How to set a DOXY\_ADJUSTED\_ERROR in Coriolis NetCDF files

This note, that follows #RD4 recommendations, explains how the DOXY\_ADJUSTED\_ERROR information should be sent to Coriolis (to [Vincent.Bernard@ifremer.fr](mailto:Vincent.Bernard@ifremer.fr)) so that it will be propagated in Real Time into the profiles files and how the SCIENTIFIC\_CALIB\_COMMENT (of the DOXY parameter) will be set accordingly.

The PI must provide the estimated DOXY\_ADJUSTED\_ERROR of their floats together with the estimation method used (designated with a specific number from 0 to 3, see below).

The method number is then used by the decoder: 1- to determine the method used to propagate the provided error at the profile levels; 2- to set the SCIENTIFIC\_CALIB\_COMMENT associated to DOXY parameter.

#RD4 : DOI: <http://dx.doi.org/10.13155/46542>

**History**

|  |  |
| --- | --- |
| **Date (dd/mmm/yyyy)** | **Comment** |
| 02/04/2020 | Creation of the document by V. Racapé and Catherine Schmechtig |
| 19/04/2021 | Update to include the time dependence in the propagation error method as recommended by #RD4 |
| 21/01/2022 | Add information regarding the sub method # 3\_2 : Adjustment based on last valid DM adjustment that includes a refined pressure correction coefficient |

# List of available propagation error methods

**Case 1 : Constant error in PPOX**

Propagation ERROR Method # = 1

Description of the method: propagation error of 10mbar by default or provided by PI in mbar

DOXY\_ADJUSTED\_ERROR = [X] µmol/kg is recomputed from CALIB\_RT\_ADJUSTED\_ERROR

**Case 2 : increasing error in time in PPOX**

Propagation ERROR Method # = 2

Description of the method: propagation error of 10mbar by default or provided by PI in mbar with an increase of 1mbar per year since the last (RT or DM) adjustment

DOXY\_ADJUSTED\_ERROR = [X] µmol/kg is recomputed from CALIB\_RT\_ADJUSTED\_ERROR

# List of available estimation methods

**Method # = 1**

Description of the method: gain estimated from the comparison between in water PSAT or PPOX from float and PSAT or PPOX from WOAat mostinthe upper 20 dbar of the water column. WOA PPOX is computed from WOA PSAT and from TEMP and PSAL float data at the atmospheric pressure of 1 atm.

**Case 1\_1: Adjustment by comparison of in water float data to WOA based on PSAT or PPOX, error in PPOX**

SCIENTIFIC\_CALIB\_COMMENT = "DOXY\_ADJUSTED is computed from an adjustment of in water PSAT or PPOX float data at surface by comparison to WOA PSAT climatology or WOA PPOX in using PSATWOA and TEMPand PSALfloat at 1 atm, DOXY\_ADJUSTED\_ERROR is computed from a PPOX\_ERROR of xx.x mbar"

Propagation ERROR Method # = 1

**Case 1\_2: Adjustment by comparison of in water float data to WOA based on PSAT or PPOX, error in PPOX increasing with time**

SCIENTIFIC\_CALIB\_COMMENT = "DOXY\_ADJUSTED is computed from an adjustment of in water PSAT or PPOX float data at surface by comparison to woaPSAT climatology or woaPPOX{woaPSAT,floatTEMP,floatPSAL}at 1 atm, DOXY\_ADJUSTED\_ERROR is computed from a PPOX\_ERROR of xx.x mbar +1mb/year"

Propagation ERROR Method # = 2

**Method # = 2**

Description of the method: gain estimated from the comparison between in air PPOXfloat and PPOXNCEP.

**Case 2\_1: Adjustment by comparison of in air float data to NCEP reanalysis atmospheric data based on PPOX, error in PPOX**

SCIENTIFIC\_CALIB\_COMMENT = "DOXY\_ADJUSTED is estimated from an adjustment of in air PPOX float data by comparison to NCEP reanalysis, DOXY\_ADJUSTED\_ERROR is recomputed from a PPOX\_ERROR = xx.x mbar"

Propagation ERROR Method # = 1

**Case 2\_2: Adjustment by comparison of in air float data to NCEP reanalysis atmospheric data based on PPOX, error in PPOX increasing with time**

SCIENTIFIC\_CALIB\_COMMENT = "DOXY\_ADJUSTED is estimated from an adjustment of in air PPOX float data by comparison to NCEP reanalysis, DOXY\_ADJUSTED\_ERROR is recomputed from a PPOX\_ERROR = xx.x mbar with an increase of 1mbar/year"

Propagation ERROR Method # = 2

**Method # = 3**

Description of the method: Adjustment information (SLOPE – OFFSET – DRIFT – INCLINE\_T) provided by the last valid cycle with DM adjustment. In some cases, additional correction beyond the common scientific equation could be recorded in the SCIENTIFIC\_CALIB section by the expansion of the N\_CALIB dimension. In this way, the sub method 3\_2 takes into consideration the fine-tuning of the pressure correction coefficient as well.

**Case 3\_2: Adjustment based on last valid DM adjustment, error in PPOX increasing with time**

SCIENTIFIC\_CALIB\_COMMENT = "DOXY\_ADJUSTED is estimated from the last valid cycle with DM adjustment, DOXY\_ADJUSTED\_ERROR is recomputed from a PPOX\_ERROR = xx.x mbar with an increase of 1mbar/year"

Propagation ERROR Method # = 2

**Case 3\_2\_2 Adjustment based on last valid DM adjustment that includes a refined pressure correction coefficient, error in PPOX increasing with time**

SCIENTIFIC\_CALIB\_COMMENT = "DOXY\_ADJUSTED is estimated from the last valid cycle with DM adjustment that includes a refined pressure correction coefficient, DOXY\_ADJUSTED\_ERROR is recomputed from a PPOX\_ERROR = xx.x mbar with an increase of 1mbar/year"

Propagation ERROR Method # = 2

# Matlab code of DOXY real time adjustment

% ------------------------------------------------------------------------------

% Perform real time adjustment on DOXY profile data.

%

% SYNTAX :

% [o\_tabProfiles] = compute\_rt\_adjusted\_doxy(a\_tabProfiles, a\_launchDate, a\_decoderId)

%

% INPUT PARAMETERS :

% a\_tabProfiles : input profile structures

% a\_launchDate : float launch date

% a\_decoderId : float decoder Id

%

% OUTPUT PARAMETERS :

% o\_tabProfiles : output profile structures

%

% EXAMPLES :

%

% SEE ALSO :

% AUTHORS : Jean-Philippe Rannou (Altran)(jean-philippe.rannou@altran.com)

% ------------------------------------------------------------------------------

% RELEASES :

% 07/03/2019 - RNU - creation

% ------------------------------------------------------------------------------

function [o\_tabProfiles] = compute\_rt\_adjusted\_doxy(a\_tabProfiles, a\_launchDate, a\_decoderId)

% output parameters initialization

o\_tabProfiles = a\_tabProfiles;

% current float WMO number

global g\_decArgo\_floatNum;

% arrays to store RT offset information

global g\_decArgo\_rtOffsetInfo;

% global default values

global g\_decArgo\_dateDef;

global g\_decArgo\_nbHourForProfDateCompInRtOffsetAdj;

global g\_decArgo\_janFirst1950InMatlab;

% to store information on DOXY adjustment

global g\_decArgo\_paramAdjInfo;

global g\_decArgo\_paramAdjId;

% look for DOXY profiles

noDoxyProfile = 1;

for idProf = 1:length(o\_tabProfiles)

if (any(strcmp({o\_tabProfiles(idProf).paramList.name}, 'DOXY')))

noDoxyProfile = 0;

break

end

end

if (noDoxyProfile)

return

end

% retrieve information on DOXY adjustement in RT\_OFFSET information of META.json file

doSlope = '';

doOffset = '';

doDrift = '';

doInclineT = '';

doDate = '';

doAdjError = '';

doAdjErrorStr = '';

doAdjErrMethod = '';

if (~isempty(g\_decArgo\_rtOffsetInfo))

for idF = 1:length(g\_decArgo\_rtOffsetInfo.param)

if (strcmp(g\_decArgo\_rtOffsetInfo.param{idF}, 'DOXY'))

% mandatory fields

doSlope = g\_decArgo\_rtOffsetInfo.slope{idF};

doOffset = g\_decArgo\_rtOffsetInfo.value{idF};

doDate = g\_decArgo\_rtOffsetInfo.date{idF};

% not mandatory fields

if (isfield(g\_decArgo\_rtOffsetInfo, 'adjError'))

doAdjError = g\_decArgo\_rtOffsetInfo.adjError{idF};

doAdjErrorStr = g\_decArgo\_rtOffsetInfo.adjErrorStr{idF}{:};

doAdjErrMethod = g\_decArgo\_rtOffsetInfo.adjErrorMethod{idF}{:};

end

% new fields (possibly not filled for old adjustments)

doDrift = 0;

if (isfield(g\_decArgo\_rtOffsetInfo, 'drift'))

doDrift = g\_decArgo\_rtOffsetInfo.drift{idF};

end

doInclineT = 0;

if (isfield(g\_decArgo\_rtOffsetInfo, 'inclineT'))

doInclineT = g\_decArgo\_rtOffsetInfo.inclineT{idF};

end

break

end

end

end

% adjust DOXY profiles

if (~isempty(doSlope))

for idProf = 1:length(o\_tabProfiles)

profile = o\_tabProfiles(idProf);

if (any(strcmp({profile.paramList.name}, 'DOXY')) && ...

(profile.date ~= g\_decArgo\_dateDef) && ...

((profile.date + g\_decArgo\_nbHourForProfDateCompInRtOffsetAdj/24) >= doDate))

% retrieve associated profiles (needed for 'real' BGC floats since

% PTS are in separate profiles)

idProfs = find(([o\_tabProfiles.outputCycleNumber] == profile.outputCycleNumber) & ...

([o\_tabProfiles.direction] == profile.direction) & ...

([o\_tabProfiles.sensorNumber] < 100)); % AUX profiles should not be considered

% some cases need the PPOX\_ERROR to increase with time

startDateToIncreasePpoxErrorWithTime = '';

if (~isnan(doAdjError))

switch (doAdjErrMethod)

case {'1\_1', '2\_1'}

startDateToIncreasePpoxErrorWithTime = '';

case {'1\_2', '2\_2', '3\_2'}

startDateToIncreasePpoxErrorWithTime = doDate;

end

end

% adjust DOXY for this profile

[ok, profile] = adjust\_doxy\_profile(profile, o\_tabProfiles(setdiff(idProfs, idProf)), ...

doSlope, doOffset, doDrift, doInclineT, doAdjError, startDateToIncreasePpoxErrorWithTime, a\_launchDate, a\_decoderId);

if (ok)

profile.rtParamAdjIdList = [profile.rtParamAdjIdList g\_decArgo\_paramAdjId];

o\_tabProfiles(idProf) = profile;

% fill structure to store DOXY adjustment information

if (profile.direction == 'A')

direction = 2;

else

direction = 1;

end

equation = ['PPOX=f(DOXY), ' ...

'PPOX\_DOXY\_ADJUSTED=(SLOPE\*(1+DRIFT/100\*(profile\_date\_juld-launch\_date\_juld)/365)+INCLINE\_T\*TEMP)\*(PPOX\_DOXY+OFFSET), ' ...

'DOXY\_ADJUSTED=f(PPOX\_DOXY\_ADJUSTED)'];

coefficient = sprintf('OFFSET = %.2f, SLOPE = %.4f, DRIFT = %.3f, INCLINE\_T = %.6f, launch\_date\_juld = %s', ...

doOffset, doSlope, doDrift, doInclineT, datestr(a\_launchDate + g\_decArgo\_janFirst1950InMatlab, 'yyyymmddHHMMSS'));

comment = '';

if (~isnan(doAdjError))

switch (doAdjErrMethod)

case '1\_1'

comment = sprintf(['DOXY\_ADJUSTED is computed from an adjustment ' ...

'of in water PSAT or PPOX float data at surface by comparison to woaPSAT ' ...

'climatology or WOA PPOX in using woaPSAT and floatTEMP and PSAL at 1 atm, ' ...

'DOXY\_ADJUSTED\_ERROR is computed from a PPOX\_ERROR of %s mbar'], doAdjErrorStr);

case '1\_2'

comment = sprintf(['DOXY\_ADJUSTED is computed from an adjustment ' ...

'of in water PSAT or PPOX float data at surface by comparison to woaPSAT ' ...

'climatology or woaPPOX{woaPSAT,floatTEMP,floatPSAL} at 1 atm, ' ...

'DOXY\_ADJUSTED\_ERROR is computed from a PPOX\_ERROR of %s mbar +1mb/year'], doAdjErrorStr);

case '2\_1'

comment = sprintf(['DOXY\_ADJUSTED is estimated from an adjustment ' ...

'of in air PPOX float data by comparison to NCEP reanalysis, ' ...

'DOXY\_ADJUSTED\_ERROR is recomputed from a PPOX\_ERROR = %s mbar'], doAdjErrorStr);

case '2\_2'

comment = sprintf(['DOXY\_ADJUSTED is estimated from an adjustment ' ...

'of in air PPOX float data by comparison to NCEP reanalysis, ' ...

'DOXY\_ADJUSTED\_ERROR is recomputed from a PPOX\_ERROR = %s mbar ' ...

'with an increase of 1mbar/year'], doAdjErrorStr);

case '3\_2'

comment = sprintf(['DOXY\_ADJUSTED is estimated from the last valid cycle ' ...

'with DM adjustment, DOXY\_ADJUSTED\_ERROR is recomputed from a ' ...

'PPOX\_ERROR = %s mbar with an increase of 1mbar/year'], doAdjErrorStr);

otherwise

fprintf('ERROR: Float #%d Cycle #%d%c: input CALIB\_RT\_ADJ\_ERROR\_METHOD (''%s'') of DOXY adjustment is not implemented yet - SCIENTIFIC\_CALIB\_COMMENT of DOXY parameter not set\n', ...

g\_decArgo\_floatNum, ...

profile.outputCycleNumber, profile.direction, doAdjErrMethod);

end

end

date = datestr(doDate+g\_decArgo\_janFirst1950InMatlab, 'yyyymmddHHMMSS');

g\_decArgo\_paramAdjInfo = [g\_decArgo\_paramAdjInfo;

g\_decArgo\_paramAdjId profile.outputCycleNumber direction ...

{'DOXY'} {equation} {coefficient} {comment} {date}];

g\_decArgo\_paramAdjId = g\_decArgo\_paramAdjId + 1;

end

end

end

end

return

% ------------------------------------------------------------------------------

% Perform real time adjustment on one DOXY profile data.

%

% SYNTAX :

% [o\_ok, o\_profile] = adjust\_doxy\_profile(a\_profile, a\_tabProfiles, ...

% a\_slope, a\_offset, a\_doDrift, a\_doInclineT, a\_adjError, a\_adjDate, a\_launchDate, a\_decoderId)

%

% INPUT PARAMETERS :

% a\_profile : input DOXY profile structure

% a\_tabProfiles : profile structures with the same cycle number and

% direction as the DOXY one

% a\_slope : slope to be used for PPOX\_DOXY adjustment

% a\_offset : offset to be used for PPOX\_DOXY adjustment

% a\_doDrift : drift to be used for PPOX\_DOXY adjustment

% a\_doInclineT : incline\_t to be used for PPOX\_DOXY adjustment

% a\_adjError : error on PPOX\_DOXY adjusted values

% a\_adjDate : start date to apply adjustment

% a\_launchDate : float launch date

% a\_decoderId : float decoder Id

%

% OUTPUT PARAMETERS :

% o\_ok : 1 if the adjustment has been performed, 0 otherwise

% o\_profile : output DOXY profile structure

%

% EXAMPLES :

%

% SEE ALSO :

% AUTHORS : Jean-Philippe Rannou (Altran)(jean-philippe.rannou@altran.com)

% ------------------------------------------------------------------------------

% RELEASES :

% 07/03/2019 - RNU - creation

% ------------------------------------------------------------------------------

function [o\_ok, o\_profile] = adjust\_doxy\_profile(a\_profile, a\_tabProfiles, ...

a\_slope, a\_offset, a\_doDrift, a\_doInclineT, a\_adjError, a\_adjDate, a\_launchDate, a\_decoderId)

% output parameters initialization

o\_ok = 0;

o\_profile = [];

% current float WMO number

global g\_decArgo\_floatNum;

% QC flag values (numerical)

global g\_decArgo\_qcDef;

global g\_decArgo\_qcNoQc;

% lists of managed decoders

global g\_decArgo\_decoderIdListBgcFloatAll;

% involved parameter information

paramPres = get\_netcdf\_param\_attributes('PRES');

paramTemp = get\_netcdf\_param\_attributes('TEMP');

paramPsal = get\_netcdf\_param\_attributes('PSAL');

paramDoxy = get\_netcdf\_param\_attributes('DOXY');

% retrieve or interpolate PTS measurements

presValues = [];

tempValues = [];

psalValues = [];

doxyValues = [];

idPres = find(strcmp({a\_profile.paramList.name}, 'PRES'));

idTemp = find(strcmp({a\_profile.paramList.name}, 'TEMP'));

idPsal = find(strcmp({a\_profile.paramList.name}, 'PSAL'));

idDoxy = find(strcmp({a\_profile.paramList.name}, 'DOXY'));

if (~ismember(a\_decoderId, g\_decArgo\_decoderIdListBgcFloatAll))

% case of a PTSO float

presValues = a\_profile.data(:, idPres);

tempValues = a\_profile.data(:, idTemp);

psalValues = a\_profile.data(:, idPsal);

doxyValues = a\_profile.data(:, idDoxy);

else

% case of a 'real' BGC float

% create a PTS profile by concatenating the near-surface and the primary

% sampling profiles

idNssProf = [];

idPsProf = [];

for idProf = 1:length(a\_tabProfiles)

profile = a\_tabProfiles(idProf);

if (strncmp(profile.vertSamplingScheme, 'Near-surface sampling:', length('Near-surface sampling:')))

idNssPres = find(strcmp({profile.paramList.name}, 'PRES'));

idNssTemp = find(strcmp({profile.paramList.name}, 'TEMP'));

idNssPsal = find(strcmp({profile.paramList.name}, 'PSAL'));

if (~isempty(idNssPres) && ~isempty(idNssTemp) && ~isempty(idNssPsal))

idNssProf = idProf;

end

elseif (strncmp(profile.vertSamplingScheme, 'Primary sampling:', length('Primary sampling:')))

idPsPres = find(strcmp({profile.paramList.name}, 'PRES'));

idPsTemp = find(strcmp({profile.paramList.name}, 'TEMP'));

idPsPsal = find(strcmp({profile.paramList.name}, 'PSAL'));

if (~isempty(idPsPres) && ~isempty(idPsTemp) && ~isempty(idPsPsal))

idPsProf = idProf;

end

end

if (~isempty(idNssProf) && ~isempty(idPsProf))

break

end

end

if (~isempty(idNssProf) && ~isempty(idPsProf))

ctdPresData = [a\_tabProfiles(idPsProf).data(:, idPsPres); a\_tabProfiles(idNssProf).data(:, idNssPres)];

ctdTempData = [a\_tabProfiles(idPsProf).data(:, idPsTemp); a\_tabProfiles(idNssProf).data(:, idNssTemp)];

ctdPsalData = [a\_tabProfiles(idPsProf).data(:, idPsPsal); a\_tabProfiles(idNssProf).data(:, idNssPsal)];

elseif (~isempty(idPsProf))

ctdPresData = a\_tabProfiles(idPsProf).data(:, idPsPres);

ctdTempData = a\_tabProfiles(idPsProf).data(:, idPsTemp);

ctdPsalData = a\_tabProfiles(idPsProf).data(:, idPsPsal);

elseif (~isempty(idNssProf))

ctdPresData = a\_tabProfiles(idNssProf).data(:, idNssPres);

ctdTempData = a\_tabProfiles(idNssProf).data(:, idNssTemp);

ctdPsalData = a\_tabProfiles(idNssProf).data(:, idNssPsal);

else

ctdPresData = [];

ctdTempData = [];

ctdPsalData = [];

end

% clean fill values

idNoDefPts = find((ctdPresData ~= paramPres.fillValue) & ...

(ctdTempData ~= paramTemp.fillValue) & ...

(ctdPsalData ~= paramPsal.fillValue));

ctdPresData = ctdPresData(idNoDefPts);

ctdTempData = ctdTempData(idNoDefPts);

ctdPsalData = ctdPsalData(idNoDefPts);

if (~isempty(ctdPresData))

% interpolate and extrapolate the PTS data at the pressures of the

% DOXY measurements

ctdIntData = compute\_interpolated\_CTD\_measurements(...

[ctdPresData ctdTempData ctdPsalData], a\_profile.data(:, idPres), a\_profile.direction);

presValues = ctdIntData(:, 1);

tempValues = ctdIntData(:, 2);

psalValues = ctdIntData(:, 3);

doxyValues = a\_profile.data(:, idDoxy);

else

fprintf('WARNING: Float #%d Cycle #%d%c: unable to find the associated CTD profile to adjust DOXY parameter - DOXY data cannot be adjusted\n', ...

g\_decArgo\_floatNum, ...

a\_profile.outputCycleNumber, a\_profile.direction);

end

end

if (~isempty(presValues))

% adjust DOXY data

[doxyAdjValues, doxyAdjErrValues] = compute\_DOXY\_ADJUSTED( ...

presValues, tempValues, psalValues, doxyValues, ...

paramPres.fillValue, paramTemp.fillValue, paramPsal.fillValue, paramDoxy.fillValue, ...

a\_slope, a\_offset, a\_doDrift, a\_doInclineT, a\_launchDate, a\_adjError, a\_adjDate, a\_profile);

if (any(doxyAdjValues ~= paramDoxy.fillValue))

% create array for adjusted data

profile = a\_profile;

paramFillValue = get\_prof\_param\_fill\_value(profile);

if (isempty(profile.dataAdj))

profile.paramDataMode = repmat(' ', 1, length(profile.paramList));

profile.dataAdj = repmat(double(paramFillValue), size(profile.data, 1), 1);

profile.dataAdjQc = ones(size(profile.dataAdj, 1), length(profile.paramList))\*g\_decArgo\_qcDef;

end

if (isempty(profile.dataAdjError) && ~isempty(doxyAdjErrValues))

profile.dataAdjError = repmat(double(paramFillValue), size(profile.data, 1), 1);

end

% store adjusted data

profile.paramDataMode(idDoxy) = 'A';

profile.dataAdj(:, idDoxy) = doxyAdjValues;

idNoDef = find(doxyAdjValues ~= paramDoxy.fillValue);

profile.dataAdjQc(idNoDef, idDoxy) = g\_decArgo\_qcNoQc;

if (~isempty(doxyAdjErrValues))

profile.dataAdjError(:, idDoxy) = doxyAdjErrValues;

end

% output parameters

o\_ok = 1;

o\_profile = profile;

end

end

return

% ------------------------------------------------------------------------------

% Interpolate the T and S measurements of a CTD profile at given P levels.

%

% SYNTAX :

% [o\_ctdIntData] = compute\_interpolated\_CTD\_measurements( ...

% a\_ctdMeasData, a\_presData, a\_presData, a\_profDir)

%

% INPUT PARAMETERS :

% a\_ctdMeasData : CTD profile measurements

% a\_presData : P levels of T and S measurement interpolation

% a\_profDir : profile direction

%

% OUTPUT PARAMETERS :

% o\_ctdIntData : CTD interpolated data

%

% EXAMPLES :

%

% SEE ALSO :

% AUTHORS : Jean-Philippe Rannou (Altran)(jean-philippe.rannou@altran.com)

% ------------------------------------------------------------------------------

% RELEASES :

% 06/02/2014 - RNU - creation

% ------------------------------------------------------------------------------

function [o\_ctdIntData] = compute\_interpolated\_CTD\_measurements( ...

a\_ctdMeasData, a\_presData, a\_profDir)

% output parameters initialization

o\_ctdIntData = [];

if (isempty(a\_ctdMeasData))

return

end

paramPres = get\_netcdf\_param\_attributes('PRES');

paramTemp = get\_netcdf\_param\_attributes('TEMP');

paramSal = get\_netcdf\_param\_attributes('PSAL');

% get the measurement levels of output data

idNoDefOutput = find((a\_presData ~= paramPres.fillValue));

% interpolate the T and S measurements at the output P levels

idNoDefInput = find(~((a\_ctdMeasData(:, 1) == paramPres.fillValue) | ...

(a\_ctdMeasData(:, 2) == paramTemp.fillValue) | ...

(a\_ctdMeasData(:, 3) == paramSal.fillValue)));

if (~isempty(idNoDefInput))

% get PTS measurements

ctdPresData = a\_ctdMeasData(idNoDefInput, 1);

ctdTempData = a\_ctdMeasData(idNoDefInput, 2);

ctdPsalData = a\_ctdMeasData(idNoDefInput, 3);

% if it is a ascending profile, flip measurements up to down

% if (length(find(diff(ctdPresData)<0)) > length(ctdPresData)/2)

if (a\_profDir == 'A')

ctdPresData = flipud(ctdPresData);

ctdTempData = flipud(ctdTempData);

ctdPsalData = flipud(ctdPsalData);

end

if (length(ctdPresData) > 1)

% consider increasing pressures only (we start the algorithm from the middle

% of the profile)

idToDelete = [];

idStart = fix(length(ctdPresData)/2);

pMin = ctdPresData(idStart);

for id = idStart-1:-1:1

if (ctdPresData(id) >= pMin)

idToDelete = [idToDelete id];

else

pMin = ctdPresData(id);

end

end

pMax = ctdPresData(idStart);

for id = idStart+1:length(ctdPresData)

if (ctdPresData(id) <= pMax)

idToDelete = [idToDelete id];

else

pMax = ctdPresData(id);

end

end

ctdPresData(idToDelete) = [];

ctdTempData(idToDelete) = [];

ctdPsalData(idToDelete) = [];

end

if (~isempty(ctdPresData))

% duplicate T&S values 10 dbar above the shallowest level

ctdPresData = [ctdPresData(1)-10; ctdPresData];

ctdTempData = [ctdTempData(1); ctdTempData];

ctdPsalData = [ctdPsalData(1); ctdPsalData];

% duplicate T&S values 50 dbar below the deepest level

ctdPresData = [ctdPresData; ctdPresData(end)+50];

ctdTempData = [ctdTempData; ctdTempData(end)];

ctdPsalData = [ctdPsalData; ctdPsalData(end)];

tempIntData = interp1(ctdPresData, ...

ctdTempData, ...

a\_presData(idNoDefOutput), 'linear');

psalIntData = interp1(ctdPresData, ...

ctdPsalData, ...

a\_presData(idNoDefOutput), 'linear');

tempIntData(isnan(tempIntData)) = paramTemp.fillValue;

psalIntData(isnan(psalIntData)) = paramSal.fillValue;

% output parameters

o\_ctdIntData = [ ...

a\_presData ...

ones(length(a\_presData), 1)\*paramTemp.fillValue ...

ones(length(a\_presData), 1)\*paramSal.fillValue];

o\_ctdIntData(idNoDefOutput, 2) = tempIntData;

o\_ctdIntData(idNoDefOutput, 3) = psalIntData;

end

end

return

% ------------------------------------------------------------------------------

% Adjust DOXY measurements.

% DOXY\_ADJUSTED is estimated from an adjustment of PPOX\_DOXY at surface on WOA

% climatology.

%

% SYNTAX :

% [o\_DOXY\_ADJUSTED, o\_DOXY\_ADJUSTED\_ERROR] = compute\_DOXY\_ADJUSTED( ...

% a\_PRES, a\_TEMP, a\_PSAL, a\_DOXY, ...

% a\_PRES\_fillValue, a\_TEMP\_fillValue, a\_PSAL\_fillValue, a\_DOXY\_fillValue, ...

% a\_slope, a\_offset, a\_doDrift, a\_doInclineT, a\_launchDate, a\_adjError, a\_adjDate, a\_profOptode)

%

% INPUT PARAMETERS :

% a\_PRES : input PRES data

% a\_TEMP : input TEMP data

% a\_PSAL : input PSAL data

% a\_DOXY : input DOXY data

% a\_PRES\_fillValue : fill value for input PRES data

% a\_TEMP\_fillValue : fill value for input TEMP data

% a\_PSAL\_fillValue : fill value for input PSAL data

% a\_DOXY\_fillValue : fill value for input DOXY data

% a\_DOXY\_fillValue : fill value for input DOXY data

% a\_slope : slope of PPOX\_DOXY adjustment

% a\_offset : slope of PPOX\_DOXY adjustment

% a\_doDrift : drift to be used for PPOX\_DOXY adjustment

% a\_doInclineT : incline\_t to be used for PPOX\_DOXY adjustment

% a\_launchDate : float launch date

% a\_adjError : error on PPOX\_DOXY adjusted values

% a\_adjDate : start date to apply adjustment

% a\_profOptode : OPTODE profile structure

%

% OUTPUT PARAMETERS :

% o\_DOXY\_ADJUSTED : output DOXY adjusted data

% o\_DOXY\_ADJUSTED\_ERROR : output error on DOXY adjusted data

%

% EXAMPLES :

%

% SEE ALSO :

% AUTHORS : Jean-Philippe Rannou (Altran)(jean-philippe.rannou@altran.com)

% ------------------------------------------------------------------------------

% RELEASES :

% 07/04/2019 - RNU - creation

% ------------------------------------------------------------------------------

function [o\_DOXY\_ADJUSTED, o\_DOXY\_ADJUSTED\_ERROR] = compute\_DOXY\_ADJUSTED( ...

a\_PRES, a\_TEMP, a\_PSAL, a\_DOXY, ...

a\_PRES\_fillValue, a\_TEMP\_fillValue, a\_PSAL\_fillValue, a\_DOXY\_fillValue, ...

a\_slope, a\_offset, a\_doDrift, a\_doInclineT, a\_launchDate, a\_adjError, a\_adjDate, a\_profOptode)

% output parameters initialization

o\_DOXY\_ADJUSTED = ones(length(a\_DOXY), 1)\*a\_DOXY\_fillValue;

if (~isnan(a\_adjError))

o\_DOXY\_ADJUSTED\_ERROR = ones(length(a\_DOXY), 1)\*a\_DOXY\_fillValue;

else

o\_DOXY\_ADJUSTED\_ERROR = [];

end

% current float WMO number

global g\_decArgo\_floatNum;

% retrieve global coefficient default values

global g\_decArgo\_doxy\_202\_205\_304\_d0;

global g\_decArgo\_doxy\_202\_205\_304\_d1;

global g\_decArgo\_doxy\_202\_205\_304\_d2;

global g\_decArgo\_doxy\_202\_205\_304\_d3;

global g\_decArgo\_doxy\_202\_205\_304\_b0;

global g\_decArgo\_doxy\_202\_205\_304\_b1;

global g\_decArgo\_doxy\_202\_205\_304\_b2;

global g\_decArgo\_doxy\_202\_205\_304\_b3;

global g\_decArgo\_doxy\_202\_205\_304\_c0;

global g\_decArgo\_doxy\_202\_205\_304\_pCoef2;

global g\_decArgo\_doxy\_202\_205\_304\_pCoef3;

% global default values

global g\_decArgo\_dateDef;

if (isempty(a\_PRES) || isempty(a\_TEMP) || isempty(a\_PSAL) || isempty(a\_DOXY))

return

end

idDef = find( ...

(a\_PRES == a\_PRES\_fillValue) | ...

(a\_TEMP == a\_TEMP\_fillValue) | ...

(a\_PSAL == a\_PSAL\_fillValue) | ...

(a\_DOXY == a\_DOXY\_fillValue));

idNoDef = setdiff(1:length(a\_DOXY), idDef);

if (~isempty(idNoDef))

presValues = a\_PRES(idNoDef);

tempValues = a\_TEMP(idNoDef);

psalValues = a\_PSAL(idNoDef);

doxyValues = a\_DOXY(idNoDef);

% convert DOXY into DOXY\_in\_molar\_units

% units convertion (micromol/kg to micromol/L)

[measLon, measLat] = get\_meas\_location(a\_profOptode.cycleNumber, a\_profOptode.profileNumber, a\_profOptode);

rho = potential\_density\_gsw(presValues, tempValues, psalValues, 0, measLon, measLat);

rho = rho/1000;

molarDoxyValues = doxyValues .\* rho;

% pressure effect un-correction:

% at presValue, optode quenched by different pO2 inside membrane than pO2

% outside in seawater due to re-equilibration effect

% translate already corrected value (outside conditions) back to sensed value

% (inside membrane)

oxygenPresUncomp = calcoxy\_presuncomp(molarDoxyValues, presValues, tempValues, ...

g\_decArgo\_doxy\_202\_205\_304\_pCoef2, ...

g\_decArgo\_doxy\_202\_205\_304\_pCoef3 ...

);

% convert DOXY\_in\_molar\_units\_and\_inside\_conditions into PPOX\_DOXY

% units convertion (micromol/L to hPa)

ppoxDoxyValues = O2ctoO2p(oxygenPresUncomp, tempValues, psalValues, presValues, ...

g\_decArgo\_doxy\_202\_205\_304\_d0, ...

g\_decArgo\_doxy\_202\_205\_304\_d1, ...

g\_decArgo\_doxy\_202\_205\_304\_d2, ...

g\_decArgo\_doxy\_202\_205\_304\_d3, ...

g\_decArgo\_doxy\_202\_205\_304\_b0, ...

g\_decArgo\_doxy\_202\_205\_304\_b1, ...

g\_decArgo\_doxy\_202\_205\_304\_b2, ...

g\_decArgo\_doxy\_202\_205\_304\_b3, ...

g\_decArgo\_doxy\_202\_205\_304\_c0 ...

);

% adjust PPOX\_DOXY

if (a\_profOptode.date ~= g\_decArgo\_dateDef)

ppoxDoxyAdjValues = (a\_slope \* (1 + a\_doDrift/100 \* (a\_profOptode.date-a\_launchDate)/365) + a\_doInclineT\*tempValues) .\* (ppoxDoxyValues + a\_offset);

else

fprintf('WARNING: Float #%d Cycle #%d%c: profile is not dated - DOXY\_ADJUSTED set to FillValue\n', ...

g\_decArgo\_floatNum, ...

a\_profOptode.outputCycleNumber, a\_profOptode.direction);

end

% convert PPOX\_ADJUSTED into DOXY\_ADJUSTED\_in\_molar\_units\_and\_inside\_conditions

% units convertion (hPa to micromol/L)

oxygenAdjPresUncomp = O2ptoO2c(ppoxDoxyAdjValues, tempValues, psalValues, presValues, ...

g\_decArgo\_doxy\_202\_205\_304\_d0, ...

g\_decArgo\_doxy\_202\_205\_304\_d1, ...

g\_decArgo\_doxy\_202\_205\_304\_d2, ...

g\_decArgo\_doxy\_202\_205\_304\_d3, ...

g\_decArgo\_doxy\_202\_205\_304\_b0, ...

g\_decArgo\_doxy\_202\_205\_304\_b1, ...

g\_decArgo\_doxy\_202\_205\_304\_b2, ...

g\_decArgo\_doxy\_202\_205\_304\_b3, ...

g\_decArgo\_doxy\_202\_205\_304\_c0 ...

);

% pressure effect re-correction:

% at presValue, optode quenched by different pO2 inside membrane than pO2

% outside in seawater due to re-equilibration effect

% translate adjusted sensed value (inside membrane) to adjusted corrected

% value (outside conditions)

molarDoxyAdjValues = calcoxy\_prescomp(oxygenAdjPresUncomp, presValues, tempValues, ...

g\_decArgo\_doxy\_202\_205\_304\_pCoef2, ...

g\_decArgo\_doxy\_202\_205\_304\_pCoef3 ...

);

% convert DOXY\_ADJUSTED\_in\_molar\_units into DOXY\_ADJUSTED

% units convertion (micromol/L to micromol/kg)

doxyAdjValues = molarDoxyAdjValues ./ rho;

o\_DOXY\_ADJUSTED(idNoDef) = doxyAdjValues;

% compute DOXY\_ADJUSTED\_ERROR

if (~isnan(a\_adjError))

% use PPOX\_DOXY\_ADJUSTED\_ERROR from META-DATA

ppoxDoxyAdjErrValues = a\_adjError;

% increase PPOX\_DOXY\_ADJUSTED\_ERROR with time (1 mbar/year)

if (~isempty(a\_adjDate))

ppoxDoxyAdjErrValues = ppoxDoxyAdjErrValues + (a\_profOptode.date - a\_adjDate)/365;

end

% convert PPOX\_ADJUSTED\_ERROR into DOXY\_ADJUSTED\_ERROR\_in\_molar\_units\_and\_inside\_conditions

% units convertion (hPa to micromol/L)

oxygenAdjErrPresUncomp = O2ptoO2c(ppoxDoxyAdjErrValues, tempValues, psalValues, presValues, ...

g\_decArgo\_doxy\_202\_205\_304\_d0, ...

g\_decArgo\_doxy\_202\_205\_304\_d1, ...

g\_decArgo\_doxy\_202\_205\_304\_d2, ...

g\_decArgo\_doxy\_202\_205\_304\_d3, ...

g\_decArgo\_doxy\_202\_205\_304\_b0, ...

g\_decArgo\_doxy\_202\_205\_304\_b1, ...

g\_decArgo\_doxy\_202\_205\_304\_b2, ...

g\_decArgo\_doxy\_202\_205\_304\_b3, ...

g\_decArgo\_doxy\_202\_205\_304\_c0 ...

);

% pressure effect re-correction:

% at presValue, optode quenched by different pO2 inside membrane than pO2

% outside in seawater due to re-equilibration effect

% translate adjusted sensed value (inside membrane) to adjusted corrected

% value (outside conditions)

molarDoxyAdjErrValues = calcoxy\_prescomp(oxygenAdjErrPresUncomp, presValues, tempValues, ...

g\_decArgo\_doxy\_202\_205\_304\_pCoef2, ...

g\_decArgo\_doxy\_202\_205\_304\_pCoef3 ...

);

% convert DOXY\_ADJUSTED\_ERROR\_in\_molar\_units into DOXY\_ADJUSTED\_ERROR

% units convertion (micromol/L to micromol/kg)

doxyAdjErrValues = molarDoxyAdjErrValues ./ rho;

o\_DOXY\_ADJUSTED\_ERROR(idNoDef) = doxyAdjErrValues;

end

end

return

% ------------------------------------------------------------------------------

% Undo Correct DO (in micromol/L) from pressure effect.

%

% SYNTAX :

% [o\_oxygen] = calcoxy\_presuncomp(a\_oxygenPrescomp, a\_pres, a\_temp, ...

% a\_pCoef2, a\_pCoef3)

%

% INPUT PARAMETERS :

% o\_oxygenPrescomp : DO values (in micromol/L) corrected from pressure effect

% a\_pres : PRES values

% a\_temp : TEMP values

% a\_pCoef2 and a\_pCoef3 : additional coefficient values

%

% OUTPUT PARAMETERS :

% a\_oxygen : DO values

%

% EXAMPLES :

%

% SEE ALSO :

% AUTHORS : Jean-Philippe Rannou (Altran)(jean-philippe.rannou@altran.com)

% ------------------------------------------------------------------------------

% RELEASES :

% 05/20/2011 - Virginie THIERRY - creation

% 05/17/2016 - RNU - update

% ------------------------------------------------------------------------------

function [o\_oxygen] = calcoxy\_presuncomp(a\_oxygenPrescomp, a\_pres, a\_temp, ...

a\_pCoef2, a\_pCoef3)

% pressure compensation correction

o\_oxygen = a\_oxygenPrescomp ./ (1 + ((a\_pCoef2 .\* a\_temp) + a\_pCoef3) .\* a\_pres/1000);

return

% ------------------------------------------------------------------------------

% Correct DO (in micromol/L) from pressure effect.

%

% SYNTAX :

% [o\_oxygenPrescomp] = calcoxy\_prescomp(a\_oxygen, a\_pres, a\_temp, ...

% a\_pCoef2, a\_pCoef3)

%

% INPUT PARAMETERS :

% a\_oxygen : DO values

% a\_pres : PRES values

% a\_temp : TEMP values

% a\_pCoef2 and a\_pCoef3 : additional coefficient values

%

% OUTPUT PARAMETERS :

% o\_oxygenPrescomp : DO values (in micromol/L) corrected from pressure effect

%

% EXAMPLES :

%

% SEE ALSO :

% AUTHORS : Jean-Philippe Rannou (Altran)(jean-philippe.rannou@altran.com)

% ------------------------------------------------------------------------------

% RELEASES :

% 05/20/2011 - Virginie THIERRY - creation

% 05/17/2016 - RNU - update

% ------------------------------------------------------------------------------

function [o\_oxygenPrescomp] = calcoxy\_prescomp(a\_oxygen, a\_pres, a\_temp, ...

a\_pCoef2, a\_pCoef3)

% pressure compensation correction

o\_oxygenPrescomp = a\_oxygen .\* (1 + ((a\_pCoef2 .\* a\_temp) + a\_pCoef3) .\* a\_pres/1000);

return