DATA PRODUCT SPECIFICATION FOR DENSITY

Version 1-01
Document Control Number 1341-00050
2012-01-06

Consortium for Ocean Leadership
1201 New York Ave NW, 4th Floor, Washington DC 20005
www.OceanLeadership.org

in Cooperation with

University of California, San Diego
University of Washington
Woods Hole Oceanographic Institution
Oregon State University
Scripps Institution of Oceanography
Rutgers University

Document Control Sheet

Version	Date	Description	Author
0-01	2011-12-13	Initial Draft	L. Heilman
1-00	2011-12-15	Initial Release	E. Chapman
1-01	2012-01-06	Changed PMO signature authority to Chief Project Scientist; corrected absolute salinity anomaly def'n	S. Webster

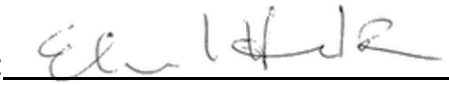
Signature Page

This document has been reviewed and approved for release to Configuration Management.

OOI Chief Systems Engineer: 

Date: 15 Dec 11

This document has been reviewed and meets the needs of the OOI Cyberinfrastructure for the purpose of coding and implementation.

OOI CI Signing Authority: 

Date: 12-15-11

TABLE OF CONTENTS

1 Abstract

This document describes the computation used to calculate the OOI Level 2 Density core data product, which is calculated using the Thermodynamic Equations of Seawater – 2010 (TEOS-10) with data from the conductivity, temperature and depth (CTD) family of instruments. This document is intended to be used by OOI programmers to construct appropriate processes to create the L2 density product.

2 Introduction

2.1 Author Contact Information

Please contact Sarah Webster (swebster@oceanleadership.org) or the Data Product Specification lead (DPS@lists.oceanobservatories.org) for more information concerning the algorithm and other items in this document.

2.2 Metadata Information

2.2.1 Data Product Name

The OOI Core Data Product Name for this product is

- DENSITY

The OOI Core Data Product Descriptive Name for this product is

- Density

2.2.2 Data Product Abstract (for Metadata)

The OOI Level 2 Density core data product is computed using the TEOS-10 equations with data from the conductivity, temperature and depth (CTD) family of instruments.

2.2.3 Computation Name

Not required for data product algorithms.

2.2.4 Computation Abstract (for Metadata)

This algorithm computes the OOI Level 2 Density core data product, which is calculated using the TEOS-10 equations with data from the conductivity, temperature and depth (CTD) family of instruments.

2.2.5 Instrument-Specific Metadata

See Section 4.4 for instrument-specific metadata that must be included with the output of this algorithm.

2.2.6 Synonyms

Synonyms for this data product are

- Density
- Seawater density

2.2.7 Similar Data Products

Similar products that this data product may be confused with are density calculated using equation-of-state definition EOS-80, or any other equation-of-state definitions that are not TEOS-10. In particular, density as calculated by the Sea-Bird instruments using the company's proprietary SeaSoft software is calculated using the EOS-80 equation-of-state and will give a different density than that calculated by the computation described herein.

Additional information regarding the similar products can be found at the TEOS-10 website (http://www.teos-10.org) and references listed in Section 2.4.

2.3 Instruments

For information on the instruments from which the L2 Density core data product inputs are obtained, see the CTD Processing Flow document (DCN 1342-00001). This document contains information on instrument classes and make/models; it also describes the flow of data from the CTDs through all of the relevant QC, calibration, and data product computations and procedures.

Please see the Instrument Application in the SAF for specifics on instrument locations and platforms.

2.4 Literature and Reference Documents

Feistel, R. (2008). "A Gibbs function for seawater thermodynamics for -6 to 80◦ C and salinity up to 120 g kg^{-1}." Deep Sea Research **I**(55): 1639-1671.

Fofonoff, N. P. and J. R.C. Millard (1983). "Algorithms for computation of fundamental properties of seawater." UNESCO technical papers in marine science **44**: 1-53.

Gill, A. E. (1982). Atmosphere-Ocean Dynamics. New York, Academic Press.

IOC, et al. (2010). The international thermodynamic equation of seawater-2010: Calculation and use of thermodynamic properties, UNESCO.

McDougal, T. J., et al. (2009). "An algorithm for estimating Absolute Salinity in the global ocean." Ocean Science Discussions **6**: 215-242.

McDougall T.J., P.M. Barker, R. Feistel and D.R. Jackett (2011). “A computationally efficient 48-term expression for the density of seawater in terms of Conservative Temperature, and related properties of seawater.” To be submitted to Ocean Science Discussions.

Millero, F. J., et al. (2008). "The composition of Standard Seawater and the definition of the Reference-Composition Salinity Scale." Deep Sea Research **I**(55): 50-72.

Pawlowicz, R. (2010). What every oceanographer needs to know about TEOS-10 (The TEOS-10 Primer). *Thermodynamic Equation Of Seawater - 2010 (TEOS-10) website: http://www.teos-10.org/*

Sea-Bird (2009). Absolute Salinity and TEOS-10: Sea-Bird's Implementation Plans. Application Note 90, Sea-Bird Electronics, Inc.

TEOS-10 (2011). Getting started with TEOS-10 and the GSW Oceanographic Toolbox. *http://www.teos-10.org/software.htm#1*

2.5 Terminology

2.5.1 Definitions

The following terms are defined here for use throughout this document. Definitions of general OOI terminology are contained in the Level 2 Reference Module in the OOI requirements database (DOORS).

Density (of seawater) (ρ): Mass per unit volume in SI units of kg/m^3. In the Thermodynamic Equations of State (TEOS) for seawater, it is calculated as the reciprocal of specific volume as defined below.

Practical Salinity (S_P): The measure of salinity defined by the Practical Salinity Scale 1978 (PSS-78). Practical Salinity is a unitless quantity that is approximately equivalent to the mass fraction of dissolved solute in seawater, but is not interchangeable with Absolute Salinity. Practical Salinity is an analogue for conductivity of seawater adjusted for temperature and pressure.

Absolute Salinity (S_A): Measure of absolute salinity adopted and defined jointly by the Intergovernmental Oceanographic Commission (IOC), International Association for the Physical Sciences of the Oceans (IAPSO) and the Scientific Committee on Oceanic Research (SCOR) in 2010 as part of the new standard for calculating the thermodynamic properties of seawater. Units of Absolute Salinity are g kg^{-1}, the mass fraction of dissolved salts in seawater. Absolute Salinity represents, to the best available accuracy, the mass fraction of dissolved solute in a sample of Standard Seawater of the same density as the observed sample (Pawlowicz, 2010).

Density Salinity (S_A^{dens}): Another term for Absolute Salinity.

Absolute Salinity anomaly (δS_A): A geographically-dependent offset in the linear scaling of absolute to practical salinity.

2.5.2 Acronyms, Abbreviations and Notations

General OOI acronyms, abbreviations and notations are contained in the Level 2 Reference Module in the OOI requirements database (DOORS). The following acronyms and abbreviations are defined here for use throughout this document.

TEOS-10	Thermodynamic Equations of Seawater – 2010
EOS-80	Equations of State of Seawater adopted by UNESCO in 1980 (the standard before TEOS-10)
GSW	Gibbs seawater or Gibbs function for seawater
ITS-90	International Temperature Scale of 1990
PSS-78	Practical Salinity Scale 1978

2.5.3 Variables and Symbols

The following variables and symbols are defined here for use throughout this document.

ρ, `rho`	Density
p, `p`	Sea pressure (*in situ*)
t, T, `t`	Temperature (*in situ*)
`CT`	Temperature (conservative)
S_A, `SA`	Absolute salinity
S_P, `SP`	Practical salinity
g	Gibbs function of seawater
g^W	Pure liquid water part of Gibbs function of seawater
g^S	Saline part of Gibbs function of seawater
φ, `lat`	Latitude
λ, `long`	Longitude

3 Theory

3.1 Description

The density of seawater is a function of temperature, salinity, pressure, and latitude and longitude. Temperature, salinity, pressure are the Level 1 sensor products from the CTD instrument. Latitude and longitude are metadata associated with the Level 1 sensor products.

Historically, the formula for calculating density has been based on practical salinity (S_P). A description of practical salinity as defined in 1980 by the UNESCO/ICES/SCOR/IAPSO joint committee is found in Fofonoff and Millard (1983), Gill (1982) and in the OOI Salinity Data Product Specification (DCN 1341-00040).

In 2010 the Intergovernmental Oceanographic Commission (IOC) of UNESCO convened and developed a more accurate calculation for salinity, denoted absolute salinity (S_A), which takes into account the previously ignored spatial differences in solute composition across the different ocean basins (Feistel, 2008) and provides a better estimate of the mass fraction of salts in seawater (IOC, 2010; McDougal, 2009). In concert with this update to the salinity calculation the Intergovernmental Oceanographic Commission (IOC) of UNESCO also revised the formula for calculating density based on the TEOS-10 Gibbs function for the equation of state of seawater, which is a function of absolute salinity (S_A).

This standard is being adopted by the worldwide scientific community and is the standard that is used by OOI. The Gibbs formulation is the most accurate density approximation formula in use today. Absolute salinity will be calculated within this algorithm to enable the calculation of density using the Gibbs formulation.

3.2 Mathematical Theory

Detailed derivations of these formulae can be found in the references cited.

Density (ρ) is a function of absolute salinity (S_A), *in situ* temperature (t), and *in situ* sea pressure (p).

$$\rho = \rho(S_A, t, p)$$

Absolute salinity (S_A) is defined in terms of practical salinity (S_P) and absolute salinity anomaly (δS_A),

$$S_A = (35.16504\ g\ \ kg^{-1}/35)S_P + \delta S_A(\varphi, \lambda, p)$$

where δS_A is found through a look up table using latitude (φ), longitude (λ), and *in situ* sea pressure (p) in dbar (IOC et al, 2010).

Density is given by the inverse of the pressure derivative of the Gibbs function (g) at constant absolute salinity and *in situ* temperature

$$\rho(S_A, t, p) = \left(\frac{\partial g}{\partial P}\Big|_{S_A,T}\right)^{-1}$$

where ∂P is the pressure derivative and T is used to represent *in situ* temperature in the partial derivative to avoid confusing t with time.

The Gibbs function (g) is defined as the sum of a pure liquid water part (g^W) and a saline part (g^S).

$$g(S_A, t, p) = g^W(t, p) + g^S(S_A, t, p)$$

Both g^W and g^S are calculated using multi-term polynomials, whose details are described in the TEOS-10 Manual. There are several different standards that can be used to define the coefficients in these polynomials. As per the suggestion in the TEOS-10 Manual, OOI has adopted the IAPWS-09 standard for calculating the pure liquid water part of the Gibbs function and the IAPWS-08 standard for calculating the saline part of the Gibbs function. For the exact coefficients and a discussion of the alternate standards and their relative merits the reader is directed to the TEOS-10 Manual.

See (Feistel, 2008) for a more detailed discussion of the Gibbs function. See (McDougal, 2009) for a discussion of the estimation of the absolute salinity anomaly. See (McDougal, 2011) for a

discussion of the estimation of density. See (IOC et al, 2010) for a more detailed discussion of all of the above.

3.3 Known Theoretical Limitations

The theoretical limitations of the density calculation are defined by the limitations of the Gibbs function. The limitations of the Gibbs function in turn depend on the standard chosen to calculate the pure liquid water and the saline parts. As noted in the previous section OOI has adopted the standards IAPWS-09 and IAPWS-08 for calculating the pure water and saline parts respectively, leading to the following limitations.

Absolute salinity:	$0 < S_A < 42$ g/kg
Temperature:	freezing temperature of seawater** $< t <$ 40 °C
Pressure:	$0 < p < 10^4$ dbar

** Specifically, the lower limit is $-(2.65 + (p + 0.101325) * 0.0743)$ °C for p given in MPa, as per the IAPWS-09 standard for the pure liquid water component of the Gibbs function.

Note that this a limitation based on absolute salinity, which is an intermediate calculation in this computation of density, not practical salinity, the input to the algorithm.

3.4 Revision History

No revisions to date.

4 Implementation

4.1 Overview

The density algorithm is implemented using a computationally efficient 48-term polynomial expression. A complete description of the 48-term polynomial can be found in IOC et al. (2010). The example Matlab code provided in the Appendix implements the 48-term polynomial. All of the functions described here are from the GSW Toolbox provided at the TEOS-10 website (http://www.teos-10.org). As of writing, GSW Toolbox 3.0 is available in Matlab, C, and Fortran. Appendix A contains copies of all of the functions called to document exactly how this function was implemented.

Note that according to the TEOS-10 Manual for the GSW Toolbox used to implement the calculation for density (and absolute salinity in order to calculate density):

> *The thermodynamic description of seawater and of ice Ih as defined in IAPWS-08 and IAPWS-06 has been adopted as the official description of seawater and of ice Ih by the Intergovernmental Oceanographic Commission in June 2009. These thermodynamic descriptions of seawater and ice were endorsed recognizing that the techniques for estimating Absolute Salinity will likely improve over the coming decades. The algorithm for evaluating Absolute Salinity in terms of Practical Salinity, latitude, longitude and pressure, will likely be updated from time to time, after relevant appropriately peer-reviewed publications have appeared, and such an updated algorithm will appear on the www.TEOS-10.org web site. Users of this software should state in their published work which version of the software was used to calculate Absolute Salinity.*
>
> *- IOC et al, (2010), p.15*

Therefore it is imperative that the version of the GSW Toolbox used to perform the calculation be cited by scientists using this data. This information, as noted in Section 4.4, will be included as part of the metadata. This information will enable future science users to understand precisely how density was calculated, in the event that the GSW Toolbox used to run this algorithm is updated during the lifetime of the observatory.

4.2 Inputs

Inputs

- L1 Temperature [°C] (see 1341-00010_Data Product_SPEC_TEMPWAT)
- L1 Pressure (sea pressure) [dbar] (1341-00020_Data_Product_SPEC_PRESWAT)
- L2 Practical Salinity (PSS-78) [unitless]
  (see 1341-00040_Data_Product_SPEC_PRACSAL)
- Latitude and longitude where the input data was collected is required. For the SBE 37IM and the SBE 16plusV2 this information is the lat/long of the mooring on which the instrument is fixed and is part of the metadata. For the SBE GPCTD this information is from the position of the glider or AUV when the CTD data were collected. The presentation of data from the OMC to CI is still TBD and this interface, including how glider position is stored and made accessible, needs to be worked out between CG and CI with help from the PMO. Once decided this section will be updated by the DPS author.

All inputs are double precision floating point numbers.

The computation described here in only produces valid results when the inputs are within the range of standard oceanographic values:

Absolute salinity:	0 < SA < 42 g/kg
Temperature:	-2.65** < t < 40 °C
Pressure:	0 < p < 10,000 dbar

** Technically, the lower limit is `-(2.65 + (p + 0.101325)*0.0743)` degrees C for `p` given in MPa, as per the IAPWS-09 standard for the pure liquid water component of the Gibbs function. However, a lower limit of – 2.65 degrees C can be used as a static replacement for OOI implementation.

Range checks on the inputs to this algorithm will have already been applied as part of the global range check (GLBLRNG, DCN 1341-10004) specified in the CTD Processing Flow document (DCN 1342-0001). A separate range check on the inputs for this algorithm does not need to be applied.

Input Data Formats
Not applicable for Level 2 data products.

4.3 Processing Flow

The specific steps necessary to create all calibrated and quality controlled data products for the CTD are described in the CTD Processing Flow document (DCN 1342-00001). This processing flow document contains a flow diagram showing all of the specific algorithms (data products and QC) necessary to compute all data products from the CTD and the order in which the algorithms must be applied.

The processing flow for the density algorithm code is as follows (in Matlab syntax):

Step 1:
Absolute salinity (`SA`) is calculated from practical salinity (`SP`, L2 input), sea pressure (`p`, L1 input), latitude (`lat`, L1 input metadata) and longitude (`long`, L1 input metadata) using the function

```
[SA, in_ocean] = gsw_SA_from_SP(SP,p,long,lat)
```

where `in_ocean` is a flag indicating that the lat/long is well inside the boundaries of dry land.

Step 2:
Conservative temperature is computed from absolute salinity (`SA`, step 1 output), *in situ* temperature (`t`, L1 input), and sea pressure (`p`, L1 input).

```
CT = gsw_CT_from_t(SA,t,p)
```

Step 3:
In situ density (`rho`) is calculated by the function

```
rho = gsw_rho(SA,CT,p)
```

from absolute salinity (`SA`, step 1 output), conservative temperature (`CT`, step 2 output), and sea pressure (`p`, L1 input), using the computationally-efficient 48-term expression for density (McDougall et al., 2011).

Note that there are two cases in which the functions called during this algorithm automatically perform quality control checks, neither of which are used in this processing flow:

1) The function `gsw_SA_from_SP`, which calculates absolute salinity from practical salinity, changes negative values of practical salinity to zero before performing the calculation because practical salinity is by definition positive. Negative salinity values will already have been flagged as bad data by the L2 salinity algorithm, therefore nothing additional needs to be done.
2) The function `gsw_SA_from_SP` also returns a flag (`in_ocean`) to indicate if the latitude and longitude are far within the boundaries of land. We are ignoring this flag because it indicates a bad position value, not a bad density value, and the position information is flagged elsewhere.

4.4 Outputs

The output of the density computation is

- Density of seawater in kg m^{-3} as a double precision floating point number.

The metadata that must be included with the output are

- the version of the GSW software used (GSW Toolbox 3.0)
- the version of the Matlab execution engine, if used.

See Appendix B for a discussion of the accuracy of the algorithm output.

4.5 Computational and Numerical Considerations

4.5.1 Numerical Programming Considerations

None.

4.5.2 Computational Requirements

Not significant.

4.6 Code Verification and Test Data Sets

The algorithm code will be verified using the test data set provided, which contain inputs and their associated correct outputs. CI will verify that the algorithm code is correct by checking that the algorithm density output, generated using the test data inputs, is identical to the test data output, and that the correct metadata have been applied.

A test data set below provides a few data points over the extent of the water column. It includes inputs, outputs, and the intermediate calculation of Absolute Salinity.

Inputs					Interm. Calc	Output
Practical Salinity [PSS-78]	Temp [deg C] [ITS-90]	Sea Pressure [dbar]	Latitude [deg N]	Longitude [deg E]	Absolute Salinity [TEOS-10]	Density [kg m-3] [TEOS-10]
33.5	28	0	15.00	-55.00	33.65829	1021.26851
33.5	28	10	15.00	-55.00	33.65832	1021.31148
37	20	150	15.00	-55.00	37.17493	1026.94422
34.9	6	800	15.00	-55.00	35.06912	1031.13498
35	3	2500	15.00	-55.00	35.16948	1039.28768
35	2	5000	15.00	-55.00	35.17522	1050.30616

Also, the output should have new metadata fields containing the version of the GSW Toolbox used (GSW Toolbox 3.0) and the version of the Matlab engine, if used.

Appendix A Example Code

This Appendix contains all Matlab subroutines necessary for calculating Density from Temperature, Salinity, Pressure using the GSW toolbox (TEOS-10) available under Creative Commons license from http://www.teos-10.org/. This code has been verified by the originators of the code using examples from the oceanographic community. The current version at the time of writing is GSW Toolbox 3.0.

Matlab, Fortran, and C routines for calculating density are available in the GSW toolbox from the TEOS-10 website.

http://www.teos-10.org/
http://www.teos-10.org/software.htm#1

A copy of the toolbox has been placed on Alfresco.
REFERENCE >> DPS Artifacts >> DENSITY_Code_gsw_matlab_v30_0.zip

A.1 In-situ Density (48-term equation)

USAGE:	rho = gsw_rho(SA,CT,p)
DESCRIPTION:	Calculates in-situ density from Absolute Salinity and Conservative Temperature, using the computationally-efficient 48-term expression for density in terms of SA, CT and p (McDougall et al., 2011). Note that potential density with respect to reference pressure, p_ref, is obtained by calling this function with the pressure argument being p_ref (i.e. "gsw_rho(SA,CT,p_ref)"). Note that the 48-term equation has been fitted in a restricted range of parameter space, and is most accurate inside the "oceanographic funnel" described in McDougall et al. (2011). The GSW library function "gsw_infunnel(SA,CT,p)" is available to be used if one wants to test if some of one's data lies outside this "funnel".
INPUT:	SA = Absolute Salinity [g/kg] CT = Conservative Temperature [deg C] p = sea pressure [dbar] (i.e. absolute pressure - 10.1325 dbar) SA & CT need to have the same dimensions. p may have dimensions 1x1 or Mx1 or 1xN or MxN, where SA & CT are MxN.
OUTPUT:	rho = in-situ density [kg m^-3]
AUTHOR:	Paul Barker and Trevor McDougall
VERSION NUMBER:	3.0 (23rd May, 2011)
REFERENCES:	IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of seawater - 2010: Calculation and use of thermodynamic properties. Intergovernmental Oceanographic Commission, Manuals and Guides No. 56, UNESCO (English), 196 pp. Available from the TEOS-10 web site. See appendix A.20 and appendix K of this TEOS-10 Manual.

```
                        McDougall T.J., P.M. Barker, R. Feistel and D.R. Jackett, 2011:  A
                        computationally efficient 48-term expression for the density of seawater
                        in terms of Conservative Temperature, and related properties of
                        seawater.  To be submitted to Ocean Science Discussions.
function rho = gsw_rho(SA,CT,p)

% gsw_rho                       in-situ density (48-term equation)
%==========================================================================
%
% USAGE:
%  rho = gsw_rho(SA,CT,p)
%
% DESCRIPTION:
%  Calculates in-situ density from Absolute Salinity and Conservative
%  Temperature, using the computationally-efficient 48-term expression for
%  density in terms of SA, CT and p (McDougall et al., 2011).
%
%  Note that potential density with respect to reference pressure, pr, is
%  obtained by calling this function with the pressure argument being pr
%  (i.e. "gsw_rho(SA,CT,pr)").
%
%  Note that the 48-term equation has been fitted in a restricted range of
%  parameter space, and is most accurate inside the "oceanographic funnel"
%  described in McDougall et al. (2011).  The GSW library function
%  "gsw_infunnel(SA,CT,p)" is avaialble to be used if one wants to test if
%  some of one's data lies outside this "funnel".
%
% INPUT:
%  SA  =  Absolute Salinity                          [ g/kg ]
%  CT  =  Conservative Temperature (ITS-90)                  [ deg C ]
%  p   =  sea pressure                               [ dbar ]
%         ( i.e. absolute pressure - 10.1325 dbar )
%
%  SA & CT need to have the same dimensions.
%  p may have dimensions 1x1 or Mx1 or 1xN or MxN, where SA & CT are MxN.
%
% OUTPUT:
%  rho  =  in-situ density                           [ kg/m ]
%
% AUTHOR:
%  Paul Barker and Trevor McDougall                  [ help_gsw@csiro.au ]
%
% VERSION NUMBER: 3.0 (18th March, 2011)
%
% REFERENCES:
%  IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of
%   seawater - 2010: Calculation and use of thermodynamic properties.
%   Intergovernmental Oceanographic Commission, Manuals and Guides No. 56,
%   UNESCO (English), 196 pp.  Available from http://www.TEOS-10.org
%    See appendix A.20 and appendix K of this TEOS-10 Manual.
%
%  McDougall T.J., P.M. Barker, R. Feistel and D.R. Jackett, 2011:  A
%   computationally efficient 48-term expression for the density of
%   seawater in terms of Conservative Temperature, and related properties
%   of seawater.  To be submitted to Ocean Science Discussions.
%
```

```
% The software is available from http://www.TEOS-10.org
%
%==========================================================================

%--------------------------------------------------------------------------
% Check variables and resize if necessary
%--------------------------------------------------------------------------

if ~(nargin == 3)
  error('gsw_rho:  Requires three inputs')
end %if

[ms,ns] = size(SA);
[mt,nt] = size(CT);
[mp,np] = size(p);

if (mt ~= ms | nt ~= ns)
   error('gsw_rho: SA and CT must have same dimensions')
end

if (mp == 1) & (np == 1)                % p scalar - fill to size of SA
   p = p*ones(size(SA));
elseif (ns == np) & (mp == 1)           % p is row vector,
   p = p(ones(1,ms), :);                % copy down each column.
elseif (ms == mp) & (np == 1)           % p is column vector,
   p = p(:,ones(1,ns));                 % copy across each row.
elseif (ns == mp) & (np == 1)           % p is a transposed row vector,
   p = p.';                             % transposed then
   p = p(ones(1,ms), :);                % copy down each column.
elseif (ms == mp) & (ns == np)
   % ok
else
   error('gsw_rho: Inputs array dimensions arguments do not agree')
end %if

if ms == 1
   SA = SA';
   CT = CT';
   p = p';
   transposed = 1;
else
   transposed = 0;
end

%--------------------------------------------------------------------------
% Start of the calculation
%--------------------------------------------------------------------------

% These few lines ensure that SA is non-negative.
[I_neg_SA] = find(SA < 0);
if ~isempty(I_neg_SA)
   SA(I_neg_SA) = 0;
end

v01 =  9.998420897506056e+2;
v02 =  2.839940833161907;
```

```
v03 = -3.147759265588511e-2;
v04 =  1.181805545074306e-3;
v05 = -6.698001071123802;
v06 = -2.986498947203215e-2;
v07 =  2.327859407479162e-4;
v08 = -3.988822378968490e-2;
v09 =  5.095422573880500e-4;
v10 = -1.426984671633621e-5;
v11 =  1.645039373682922e-7;
v12 = -2.233269627352527e-2;
v13 = -3.436090079851880e-4;
v14 =  3.726050720345733e-6;
v15 = -1.806789763745328e-4;
v16 =  6.876837219536232e-7;
v17 = -3.087032500374211e-7;
v18 = -1.988366587925593e-8;
v19 = -1.061519070296458e-11;
v20 =  1.550932729220080e-10;
v21 =  1.0;
v22 =  2.775927747785646e-3;
v23 = -2.349607444135925e-5;
v24 =  1.119513357486743e-6;
v25 =  6.743689325042773e-10;
v26 = -7.521448093615448e-3;
v27 = -2.764306979894411e-5;
v28 =  1.262937315098546e-7;
v29 =  9.527875081696435e-10;
v30 = -1.811147201949891e-11;
v31 = -3.303308871386421e-5;
v32 =  3.801564588876298e-7;
v33 = -7.672876869259043e-9;
v34 = -4.634182341116144e-11;
v35 =  2.681097235569143e-12;
v36 =  5.419326551148740e-6;
v37 = -2.742185394906099e-5;
v38 = -3.212746477974189e-7;
v39 =  3.191413910561627e-9;
v40 = -1.931012931541776e-12;
v41 = -1.105097577149576e-7;
v42 =  6.211426728363857e-10;
v43 = -1.119011592875110e-10;
v44 = -1.941660213148725e-11;
v45 = -1.864826425365600e-14;
v46 =  1.119522344879478e-14;
v47 = -1.200507748551599e-15;
v48 =  6.057902487546866e-17;

sqrtSA = sqrt(SA);

v_hat_denominator = v01 + CT.*(v02 + CT.*(v03 + v04*CT))  ...
          + SA.*(v05 + CT.*(v06 + v07*CT) ...
        + sqrtSA.*(v08 + CT.*(v09 + CT.*(v10 + v11*CT)))) ...
            + p.*(v12 + CT.*(v13 + v14*CT) + SA.*(v15 + v16*CT) ...
            + p.*(v17 + CT.*(v18 + v19*CT) + v20*SA));

v_hat_numerator = v21 + CT.*(v22 + CT.*(v23 + CT.*(v24 + v25*CT))) ...
```

```
          + SA.*(v26 + CT.*(v27 + CT.*(v28 + CT.*(v29 + v30*CT))) + v36*SA ...
      + sqrtSA.*(v31 + CT.*(v32 + CT.*(v33 + CT.*(v34 + v35*CT)))))  ...
          + p.*(v37 + CT.*(v38 + CT.*(v39 + v40*CT))  ...
          + SA.*(v41 + v42*CT) ...
          + p.*(v43 + CT.*(v44 + v45*CT + v46*SA) ...
          + p.*(v47 + v48*CT)));

rho = v_hat_denominator./v_hat_numerator;

%--------------------------------------------------------------------------
% This function calculates rho using the computationally-efficient
% 48-term expression for density in terms of SA, CT and p. If one wanted to
% compute rho from SA, CT, and p with the full TEOS-10 Gibbs function,
% the following lines of code will enable this.
%
%    pt0 = gsw_pt_from_CT(SA,CT);
%    pr0 = zeros(size(SA));
%    t = gsw_pt_from_t(SA,pt0,pr0,p);
%    rho = gsw_rho_t_exact(SA,t,p);
%
%    or call the following, it is identical to the lines above.
%
%    rho = gsw_rho_CT_exact(SA,CT,p)
%
%    or call the following, it is identical to the lines above.
%
%    [rho, ~, ~] = gsw_rho_alpha_beta_CT_exact(SA,CT,p)
%
%------------------This is the end of the alternative code-------------------

if transposed
   rho = rho.';
end

end
```

A.2 Absolute Salinity from Practical Salinity

USAGE: [SA, in_ocean] = gsw_SA_from_SP(SP,p,long,lat)

DESCRIPTION: Calculates absolute salinity from practical salinity. Since SP is non-negative by definition, this function changes any negative input values of SP to be zero.

INPUT:
SP = Practical Salinity (PSS-78) [unitless]
p = sea pressure [dbar] (i.e., absolute pressure - 10.1325 dbar)
long = longitude in decimal degrees [0 ... +360] or [-180 ... +180]
lat = latitude in decimal degrees north [-90 ... +90]

p, lat & long may have dimensions 1x1 or Mx1 or 1xN or MxN, where SP is MxN.

OUTPUT:
SA = Absolute Salinity [g/kg]
in_ocean = 0, if long and lat are a long way from the ocean
= 1, if long and lat are in the ocean

Note. This flag is only set when the observation is well and truly on dry land; often the warning flag is not set until one is several hundred kilometres inland from the coast.

AUTHOR: Trevor McDougall, Paul Barker & David Jackett

VERSION NUMBER: 3.0 (23rd May, 2011)

REFERENCES: IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of seawater - 2010: Calculation and use of thermodynamic properties. Intergovernmental Oceanographic Commission, Manuals and Guides No. 56, UNESCO (English), 196 pp. Available from the TEOS-10 web site. See section 2.5 and appendices A.4 and A.5 of this TEOS-10 Manual.

McDougall, T.J., D.R. Jackett and F.J. Millero, 2010: An algorithm for estimating Absolute Salinity in the global ocean. Submitted to Ocean Science. A preliminary version is available at Ocean Sci. Discuss., 6, 215-242. http://www.ocean-sci-discuss.net/6/215/2009/osd-6-215-2009-print.pdf

```
function [SA, in_ocean] = gsw_SA_from_SP(SP,p,long,lat)

% gsw_SA_from_SP                Absolute Salinity from Practical Salinity
%==========================================================================
%
% USAGE:
%  [SA, in_ocean] = gsw_SA_from_SP(SP,p,long,lat)
%
% DESCRIPTION:
%  Calculates Absolute Salinity from Practical Salinity.  Since SP is
%  non-negative by definition, this function changes any negative input
%  values of SP to be zero.
%
% INPUT:
```

```
%  SP   =  Practical Salinity  (PSS-78)                    [ unitless ]
%  p    =  sea pressure                                    [ dbar ]
%        ( i.e. absolute pressure - 10.1325 dbar )
%  long =  longitude in decimal degrees                    [ 0 ... +360 ]
%                                       or  [ -180 ... +180 ]
%  lat  =  latitude in decimal degrees north               [ -90 ... +90 ]
%
%  p, lat & long may have dimensions 1x1 or Mx1 or 1xN or MxN,
%  where SP is MxN.
%
% OUTPUT:
%  SA       =  Absolute Salinity                           [ g/kg ]
%  in_ocean  =  0, if long and lat are a long way from the ocean
%           =  1, if long and lat are in the ocean
%  Note. This flag is only set when the observation is well and truly on
%    dry land; often the warning flag is not set until one is several
%    hundred kilometres inland from the coast.
%
% AUTHOR:
%  David Jackett, Trevor McDougall & Paul Barker      [ help_gsw@csiro.au ]
%
% VERSION NUMBER: 3.0 (31st May, 2011)
%
% REFERENCES:
%  IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of
%   seawater - 2010: Calculation and use of thermodynamic properties.
%   Intergovernmental Oceanographic Commission, Manuals and Guides No. 56,
%   UNESCO (English), 196 pp.  Available from http://www.TEOS-10.org
%    See section 2.5 and appendices A.4 and A.5 of this TEOS-10 Manual.
%
%  McDougall, T.J., D.R. Jackett and F.J. Millero, 2010: An algorithm
%   for estimating Absolute Salinity in the global ocean.  Submitted to
%   Ocean Science. A preliminary version is available at Ocean Sci. Discuss.,
%   6, 215-242.
%   http://www.ocean-sci-discuss.net/6/215/2009/osd-6-215-2009-print.pdf
%
%  The software is available from http://www.TEOS-10.org
%
%==========================================================================

%--------------------------------------------------------------------------
% Check variables and resize if necessary
%--------------------------------------------------------------------------

if ~(nargin==4)
   error('gsw_SA_from_SP:  Requires four inputs')
end %if

[ms,ns] = size(SP);
[mp,np] = size(p);

if (mp == 1) & (np == 1)            % p is a scalar - fill to size of SP
   p = p*ones(size(SP));
elseif (ns == np) & (mp == 1)       % p is row vector,
   p = p(ones(1,ms), :);            % copy down each column.
elseif (ms == mp) & (np == 1)       % p is column vector,
```

```
   p = p(:,ones(1,ns));                % copy across each row.
elseif (ns == mp) & (np == 1)          % p is a transposed row vector,
   p = p.';                            % transposed then
   p = p(ones(1,ms), :);               % copy down each column.
elseif (ms == mp) & (ns == np)
   % ok
else
   error('gsw_SA_from_SP: Inputs array dimensions arguments do not agree')
end %if

[mla,nla] = size(lat);

if (mla == 1) & (nla == 1)             % lat is a scalar - fill to size of SP
   lat = lat*ones(size(SP));
elseif (ns == nla) & (mla == 1)        % lat is a row vector,
   lat = lat(ones(1,ms), :);           % copy down each column.
elseif (ms == mla) & (nla == 1)        % lat is a column vector,
   lat = lat(:,ones(1,ns));            % copy across each row.
elseif (ns == mla) & (nla == 1)        % lat is a transposed row vector,
   lat = lat.';                        % transposed then
   lat = lat(ones(1,ms), :);           % copy down each column.
elseif (ms == mla) & (ns == nla)
   % ok
else
   error('gsw_SA_from_SP: Inputs array dimensions arguments do not agree')
end %if

[mlo,nlo] = size(long);
[Iwest] =find(long < 0);
if ~isempty(Iwest)
   long(Iwest) = long(Iwest) + 360;
end

if (mlo == 1) & (nlo == 1)             % long is a scalar - fill to size of SP
   long = long*ones(size(SP));
elseif (ns == nlo) & (mlo == 1)        % long is a row vector,
   long = long(ones(1,ms), :);         % copy down each column.
elseif (ms == mlo) & (nlo == 1)        % long is a column vector,
   long = long(:,ones(1,ns));          % copy across each row.
elseif (ns == mlo) & (nlo == 1)        % long is a transposed row vector,
   long = long.';                      % transposed then
   long = long(ones(1,ms), :);         % copy down each column.
elseif (ms == nlo) & (mlo == 1)        % long is a transposed column vector,
   long = long.';                      % transposed then
   long = long(:,ones(1,ns));          % copy down each column.
elseif (ms == mlo) & (ns == nlo)
   % ok
else
   error('gsw_SA_from_SP: Inputs array dimensions arguments do not agree')
end %if

if ms == 1
   SP = SP.';
   p = p.';
   lat = lat.';
   long = long.';
```

```
   transposed = 1;
else
   transposed = 0;
end

[Iout_of_range] = find(p < 100 & SP > 120);
SP(Iout_of_range) = NaN;
[Iout_of_range] = find(p >= 100 & SP > 42);
SP(Iout_of_range) = NaN;

[Inan] = find(abs(SP) == 99999 | abs(SP) == 999999);
SP(Inan) = NaN;
[Inan] = find(abs(p) == 99999 | abs(p) == 999999);
p(Inan) = NaN;
[Inan] = find(abs(long) == 9999 | abs(long) == 99999);
long(Inan) = NaN;
[Inan] = find(abs(lat) == 9999 | abs(lat) == 99999);
lat(Inan) = NaN;

if ~isempty(find(p < -1.5 | p > 12000))
   error('gsw_SA_from_SP: pressure is out of range')
end
if ~isempty(find(long < 0 | long > 360))
   error('gsw_SA_from_SP: longitude is out of range')
end
if ~isempty(find(abs(lat) > 90))
   error('gsw_SA_from_SP: latitude is out of range')
end

%--------------------------------------------------------------------------
% Start of the calculation
%--------------------------------------------------------------------------

% These few lines ensure that SP is non-negative.
[I_neg_SP] = find(SP < 0);
if ~isempty(I_neg_SP)
   SP(I_neg_SP) = 0;
end

[Iocean] = find(~isnan(SP.*p.*lat.*long));

SA = nan(size(SP));
SAAR = nan(size(SP));
in_ocean = nan(size(SP));

% The following function (gsw_SAAR) finds SAAR in the non-Baltic parts of
% the world ocean.  (Actually, this gsw_SAAR look-up table returns values
% of zero in the Baltic Sea since SAAR in the Baltic is a function of SP,
% not space.
[SAAR(Iocean), in_ocean(Iocean)] = gsw_SAAR(p(Iocean),long(Iocean),lat(Iocean));

SA(Iocean) = (35.16504/35)*SP(Iocean).*(1 + SAAR(Iocean));

% Here the Practical Salinity in the Baltic is used to calculate the
% Absolute Salinity there.
SA_baltic(Iocean) = gsw_SA_from_SP_Baltic(SP(Iocean),long(Iocean),lat(Iocean));
```

```
[Ibaltic] = find(~isnan(SA_baltic(Iocean)));

SA(Iocean(Ibaltic)) = SA_baltic(Iocean(Ibaltic));

if transposed
   SA = SA';
   in_ocean = in_ocean';
end

end
```

A.3 Conservative Temperature from in-situ Temperature

USAGE: CT = gsw_CT_from_t(SA,t,p)

DESCRIPTION: Calculates Conservative Temperature of seawater from in-situ temperature.

INPUT: SA = Absolute Salinity [g/kg]
t = in-situ temperature (ITS-90) [deg C]
p = sea pressure [dbar] (i.e. absolute pressure - 10.1325 dbar)

SA & t need to have the same dimensions.
p may have dimensions 1x1 or Mx1 or 1xN or MxN, where SA & t are MxN.

OUTPUT: CT = Conservative Temperature [deg C]

AUTHOR: David Jackett, Trevor McDougall and Paul Barker

VERSION NUMBER: 3.0 (16th May, 2011)

REFERENCES: IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of seawater - 2010: Calculation and use of thermodynamic properties. Intergovernmental Oceanographic Commission, Manuals and Guides No. 56, UNESCO (English), 196 pp. Available from the TEOS-10 web site. See section 3.3 of this TEOS-10 Manual.

```
function CT = gsw_CT_from_t(SA,t,p)

% gsw_CT_from_t          Conservative Temperature from in-situ temperature
%==========================================================================
%
% USAGE:
%  CT = gsw_CT_from_t(SA,t,p)
%
% DESCRIPTION:
%  Calculates Conservative Temperature of seawater from in-situ
%  temperature.
%
% INPUT:
%  SA  =  Absolute Salinity                          [ g/kg ]
%  t   =  in-situ temperature (ITS-90)                 [ deg C ]
%  p   =  sea pressure                               [ dbar ]
%         ( i.e. absolute pressure - 10.1325 dbar )
%
%  SA & t need to have the same dimensions.
%  p may have dimensions 1x1 or Mx1 or 1xN or MxN, where SA & t are MxN.
%
% OUTPUT:
%  CT  =  Conservative Temperature (ITS-90)                [ deg C ]
%
% AUTHOR:
%  David Jackett, Trevor McDougall and Paul Barker    [ help_gsw@csiro.au ]
%
```

```
% VERSION NUMBER: 3.0 (27th March, 2011)
%  This function is unchanged from version 2.0 (24th September, 2010).
%
% REFERENCES:
%  IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of
%   seawater - 2010: Calculation and use of thermodynamic properties.
%   Intergovernmental Oceanographic Commission, Manuals and Guides No. 56,
%   UNESCO (English), 196 pp.  Available from http://www.TEOS-10.org
%    See section 3.3 of this TEOS-10 Manual.
%
%  The software is available from http://www.TEOS-10.org
%
%==========================================================================

%--------------------------------------------------------------------------
% Check variables and resize if necessary
%--------------------------------------------------------------------------

if ~(nargin==3)
   error('gsw_CT_from_t:  Requires three inputs')
end %if

[ms,ns] = size(SA);
[mt,nt] = size(t);
[mp,np] = size(p);

if (mt ~= ms | nt ~= ns)
    error('gsw_CT_from_t: SA and t must have same dimensions')
end

if (mp == 1) & (np == 1)              % p scalar - fill to size of SA
    p = p*ones(size(SA));
elseif (ns == np) & (mp == 1)         % p is row vector,
    p = p(ones(1,ms), :);             % copy down each column.
elseif (ms == mp) & (np == 1)         % p is column vector,
    p = p(:,ones(1,ns));              % copy across each row.
elseif (ns == mp) & (np == 1)         % p is a transposed row vector,
    p = p.';                          % transposed then
    p = p(ones(1,ms),:);              % copy down each column.
elseif (ms == mp) & (ns == np)
    % ok
else
    error('gsw_CT_from_t: Inputs array dimensions arguments do not agree')
end %if

[Iout_of_range] = find(p < 100 & (t > 80 | t < -12));
if (~isempty(Iout_of_range))
    t(Iout_of_range) = NaN;
end
[Iout_of_range] = find(p >= 100 & (t > 40 | t < -12));
if (~isempty(Iout_of_range))
    t(Iout_of_range) = NaN;
end

if ms == 1
    SA = SA.';
```

```
    t = t.';
    p = p.';
    transposed = 1;
else
    transposed = 0;
end

%--------------------------------------------------------------------------
% Start of the calculation
%--------------------------------------------------------------------------

pt0 = gsw_pt0_from_t(SA,t,p);
CT = gsw_CT_from_pt(SA,pt0);

if transposed
    CT = CT.';
end

end
```

A.4 Potential Temperature w/ Ref Sea Pressure of Zero dbar

```
function pt0 = gsw_pt0_from_t(SA,t,p)

% gsw_pt0_from_t                    potential temperature with a
%                                 reference sea pressure of zero dbar
%
==========================================================================
%
% USAGE:
%  pt0 = gsw_pt0_from_t(SA,t,p)
%
% DESCRIPTION:
%  Calculates potential temperature with reference pressure, p_ref = 0 dbar.
%  The present routine is computationally faster than the more general
%  function "gsw_pt_from_t(SA,t,p,p_ref)" which can be used for any reference
%  pressure value.
%  This subroutine calls "gsw_entropy_part(SA,t,p)",
%  "gsw_entropy_part_zerop(SA,pt0)" and "gsw_gibbs_pt0_pt0(SA,pt0)".
%
% INPUT:
%  SA  =  Absolute Salinity                                  [ g/kg ]
%  t   =  in-situ temperature (ITS-90)                         [ deg C ]
%  p   =  sea pressure                                      [ dbar ]
%       ( i.e. absolute pressure - 10.1325 dbar )
%
%  SA & t need to have the same dimensions.
%  p may have dimensions 1x1 or Mx1 or 1xN or MxN, where SA & t are MxN.
%
% OUTPUT:
%  pt0  =  potential temperature                               [ deg C ]
%        with reference sea pressure (p_ref) = 0 dbar.
%  Note. The reference sea pressure of the output, pt0, is zero dbar.
%
% AUTHOR:
%  Trevor McDougall, David Jackett, Claire Roberts-Thomson and Paul Barker.
%                                        [ help_gsw@csiro.au ]
%
% VERSION NUMBER: 3.0 (29th March, 2011)
%  This function is unchanged from version 2.0 (24th September, 2010).
%
% REFERENCES:
%  IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of
%   seawater - 2010: Calculation and use of thermodynamic properties.
%   Intergovernmental Oceanographic Commission, Manuals and Guides No. 56,
%   UNESCO (English), 196 pp.  Available from http://www.TEOS-10.org
%    See section 3.1 of this TEOS-10 Manual.
%
%  McDougall T.J., P.M. Barker, R. Feistel and D.R. Jackett, 2011:  A
%   computationally efficient 48-term expression for the density of
%   seawater in terms of Conservative Temperature, and related properties
%   of seawater.  To be submitted to Ocean Science Discussions.
%
%  The software is available from http://www.TEOS-10.org
%
```

```
%==========================================================================

%--------------------------------------------------------------------------
% Check variables and resize if necessary
%--------------------------------------------------------------------------

if ~(nargin == 3)
   error('gsw_pt0_from_t: Requires 3 inputs - Absolute Salinity, temperature, and pressure')
end %if

[ms,ns] = size(SA);
[mt,nt] = size(t);
[mp,np] = size(p);

if (ms ~= mt | ns ~= nt )
   error('gsw_pt0_from_t: Input arguments do not have the same dimensions')
end %if

if (mp == 1) & (np == 1)              % p scalar - fill to size of SA
   p = p*ones(size(SA));
elseif (ns == np) & (mp == 1)         % p is row vector,
   p = p(ones(1,ms), :);              % copy down each column.
elseif (ms == mp) & (np == 1)         % p is column vector,
   p = p(:,ones(1,ns));               % copy across each row.
elseif (ns == mp) & (np == 1)         % p is a transposed row vector,
   p = p.';                           % transposed then
   p = p(ones(1,ms), :);              % copy down each column.
elseif (ms == mp) & (ns == np)
   % ok
else
   error('gsw_pt0_from_t: Inputs array dimensions arguments do not agree')
end %if

if ms == 1
   SA = SA.';
   t = t.';
   p = p.';
   transposed = 1;
else
   transposed = 0;
end

%--------------------------------------------------------------------------
% Start of the calculation
%--------------------------------------------------------------------------

% These few lines ensure that SA is non-negative.
[I_neg_SA] = find(SA < 0);
if ~isempty(I_neg_SA)
   SA(I_neg_SA) = 0;
end

cp0 = 3991.86795711963;          % from Eqn. (3.3.3) of IOC et al. (2010).
SSO = 35.16504;                  % from section 2.4 of IOC et al. (2010).

s1 = SA*(35./SSO);
```

```
pt0 = t + p.*( 8.65483913395442e-6  - ...
        s1.*  1.41636299744881e-6  - ...
         p.*  7.38286467135737e-9  + ...
         t.*(-8.38241357039698e-6  + ...
        s1.*  2.83933368585534e-8  + ...
         t.*  1.77803965218656e-8  + ...
         p.*  1.71155619208233e-10));

dentropy_dt = cp0./((273.15 + pt0).*(1-0.05.*(1 - SA./SSO)));

true_entropy_part = gsw_entropy_part(SA,t,p);

for Number_of_iterations = 1:2
   pt0_old = pt0;
   dentropy = gsw_entropy_part_zerop(SA,pt0_old) - true_entropy_part;
   pt0 = pt0_old - dentropy./dentropy_dt ; % this is half way through the modified method
   pt0m = 0.5*(pt0 + pt0_old);
   dentropy_dt = -gsw_gibbs_pt0_pt0(SA,pt0m);
   pt0 = pt0_old - dentropy./dentropy_dt;
end

if transposed
   pt0 = pt0.';
end

% maximum error of 6.3x10^-9 degrees C for one iteration.
% maximum error is 1.8x10^-14 degrees C for two iterations
% (two iterations is the default, "for Number_of_iterations = 1:2").
% These errors are over the full "oceanographic funnel" of
% McDougall et al. (2011), which reaches down to p = 8000 dbar.

end
```

A.5 Conservative Temperature from Potential Temperature

```
function CT = gsw_CT_from_pt(SA,pt)

% gsw_CT_from_pt        Conservative Temperature from potential temperature
%==========================================================================
%
% USAGE:
%  CT = gsw_CT_from_pt(SA,pt)
%
% DESCRIPTION:
%  Calculates Conservative Temperature of seawater from potential
%  temperature (whose reference sea pressure is zero dbar).
%
% INPUT:
%  SA  =  Absolute Salinity                                  [ g/kg ]
%  pt  =  potential temperature (ITS-90)                        [ deg C ]
%
%  SA & pt need to have the same dimensions.
%
% OUTPUT:
%  CT  =  Conservative Temperature (ITS-90)                     [ deg C ]
%
% AUTHOR:
%  David Jackett, Trevor McDougall and Paul Barker    [ help_gsw@csiro.au ]
%
% VERSION NUMBER: 3.0 (29th March, 2011)
%  This function is unchanged from version 2.0 (24th September, 2010).
%
% REFERENCES:
%  IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of
%   seawater - 2010: Calculation and use of thermodynamic properties.
%   Intergovernmental Oceanographic Commission, Manuals and Guides No. 56,
%   UNESCO (English), 196 pp.  Available from http://www.TEOS-10.org
%    See section 3.3 of this TEOS-10 Manual.
%
%  The software is available from http://www.TEOS-10.org
%
%==========================================================================

%--------------------------------------------------------------------------
% Check variables and resize if necessary
%--------------------------------------------------------------------------

if ~(nargin == 2)
   error('gsw_CT_from_pt:  Requires two inputs')
end %if

[ms,ns] = size(SA);
[mt,nt] = size(pt);

if (mt ~= ms | nt ~= ns)
    error('gsw_CT_from_pt: SA and pt must have same dimensions')
end
```

```
if ms == 1
   SA = SA.';
   pt = pt.';
   transposed = 1;
else
   transposed = 0;
end

%--------------------------------------------------------------------------
% Start of the calculation
%--------------------------------------------------------------------------

% These few lines ensure that SA is non-negative.
[I_neg_SA] = find(SA < 0);
if ~isempty(I_neg_SA)
   SA(I_neg_SA) = 0;
end

cp0 = 3991.86795711963;     % defined in Eqn. (3.3.3) of IOC et al. (2010).

sfac = 0.0248826675584615;                  % sfac = 1/(40.*(35.16504/35)).

x2 = sfac.*SA;
x = sqrt(x2);
y = pt.*0.025;                              % normalize for F03 and F08.

pot_enthalpy =  61.01362420681071 + y.*(168776.46138048015 + ...
   y.*(-2735.2785605119625 + y.*(2574.2164453821433 + ...
   y.*(-1536.6644434977543 + y.*(545.7340497931629 + ...
   (-50.91091728474331 - 18.30489878927802.*y).*y))))) + ...
   x2.*(268.5520265845071 + y.*(-12019.028203559312 + ...
   y.*(3734.858026725145 + y.*(-2046.7671145057618 + ...
   y.*(465.28655623826234 + (-0.6370820302376359 - ...
   10.650848542359153.*y).*y)))) + ...
   x.*(937.2099110620707 + y.*(588.1802812170108 + ...
   y.*(248.39476522971285 + (-3.871557904936333 - ...
   2.6268019854268356.*y).*y)) + ...
   x.*(-1687.914374187449 + x.*(246.9598888781377 + ...
   x.*(123.59576582457964 - 48.5891069025409.*x)) + ...
   y.*(936.3206544460336 + ...
   y.*(-942.7827304544439 + y.*(369.4389437509002 + ...
   (-33.83664947895248 - 9.987880382780322.*y).*y))))));

%--------------------------------------------------------------------------
% The above polynomial for pot_enthalpy is the full expression for
% potential entahlpy in terms of SA and pt, obtained from the Gibbs
% function as below.  The above polynomial has simply collected like powers
% of x and y so that it is computationally faster than calling the Gibbs
% function twice as is done in the commented code below.  When this code
% below is run, the results are identical to calculating pot_enthalpy as
% above, to machine precision.
%
%  n0 = 0;
%  n1 = 1;
%  pr0 = zeros(size(SA));
%  pot_enthalpy = gsw_gibbs(n0,n0,n0,SA,pt,pr0) - ...
```

```
%                   (273.15 + pt).*gsw_gibbs(n0,n1,n0,SA,pt,pr0);
%
%----------------This is the end of the alternative code----------------

CT = pot_enthalpy./cp0;

if transposed
   CT = CT.';
end

end
```

A.6 Partial Calculation of Entropy

```
function entropy_part = gsw_entropy_part(SA,t,p)

%gsw_entropy_part    entropy minus the terms that are a function of only SA
%==========================================================================
% This function calculates entropy, except that it does not evaluate any
% terms that are functions of Absolute Salinity alone.  By not calculating
% these terms, which are a function only of Absolute Salinity, several
% unnecessary computations are avoided (including saving the computation
% of a natural logarithm).  These terms are a necessary part of entropy,
% but are not needed when calculating potential temperature from in-situ
% temperature.
%
% VERSION NUMBER: 3.0 (29th March, 2011)
%  This function is unchanged from version 2.0 (24th September, 2010).
%
%==========================================================================

% These few lines ensure that SA is non-negative.
[I_neg_SA] = find(SA < 0);
if ~isempty(I_neg_SA)
   SA(I_neg_SA) = 0;
end

sfac = 0.0248826675584615;              % sfac = 1/(40*(35.16504/35));

x2 = sfac.*SA;
x = sqrt(x2);
y = t.*0.025d0;
z = p.*1d-4;

g03 = z.*(-270.983805184062 + ...
   z.*(776.153611613101 + z.*(-196.51255088122 + (28.9796526294175 -
2.13290083518327.*z).*z))) + ...
   y.*(-24715.571866078 + z.*(2910.0729080936 + ...
   z.*(-1513.116771538718 + z.*(546.959324647056 + z.*(-111.1208127634436 +
8.68841343834394.*z)))) + ...
   y.*(2210.2236124548363 + z.*(-2017.52334943521 + ...
   z.*(1498.081172457456 + z.*(-718.6359919632359 + (146.4037555781616 -
4.9892131862671505.*z).*z))) + ...
   y.*(-592.743745734632 + z.*(1591.873781627888 + ...
   z.*(-1207.261522487504 + (608.785486935364 - 105.4993508931208.*z).*z)) + ...
   y.*(290.12956292128547 + z.*(-973.091553087975 + ...
   z.*(602.603274510125 + z.*(-276.361526170076 + 32.40953340386105.*z))) + ...
   y.*(-113.90630790850321 + y.*(21.35571525415769 - 67.41756835751434.*z) + ...
   z.*(381.06836198507096 + z.*(-133.7383902842754 + 49.023632509086724.*z)))))));

g08 = x2.*(z.*(729.116529735046 + ...
   z.*(-343.956902961561 + z.*(124.687671116248 + z.*(-31.656964386073 +
7.04658803315449.*z)))) + ...
   x.*( x.*(y.*(-137.1145018408982 + y.*(148.10030845687618 + y.*(-68.5590309679152 +
12.4848504784754.*y))) - ...
   22.6683558512829.*z) + z.*(-175.292041186547 + (83.1923927801819 -
29.483064349429.*z).*z) + ...
```

```
   y.*(-86.1329351956084 + z.*(766.116132004952 + z.*(-108.3834525034224 +
51.2796974779828.*z)) + ...
   y.*(-30.0682112585625 - 1380.9597954037708.*z + y.*(3.50240264723578 +
938.26075044542.*z)))) + ...
   y.*(1760.062705994408 + y.*(-675.802947790203 + ...
   y.*(365.7041791005036 + y.*(-108.30162043765552 + 12.78101825083098.*y) + ...
   z.*(-1190.914967948748 + (298.904564555024 - 145.9491676006352.*z).*z)) + ...
   z.*(2082.7344423998043 + z.*(-614.668925894709 + (340.685093521782 -
33.3848202979239.*z).*z))) + ...
   z.*(-1721.528607567954 + z.*(674.819060538734 + ...
   z.*(-356.629112415276 + (88.4080716616 - 15.84003094423364.*z).*z)))));

entropy_part = -(g03 + g08).*0.025d0;

end
```

A.7 Partial Calculation of Entropy at Sea Pressure of Zero

```
function entropy_part_zerop = gsw_entropy_part_zerop(SA,pt0)

% gsw_entropy_part_zerop          entropy_part evaluated at the sea surface
%==========================================================================
% This function calculates entropy at a sea pressure of zero, except that
% it does not evaluate any terms that are functions of Absolute Salinity
% alone.  By not calculating these terms, which are a function only of
% Absolute Salinity, several unnecessary computations are avoided
% (including saving the computation of a natural logarithm). These terms
% are a necessary part of entropy, but are not needed when calculating
% potential temperature from in-situ temperature.
% The inputs to "gsw_entropy_part_zerop(SA,pt0)" are Absolute Salinity
% and potential temperature with reference sea pressure of zero dbar.
%
% VERSION NUMBER: 3.0 (29th March, 2011)
%  This function is unchanged from version 2.0 (24th September, 2010).
%
%==========================================================================

% These few lines ensure that SA is non-negative.
[I_neg_SA] = find(SA < 0);
if ~isempty(I_neg_SA)
   SA(I_neg_SA) = 0;
end

sfac = 0.0248826675584615;              % sfac = 1/(40*(35.16504/35));

x2 = sfac.*SA;
x = sqrt(x2);
y = pt0.*0.025d0;

g03 =  y.*(-24715.571866078 + y.*(2210.2236124548363 + ...
   y.*(-592.743745734632 + y.*(290.12956292128547 + ...
   y.*(-113.90630790850321 + y.*21.35571525415769)))));

g08 = x2.*(x.*( x.*(y.*(-137.1145018408982 + y.*(148.10030845687618 + ...
   y.*(-68.5590309679152 + 12.4848504784754.*y)))) + ...
   y.*(-86.1329351956084 + y.*(-30.0682112585625 + y.*3.50240264723578))) + ...
   y.*(1760.062705994408 + y.*(-675.802947790203 + ...
   y.*(365.7041791005036 + y.*(-108.30162043765552 + 12.78101825083098.*y)))));


entropy_part_zerop = -(g03 + g08).*0.025d0;

end
```

A.8 Second Derivative of the Gibbs Function

```
function gibbs_pt0_pt0 = gsw_gibbs_pt0_pt0(SA,pt0)

% gsw_gibbs_pt0_pt0                         gibbs_tt at (SA,pt,0)
%==========================================================================
% This function calculates the second derivative of the specific Gibbs
% function with respect to temperature at zero sea pressure.  The inputs
% are Absolute Salinity and potential temperature with reference sea
% pressure of zero dbar.  This library function is called by both
% "gsw_pt_from_CT(SA,CT)" ,"gsw_pt0_from_t(SA,t,p)" and
% "gsw_pt_from_entropy(SA,entropy)".
%
% VERSION NUMBER: 3.0 (29th March, 2011)
%  This function is unchanged from version 2.0 (24th September, 2010).
%
%==========================================================================

% These few lines ensure that SP is non-negative.
[I_neg_SA] = find(SA < 0);
if ~isempty(I_neg_SA)
   SA(I_neg_SA) = 0;
end

sfac = 0.0248826675584615;              % sfac = 1/(40*(35.16504/35));

x2 = sfac.*SA;
x = sqrt(x2);
y = pt0.*0.025d0;

g03 = -24715.571866078 + ...
   y.*(4420.4472249096725 + ...
   y.*(-1778.231237203896 + ...
   y.*(1160.5182516851419 + ...
   y.*(-569.531539542516 + y.*128.13429152494615))));

g08 = x2.*(1760.062705994408 + x.*(-86.1329351956084 + ...
   x.*(-137.1145018408982 + y.*(296.20061691375236 + ...
   y.*(-205.67709290374563 + 49.9394019139016.*y))) + ...
   y.*(-60.136422517125 + y.*10.50720794170734)) + ...
   y.*(-1351.605895580406 + y.*(1097.1125373015109 +  ...
   y.*(-433.20648175062206 + 63.905091254154904.*y))));

gibbs_pt0_pt0 = (g03 + g08).*0.000625;

end
```

A.9 Absolute Salinity Anomaly Ratio

```
function [SAAR, in_ocean] = gsw_SAAR(p,long,lat)

% gsw_SAAR       Absolute Salinity Anomaly Ratio (excluding the Baltic Sea)
%==========================================================================
%
% USAGE:
%  [SAAR, in_ocean] = gsw_SAAR(p,long,lat)
%
% DESCRIPTION:
%  Calculates the Absolute Salinity Anomaly Ratio, SAAR, in the open ocean
%  by spatially interpolating the global reference data set of SAAR to the
%  location of the seawater sample.
%
%  This function uses version 3.0 of the SAAR look up table.
%
%  The Absolute Salinity Anomaly Ratio in the Baltic Sea is evaluated
%  separately, since it is a function of Practical Salinity, not of space.
%  The present function returns a SAAR of zero for data in the Baltic Sea.
%  The correct way of calculating Absolute Salinity in the Baltic Sea is by
%  calling gsw_SA_from_SP.
%
% INPUT:
%  p     =  sea pressure                                    [ dbar ]
%         ( i.e. absolute pressure - 10.1325 dbar )
%  long  =  Longitude in decimal degrees                      [ 0 ... +360 ]
%                                       or [ -180 ... +180 ]
%  lat   =  Latitude in decimal degrees north                 [ -90 ... +90 ]
%
%  p, long & lat need to be vectors and have the same dimensions.
%
% OUTPUT:
%  SAAR      =  Absolute Salinity Anomaly Ratio               [ unitless ]
%  in_ocean  =  0, if long and lat are a long way from the ocean
%            =  1, if long and lat are in the ocean
%  Note. This flag is only set when the observation is well and truly on
%    dry land; often the warning flag is not set until one is several
%    hundred kilometres inland from the coast.
%
% AUTHOR:
%  David Jackett                                  [ help_gsw@csiro.au ]
%
% MODIFIED:
%  Paul Barker and Trevor McDougall
%
% VERSION NUMBER: 3.0 (31st May, 2011)
%
% REFERENCES:
%  IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of
%   seawater - 2010: Calculation and use of thermodynamic properties.
%   Intergovernmental Oceanographic Commission, Manuals and Guides No. 56,
%   UNESCO (English), 196 pp.  Available from http://www.TEOS-10.org
%
%  McDougall, T.J., D.R. Jackett and F.J. Millero, 2010: An algorithm
%   for estimating Absolute Salinity in the global ocean.  Submitted to
```

```
%   Ocean Science. A preliminary version is available at Ocean Sci.
%   Discuss., 6, 215-242.
%   http://www.ocean-sci-discuss.net/6/215/2009/osd-6-215-2009-print.pdf
%
%  The software is available from http://www.TEOS-10.org
%
%==========================================================================

%--------------------------------------------------------------------------
% Check variables and resize if necessary
%--------------------------------------------------------------------------

if ~(nargin == 3)
   error('gsw_SAAR:  Requires three inputs')
end %if

[mp,np] = size(p);
[mla,nla] = size(lat);
[mlo,nlo] = size(long);

if (mp ~= mla) | (mp ~=mlo) | (np ~= nla) | (np ~= nlo)
    error('gsw_SAAR: Inputs need be of the same size')
end %if

if ~isempty(find(p < -1.5))
    error('gsw_SAAR: pressure needs to be positive')
end

[Ipn] = find(p < 0 & p > -1.5);
if ~isempty(Ipn)
    p(Ipn) = 0;
end

%--------------------------------------------------------------------------
% Start of the calculation (extracting from a look up table)
%--------------------------------------------------------------------------

gsw_data = 'gsw_data_v3_0.mat';

gsw_data_file = which(gsw_data);

load (gsw_data_file,'SAAR_ref','lats_ref','longs_ref','p_ref', ...
    'ndepth_ref');

nx = length(longs_ref);
ny = length(lats_ref);
nz = length(p_ref);
n0 = length(p);

dlongs_ref = longs_ref(2) - longs_ref(1);
dlats_ref = lats_ref(2) - lats_ref(1);

indsx0 = floor(1 + (nx-1)*(long - longs_ref(1))./(longs_ref(nx) - longs_ref(1)));
indsx0 = indsx0(:);
inds = find(indsx0 == nx);
indsx0(inds) = nx - 1;
```

```
indsy0 = floor(1 + (ny-1)*(lat - lats_ref(1))./(lats_ref(ny) - lats_ref(1)));
indsy0 = indsy0(:);
inds = find(indsy0 == ny);
indsy0(inds) = ny - 1;

indsz0 = sum(ones(nz,1)*p(:)' >= p_ref(:)*ones(1,n0));
indsz0 = indsz0(:);                          % adjust in the vertical

indsn1 = sub2ind([ny,nx],indsy0,indsx0);        % casts containing data
indsn2 = sub2ind([ny,nx],indsy0,indsx0+1);
indsn3 = sub2ind([ny,nx],indsy0+1,indsx0+1);
indsn4 = sub2ind([ny,nx],indsy0+1,indsx0);

nmax = max([ndepth_ref(indsn1)';ndepth_ref(indsn2)';ndepth_ref(indsn3)';ndepth_ref(indsn4)']);

inds1 = find(indsz0(:)' > nmax);               % casts deeper than GK maximum

p(inds1) = p_ref(nmax(inds1));                 % have reset p here so have to reset indsz0

indsz0 = sum(ones(nz,1)*p(:)' >= p_ref(:)*ones(1,n0));
indsz0 = indsz0(:);
inds = find(indsz0 == nz);
indsz0(inds) = nz - 1;

inds0 = sub2ind([nz,ny,nx],indsz0,indsy0,indsx0);

data_indices = [indsx0,indsy0,indsz0,inds0];
data_inds = data_indices(:,3);

r1 = (long(:) - longs_ref(indsx0))./(longs_ref(indsx0+1) - longs_ref(indsx0));
s1 = (lat(:) - lats_ref(indsy0))./(lats_ref(indsy0+1) - lats_ref(indsy0));
t1 = (p(:) - p_ref(indsz0))./(p_ref(indsz0+1) - p_ref(indsz0));

nksum = 0;
no_levels_missing = 0;

sa_upper = nan(size(data_inds));
sa_lower = nan(size(data_inds));
SAAR = nan(size(data_inds));
in_ocean = ones(size(SAAR));

for k = 1:nz-1

   inds_k = find(indsz0 == k);
   nk = length(inds_k);

   if nk>0
      nksum = nksum+nk;
      indsx = indsx0(inds_k);
      indsy = indsy0(inds_k);
      indsz = k*ones(size(indsx));
      inds_di = find(data_inds == k);          % level k interpolation
      saar = nan(4,n0);
      inds1 = sub2ind([nz,ny,nx], indsz, indsy, indsx);
      saar(1,inds_k) = SAAR_ref(inds1);
```

```
inds2 = sub2ind([nz,ny,nx], indsz, indsy, indsx+1);
saar(2,inds_k) = SAAR_ref(inds2);                % inds0 + ny*nz
inds3 = sub2ind([nz,ny,nx], indsz, indsy+1, indsx+1);
saar(3,inds_k) = SAAR_ref(inds3);            % inds0 + ny*nz + nz
inds4 = sub2ind([nz ny,nx], indsz, indsy+1, indsx);
saar(4,inds_k) = SAAR_ref(inds4);                % inds0 + nz

inds = find(260<=long(:) & long(:)<=295.217 & ...
   0<=lat(:) & lat(:)<=19.55 & indsz0(:)==k);
if ~isempty(inds)
   saar(:,inds) = gsw_saar_add_barrier(saar(:,inds),long(inds), ...
      lat(inds),longs_ref(indsx0(inds)),lats_ref(indsy0(inds)),dlongs_ref,dlats_ref);
end

inds = find(isnan(sum(saar))' & indsz0==k);
if ~isempty(inds)
   saar(:,inds) = gsw_saar_add_mean(saar(:,inds));
end

sa_upper(inds_di) = (1-s1(inds_di)).*(saar(1,inds_k)' + ...
   r1(inds_di).*(saar(2,inds_k)'-saar(1,inds_k)')) + ...
   s1(inds_di).*(saar(4,inds_k)' + ...
   r1(inds_di).*(saar(3,inds_k)'-saar(4,inds_k)'));  % level k+1 interpolation

saar = nan(4,n0);
inds1 = sub2ind([nz,ny,nx], indsz+1, indsy, indsx);
saar(1,inds_k) = SAAR_ref(inds1);
inds2 = sub2ind([nz,ny,nx], indsz+1, indsy, indsx+1);
saar(2,inds_k) = SAAR_ref(inds2);                % inds1 + ny*nz
inds3 = sub2ind([nz,ny,nx], indsz+1, indsy+1, indsx+1);
saar(3,inds_k) = SAAR_ref(inds3);            % inds1 + ny*nz + nz
inds4 = sub2ind([nz ny,nx], indsz+1, indsy+1, indsx);
saar(4,inds_k) = SAAR_ref(inds4);                % inds1 + nz

inds = find(260<=long(:) & long(:)<=295.217 & ...
   0<=lat(:) & lat(:)<=19.55 & indsz0(:)==k);
if ~isempty(inds)
   saar(:,inds) = gsw_saar_add_barrier(saar(:,inds),long(inds), ...
      lat(inds),longs_ref(indsx0(inds)),lats_ref(indsy0(inds)),dlongs_ref,dlats_ref);
end

inds = find(isnan(sum(saar))' & indsz0==k);

if ~isempty(inds)
   saar(:,inds) = gsw_saar_add_mean(saar(:,inds));
end

sa_lower(inds_di) = (1-s1(inds_di)).*(saar(1,inds_k)' + ...
   r1(inds_di).*(saar(2,inds_k)'-saar(1,inds_k)')) + ...
   s1(inds_di).*(saar(4,inds_k)' + ...
   r1(inds_di).*(saar(3,inds_k)'-saar(4,inds_k)'));

inds_different = find(isfinite(sa_upper(inds_di)) & isnan(sa_lower(inds_di)));

if ~isempty(inds_different)
   sa_lower(inds_di(inds_different)) = sa_upper(inds_di(inds_different));
```

```
        end

        SAAR(inds_di) = sa_upper(inds_di) + t1(inds_di).*(sa_lower(inds_di) - sa_upper(inds_di));

    else
        no_levels_missing = no_levels_missing + 1;
    end
end

inds = find(~isfinite(SAAR));
SAAR(inds) = 0;

in_ocean(inds) = 0;

end

%##########################################################################

function SAAR = gsw_saar_add_mean(saar)

% gsw_saar_add_mean
%==========================================================================
%
% USAGE:
%  SAAR = gsw_saar_add_mean(saar)
%
% DESCRIPTION:
%  Replaces NaN's with nanmean of the 4 adjacent neighbours
%
% INPUT:
%  saar  =  Absolute Salinity Anomaly Ratio of the 4 adjacent neighbours
%                                          [ unitless ]
%
% OUTPUT:
%  SAAR  =  nanmean of the 4 adjacent neighbours              [ unitless ]
%
% AUTHOR:
%  David Jackett
%
% MODIFIED:
%  Paul Barker and Trevor McDougall
%
% VERSION NUMBER: 3.0
%
% REFERENCES:
%  IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of
%   seawater - 2010: Calculation and use of thermodynamic properties.
%   Intergovernmental Oceanographic Commission, Manuals and Guides No. 56,
%   UNESCO (English), 196 pp.  Available from http://www.TEOS-10.org
%
%  McDougall, T. J., D. R. Jackett and F. J. Millero, 2010: An algorithm
%   for estimating Absolute Salinity in the global ocean.  Submitted to
%   Ocean Science, a preliminary version is available at Ocean Sci.
%   Discuss., 6, 215-242.
%   http://www.ocean-sci-discuss.net/6/215/2009/osd-6-215-2009-print.pdf
%   and the computer software is available from http://www.TEOS-10.org
```

```
%
%==========================================================================

saar_mean = mean(saar);
inds_nan = find(isnan(saar_mean));
no_nan = length(inds_nan);

for kk = 1:no_nan
   col = inds_nan(kk);
   inds_kk = find(isnan(saar(:,col)));
   [Inn] = find(~isnan(saar(:,col)));
   if ~isempty(Inn)
      saar(inds_kk,col) = mean(saar(Inn,col));
   end
end

SAAR = saar;

end

%###########################################################################

function SAAR = gsw_saar_add_barrier(saar,long,lat,longs_ref,lats_ref,dlongs_ref,dlats_ref)

% gsw_saar_add_barrier
%==========================================================================
%
% USAGE:
%  SAAR = gsw_saar_add_barrier(saar,long,lat,longs_ref,lats_ref,dlongs_ref,dlats_ref)
%
% DESCRIPTION:
%  Adds a barrier through Central America (Panama) and then averages
%  over the appropriate side of the barrier
%
% INPUT:
%  saar       =  Absolute Salinity Anomaly Ratio                      [ unitless ]
%  long       =  Longitudes of data in decimal degrees east             [ 0 ... +360 ]
%  lat        =  Latitudes of data in decimal degrees north             [ -90 ... +90 ]
%  longs_ref  =  Longitudes of regular grid in decimal degrees east     [ 0 ... +360 ]
%  lats_ref   =  Latitudes of regular grid in decimal degrees north     [ -90 ... +90 ]
%  dlongs_ref =  Longitude difference of regular grid in decimal degrees  [ deg longitude ]
%  dlats_ref  =  Latitude difference of regular grid in decimal degrees   [ deg latitude ]
%
% OUTPUT:
%  SAAR       =  Absolute Salinity Anomaly Ratio                      [ unitless ]
%
% AUTHOR:
%  David Jackett
%
% MODIFIED:
%  Paul Barker and Trevor McDougall
%
% VERSION NUMBER: 3.0
%
% REFERENCES:
%  IOC, SCOR and IAPSO, 2010: The international thermodynamic equation of
```

```
%   seawater - 2010: Calculation and use of thermodynamic properties.
%   Intergovernmental Oceanographic Commission, Manuals and Guides No. 56,
%   UNESCO (English), 196 pp.  Available from http://www.TEOS-10.org
%
%  McDougall, T. J., D. R. Jackett and F. J. Millero, 2010: An algorithm
%   for estimating Absolute Salinity in the global ocean.  Submitted to
%   Ocean Science, a preliminary version is available at Ocean Sci.
%   Discuss., 6, 215-242.
%   http://www.ocean-sci-discuss.net/6/215/2009/osd-6-215-2009-print.pdf
%   and the computer software is available from http://www.TEOS-10.org
%
%==========================================================================

longs_pan = [260.0000 272.5900 276.5000 278.6500 280.7300 295.2170];

lats_pan  = [ 19.5500  13.9700   9.6000   8.1000   9.3300   0];

lats_lines0 = interp1(longs_pan,lats_pan,long);

lats_lines1 = interp1(longs_pan,lats_pan,longs_ref);
lats_lines2 = interp1(longs_pan,lats_pan,(longs_ref+dlongs_ref));

for k0 = 1:length(long)
   if lats_lines0(k0) <= lat(k0)
      above_line0 = 1;
   else
      above_line0 = 0;
   end
   if lats_lines1(k0) <= lats_ref(k0)
      above_line(1) = 1;
   else
      above_line(1) = 0;
   end
   if lats_lines1(k0) <= (lats_ref(k0) + dlats_ref)
      above_line(4) = 1;
   else
      above_line(4) = 0;
   end
   if lats_lines2(k0) <= lats_ref(k0)
      above_line(2) = 1;
   else
      above_line(2) = 0;
   end
   if lats_lines2(k0) <= (lats_ref(k0) + dlats_ref)
      above_line(3) = 1;
   else
      above_line(3) = 0;
   end
   inds = find(above_line ~= above_line0);      % indices of different sides of CA line
   saar(inds,k0) = nan;
end

saar_mean = mean(saar);
inds_nan = find(isnan(saar_mean));
no_nan = length(inds_nan);
```

```
for kk = 1:no_nan
   col = inds_nan(kk);
   inds_kk = find(isnan(saar(:,col)));
   [Inn] = find(~isnan(saar(:,col)));
   if ~isempty(Inn)
    saar(inds_kk,col) = mean(saar(Inn,col));
   end
end

SAAR = saar;

end
```

A.10 GSW v. 3.0 Absolute Salinity Anomaly Ratio (SAAR) Lookup Table

This lookup table is in stored in the Matlab file gsw_data_v3_0.mat, a copy of which is in the Data_Product_SPEC_Artifacts directory on Alfresco.

Appendix B Output Accuracy

There is no accuracy requirement for Density in the OOI requirements database (DOORS).

However, the accuracy of the density calculated by this algorithm can be computed as a function of the accuracy of the algorithm inputs: pressure, temperature, and salinity. To calculate the output accuracy we will use the accuracies given in the Specifications for Conductivity Temperature Depth (CTD) Instruments on Mobile Assets and Profilers (DCN 1331-0001_SPEC_CTD_OOI) and the Specifications for Conductivity Temperature Depth (CTD) Instruments on Fixed Platforms (DCN 1336-0001_SPEC_CTD_OOI) for the algorithm inputs:

±0.005 for practical salinity (SALT-003 fixed; SALT-001 mobile)

±0.002 [°C] for temperature (TEMP-003 fixed; TEMP-001 mobile)

±0.1% of the maximum operational depth range for pressure (PRES-001 fixed; PRES-003 mobile)

Assuming typical ocean values, 5 °C and practical salinity of 35, and a maximum operational depth range of 3000 m (approx. 3 dbar of error in pressure) , the resulting density inaccuracy varies from approximately 0.021 kg m^{-3} at the surface to 0.022 kg m^{-3} at 3000 m (mainly from pressure).

The accuracies of the TEOS-10 algorithms are described in detail in the references.

Appendix C Sensor Calibration Effects

None.