durée, enchainement), les positions et dates des localisations argos de surface, les mesures de température/salinité/pression en profondeur, et la variable 'grounded' qui indique si le flotteur a touché le fond ou non. L'ensemble des alertes associées aux incohérences ou problèmes repérés sont listés dans un fichier *RT_alertes.txt* consultable à la fin du programme. Dans certains cas, le flag peut être modifié. Les numéros de flotteurs et les cycles concernés par chaque alerte (de 1 à 30) sont récupérés dans une variable de type *alerte[1à 30].txt*. Les alertes ont également été regroupées en 8 types afin de pouvoir effectuer des comparaisons avec les alertes Altran.

## II. DESCRIPTION DE L'ALGORITHME

### II. 1 INITIALISATION

On récupère les path (emplacement des fichiers flotteurs, liste des flotteurs, de la climatologie, de la bathy, des répertoires qui seront utilisés, etc) et les différents paramètres à partir du fichier config. On définit également les fichiers .txt des alertes et on initialise certains tableaux et variables globales.

### II. 2 BOUCLE SUR LES FLOTTEURS

Pour chaque flotteur :

- on récupère le fichier trajectoire et le fichier META.

- on récupère les indices des localisations de surface (idLoc) et de dérive en profondeur (idDrift). Pour cela on se sert du measurement_code (703 pour la surface et 290, 296, 299 pour les mesures de parking). On alerte si jamais le measurement_code n'est pas présent.

- on créé la liste des numéros de cycles théoriques
`cycles_1 = [numCycleMinTraj:numCycleMaxTraj];`
et réelle : `cycles = unique(CycLoc(CycLoc>=0))` avec
`CycLoc=T.cycle_number.data(idLoc);`

-On calcule la médiane des durées de cycles (dureeMedianCycle) pour chaque numéro de mission.
Pour cela on commence par récupérer les cycles concernés par le numéro de mission donné (cycles_m) et on calcul la durée de chaque cycle (duree_cycle_m).
Si l'écart-type des durées de cycle pour un numéro de mission donné est < 100 (et qu'il y a plus d'un cycle) :

`dureeMedianCycle(m)=median(duree_cycle_m);`

On teste ensuite la cohérence entre la médiane de la durée de cycle et celle fournie par le fichier méta (M.CycleTime). Si la valeur absolue de la différence entre la médiane et M.CycleTIme pour un numéro de mission donné est > 0.1, on alerte sur une incohérence META/TRAJ.

Par ailleurs, il arrive que le nombre de numéro de missions donné dans le fichier TRAJ diffère de celui indiqué dans le fichier META (M.CycleTime). On compare donc les deux et on alerte si c'est le cas.

### II.2.1 CONTROLE DES CYCLES (Test_cycles)

Le test vérifie la présence de doublons dans les numéros de cycles, de numéros de cycles non croissants, et de mauvaise durée de cycle.

On récupère les indices des locs de surface d'un cycle donné (idCyc) et du cycle précédent (idCycprec).

1. **Alerte double cycle**

Pour les cycles >1, si les locs de surface pour le cycle n existent, on récupère la date associée.
On compare ensuite les dates à l'indice i et celle à l'indice i+1, si elles sont identiques on alerte sur un doublon de cycle.
`if(isempty(find(ismember(cyc(1:id-1,:),cyc(id,:))==1))==0 & id>1)`

2. **Alerte cycles non croissants**

On trie les numéros de cycles dans l'ordre croissant (a_cycles_sorted). Si ce que l'on obtient est différent du tableau de cycles initial (a_cycles) on alerte sur les cycles non croissants.

```
if isequal(a_cycles_sorted,a_cycles)==0
```

3. **Alerte durée cycle**

On récupère le numéro de mission associé au cycle n (idMis).

Si le nombre de numéro de mission est inférieur au nombre donné par M.CycleTime et que le numéro de mission est différent de NaN.
Pour estimer la durée du cycle donné, on récupère la différence entre la date de la dernière loc du cycle donné et la date de la dernière loc du cycle précédent. Si la différence (Cyct) est trop éloignée de la durée de cycle médiane calculée pour le numéro de mission idMis donné ou bien de la durée donnée par le fichier méta, on alerte sur la durée du cycle.

```
if id > 2 &
(Cyct<=a_dureeMedianCycle(idMis)*(a_cycles_sorted(id)-
a_cycles_sorted(id-1))-1 |
Cyct>=a_dureeMedianCycle(idMis)*(a_cycles_sorted(id)-
a_cycles_sorted(id-1))+1) | Cyct<M.CycleTime(numMis)-0.15 |
Cyct>M.CycleTime(numMis)+0.15
```

S'il y a incohérence entre la durée médiane calculée et la durée de cycle fournie par le fichier méta, et qu'il y a au moins une durée de cycle <0, on indique que le problème dans la durée du cycle est probablement dû à un mauvais arrangement des transmissions argos.

Par ailleurs, dans le cas où le nombre de numéro de mission indiqué dans le fichier méta est < au numéro de mission donné isMis, on indique que l'on ne peut pas comparer la valeur médiane avec la valeur de durée de cycle fournie par le fichier méta.

**4. Alerte découpage des cycles**

On vérifie le découpage des cycles en comparant le tableau des cycles trié de façon croissante (a_cycle_sorted) avec le tableau non trié (a_cycles_1). Si des différences existent-elles sont récupérées dans cycles_missing . On récupère également les cycles précédents et les cycles suivants les cycles manquants pour estimer la durée écoulée.
Afin de déterminer s'il y a un problème dans le découpage des cycles, on calcule ensuite la durée entre deux cycles encadrants le ou les cycle(s) manquant(s) (=dureeCyclem).
Puis on l'a compare à la durée estimée en récupérant les dates des cycles situés avant et après le(s) cycle(s) manquants :

```
if ~isnan(dureeCyclem) & (T.juld.data(idCycmpost(end))-
T.juld.data(idCycmpre(end))>(cycles_post_missing(im)-
cycles_pre_missing(im))*(dureeCyclem+0.5) ...
            |T.juld.data(idCycmpost(end))-
T.juld.data(idCycmpre(end))<(cycles_post_missing(im)-
cycles_pre_missing(im))*(dureeCyclem-0.5))
```

Si la condition ci-dessus est vérifiée, on alerte sur un saut de cycles.

***Fin de la fonction Test_cycles***

**II.2. 2 CONTROLE DES LOCS ARGOS**

***Fonction Test_locs***

1. **Contrôle des dates.**

On récupère les dates des locs de surface et on vérifie qu'elles sont bien > 01/01/1997 et <= la date d'aujourd'hui. Pour les cas contraires, les dates sont erronées. On alerte alors sur les dates de loc non réalistes et le flag de juld_q est mi à 6.

2. **Contrôle des Loc Argos**

On récupère les longitudes et latitudes des locs argos et on sélectionne celles qui sont réalistes, c'est-à-dire >=-180 et <=180 pour les longitudes et >=-90 et <=90 pour les latitudes.
S'il y existe des positions erronées, on alerte sur les loc non réalistes. Et le flag position est mis à 6.

3. **Contrôle des dates et locs de lancement.**

On compare la latitude, la longitude et la date de lancement fournie par le fichier trajectoire (LaunchLat, LaunchLon, LaunchDate) et celle fournie par le fichier META (M.launch_latitude, M.launch_longitude,Launchdate_meta).
Si la différence en valeur absolue est >0.1 pour la longitude/latitude ou si Launchdate et Launchdate_meta diffèrent, on alerte sur l'incohérence entre META/TRAJ pour la position et/ou date de lancement.

***Fin de la fonction Test_locs***

### II.2.4 CONTROLE DES MESURES T/S/P°

On veut vérifier que les mesures des paramètres physiques sont cohérentes en comparant avec le modèle climato (ici ISAS).

Pour chaque cycle :
- On récupère les indices des locs de surface (idCyc) et des dérives en profondeur (idCyc_drift) associées au numéro de cycle.
- On récupère les dates/lon/lat/qc/temp/date/position_qc des locs de surface.
- On fixe une valeur limite de temp et psal à 50.

S'il y a bien eu dérive en profondeur (idCycdrift existe) et que les valeurs de pression sont différentes de NaN :
.On récupère la valeur de la longitude et la latitude à la première loc en surface.
et à la dernière loc du cycle précédent.
. On calcule la valeur absolue de la moyenne des pressions (pres_drift_mes), températures (temp_drift_mes), salinité (psal_drift_mes) pendant la dérive.
. On récupère l'altitude de la première position du cycle n et de la dernière position du cycle n-1 pour déterminer par la suite si le flotteur est grounded ou non.
. On détermine les indices ISAS en longitude (ilong_drift), latitude (ilat_drift), pression (idepth_drift) ; on convertit les échelles de température et de salinité mesurée et on calcule l'écart type pour la température (temp_std) et la salinité (psal_std).

#### 1. Contrôle de la température et de la salinité *(Fonction Test_TS)*

- Si la température mesurée en profondeur de dérive (temp_drift_mes) est < à la température estimée par ISAS (temp_drift_th) – 10* l'écart-type. OU si la température mesurée est > la température estimée par ISAS+ 10* l'écart-type ET si la température estimée est < 50, on alerte sur un problème de température.
  temp_alert est mis = 1 et temp_non_ref et mis à 0

Si la température estimée par ISAS est > 5, on alerte sur la température non référencée et Temp_non_ref est mis = 1 et temp_alert = 0 ;

Sinon temp_alert = 0 et temp_non_ref = 0 ;

- Si la salinité mesurée en profondeur de dérive est < la salinité théorique – psal_hold * psal_std. OU la salinité mesurée en profondeur de dérive est > la salinité théorique +

psal_hold * psal_std.  ET la salinité théorique < 60, on alerte sur un problème de salinité.
Si la salinité théorique > 60, on alerte sur la salinité non référencée.

***Fin de la fonction Test_TS.***

## 2. QC des mesures pressions avant vérification

On sélectionne les mesures de pressions fiables (qc différent de 4 et 6 et différent de NaN) .
On récupère ensuite les pressions  pour les fonds < -1500m et on calcule la valeur absolue de la moyenne des pressions sélectionnées plus haut, sans considérer les pressions <50 dbar. Le résultat pour chaque cycle est stoké dans pres_ok.
Dans le cas où il n'y a pas de mesures de dérive pour un cycle donné, pres_ok est mis à NaN.

## 3. Contrôle de la pression

### a. Calcul de la pression médiane pour chaque numéro de mission

Le calcul de la pression médiane pour chaque numéro de mission va permettre de vérifier la cohérence entre la valeur de la profondeur de parking indiquée dans le fichier trajectoire et celle fournie par le fichier META. Elle sera également utile pour vérifier si le flotteur est « grounded » ou non.

Ainsi, pour chaque numéro de mission, dans le cas où l'on dispose de mesure en profondeur et si une valeur au moins n'est pas mise à NaN, on récupère les valeurs de pression dans pres_ok_m.
Si l'écart type des valeurs de pression >200 et <2000 est inférieure à 200 :
`presMedianDrift(m)=-abs(median(pres(pres>200&pres<2000)`
Sinon  la pression médiane sera la valeur de la pression de parking fournie par le fichier META (M.ParkPressure) pour le numéro de mission donné

### b. Cohérence Fichier TRAJ/META.

Afin de vérifier la cohérence entre la valeur de pression de parking fournie par le fichier TRAJ et le fichier META, on calcule :

`abs(presMedianDrift(m)- (-M.ParkPressure(missions(m)))`

Si le résultat est >20, on alerte sur une incohérence entre fichier TRAJ/META.

## 4. Contrôle du grounded

### a. détermination de la profondeur de parking (park_prof)

On cherche à vérifier si le flotteur a touché le fond lors de sa descente ou non. On commence par récupérer la valeur de la profondeur de parking du flotteur (park_prof) pour chaque numéro de mission.
Si la valeur médiane de la pression pour un numéro de mission donné est proche de la profondeur de parking indiquée dans le fichier META, alors `park_prof(m) = -M.ParkPressure(m)`
Sinon, dans le cas où des numéros de mission manquent dans le fichier META ou si la pression médiane est éloignée de la prof de parking indiquée dans le fichier META, on prend la valeur de la médiane pour la valeur de park_prof.

### b. Comparaison de la pression mesurée avec la pression estimée par le modèle ISAS.

- Si la pression mesurée est trop éloignée de la pression médiane,

```
if (pres_drift_mes(i_rpp)<presMedianDrift(idMis)-100 ...
                          |
pres_drift_mes(i_rpp)>presMedianDrift(idMis)+100)
```

alors on alerte sur un problème de pression/profondeur de parking et pres_alert est mis =1.

- Si une alerte sur la pression/profondeur de parking s'accompagne d'une alerte sur la température mesurée :

```
if pres_alert(i_rpp)==1 & temp_alert(i_rpp)==1
```

alors on alerte sur la possibilité d'un problème de capteur pression.

- Si la bathy de la dernière loc du cycle précédent ou celle de la première loc du cycle n est très éloignée de la profondeur de parking OU s'il y a une alerte de température non référencée par le modèle climato et une alerte de pression :

```
if  (elev_end_all(i_rpp)>=park_prof(idMis)+30 ||
elev_all(i_rpp)>=park_prof(idMis)+30) ||
(temp_non_ref(i_rpp)==1 & pres_alert(i_rpp)==1)
```

alors on alerte sur la possibilité qu'il y ait grounded et alerte_grounded est mis à 1.

On regarde ensuite la valeur de la variable grounded présente dans le fichier trajectoire qui indique 'Y' si le flotteur est grounded, 'N' s'il ne l'est pas, et 'U' pour 'Unknown'.

Les lettres ont été préalablement associées à des chiffres : 1 pour 'Y', 2 pour 'U'et 3 pour 'U' dans la variable GroundedNum.

Si GroundedNum=2 et alerte_grounded=1, on alerte sur l'incohérence avec le résultat du test et la variable Grounded et la variable T.grounded.data est mise à 'Y'.
Si GRoundedNum=1 et alerte_grounded=0, on alerte sur l'incohérence et la variable T.grounded.data est mise à 'P'.
Si GroundedNum=3 et alerte_grounded=0, la variable T.grounded.data est mis à 'N'.

### II.2.5 CONTROLE DES LOCS PAR CYCLE

#### 1. Contrôle de la présence de doubles dates de loc

On utilise la fonction find_doubles_datelatlon_30s qui repère les doublons de lat/lon associés à un doublon de date.

```
[isdouble_toremove,idDoublon_date,idDoublon_latlon] = ...
find_doubles_datelatlon_30s(locDate, locLon, locLat, locQc,
locTemp)
```

Si isdouble_toremove n'est pas vide, on alerte sur la présence de doubles dates de loc.
Le qc pour les dates doubles est mis à 6.

```
T.juld_qc.data(idCyc(isdouble_toremove))=6;
```

Si idDoublon_date et idDoublon_latlon ne sont pas vides, on alerte sur la présence de doublons de dates associés à des doublons lon-lat, et flag à 6 les lon/lat doublés dont le qc est > 1.

```
T.position_qc.data(idCyc(idDoublon_latlon(find(T.position_qc.d
ata(idCyc(idDoublon_latlon)))>1)))=6;
```

#### 2. Contrôle de la croissance des dates de loc

On récupère les dates de locs (locDateok) dont le qc est différent de 4 et 6 et on les trie de façon croissante (LocDateok_sorted). Si locdate diffère de locdateok_sorted, on alerte sur des dates de loc non croissantes.

### 3. Contrôle de l'écart max entre deux locs

On calcul l'écart moyen et maximum entre deux dates de locs consécutives d'un même cycle pour détecter si des cycles sont accolés par erreur.

```
ecartMeanLocCycle(istat)=median(diff(LocDate_sorted(locDate_qc
_sorted<6)))*24;
ecartMaxLocCycle(istat)=
max(diff(LocDate_sorted((locDate_qc_sorted<6))))*24;
```

Si l'écart max entre 2 locs est >10 on alerte.

### 5. Contrôle de l'écart entre la 1ere et la dernière loc pour un même cycle

Pour les dates de locs qui ont un qc différent de 6 et 4, on calcule cet écart :

```
ecartFirstLastLoc(istat)= LocDate_sorted(end)-
LocDate_sorted(1)
```

Si l'écart est supérieur à l'écart moyen entre deux locs (10h pour les flotteurs iridium et 24h pour les Argos), on alerte. On alerte également si l'écart est > 60h, en indiquant que le flotteur est peut être en EOL (end of life).

### 6. Contrôle des premières localisations

Il arrive que le flotteur soit mis en route sur le bateau et localisé par Argos avant d'être mis à l'eau. On élimine ces localisations qui reflètent le déplacement du bateau. Pour cela :
-Lorsque le premier numéro de cycle est 0 ou 1 et que le M.launch_qc=1, on récupère la date de lancement LaunchDate = T.juld.data(iLaunch) et on vérifie qu'elle est réaliste, soit > au 01/01/1997 et < à la date du jour. Sinon on alerte sur une date de lancement non réaliste.

-Pour les dates réalistes on calcule la différence entre la date de la première loc et la date de lancement (ecartLaunchDateFirstLoc). Il se peut que la Launchdate, qui est une méta donnée rentrée à la main, puisse être erronée.

On commence par vérifier la différence entre la launchdate et la Startup_Date (dans le cas où elle est présente dans le fichier META) qui correspond à l'allumage de flotteur, qui doit être antérieure à la launchdate : `Diff_LS = LaunchDate+datenum('01011950','ddmmyyyy') - StartDate;`

Si M.startup_date_qc=1 et Diff_LS<0 alors on alerte sur le fait que la launchdate est antérieure à la startup date et on flag à 6 le M.launch_qc.

Si M.startup_date_qc=1 et Diff_LS > 1 alors on alerte sur un écart entre la launchdate et la startup_date est > 1 jour.

On vérifie que les dates de loc du premier cycle sont supérieures à la launchdate. On définit des seuils (en jours) au-delà desquels il est fort probable qu'elle soit erronée, auquel cas il ne faut pas flagguer à 4/6 les dates de loc.
Si la launchdate est postérieure aux locs de3 jours (3h), la launchdate est considérée comme douteuse. En effet, les mises en route sur le bateau se font quelques minutes avant.

Si `ecartLaunchDateFirstLoc(k)>-0.125 (=3h)`, on vérifie que les dates de locs du premier cycle sont supérieures à la launchdate . Si ce n'est pas le cas on alerte sur des dates de locs antérieures à la launchdate et on les flagge.
Si c'est le cas, alors le problème vient de la launchdate donc on alerte.
Par ailleurs, si `ecartLaunchDateFirstLoc(k)>10.5` on l'alerte également en supposant un DPF (Deep Profile First) sans prélude.

Remarque : ecartLaunchDateFirstLoc est mis à NaN si le premier numéro de cycle est 1.
Les vérifications ci-dessus ont été faites dans le cas où le premier cycle est = 0 ou 1, dans le cas contraire, on alerte sur le fait qu'il y ait qu'une seule loc, et si `ecartLaunchDateFirstLoc > 100` on alerte sur le fait qu'il n'y ait qu'une seule loc et une date incohérente.
*A VOIR*

### 7. Contrôle de la phase de prélude

Pour confirmer ou non le cas d'un DPF sans prélude on teste la présence du cycle 0. S'il est absent et si le measurement_code(2) = 100 on indique l'absence du cycle 0 et la présence d'un DPF sans prélude.

### 8. Contrôle des positions

On vérifie si les positions sont bien dans l'eau et non sur la terre. On flagge à 4 si ce n'est pas le cas.

On ne vérifie que les positions qui ont un flag bon et qui ont passé les tests précédents.
On récupère la longitude (ilong), la latitude (ilat) et l'altitude (elev_float) d'après le fichier bathy choisi, pour le point donné et pour ses 4 voisins.
Les 4 voisins doivent avoir une altitude positive pour que l'on puisse considérer le point sur le continent. Si c'est le cas on alerte sur un problème de localisation sur le continent. Le flag position est mis à 4.
Si l'altitude au point donné est > 0 mais moins de 4 voisins sont également positifs, on alerte

sur une loc terre, le flotteur est possiblement coincé dans un atoll ou récupéré et proche de la terre.

Si l'altitude est négatif, >-200m et qu'il y a au moins un voisin avec une altitude positive, on indique que le flotteur est probablement proche de la terre.

### II.2.6 CONTROLE DES DERIVES

On établit des statistiques sur la dérive du flotteur en profondeur entre la dernière localisation du cycle n-1 et la première du cycle n, et en surface, entre la première et la dernière localisation du cycle n. Si la dérive est supérieure aux bornes données (bornesurf = 20000 pour système iridium et 100000 pour système Argos), la localisation est considérée comme douteuse.

On calcule :

```
distanceprof=a*acos(sin(dlat*pi/180)*sin(dlatlas*pi/180)+cos(dlat*pi/180)*cos(dlatlas*pi/180)*cos((dlonlas-dlon)*pi/180));
```

Avec dlatlas et dlonlas les latitudes et longitudes de la dernière loc du cycle précédent et dlat et dlon les latitudes et longitudes de la première loc du cycle n.

Si la vitesse calculée à partir de cette distance est supérieure à 3m/s on alerte en indiquant qu'une Loc Argos est probablement erronée.

On effectue le même type de calcul pour la surface :

```
distancesurf=a*acos(sin(dlatend*pi/180)*sin(dlat*pi/180)+cos(dlatend*pi/180)*cos(dlat*pi/180)*cos((dlon-dlonend)*pi/180));
```

avec dlatend et dlonend la latitude et longitude de la dernière loc de surface et dlat et dlat de la première loc de surface.
Si la distance est supérieure à bornesurf, qui varie selon le système de positionnement (iridium ou Argos), on alerte.

Afin de localiser ensuite les positions Argos erronées, on réalise un calcul de vitesses entre les locs de surface successives en utilisant une version légèrement modifiée d'un programme écrit par Jean-Philippe Rannou basé sur le passage du critère JAMSTEC sur un ensemble de positions Argos de surface (voir Kobayashi et al. 2008[1]). On récupère ainsi les indices des locs considérées comme erronées et on alerte.

[1] *Nakamura, T., Ogita, N., Kobayashi, T. : Quality control method of Argo float position data, JAMSTEC Report of Research and Development, 7, 11-18*

# III. FIN DES TESTS

A l'issue des tests, on récupère des variables utiles pour faire certaines statistiques ou vérifications dans deux fichiers nommés *variables_stat.mat* et *alertes_stat.mat*.

# IV. QUELQUES STATISTIQUES

## IV. 1 Exemples de stats sur certains paramètres.

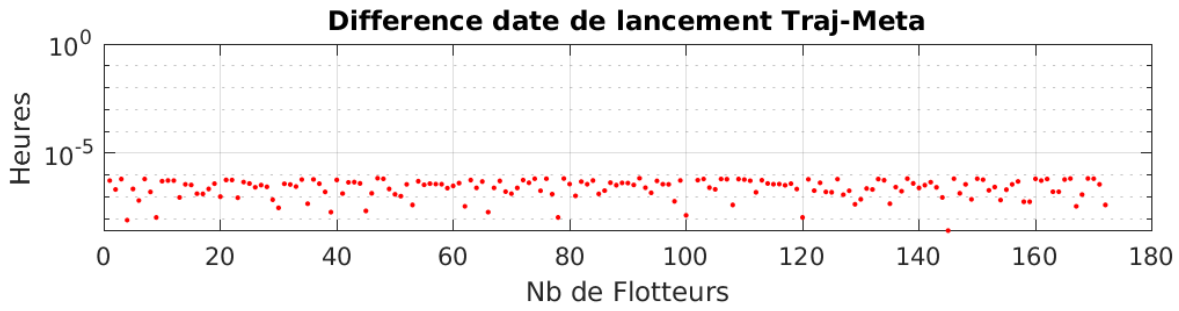


Fig. 1 – Différence en heures entre la date de lancement (LaunchDate) présent dans le fichier trajectoire et celle présente dans le fichier méta pour chaque flotteur, pour les flotteurs Apex d'AOML (172 flotteurs).

Ici l'information donnée sur la date de lancement dans le fichier trajectoire et le fichier méta était cohérente pour l'ensemble des flotteurs.

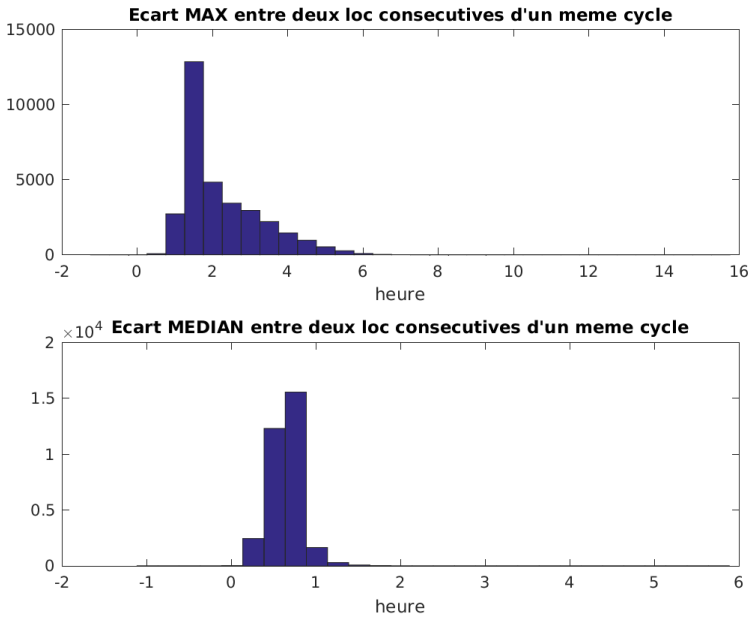


Fig.2 – Ecart max et écart médian (en heures) entre deux locs consécutives d'un même cycle, pour les flotteurs Apex d'AOML.

L'écart moyen entre deux locs d'un même cycle pour ce type de flotteurs est d'un peu moins d'une heure.

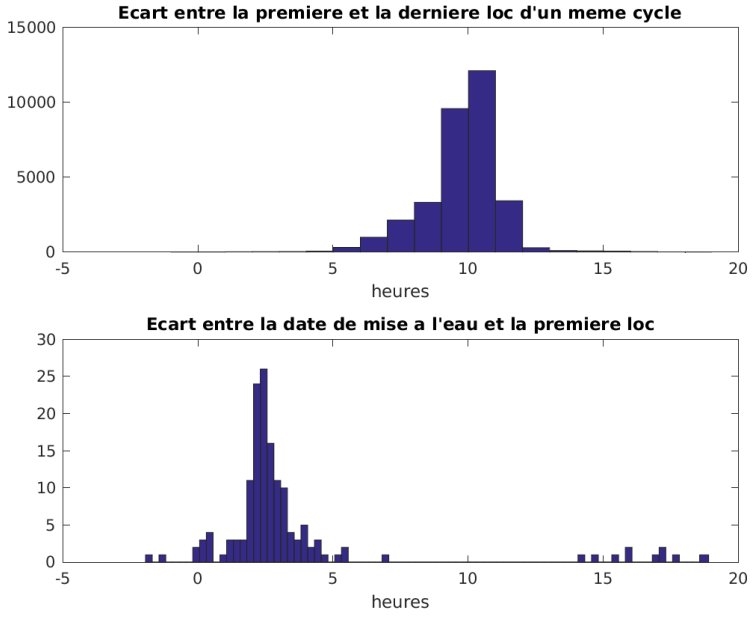


Fig.3 – haut : écart (en heures) entre la première et la dernière loc d'un même cycle ; bas : écart (en heures) entre la date de mise à l'eau (LaunchDate) et la première loc ; pour les flotteurs Apex d'AOML.

L'écart moyen entre la première et la dernière loc d'un même cycle pour ce type de flotteur (et pour les flotteurs Argos de façon générale) est ainsi d'~10h. L'écart moyen entre la de de mise à l'eau et la première loc est ici proche de 3h.

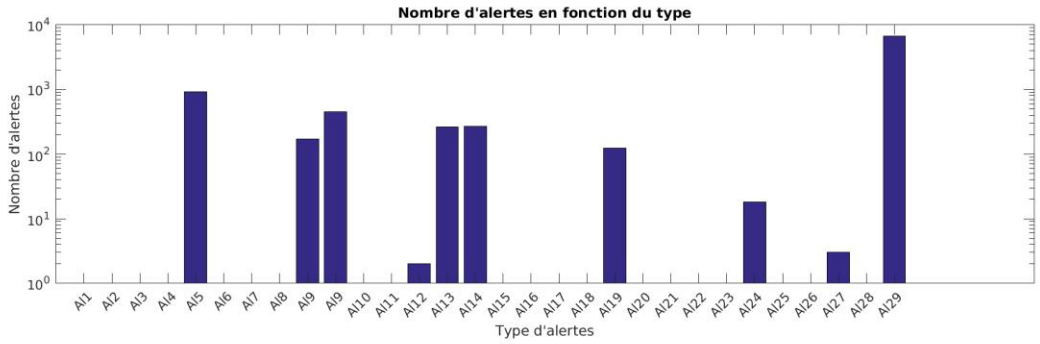


Fig.4 - Nombre d'alertes par type d'alertes pour les flotteurs Apex d'AOML.

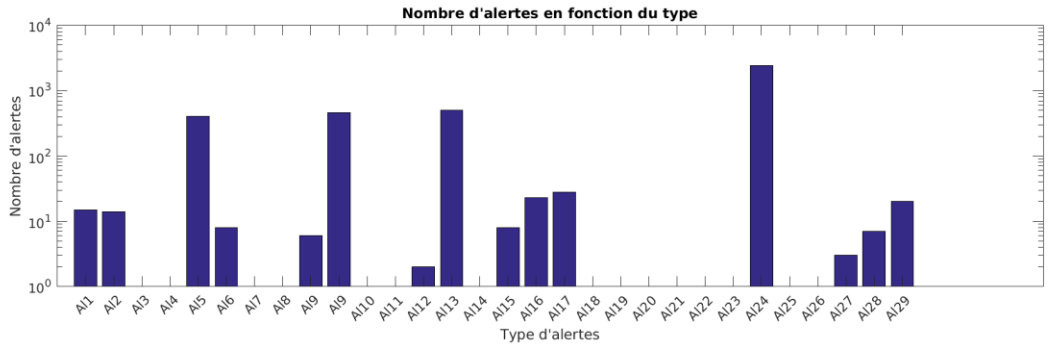


Fig.5 - Nombre d'alertes par type d'alertes pour les flotteurs iridium de Coriolis (137 flotteurs).

## IV.2 Comparaison des alertes issues du programme avec les alertes Altran.

Afin de faire quelques statistiques sur les alertes et de comparer les résultats obtenus avec ceux fournis par Altran, on fait tourner le programme pour chaque catégorie de flotteurs :

- AOML_RISER_apex_argos
- JMA_APEX_PI_JAMSTEC
- CSIRO_APEX_Argos_apf9
- CSIRO_APEX_Argos_apf8
- INCOIS_APEX_argos_apf8
- INCOIS_APEX_argos_apf9
- AOML_SOLOII_Dmode_all
- AOML_SOLOII_Dmode_all_bis

Le nombre de flotteurs à tester varie de 59 à 200 selon la catégorie. Le nombre total de flotteurs testés étant de 893. Pour chaque type de flotteur, on récupère les fichiers .txt pour chaque alerte. Par exemple pour le premier type (AOML_RISER ; 172 flotteurs) on aura : Alertes_cycles.txt, Alertes_grounded_srtm.txt, Alertes_pressure.txt. Chaque type de flotteur n'a pas nécessairement les mêmes types d'erreurs. Le même type de fichier .txt des alertes d'après Altran sont disponibles au préalable et seront comparés au fichier alertes du programme. Les alertes dans le programme ont été regroupées de façon à ce qu'elles incluent le plus fidèlement possible les alertes repérées par Altran, mais n'ayant pas le détail de ces dernières, la comparaison devient approximative pour les alertes de type « locpos », « cycle », « metatraj ».

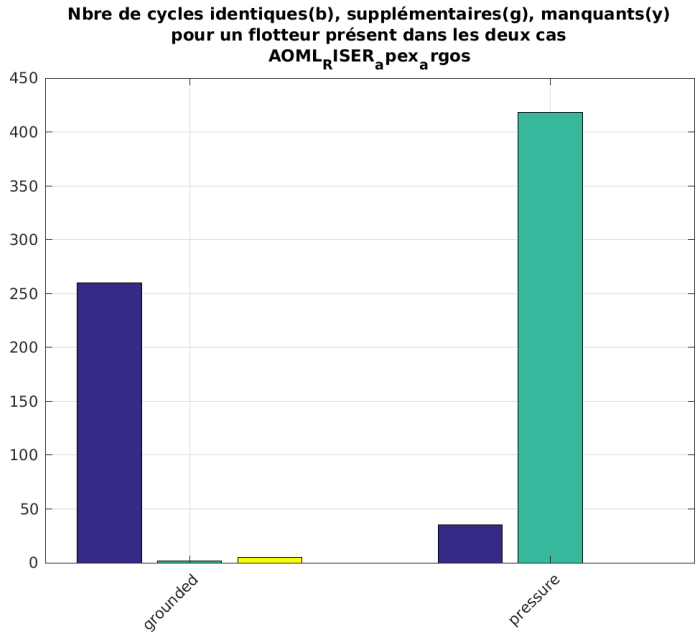


Fig.6 – Nombre de cycles identiques (bleu), supplémentaires (vert) et manquants (jaune) d'après le programme en comparaison avec Altran, pour un flotteur présent dans les deux cas ; pour l'alerte « Grounded » et « Pressure » et pour les flotteurs de type Apex (AOML).

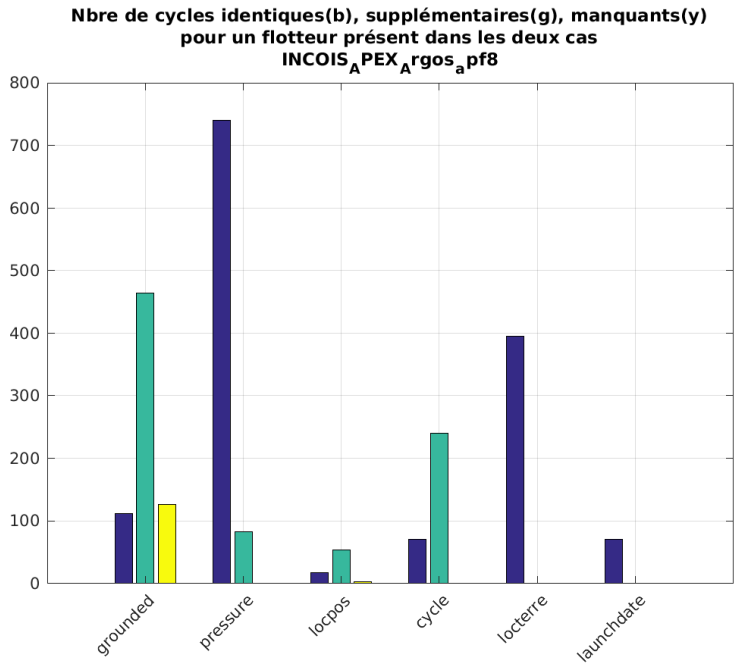


Fig.7 – Même chose que pour la figure 2 mais pour les alertes « grounded », « pressure », « locpos » , « cycle », locterre », « launchdate », pour les flotteurs de type Apex (INCOIS).

Les cycles manquants pour l'alerte « grounded » d'après le programme par rapport aux alertes Altran sont pour la plupart associés à des cas où il n'y pas de 'measurement code' 290, 299, qui indique des mesure en profondeur de parking. Il est donc logique que l'alerte « grounded » n'apparaisse pas (le fait qu'elle apparaisse dans les alertes Altran reste à déterminer).

Les alertes supplémentaires sont principalement dûes aux marges fixées dans le programme (voir section II.2.4- 4b ; +100 et -100), ainsi qu'à la bathy utilisée.

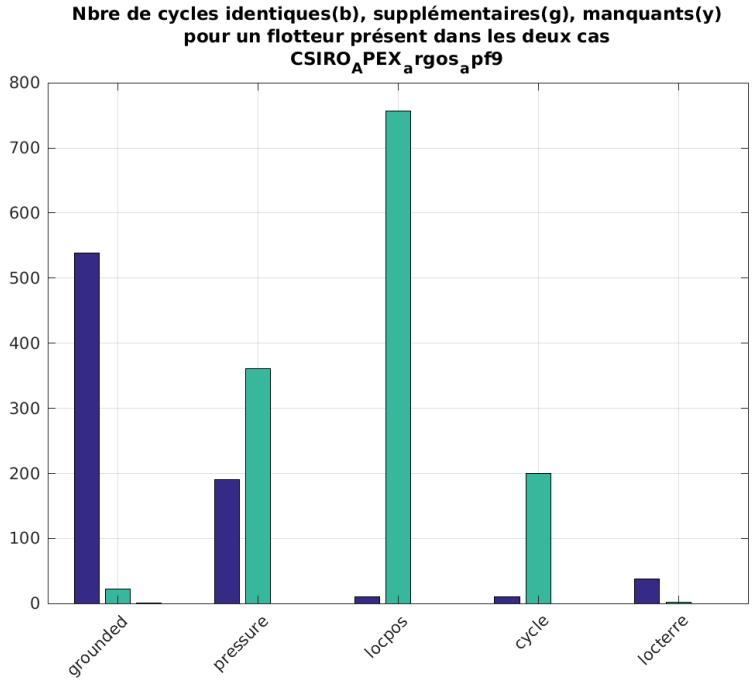


Fig.8 - Même chose que pour la figure 2 mais pour les alertes « grounded », « pressure », « locpos » , « cycle », « locterre », pour les flotteurs de type Apex (CSIRO).

De manière plus globale, la figure 4 montre le pourcentage de flotteurs identiques, supplémentaires et manquants pour l'alerte « grounded » pour l'ensemble des flotteurs.

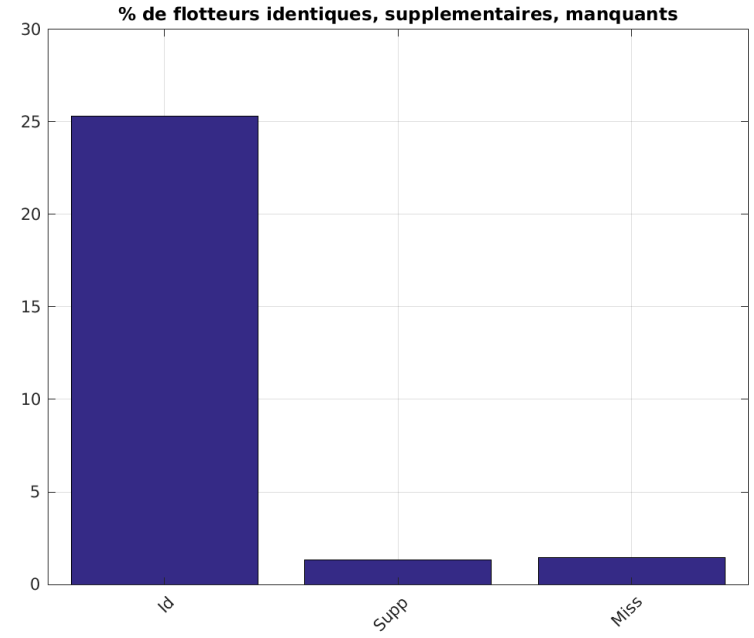


Fig.9 – Pourcentage de flotteurs identiques, supplémentaires, et manquants pour l'alerte « grounded » pour l'ensemble des flotteurs testés.