Mixing Toolbox

Institute for Marine and Antarctic Studies

Institute for Marine and Antarctic Studies
University of Tasmania

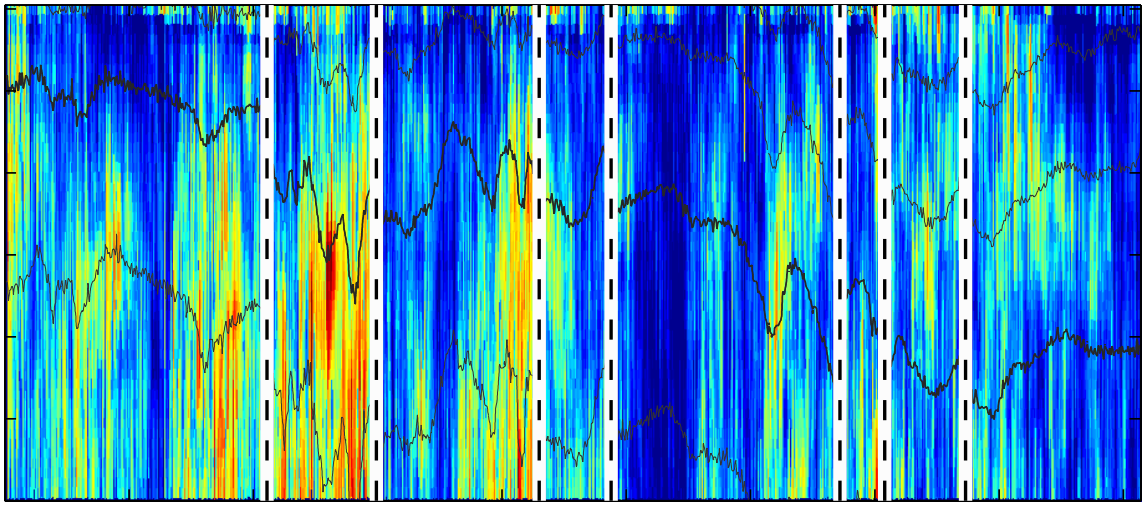
IMAS Technical Report 2014/01

Mixing (MX) Oceanographic Toolbox for EM-APEX* float data applying shear-strain finescale parameterization

*** Electromagnetic Autonomous Profiling Explorer (EM-APEX)**

Amelie Meyer, Helen E. Phillips, Bernadette M. Sloyan and Kurt L. Polzin

July 2014
Revised September 2015



ARC CENTRE OF EXCELLENCE FOR CLIMATE SYSTEM SCIENCE

IMAS Technical Report 2014/01

Mixing Toolbox
July 2014
Revised September 2015

Mixing (MX) Oceanographic Toolbox for EM-APEX* float data applying shear-strain finescale parameterization

*** Electromagnetic Autonomous Profiling Explorer (EM-APEX)**

Amelie Meyer and Helen E. Phillips

Institute for Marine and Antarctic Studies, and
ARC Centre of Excellence for Climate System Science
University of Tasmania
20 Castray Esplanade
Battery Point, Tasmania 7004

Bernadette M. Sloyan

CSIRO
Ocean and Atmosphere Flagship
GPO Box 1538
Hobart, Tasmania 7001

Kurt L. Polzin

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts

Final Report

Funded by the ARC centre of Excellence for Climate System Science, the Quantitative Marine Science program at IMAS and the 2009 CSIRO Wealth from Ocean Flagship Collaborative Fund Postgraduate Scholarship.

IMAS www.imas.utas.edu.au

Abstract: The Mixing Oceanographic Toolbox provides a framework to estimate the dissipation rate and diffusivity from Electromagnetic Autonomous Profiling Explorer (EM-APEX) float observational data. The EM-APEX floats measure the temperature, salinity, pressure, and horizontal velocity of the current. Vertical gradients of velocity and buoyancy are estimated and a finescale parameterization is applied. This method provides order of magnitude estimates of mixing as well as estimates of the regional variability of mixing.

Reference: For bibliographic purposes, this document should be cited as follows:

Meyer, A., H. E. Phillips, B. M. Sloyan, and K. L. Polzin: Mixing (MX) Oceanographic Toolbox for EM-APEX* float data applying shear-strain finescale parameterization, 69pp, Inst. for Marine and Antarctic Studies, Hobart, Technical Report, August 2014.

ISBN: 978-1-86295-770-1

Table of Contents

List of Figures

1 Introduction

In the ocean, mixing results from density overturns driven by wave breaking and Kelvin-Helmholtz instabilities. Turbulent mixing leads to the transfer of kinetic energy into heat by viscous dissipation. A key remaining challenge in physical oceanography is the understanding and parameterization of small-scale mixing in the oceans [2]. In spite of much work on new instruments and techniques to measure turbulence in the ocean, data sets of mixing are still sparse and our limited understanding of the physical processes behind turbulent mixing leads to inaccurate representations of mixing in ocean general circulation models (OGCMs) [3].

The eddy diffusion coefficient of mass across isopycnal surfaces, called diapycnal turbulent eddy diffusivity of mass and hereinafter referred to as diffusivity (K_ρ), is used to characterise turbulent mixing in the ocean. It is defined by the flux gradient relation

$$F_c = -K_c \nabla C, \tag{1}$$

where F_c is the flux of some property C, K_c is the diffusivity of that property and ∇C its gradient. Assuming C is density, a positive diffusivity flux will decrease the density gradient and conversely, a negative diffusivity flux will increase the density gradient (i.e. increase stratification).

The rate of loss of the kinetic energy of the turbulent motion per unit mass through viscosity to heat is referred to as the turbulent kinetic energy dissipation rate (ϵ), hereinafter referred to as the dissipation rate. The dissipation rate has typical values that range from $1 \times 10^{-10}\,\mathrm{W\,kg^{-1}}$ in the abyssal ocean[1], to $1 \times 10^{-1}\,\mathrm{W\,kg^{-1}}$ in areas such as the surf zone. Diffusivity can be estimated from the turbulent kinetic energy dissipation rate by applying the Osborn [4] relation:

$$K_\rho = \Gamma \frac{\epsilon}{N^2}, \tag{2}$$

[1] $1\,\mathrm{W\,kg^{-1}} = 1\,\mathrm{m^2\,s^{-3}}$

where the mixing efficiency ($\Gamma = 0.2$) is assumed to be a constant. For further details about the choice of Γ, see discussion in Polzin et al., (2014) [5].

1.1 Estimating mixing

Turbulence in the ocean is the result of a downscale energy cascade that transfers energy and momentum from large scale currents towards smaller scale internal waves, mostly as a result of nonlinear internal wave-wave interactions [6]. Diapycnal mixing can be estimated directly as an area average using tracer release experiments [7] or indirectly with microstructure profilers (measuring shear with an air-foil shear probe) [8]. Diapycnal mixing can also be indirectly estimated using finescale parameterization derived from empirical and theoretical relations based on finescale observations of the internal wave field characteristic shear and strain. The intensity of turbulent mixing is related to the energy and the shear of the local internal wave field [9]. Many variants of the finescale parameterization exist using observations of shear and strain, or either shear or strain only.

Finescale parameterization is based on (1) the assumption that most of the turbulent mixing is driven by breaking internal waves (locally and remotely generated) in the stratified ocean [10], and (2) the notion of a downscale energy cascade. The finescale parameterizations have been widely used in the past decade [e.g. 11, 12, 13, 14, 15, 16, 17, 18, 19, 20], mostly because the observations needed (profiles of vertical density and velocity) to derive the dissipation rate with this method are much more easily acquired than direct dissipation microstructure observations.

The uncertainties associated with these various finescale parameterization methods are typically $\pm 50\%$ [21, 22, 5]. The method provides order of magnitude estimates of mixing as well as estimates of the spatial gradients of mixing.

1.2 EM-APEX floats

The EM-APEX is an innovative instrument that provides relatively inexpensive, autonomous, high-resolution observations of velocity.

Float characteristics

The EM-APEX profiling float (Figure 1) is a recent addition to the array of autonomous profiling instruments and measures vertical profiles of temperature, salinity and horizontal velocity. EM-APEX floats are the result of a

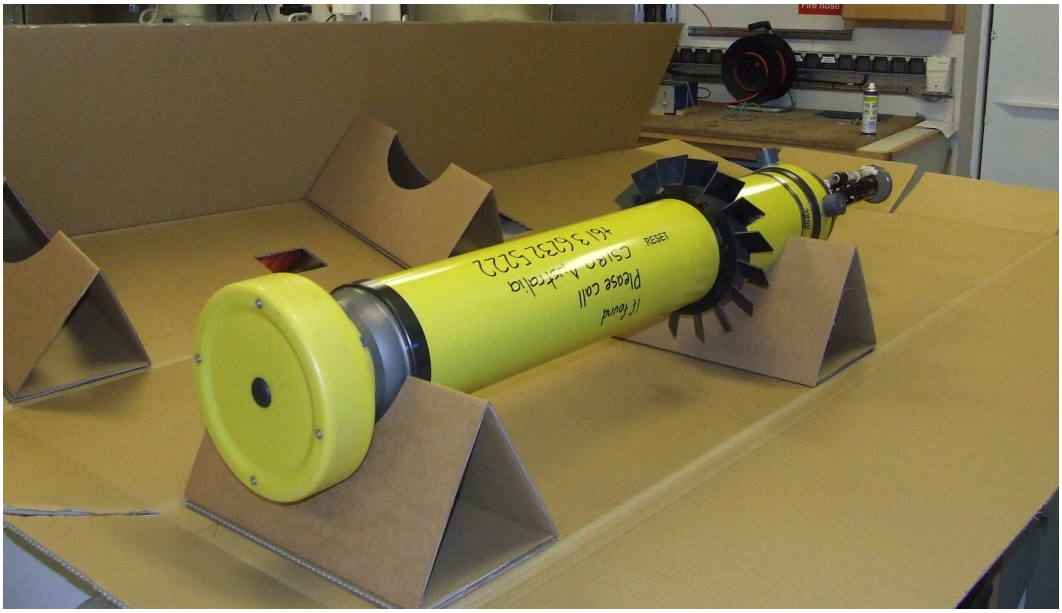


Figure 1. EM-APEX float prior to deployment in the wet lab. The cardboard box is used to protect the float during the deployment procedure. Two characteristics specific to the EM-APEX float are the black fins allowing it to rotate as it sinks through the water column and the grey electrodes close to the top of the float.

collaboration between the University of Washington Applied Physics Laboratory (APL-UW) and Teledyne Webb Research Corporation (WRC). The float combines a standard Teledyne APEX float with an electromagnetic subsystem. The main technical characteristics of the EM-APEX float used in this report are described below.

SBE-41 CTD

On the EM-APEX floats used in the SOFINE experiment, temperature (T), salinity (S) and pressure (P) are measured by a Sea Bird Electronics SBE-41 CTD. The float rate of descent and ascent has a range of 0.10 to 0.12 $\mathrm{m\,s^{-1}}$. The CTD is pumped on demand for approximately 2.5 s, delivering $40\,\mathrm{ml\,s^{-1}}$ flow. The CTD sensor accuracy provided by the manufacturer is 2 dbar for pressure, $2 \times 10^{-3}\,^{\circ}\mathrm{C}$ for temperature, and 2×10^{-3} for conductivity. The CTD data is processed in 2.2 m vertical bins for preliminary analysis and then in 3 m vertical bins when deriving mixing estimates to match the electro-magnetic subsystem data vertical resolution (see below).

EM-APEX electromagnetic subsystem

The EM-APEX electromagnetic subsystem has a compass, accelerometer and five electrodes to estimate the magnitude of horizontal currents (Figure 2). The horizontal velocity is estimated using the principle that a conductor moving through a magnetic field develops an electrical potential drop across the conductor. In this application, the conductor is seawater and the magnetic field is that of the Earth [23]. The EM-APEX electromagnetic subsystem voltmeter measures this electric potential difference across the body of the float with two independent pairs of electrodes.

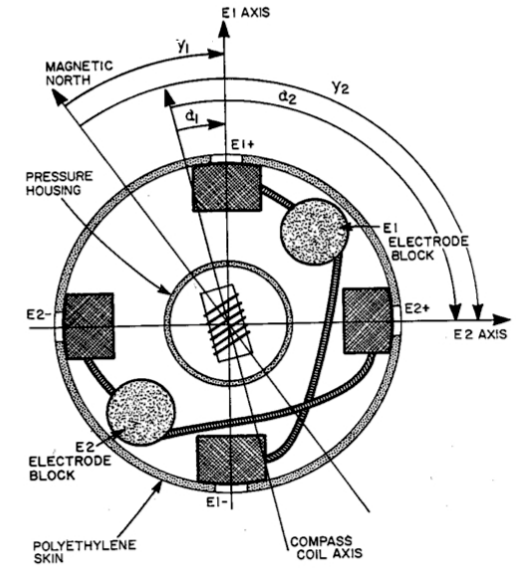


Figure 2. Cross-section of the EM-APEX electromagnetic subsystem at the level of the electrodes (Sandford 1978 [1] p191).

The float rotates with a period of 12 s due to external fins, and the motionally-induced electric field is sampled at 20 Hz and then averaged with a sinusoidal fit. The fit is made over 50 s long segments of data with 25 s between successive fits, acting as a low-pass filter [24]. The fits provide an estimate of the horizontal current and the residuals provide an estimate of the velocity noise level at a vertical resolution of approximately 3 m. Measured voltages are transmitted over the Iridium global phone system and the processing of the voltages into eastward and northward velocity components is shore-based. The velocity profiles are relative to a depth-independent offset. Given the GPS positions, by pairing profiles, the absolute velocity profile can be estimated [25]. Profiles are gridded into 3 m vertical bins for the mixing analysis.

2 Installing the Mixing (MX) Oceanographic Toolbox in Matlab

Step 1

Download the MX Oceanographic Toolbox in Matlab from:
www.mathworks.com.au/matlabcentral/fileexchange/47595-mixing–mx–oceanographic-toolbox-for-em-apex-float-data

Step 2

Create directory called 'MX' and unzip the Toolbox into this directory. Make sure that the two subfolders 'figures', and 'private1' have been extracted.

Step 3

In Matlab, add the 'MX' directory to your Matlab path using the option 'Add with subfolders...'. In the menu, go to 'File' or 'Home', 'Set Path...', 'Add with subfolders...'. Make sure that the two subfolders 'figures', and 'private1' have been added to the path.

Step 4

From the MX directory, run *mx_check_functions* to check that the toolbox is correctly installed. Using the function as such applies a mixing finescale parameterization to a set of 36 profiles from the EM-APEX float 3951. Figures from the analysis are saved in the 'MX' directory under the folder 'figures'.

Once the MX Oceanographic Toolbox is installed, you can run *mx_mixing_analysis.m* for your own EM-APEX data. First you need to define the input data and parameters. In matlab, type *help mx_mixing_analysis.m* to get a description of the input parameters and what format the input data file needs to follow. **Note that you will need the GSW Oceanographic Toolbox installed on your computer for the MX Oceanographic Toolbox to run. The GSW Oceanographic Toolbox can be downloaded from www.TEOS-10.org.**

3 Overview of the Mixing (MX) Oceanographic Toolbox library

3.1 Structure of the MX Toolbox mixing analysis

A detailed descriptions of the theory and methods applied in this library can be found in Meyer et al. (2015) [26].

3.1.1 Load and build the dataset

In this section, we load the input data set of profiles from the EM-APEX float and the parameters ('mx_parameters.mat') that will be applied in the analysis. Next we extract the relevant data from the input data set and create a structure 'initial_data.mat'.

3.1.2 Derive the fall rate of the EM-APEX float

In this short section, the fall rate of the EM-APEX float is estimated at each bin depth of each profile and added to the 'initial_data.mat' structure.

3.1.3 Grid the data

Both the CTD data (temperature and salinity) and the EM data (velocity and fall rate) are gridded on a regular pressure grid using 'mx_grid_initialdata.m'. This grid is preset to vertical intervals of 2.2 dbar for the CTD data and 3 dbar for the EM data. These values can be changed manually in the code 'mx_grid_initialdata.m'. The gridded data is saved in the same structure 'initial_data.mat'.

3.1.4 Derive initial variables

In this section, we derive some initial variables needed for the mixing calculations. First we evaluate Conservative Temperature (Θ) as a function of Absolute Salinity (S_A), in situ temperature (T) and pressure (P) using the Gibbs Seawater Oceanographic Toolbox (GSW). Next we estimate the potential density (referenced to the sea surface) anomaly using the Gibbs Seawater Oceanographic Toolbox (GSW). We also estimate the buoyancy frequency (N), the mixed layer depth (MLD) and the current speed (See **(author?)** [26] for derivation details). Each of these variables are saved in 'initial_data.mat'.

3.1.5 Plot initial variables

The initial variables are changed from a structure file into a matrix 'initial_data_gridded.mat' using 'mx_grid_all_initial_data.m'. The temperature, salinity, buoyancy frequency, mixed layer depth and current speed are then plotted and the figures are saved in a directory designated in the inputs.

3.1.6 Derive mixing variables

In this section, we derive some mixing variables needed for the mixing calculations. First we evaluate a reference background buoyancy frequency ($N2\_ref$). Next we estimate the buoyancy frequency on a 100 dbar vertical length scale ($N2\_100m$). We also estimate the strain and the shear in each EM-APEX profile. Each of these variables are saved in a new structure 'mixing_data.mat'.

3.1.7 Derive mixing

All the mixing variables are gridded onto a standard 3 dbar grid ('mx_grid_mixingdata.m') and saved into a new structure 'mixing_data_gridded.mat'. The shear-strain finescale parameterization is applied ('mx_derive_mixing.m') and the dissipation rate (ϵ) and diffusivity (K_ρ) estimated. The mixing data is saved in a new structure 'mixing_data_gridded_run.mat'.

3.1.8 Plot mixing variables

The dissipation rate (ϵ), diffusivity (K_ρ), ratio of CCW to CW rotating shear variance (ϕ_{CCW}/ϕ_{CW}) and Shear-to-strain variance ratio (R_ω) are plotted. The figures are saved in a directory designated in the inputs.

3.2 List of input parameters

$U =$ '1';	U velocity sensor: can be either '1' or '2'
$V =$ '1';	V velocity sensor: can be either '1' or '2'
$moving\_window = 20$;	Number of consecutive profiles used to estimate N2_ref.
$dzN2ref = 24$;	Vertical number of bin depth used to estimate N2_ref.
$dzN2 = 6$;	Differential length for N2 calculation: e.g. 6 (m).
$drho = 0.03$;	Density gradient to derive the mixed layer depth: 0.03 kgm^{-4}
$dz = 3$;	Main data set pressure grid interval (m).
$dzs = 6$;	Vertical scale over which shear is derived. Must be a multiple of dz.
$fftpt = 128$;	Number of points for the fast fourier transform e.g. 128 but could be 32,64,128... or any power of 2.
$lzmin\_fixed = 50$;	Minimum wavelength integration for shear/strain spectra (m).
$lzmax\_fixed = 300$;	Maximum wavelength integration for shear/strain spectra (m).

Constants:

$R = 5$;	Average shear to strain ratio for this data set. For the first run, use any value between 3 and 15 .
$gamma = 0.2$;	Mixing efficiency gamma. Can be tuned if necessary (see ref.: [22, 27, 5])
$epsilon0 = 8x10^{-10}$;	Dissipation rate from the GM76 model. Do not change.
$N0 = 0.00524$;	Buoyancy frequency from the GM76 model. Do not change.
$f0 = sw\_f(32.5)$;	Inertial frequency from the GM76 model. Do not change.
$E = 6.3e - 5$;	Dimensionless energy level from the GM76 model. Do not change.
$b = 1300$;	Scale depth of thermocline (m). Do not change.
$jstar = 3$;	Mode number. Do not change.

Spectral method: Choice involves trade-offs between confidence and variance preservation.

$spectral\_method = 'Tycho2'$;	Matlab routine where $Tycho2 = 10sin^2window$
$cospectral\_method = 'Tycho2\_cospectra'$;	Matlab routine which defines the decomposition spectral method for both CW and CCW.

Spectral corrections: Switches for various spectral corrections. Models to correct the high frequency portion of the spectra due to instrument limits.

$switch\_fd = 1$;	First-differencing correction. When set to one, the correction is on. This correction is instrument dependent and only applies to EM-APEX floats.

Wavelength of integral: Wavelength integration for shear and strain spectra to internal waveband. The finestructure epsilon comes from an integral of shear and strain power over a certain wavenumber/wavelength range - this sets the limits of wavelength integration. There are two options in the code: either have a fixed minimum value of integration (lzmin) or evaluate lzmin using the data set. The latest option is the default option and only possible because the EM-APEX floats have sufficient vertical resolution to resolve the transition into wave breaking, often nominally set at 10 m vertical wavelength. If needed, lzmin can be set to a fixed value.

$lzmin\_fixed$;	This value is sensitive. If being used, choose with care.
$lzmax\_fixed$;	300 m is a typical number.
$crit\_rat = NaN$;	Threshold to determine lzmin. lzmin is the minimum wavelength for which the noise spectra is less than critical ratio * spectrum.
$lzmin\_threshold = NaN$;	Minimum wavelength threshold (lzmin) is set to the maximum of lzmin_threshold. lzmin is determined from the noise threshold.

3.3 List of output variables

$fltid$	[1x36 double]	Float number
$profile\_number$	[1x36 double]	Profile number for this data
lon	[1x36 double]	Longitude (^{o}E)
lat	[1x36 double]	Latitude (^{o}N)
$surface\_mlt$	[1x36 double]	Profile surface time
MLD	[1x36 double]	Mixed layer depth (m)
P	[550x36 double]	Pressure (dbar)
SA	[550x36 double]	Absolute Salinity (gkg^{-1})
CT	[550x36 double]	Conservative temperature (^oC)
$sigma\_0$	[550x36 double]	Potential density ref. to sea surface (kgm^{-3})
$N2$	[550x36 double]	Buoyancy frequency (rads^{-1})
$N2\_ref$	[550x36 double]	Reference buoyancy frequency (rads^{-1})
$N2\_100m$	[550x36 double]	Buoyancy frequency over 100m vertical window (rads^{-1})
$strain$	[550x36 double]	Strain
$fallrate$	[550x36 double]	Fall rate of the EM-APEX float (ms^{-1})
U	[550x36 double]	East horizontal velocity (ms^{-1})
V	[550x36 double]	North horizontal velocity (ms^{-1})
$speed$	[550x36 double]	Current speed (ms^{-1})
$shear$	[550x36 double]	Shear (s^{-1})
$epsilon\_shear$	[52x36 double]	Dissipation rate from shear parameterization (Wkg^{-1})
Kz_shear	[52x36 double]	Diffusivity from shear parameterization (m^{2}s^{-1})
P_m_shear	[52x36 double]	Pressure for shear data (dbar)
$shear\_variance$	[52x36 double]	Shear variance
$CW\_S\_variance$	[52x36 double]	Clockwise shear variance
$CCW\_S\_variance$	[52x36 double]	Counter-clockwise shear variance
Mc_shear	[52x36 double]	Cutoff vertical wavenumber from shear (cpm)
$epsilon\_strain$	[65x36 double]	Dissipation rate from strain parameterization (Wkg^{-1})
Kz_strain	[65x36 double]	Diffusivity from strain parameterization (m^{2}s^{-1})
P_m_strain	[65x36 double]	Pressure for strain data (dbar)
$strain\_variance$	[65x36 double]	Strain variance
$critiWave\_strain$	[65x36 double]	Cutoff vertical wavenumber from strain (cpm)
$epsilon$	[550x36 double]	Dissipation rate from shear-strain parameterization (Wkg^{-1})
Kz	[550x36 double]	Diffusivity from shear-strain parameterization (m^{2}s^{-1})
P_m	[550x36 double]	Pressure grid for mixing data (dbar)
Rw	[550x36 double]	Shear-to-strain variance ratio

3.4 List of functions and data files in the MX Toolbox

The MX Oceanographic Toolbox main function *mx_mixing_analysis.m* calls the following library functions:

mx_build_initial_data	Build the initial data set
mx_derive_fallrate	Estimate the EM-APEX float fall rate
mx_grid_initialdata	Grid the CTD and EM data regular pressure grids
mx_derive_abs_T_S	Derive the Absolute Salinity and Conservative Temperature
mx_derive_potential_density_anomaly	Derive the potential density referenced to sea-surface anomaly
mx_derive_N2	Derive the buoyancy frequency N^2
mx_derive_mixed_layer_depth	Estimate the mixed layer depth
mx_derive_current_speed	Derive the current speed
mx_grid_all_initial_data	Grid all the initial data from a structure into a matrix format
mx_plot_temperature	Plot all the conservative temperature profiles
mx_plot_salinity	Plot all the absolute salinity profiles
mx_plot_mixed_layer_depth	Plot the mixed layer depths overlying the density profiles
mx_plot_current_speed	Plot all the current speed profiles
mx_derive_N2_ref	Derive a reference buoyancy frequency N^2ref
mx_derive_N2_100m	Derive a buoyancy frequency over a 100m vertical window
mx_derive_strain	Derive the strain profiles
mx_derive_shear	Derive the shear profiles
mx_grid_mixingdata	Grid both the CTD and EM data to the same pressure grid
mx_derive_mixing	Umbrella function in which the shear-strain parameterization is applied
mx_derive_mixing_shear	Shear parameterization
mx_derive_mixing_strain	Strain parameterization
mx_derive_mixing_shearstrain	Combining shear and strain parameterization to estimate mixing
mx_plot_dissipation_rate	Plot all the dissipation rate profiles
mx_plot_diffusivity	Plot all the diffusivity profiles
mx_plot_CCW_CW	Plot all the ratio of rotary shear variance profiles
mx_plot_Rw	Plot all the shear-to-strain variance ratio profiles
mx_check_functions	Check that the MX library has been properly installed

The MX data files

float_data_vmx.mat	contains 36 vertical profiles of salinity, temperature, and horizontal current velocity from the EM-APEX float 3951 at known longitude, latitude and time.
mx_parameters.mat	contains the default input parameters for the function *mx_mixing_analysis.m*.

4 Application to EM-APEX float data from the SOFine project

4.1 SOFine project

Eight EM-APEX floats were deployed during the RRS James Cook cruise JC029 in late 2008 as part of the Southern Ocean FINEstructure (SOFine) project. The SOFine project is a U.K., U.S. and Australian collaborative experiment to investigate the impact of finescale processes on the momentum balance in the Antarctic Circumpolar Current [28]. The floats were deployed on the northern edge of the Kerguelen Plateau in late 2008 to drift along the Antarctic Circumpolar Current (ACC) (Figure 3).

While drifting north of the Kerguelen Plateau, the floats were programmed to surface twice a day, measuring four profiles of temperature, salinity, pressure and horizontal velocity from the sea surface to 1600 m, with a parked drift of 8 hours at 1000 m (Figure 4). The floats spent typically 30 minutes at the surface to transmit the profile data over the Iridium satellite network as opposed to floats transmitting over the Argos communication system spending on average 10 hours at the surface. Using the Iridium communication system allows for two-way communication as well as faster data transfer and therefore the option to sample at higher resolution. The floats only sampled the top 1600 m of the water column rather than going to their maximum 2000 m so that consecutive up profiles are approximately half an inertial period apart (17 hours at 45° S latitude), and consequently the inertial frequency can be resolved in the data.

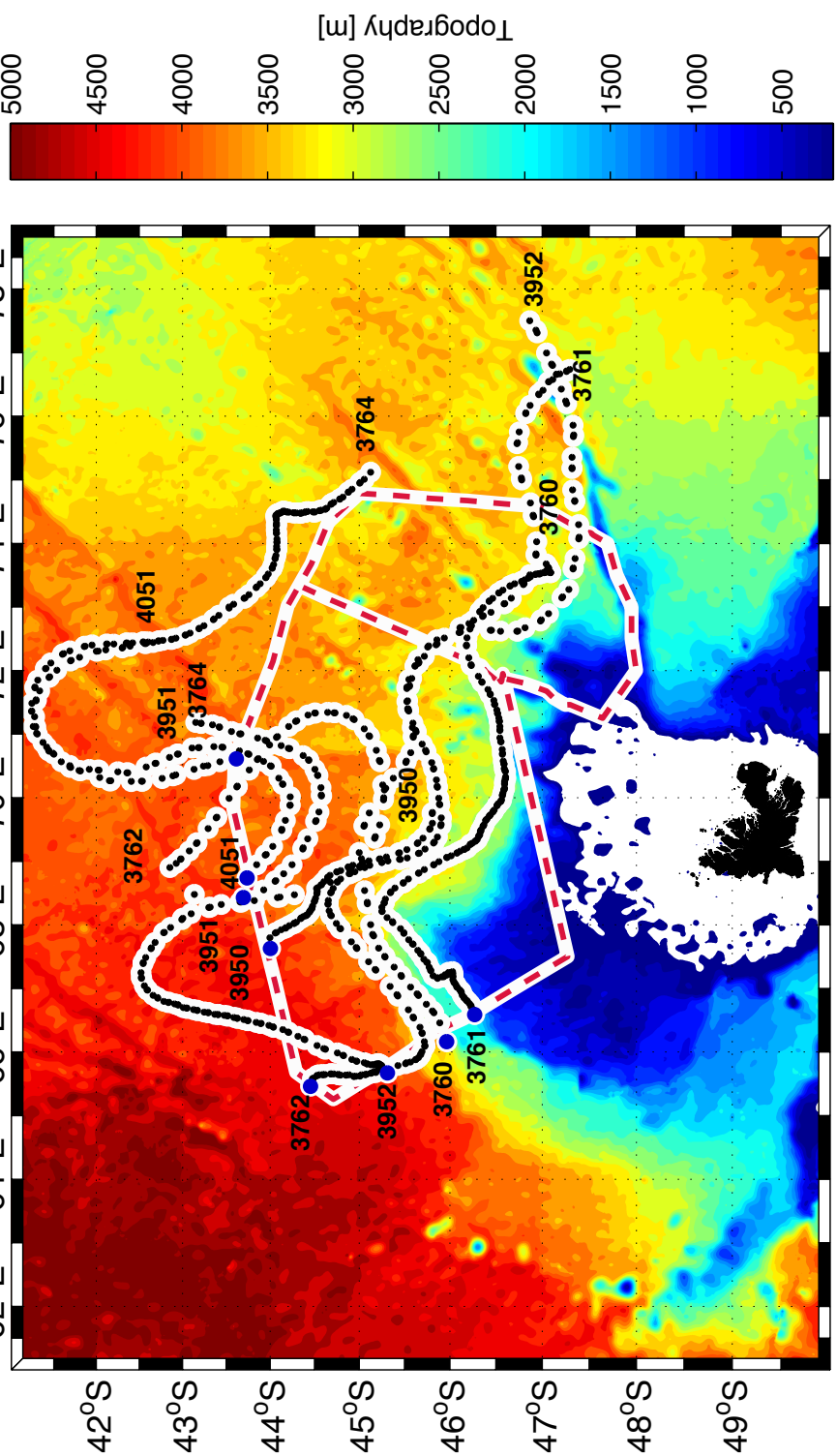


Figure 3. EM-APEX float trajectories overlying topography in colour scale ranging from 200 to 5000 m at 200 m increments. Each black dot denotes a float profile surface location and the deployment location is highlighted by a bigger blue dot. The float numbers are indicated at the first and last profile of each float. There are 914 profiles, sampled between the 18^{th} November 2008 and 30^{th} January 2009. Also shown is the voyage track of RRS James Cook JC029 (dash red line).

The cruise track with the deployment positions of the floats is shown in Figure 3. Five of the eight EM-APEX floats were deployed at a CTD station, allowing calibration of the float salinity sensor with the CTD salinity observations. The data transmitted by the floats over the Iridium phone system were received by a data server at the University of Tasmania and converted to relative velocity using software developed by John Dunlap at the University of Washington in the research group of Prof. Tom Sanford. This was followed by extensive processing to calibrate the instruments, automate the quality control of the velocity data, and to convert relative velocity to absolute velocity [25].

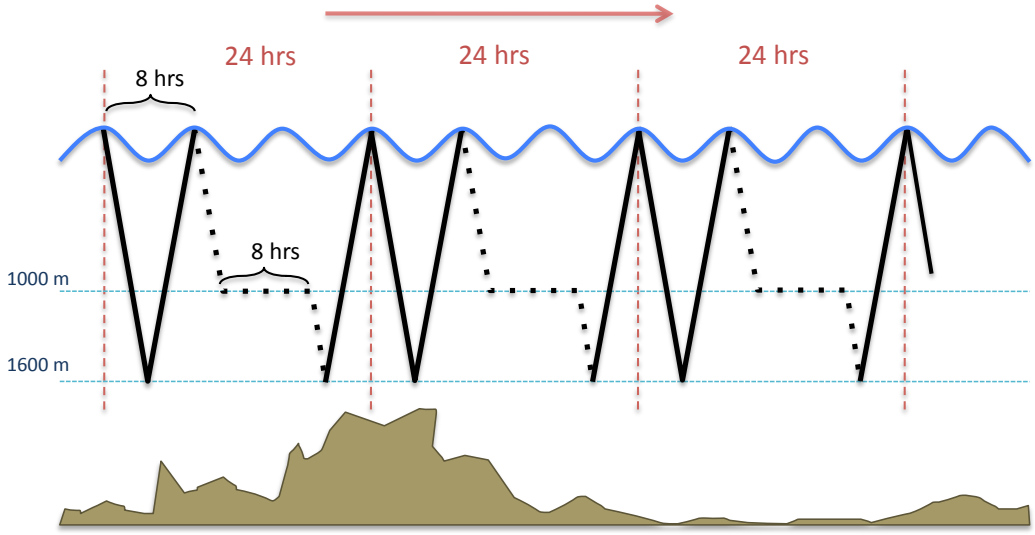


Figure 4. EM-APEX float vertical sampling strategy. The black line denotes the path of the float in the water column. The dotted line refers to profiles not used in this study.

4.2 Initial variables

A detailed description of the methods and results can be found in Meyer et al. (2015) [26].

4.2.1 Temperature

The SOFine data set mean temperature is 4.65 °C.

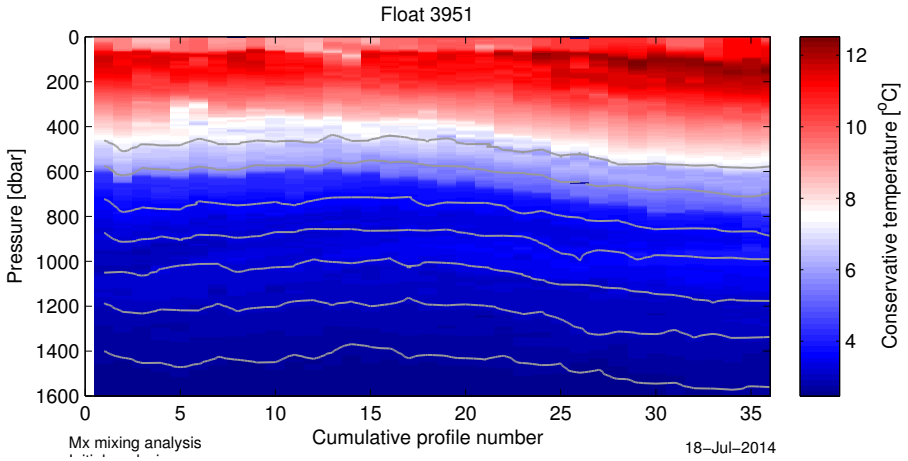


Figure 5. Vertical distribution of conservative temperature along the trajectory of float 3951. Potential density contours every 0.1 $\mathrm{kg\,m^{-3}}$ between σ_θ =27.0 and σ_θ =29.0 $\mathrm{kg\,m^{-3}}$ are shown (grey)

4.2.2 Salinity

The SOFine data set mean salinity is 34.45.

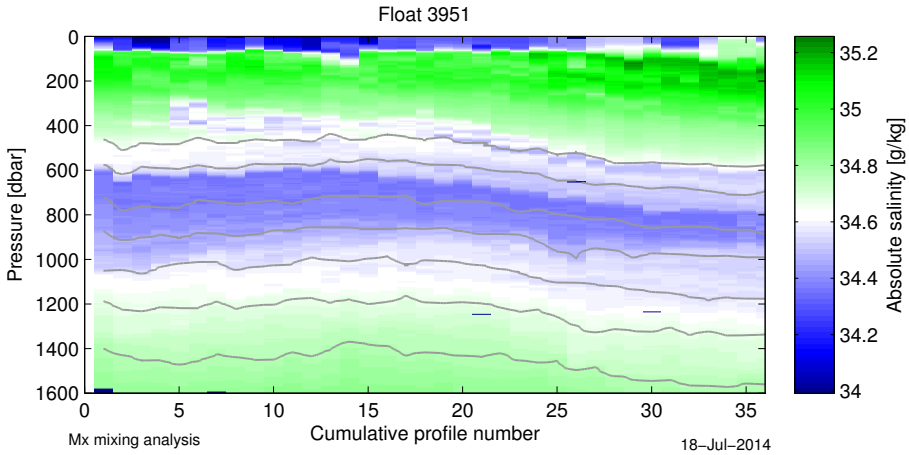


Figure 6. Vertical distribution of absolute salinity along the trajectory of float 3951. Potential density contours every 0.1 $\mathrm{kg\,m^{-3}}$ between σ_θ =27.0 and σ_θ =29.0 $\mathrm{kg\,m^{-3}}$ are shown (grey)

4.2.3 Buoyancy frequency

The SOFine data set mean buoyancy frequency is 8.6×10^{-6} rad^2 s^{-1}.

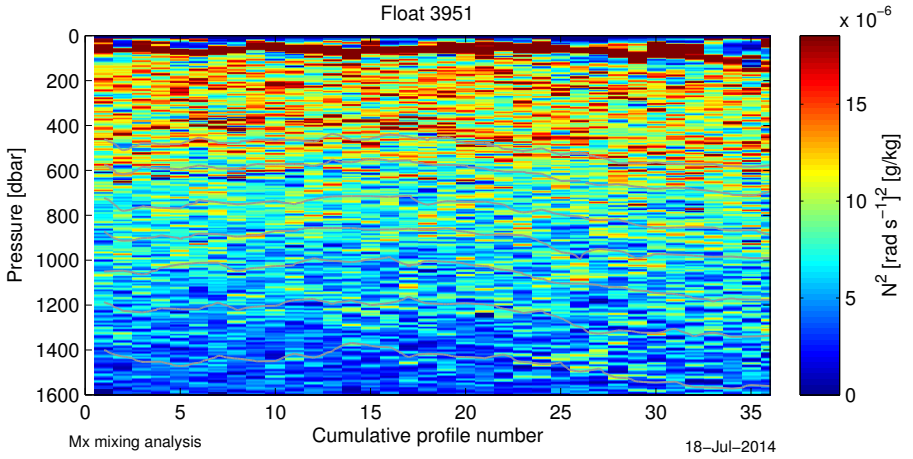


Figure 7. Vertical distribution of the squared buoyancy frequency N^2 along the trajectory of float 3951. Potential density contours every 0.1 $\mathrm{kg\,m^{-3}}$ between σ_θ =27.0 and σ_θ =29.0 $\mathrm{kg\,m^{-3}}$ are shown (grey)

4.2.4 Mixed layer depth

The SOFine data set mean mixed layer depth is 53 m.

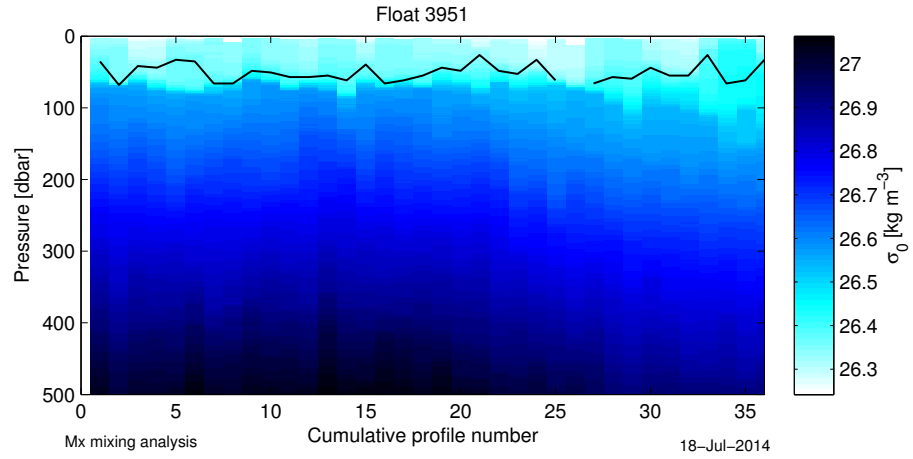


Figure 8. Mixed layer depth (black contour) overlying vertical distribution of potential density along the trajectory of float 3951.

4.2.5 Current speed

The SOFine data set mean current speed is 0.29 $\mathrm{ms^{-1}}$.

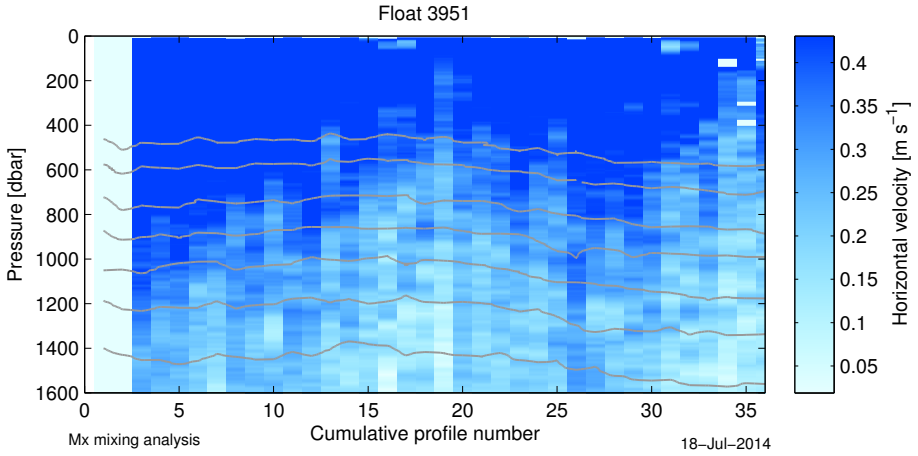


Figure 9. Vertical distribution of horizontal speed along the trajectory of float 3951. Potential density contours every 0.1 $\mathrm{kg\,m^{-3}}$ between σ_θ =27.0 and σ_θ =29.0 $\mathrm{kg\,m^{-3}}$ are shown (grey).

4.3 Mixing variables

A detailed description of the methods and results can be found in Meyer et al. (2015) [26].

4.3.1 Dissipation rate

The SOFine data set mean dissipation rate (ϵ) is 9×10^{-10} $\mathrm{W\,kg^{-1}}$ with a 90% confidence interval of 7.0×10^{-10} $\mathrm{W\,kg^{-1}}$ to 9.7×10^{-10} $\mathrm{W\,kg^{-1}}$ [26].

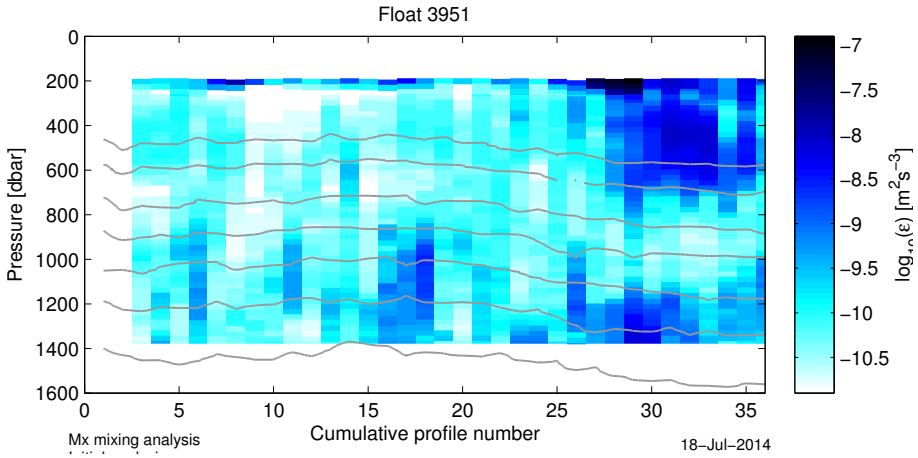


Figure 10. Vertical distribution of the dissipation rate (ϵ) along the trajectory of float 3951. Potential density contours every 0.1 $\mathrm{kg\,m^{-3}}$ between σ_θ =27.0 and σ_θ =29.0 $\mathrm{kg\,m^{-3}}$ are shown (grey).

4.3.2 Diffusivity

Values of diffusivity (K_ρ) estimated from the SOFine EM-APEX data with the shear-strain parameterization show large variability and vary by as much as four orders of magnitude in one profile [26]. Some regions show particularly weak dif-

fusivities of $O(10^{-6}\ \mathrm{m^2\,s^{-1}})$. Other regions show enhanced diffusivity values of $O(10^{-3})\ \mathrm{m^2\,s^{-1}}$ below 600 m in the vicinity of rough topography and of $O(10^{-4})\ \mathrm{m^2\,s^{-1}}$ in the upper 200 m. The overall mean diffusivity is $3 \times 10^{-5}\ \mathrm{m^2\,s^{-1}}$ with a 90% confidence interval of $2.5 \times 10^{-5}\ \mathrm{m^2\,s^{-1}}$ to $3.4 \times 10^{-5}\ \mathrm{m^2\,s^{-1}}$ [26].

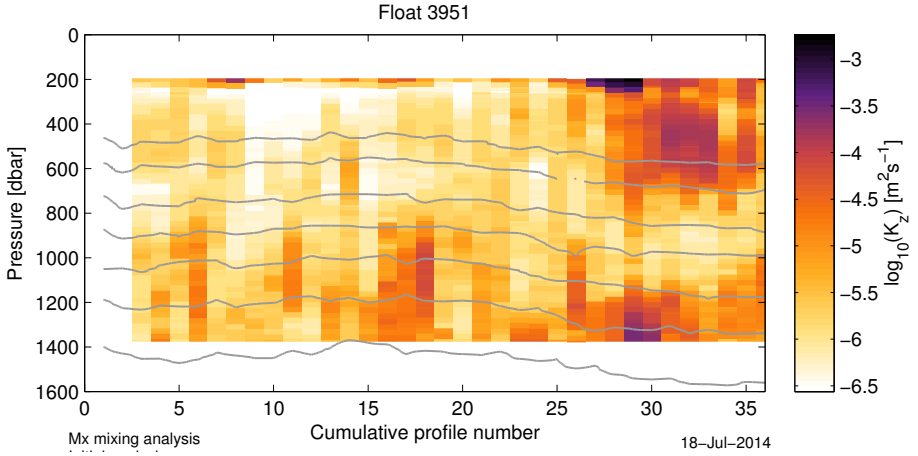


Figure 11. Vertical distribution of diffusivity (K_ρ) along the trajectory of float 3951. Potential density contours every 0.1 $\mathrm{kg\,m^{-3}}$ between σ_θ =27.0 and σ_θ =29.0 $\mathrm{kg\,m^{-3}}$ are shown (grey).

4.3.3 Ratio of rotary shear variance

The ratio of rotary shear variance (CCW/CW) can be used to infer the dominant direction of energy propagation of internal waves [29]. A dominance of CCW polarisation of the shear suggests predominantly downward energy propagation in the Southern Hemisphere (upward phase propagation). A dominance of CW polarisation of the shear indicates that upward energy propagation dominates (downward phase propagation).

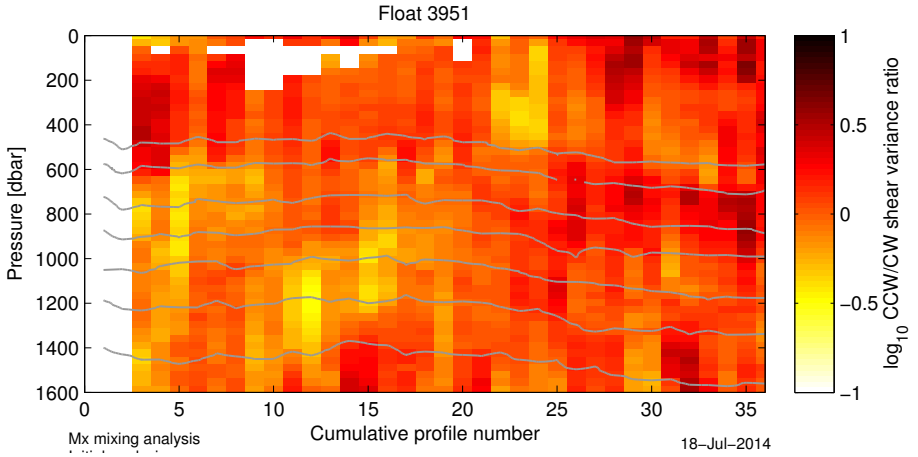


Figure 12. Vertical distribution of the ratio of rotary shear variance (CCW/CW) along the trajectory of float 3951. Potential density contours every 0.1 $\mathrm{kg\,m^{-3}}$ between σ_θ =27.0 and σ_θ =29.0 $\mathrm{kg\,m^{-3}}$ are shown (grey).

4.3.4 Shear-to-strain ratio

The shear-to-strain variance ratio (R_ω) is an estimate of the mean aspect ratio of the internal wave field. It can be used to estimate the bulk frequency of the internal wave field content [5]. Higher values of R_ω imply a dominant presence of near-inertial waves; lower values of R_ω can be attributed to the presence of more high-frequency internal waves or the presence of shear instabilities [5].

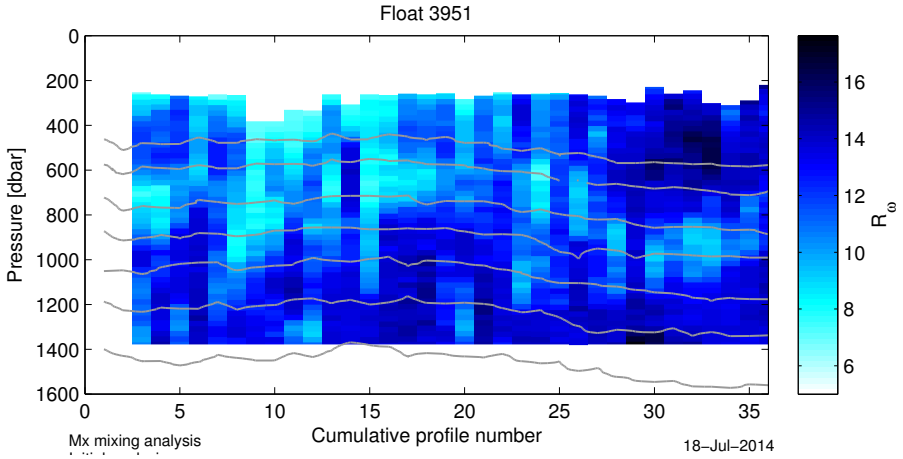


Figure 13. Vertical distribution of the shear-to-strain ratio (R_ω) along the trajectory of float 3951. Potential density contours every 0.1 $\mathrm{kg\,m^{-3}}$ between σ_θ =27.0 and σ_θ =29.0 $\mathrm{kg\,m^{-3}}$ are shown (grey).

5 Acknowledgements

We thank Alberto Naveira Gabarato and Stephanie Waterman for their valuable guidance and contributions; John Dunlap for his valuable support during the data collection and quality control stages of the project. We also thank Paul Barker for all his useful advice in creating and coding a matlab library. AM was supported by the ARC centre of Excellence for Climate System Science, the Quantitative Marine Science program at IMAS, and the 2009 CSIRO Wealth from Ocean Flagship Collaborative Fund Postgraduate Scholarship. HP is funded by the Australian Research Council Discovery Project (DP130102088). KLPs salary support was provided by Woods Hole Oceanographic Institution bridge support funds. Funding for the EM-APEX experiment was provided by the Australian Research Council Discovery Project (DP0877098) and Australian Government DEWHA Projects 3002 and 3228.

References

[1] T B Sanford, R G Drever, and J H Dunlap. A velocity profiler based on the principles of geomagnetic induction. *Deep-Sea Research*, 25(2):183–210, 1978.

[2] M H Alford, M C Gregg, V Zervakis, and H Kontoyiannis. Internal wave measurements on the Cycladic Plateau of the Aegean Sea. *Journal of Geophysical Research*, 117:C01015, 2012.

[3] Carl Wunsch and Raffaele Ferrari. Vertical Mixing, Energy, and the General Circulation of the Oceans. *Annual Review of Fluid Mechanics*, 36(1):281–314, January 2004.

[4] T R Osborn. Estimates of the local-rate of vertical diffusion from dissipation measurements. *Journal of Physical Oceanography*, 10(1):83–89, 1980.

[5] K L Polzin, A C Naveira Garabato, T N Huussen, B M Sloyan, and S N Waterman. Finescale parameterizations of turbulent dissipation. *Journal of Geophysical Research - Oceans*, 119:1383–1419, 2014.

[6] C H McComas and P Muller. The dynamic balance of internal waves. *Journal of Physical Oceanography*, 11(7):970–986, 1981.

[7] J. R. Ledwell, L. C. St. Laurent, J. B. Girton, J. M. Toole, L C St Laurent, J. B. Girton, and J. M. Toole. Diapycnal Mixing in the Antarctic Circumpolar Current. *Journal of Physical Oceanography*, 41(1):241–246, January 2011.

[8] M C Gregg. Diapycnal mixing in the thermocline - A review. *Journal of Geophysical Research*, 92(C5):5249–5286, 1987.

[9] K L Polzin, J M Toole, and R W Schmitt. Finescale parameterizations of turbulent dissipation. *Journal of Physical Oceanography*, 25(3):306–328, 1995.

[10] Matthew H Alford and Michael C Gregg. Near-inertial mixing: Modulation of shear, strain and microstructure at low latitude. *Journal of Geophysical Research*, 106(C8):16,916–947,968, 2001.

[11] C. Mauritzen, K L Polzin, M S McCartney, R C Millard, and D E West-Mack. Evidence in hydrography and density fine structure for enhanced vertical mixing over the Mid-Atlantic Ridge in the western Atlantic. *Journal of Geophysical Research*, 107(C10):11–19, 2002.

[12] A C Naveira Garabato, K L Polzin, B A King, K J Heywood, and M Visbeck. Widespread intense turbulent mixing in the Southern Ocean. *Science*, 303(5655):210–213, 2004.

[13] B. M. Sloyan. Spatial variability of mixing in the Southern Ocean. *Geophysical Research Letters*, 32(L18603):n/a–n/a, September 2005.

[14] E Kunze, E Firing, J M Hummon, T K Chereskin, A M Thurnherr, Ocean Sciences, British Columbia, San Diego, La Jolla, and Lamont-doherty Earth Observatory.

Global abyssal mixing inferred from lowered ADCP shear and CTD strain profiles. *Journal of Physical Oceanography*, 36(12):2350–2352, 2006.

[15] M. H. Alford, J. a. MacKinnon, Zhongxiang Zhao, Rob Pinkel, Jody M Klymak, and Thomas Peacock. Internal waves across the Pacific. *Geophysical Research Letters*, 34(24):L24601, December 2007.

[16] Young-Hyang H Park, Jean-Luc L Fuda, Isabelle Durand, Alberto C. Naveira Garabato, and A C N Garabato. Internal tides and vertical mixing over the Kerguelen Plateau. *Deep-Sea Research Part II-Topical Studies in Oceanography*, 55(5-7):582–593, March 2008.

[17] Ilker Fer, Ragnheid Skogseth, and Florian Geyer. Internal Waves and Mixing in the Marginal Ice Zone near the Yermak Plateau. *Journal of Physical Oceanography*, 40(7):1613–1630, July 2010.

[18] J A MacKinnon, M H Alford, R Pinkel, J Klymak, and Z Zhao. Mixing across the Pacific. *Journal of Physical Oceanography*, 2011.

[19] Lixin Wu, Zhao Jing, Steve Riser, and Martin Visbeck. Seasonal and spatial variations of Southern Ocean diapycnal mixing from Argo profiling floats. *Nature Geoscience*, 4(6):363–366, June 2011.

[20] C. B. Whalen, L. D. Talley, and J. a. MacKinnon. Spatial and temporal variability of global ocean mixing inferred from Argo profiles. *Geophysical Research Letters*, 39(18):L18612, September 2012.

[21] K Polzin, E Kunze, J Hummon, and E Firing. The finescale response of lowered ADCP velocity profiles. *Journal of Atmospheric and Oceanic Technology*, 19(2):205–224, 2002.

[22] S A Thorpe. *The Turbulent Ocean*. Number 439p. Cambridge University Press, 2005.

[23] T B Sanford, J H Dunlap, J A Carlson, D C Webb, and J B Girton. Autonomous velocity and density profiler: EM-APEX. *Proceedings of the IEEE/OES eighth working conference on current measurement technology - Proceedings*, pages 152–156, 2005.

[24] Thomas B. Sanford, James F. Price, and James B. Girton. Upper-Ocean Response to Hurricane Frances (2004) Observed by Profiling EM-APEX Floats. *Journal of Physical Oceanography*, 41(6):1041–1056, June 2011.

[25] H E Phillips and N L Bindoff. On the non-Equivalent Barotropic structure of the Antarctic Circumpolar Current: An observational perspective. *Journal of Geophysical Research*, 119:doi:10.1002/2013JC009516, 2014.

[26] A Meyer, B M Sloyan, K L Polzin, H E Phillips, and N L Bindoff. Mixing variability in the Southern Ocean. *Journal of Physical Oceanography*, 45:966–987, 2015.

[27] G.N. N Ivey, K.B. B Winters, and J.R. R Koseff. Density stratification, turbulence, but how much mixing? *Annual Review of Fluid Mechanics*, 40(1):169–184, January 2008.

[28] A C Naveira Garabato. RRS James Cook Cruise 29, 01 Nov-22 Dec 2008. SOFine Cruise Report: Southern Ocean Finestructure. Southampton Cruise Report 35, National Oceanography Centre, 2009.

[29] K D Leaman and T B Sanford. Vertical energy propagation of internal waves - Vector spectral analysis of velocity profiles. *Journal of Geophysical Research*, 80(15):1975–1978, 1975.

Appendix A: Symbols and Notations

The list below contains most of the parameters and variables used in this report with their equivalent symbol, units, and value if appropriate.

ϕ_{CCW}/ϕ_{CW}	Ratio of CCW to CW rotating shear variance	7
ϵ	Turbulent kinetic energy dissipation rate (Wkg^{-1})	1
ϵ_0	GM76 dissipation rate $= 8 \times 10^{-10}$ W kg^{-1}	8
E_0	Dimensionless energy $= 6.3 \times 10^{-5}$	8
f_0	GM76 inertial frequency $= 7.86 \times 10^{-5}$ s^{-1}	8
Γ	Mixing efficiency $= 0.2$	8
$j*$	Mode number scale $= 3$	8
K_ρ	Diapycnal turbulent eddy diffusivity of mass ($\mathrm{m}^2\mathrm{s}^{-1}$)	1
MLD	Mixed Layer Depth m	6
N	Buoyancy frequency $= 10^{-2} - 10^{-4}$ rad s^{-1}	6
N_{ref}	Background reference buoyancy frequency rad s^{-1}	7
N_{100m}	Buoyancy frequency on 100m scale rad s^{-1}	7
N_0	GM76 buoyancy frequency $= 3$ cph	8
P	Pressure (dbar)	6
R_ω	Shear-to-strain variance ratio	7
T	Temperature(°C)	3
Θ	Conservative Temperature (°C)	6
∇	Spatial gradient	1

Appendix B: Matlab code for the Mixing (MX) Oceanographic Toolbox

B.1 Analysis

```
function [mixing_data_gridded] = mx_mixing_analysis(float_data,run,fig,
    directory,mx_parameters)

% mx_mixing_analysis            Mixing estimates from EM-APEX float profiles
%=========================================================================
%
% USAGE:
%  mixing_data = mx_mixing_analysis(float_data,run,fig,directory,
    mx_parameters)
%
% DESCRIPTION:
%  This function applies a shear-strain finescale parameterization to
% vertical profiles from EM-APEX floats and outputs the dissipatino rate
% and diffusivity.
%
% INPUT:
%   float_data  = name of .mat file to be analysed e.g. 'float_data_v2'
%   run         = index of run e.g. 'a'
%   fig         = turns figure display on and off: either 'on' or 'off'.
%                 In both case, figures are saved to drive.
%   directory   = pwd by default. This is the directory in which the
%                 figures will be saved.
%
% The float_data .mat file has to be a structure array as follow:
%
% F3951 =                           % where 3951 is the float number
%
% 1x65 struct array with fields:    % where 65 is the number of profiles
%
% and where for each profile, the data format is:
%        float_wmoid: '1901142'     % float WMO ID
%              fltid: '4051'
%     profile_number: 1
%        surface_mlt: 7.3374e+05
%                lon: 68.7386
%                lat: -43.7382
%            ctd_mlt: [1x609 double]
%           Pctd_cal: [1x609 double]
%              S_cal: [1x609 double]
%              T_cal: [1x609 double]
%             ef_mlt: [1x464 double]
%            Pef_cal: [1x464 double]
%             U1_abs: [1x464 double]
%             U2_abs: [1x464 double]
%             V1_abs: [1x464 double]
%             V2_abs: [1x464 double]
%
% PARAMETERS:
% The parameters are saved in mx_parameters.mat. Some of them have to be
% changed to reflect the data type, location and resolution.
%
% U='1';              % U velocity sensor: either '1' or '2'
% V='1';              % V velocity sensor: either '1' or '2'
% moving_window=20;
% dzN2ref=24;
% dzN2=6;             % differential length for N2 calculation: e.g. 6 [dbar]
```

```
55 % drho=0.03;           % density gradient to derive the mixed layer depth:
       0.03
56 % dz=3;                % main data set pressure grid interval [m]
57 % dzs=6;
58 % fftpt=128;           % Number of points for the fast fourier transform eg
59 %                         128 but could be 32,64,128... or any power of 2
60 % lzmin_fixed=50;      % mini wavelength integration for shear/strain spectra
         [m]
61 % lzmax_fixed=300;     % maxi wavelength integration for shear/strain spectra
       [m]
62 %
63 % Constants
64 % R=5;                    % Average shear to strain ratio
65 % gamma=0.2;               % Mixing efficiency gamma
66 % epsilon0=8*10^(-10);  % from the GM76 model
67 % N0=0.00524;             % from the GM76 model
68 % f0=sw_f(32.5);          % from the GM76 model
69 % E=6.3e-5;               % dimensionless energy level from the GM76 model
70 % b=1300;                 % scale depth of thermocline [dbar]
71 % jstar=3;                % mode number
72 %
73 % Spectral method:
74 % Choice involves trade-offs between confidence and variance perservation
75 % spectral_method='Tycho2'; % Tycho2 = 10 sin^2 window
76 % Decomposition spectral method for both CW and CCW:
77 % cospectral_method='Tycho2_cospectra';
78 %
79 % Spectral corrections:
80 % Switches for various spectral corrections. Models to correct the high
81 % frequency portion of the spectra due to instrument limits.
82 % switch_fd = 1;  % first-differencing correction when 1 ->correction is on
83 %
84 % Wavelength of integral:
85 % Wavelength integration for shear and strain spectra (finestructure
86 % epsilon comes from an integral of shear and strain power over a certain
87 % wavenumber/wavelength range - this sets the limits of wavelength
88 % integration) ~ to internal waveband.
89 % lzmin_fixed;      % This value is very sensitive
90 % lzmax_fixed;      % 300m is quite a typical number
91 % Threshold to determine lzmin where lzmin is the minimum wavelength for
92 % which the noise spectra is less critical ratio * spectrum
93 % crit_rat=NaN;
94 % Minimum wavelength threshold - lzmin is set to the maximum of
95 % lzmin_threshold and lzmin determinded from the noise threshold
96 % lzmin_threshold=NaN;
97 %
98 %
99 % OUTPUT:
100 %  mixing_data_gridded.mat
101 %
102 %                  fltid: [1x36 double]    Float number
103 %          profile_number: [1x36 double]    Profile number for this data
104 %                     lon: [1x36 double]    Longitude in o
105 %                     lat: [1x36 double]    Latitude in o
106 %            surface_mlt: [1x36 double]    Profile surface time
107 %                     MLD: [1x36 double]    Mixed layer depth m
108 %                       P: [550x36 double] Pressure
109 %                      SA: [550x36 double] Absolute salinity in g/kg
110 %                      CT: [550x36 double] Conservative temperature oC
111 %                 sigma_0: [550x36 double] Potential density ref. to sea surface
        kg/m^3
112 %                      N2: [550x36 double] Buyoancy frequency rad/s
113 %                  N2_ref: [550x36 double] Reference buoyancy frequency
114 %                N2_100m: [550x36 double] Buoyancy frequency over 100m vertical
        window
115 %                  strain: [550x36 double] Strain
116 %                fallrate: [550x36 double] Fall rate of the EM-APEX float
117 %                       U: [550x36 double] East horizontal velocity
118 %                       V: [550x36 double] North horizontal velocity
119 %                   speed: [550x36 double] Current speed
```

```
120 %                 shear: [550x36 double] Shear
121 %       epsilon_shear: [52x36 double]  Dissipation rate from shear
        parameterization
122 %              Kz_shear: [52x36 double]  Diffusivity from shear
        parameterization
123 %            P_m_shear: [52x36 double]  Pressure for shear data \si{dbar}
124 %       shear_variance: [52x36 double]  Shear variance
125 %        CW_S_variance: [52x36 double]  Clockwise shear variance
126 %       CCW_S_variance: [52x36 double]  Counter-clockwise shear variance
127 %             Mc_shear: [52x36 double]  Cutoff vertical wavenumber from shear
128 %       epsilon_strain: [65x36 double]  Dissipation rate from strain
        parameterization
129 %              Kz_strain: [65x36 double]  Diffusivity from strain
        parameterization
130 %            P_m_strain: [65x36 double]  Pressure for strain data \si{dbar}
131 %      strain_variance: [65x36 double]  Strain variance
132 %     critiWave_strain: [65x36 double]  Cutoff vertical wavenumber from
        strain
133 %               epsilon: [550x36 double] Dissipation rate from shear-strain
        parameterization
134 %                    Kz: [550x36 double] Diffusivity from shear-strain
        parameterization
135 %                  P_m: [550x36 double] Pressure grid for mixing data
136 %                   Rw: [550x36 double] Shear-to-strain variance ratio
137 %
138 % AUTHOR:
139 %   Amelie MEYER
140 %
141 % VERSION NUMBER: 1.0 (16th June, 2014)
142 %
143 % RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
144 %           Bindoff. Mixing variability in the Southern Ocean. Journal of
145 %           Physical Oceanography, 45,966-987, 2015.
146 %=========================================================================
147
148 %% Check input parameters are defined
149 if ~(nargin == 5)
150     fprintf(2,'Input variables for mx_mixing_analysis are missing... default
            options will be used!!\n');
151 end
152
153 % If input variables are not defined, use the default options
154 if ~exist('directory','var')
155     directory=pwd;
156 elseif ~ischar(directory)
157     directory=pwd;
158 end
159 if ~exist('fig','var')
160     fig='off';
161 elseif ~ischar(fig)
162     fig='off';
163 end
164 if ~exist('float_data','var')
165     float_data='float_data_vmx';
166 elseif ~ischar(float_data)
167     float_data='float_data_vmx';
168 end
169 if ~exist('run','var')
170     run='test_run';
171 elseif ~ischar(run)
172     run='test_run';
173 end
174 if ~exist('mx_parameters','var')
175     mx_parameters='mx_parameters';
176 elseif ~ischar(mx_parameters)
177     mx_parameters='mx_parameters';
178 end
179
180 %% Load and build datasets
181 load(mx_parameters);
```

```
182 mx_build_initial_data(float_data)
183
184 %% Derive fallrate of the EM-APEX floats
185 mx_derive_fallrate
186
187 %% Grid CTD data onto 2.2dbar and EM on 3dbar
188 mx_grid_initialdata(parameters.U,parameters.V)
189
190 %% Derive initial variables
191 mx_derive_abs_T_S
192 mx_derive_potential_density_anomaly
193 mx_derive_N2(parameters.dzN2)
194 mx_derive_mixed_layer_depth(parameters.drho)
195 mx_derive_current_speed
196
197 %% Plot main variables
198 mx_grid_all_initial_data
199
200 mx_plot_temperature(fig,directory)
201 mx_plot_salinity(fig,directory)
202 mx_plot_N2(fig,directory)
203 mx_plot_mixed_layer_depth(fig,directory)
204 mx_plot_current_speed(fig,directory)
205
206 %% Derive mixing variables
207 mx_derive_N2_ref(parameters.moving_window,parameters.dzN2ref)
208 mx_derive_N2_100m
209 mx_derive_strain
210 mx_derive_shear(parameters.dz,parameters.dzs)
211
212 %% Derive mixing
213 mx_grid_mixingdata(parameters.dz)
214 mixing_data_gridded=mx_derive_mixing(parameters.dz,parameters.fftpt,
        parameters.lzmin_fixed,parameters.lzmax_fixed,mx_parameters,run);
215
216 %% Plot main mixing variables
217 mx_plot_dissipation_rate(fig,directory,run)
218 mx_plot_diffusivity(fig,directory,run)
219 mx_plot_CCW_CW(fig,directory,run)
220 mx_plot_Rw(fig,directory,run)
```

B.2 Load and build the dataset

B.2.1 mx_build_initial_data.m

```
1  function [] = mx_build_initial_data(float_data)
2
3  % mx_build_initial_data                          Build the initial data file
4  %=========================================================================
5  %
6  % USAGE:
7  %   [] = mx_build_initial_data(float_data)
8  %
9  % DESCRIPTION:
10 %   Builds a data set for the initial analysis using the provided float data
11 %   and saved this data set as a structure 'initial_data.mat'
12 %
13 % INPUT:
14 %    float_data  = name of .mat file to be analysed e.g. 'float_data_v2'
15 %    run         = index of run e.g. 'a'
16 %
17 % OUTPUT:
18 %   initial_data.mat
19 %
20 % AUTHOR:
21 %   Amelie MEYER
22 %
```

```
% VERSION NUMBER: 1.0 (16th June, 2014)
%
%==========================================================================
display('Building initial data file...');

% Load the data
load(float_data);

%% Identify the float ID
% List all variables:
variables=who;
% Looks for variables with a float name e.g. 'F3760'
for i=1:length(variables)
    nb_charac=cell2mat(cellfun(@size,variables(i),'uni',false));
    if nb_charac(2) == 5 % identifies variables with 5 characters
        f_charac=strfind(cellstr(variables(i)),'F');
        % looks for variables starting with the letter 'F'
        if cell2mat(f_charac)==1
            % Identifies the float ID
            float=strtok(cellstr(variables(i)),'F');
            % float ID
            flt=str2num(cell2mat(float));
        end
    end
end

%% FIELDS TO RENAME
eval(['[F' int2str(flt) '.Pctd]=F' int2str(flt) '.Pctd_cal;']);
eval(['[F' int2str(flt) '.S]=F' int2str(flt) '.S_cal;']);
eval(['[F' int2str(flt) '.T]=F' int2str(flt) '.T_cal;']);
eval(['[F' int2str(flt) '.Pef]=F' int2str(flt) '.Pef_cal;']);
eval(['[F' int2str(flt) '.U1]=F' int2str(flt) '.U1_abs;']);
eval(['[F' int2str(flt) '.U2]=F' int2str(flt) '.U2_abs;']);
eval(['[F' int2str(flt) '.V1]=F' int2str(flt) '.V1_abs;']);
eval(['[F' int2str(flt) '.V2]=F' int2str(flt) '.V2_abs;']);

%% FIELDS TO REMOVE
% Remove the below list of fields:
fields={'S_cal' 'T_cal' 'Pctd_cal' 'Pef_cal' 'U1_abs' 'U2_abs' 'V1_abs' '
    V2_abs'};
eval(['Flt' int2str(flt) '=rmfield(F' int2str(flt) ',fields);']);

%% SAVE STRUCTURE
save('initial_data.mat','-regexp','^Flt','^flt');
```

B.3 Derive the fall rate of the EM-APEX float

B.3.1 mx_derive_fallrate.m

```
function [] = mx_derive_fallrate

% mx_derive_fallrate            Derives the fallrate of the EM-APEX floats
%==========================================================================
%
% DESCRIPTION:
%  Calculates the fall rate of the EmApex and adds it as a variable to
% initial_data_I. Uses the original pressure grid and the fact that
% measurements are made every 25s to derive fall rate in m/s.
%
% INPUT:
%   initial_data.mat
%
% OUTPUT:
%       initial_data.mat
%
% AUTHOR:
%  Amelie Meyer
```

```
%
% VERSION NUMBER: 1.1 (17th June, 2014)
%
% REFERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
%             Bindoff. Mixing variability in the Southern Ocean. Journal of
%             Physical Oceanography, 45,966-987, 2015.
%=========================================================================
display('Derive float fallrate...');

set(0,'RecursionLimit',600)

load initial_data.mat
t=25;   % time in second between each measurements...

eval(['Flt' int2str(int32(flt)) '(1,1).fallrate=NaN;']);        % Creates a
    new empty variable fallrate in structure f3761...

eval(['[c profiles]=size(Flt' int2str(int32(flt)) ');']);       % Looks how
    many profiles there are
 for i=1:profiles;                                              % For each
     profile
    l=eval(['length(Flt' int2str(int32(flt)) '(i).Pef);']);     % l is the
        number of bin depth in that profile
    distance=NaN(l,1);
    % FOR MOST BIN DEPTH
    for ii=2:l;
        distance(ii,1)=eval(['Flt' int2str(int32(flt)) '(i).Pef(ii)-Flt'
            int2str(int32(flt)) '(i).Pef(ii-1);']);
    end; clear ii
    % Calculate fallrate:
    fallrate=distance./t;
    % Copy fallrate into structure
    eval(['Flt',int2str(int32(flt)),'(i).fallrate(1,1:l)=NaN;'])        %
        Fills the empty variable with NaNs
    eval(['Flt',int2str(int32(flt)),'(i).fallrate(1,2:l)=fallrate(2:l);'])
    % FOR 1ST BIN DEPTH: copies the next value
    i_min=min(find(isnan(fallrate)==0));
    eval(['Flt',int2str(int32(flt)),'(i).fallrate(1,i_min-1)=fallrate(i_min);
        '])
    clear fallrate
 end

save('initial_data.mat','-regexp','^Flt','^flt');
```

B.4 Grid the data

B.4.1 mx_grid_initialdata.m

```
function []=mx_grid_initialdata(U,V)

% mx_grid_initialdata                                Grid the initial data file
%=========================================================================
%
% USAGE:
%   [] = mx_grid_initialdata(U,V)
%
% DESCRIPTION:
%  Both the CTD data (temperature and salinity) and the EM data (velocity
% and fall rate) are gridded on a regular pressure grid. This grid is
% preset to vertical intervals of 2.2 dbar for the CTD data and
% 3 dbar for the EM data. These values can be changed manually in the code.
% The gridded data is saved in the same structure ?initial data.mat?.
%
% INPUT:
%   initial_data.mat
%   U                   = U vel sensor eg '1' or '2'
%       V                       = V vel sensor eg '1' or '2'
```

```
%
% OUTPUT:
%    initial_data.mat
%
% AUTHOR:
%    Amelie MEYER
%
% VERSION NUMBER: 1.0 (18th June, 2014)
%
% RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
%           Bindoff. Mixing variability in the Southern Ocean. Journal of
%           Physical Oceanography, 45,966-987, 2015.
%=========================================================================
display('Grid the initial data file...');

load initial_data.mat
warning('off','MATLAB:interp1:NaNinY');

eval(['[c profiles]=size(Flt' int2str(flt) ');']);% Looks how many profiles
    there are
% Copy data from initial_data_II.mat:
for i=1:profiles;
    % Grab single variables
    eval(['float' int2str(flt) '(i).float_wmoid=Flt' int2str(flt) '(i).
        float_wmoid;']);
    eval(['float' int2str(flt) '(i).fltid=Flt' int2str(flt) '(i).fltid;']);
    eval(['float' int2str(flt) '(i).profile_number=Flt' int2str(flt) '(i).
        profile_number;']);
    eval(['float' int2str(flt) '(i).surface_mlt=Flt' int2str(flt) '(i).
        surface_mlt;']);
    eval(['float' int2str(flt) '(i).lon=Flt' int2str(flt) '(i).lon;']);
    eval(['float' int2str(flt) '(i).lat=Flt' int2str(flt) '(i).lat;']);
    % Grab CTD variables and interp on a p gridd of 2.2 db
    ctdgrid=(1:2.2:1650)';                                  % Create new Pctd
        grid
    eval(['float' int2str(flt) '(i).Pctd=ctdgrid;']);       % add it to
        structure
    Pctd1=eval(['Flt' int2str(flt) '(i).Pctd']);                % Old Pct
        grid
    % Get rid of NaNs at bottom of variables:
    ind=min(find(isnan(Pctd1)))-1;      % finds the last good index of Pctd
    if isempty(ind)
        Pctd=Pctd1;
        ctd_mlt=eval(['Flt' int2str(flt) '(i).ctd_mlt;']);
        S=eval(['Flt' int2str(flt) '(i).S;']);
        T=eval(['Flt' int2str(flt) '(i).T;']);
    else
        Pctd=Pctd1(1:ind);
        ctd_mlt=eval(['Flt' int2str(flt) '(i).ctd_mlt(1:ind);']);
        S=eval(['Flt' int2str(flt) '(i).S(1:ind);']);
        T=eval(['Flt' int2str(flt) '(i).T(1:ind);']);
    end
    eval(['float' int2str(flt) '(i).S=interp1(Pctd,S,ctdgrid);']);
    eval(['float' int2str(flt) '(i).T=interp1(Pctd,T,ctdgrid);']);
    eval(['float' int2str(flt) '(i).ctd_mlt=interp1(Pctd,ctd_mlt,ctdgrid);'])
        ;
    % Grab EM variables
    emgrid=(1:3:1650)';                                     % Create new Pctd
        grid
    eval(['float' int2str(flt) '(i).Pef=emgrid;']);         % add it to
        structure
    Pef=eval(['Flt' int2str(flt) '(i).Pef']);                   % Old Pct
        grid
    ef_mlt=eval(['Flt' int2str(flt) '(i).ef_mlt']);
    fallrate=eval(['Flt' int2str(flt) '(i).fallrate']);
    u=eval(['Flt' int2str(flt) '(i).U' U '']);
    v=eval(['Flt' int2str(flt) '(i).V' V '']);
    eval(['float' int2str(flt) '(i).ef_mlt=interp1(Pef,ef_mlt,emgrid);']);
    eval(['float' int2str(flt) '(i).fallrate=interp1(Pef,fallrate,emgrid);'])
        ;
```

```
78      eval(['float' int2str(flt) '(i).U=interp1(Pef,u,emgrid);']);
79      eval(['float' int2str(flt) '(i).V=interp1(Pef,v,emgrid);']);
80      clear Pef ef_mlt fallrate u v emgrid
81  end
82
83  save('initial_data.mat','-regexp','^float','^flt');
```

B.5 Derive initial variables

B.5.1 mx_derive_abs_T_S.m

```
1   function [] = mx_derive_abs_T_S
2
3   % mx_derive_abs_T_S   Derive absolute salinity and conservative temperature
4   %==========================================================================
5   %
6   % USAGE:
7   %  [] = mx_derive_abs_T_S
8   %
9   % DESCRIPTION:
10  %  Calculates Absolute Salinity from Practical Salinity.  Since SP is
11  %   non-negative by definition, this function changes any negative input
12  %   values of SP to be zero. Calculates Conservative Temperature of
13  %   seawater from in-situ temperature.
14  %
15  % INPUT:
16  %   initial_data.mat
17  %
18  % OUTPUT:
19  %  initial_data.mat     with:
20  %  SA               =  Absolute Salinity                        [ g/kg ]
21  %  CT               =  Conservative Temperature (ITS-90)        [ deg C ]
22  %
23  % AUTHOR:
24  %  Amelie MEYER
25  %
26  % VERSION NUMBER: 1.0 (18th June, 2014)
27  %
28  % REFERENCES:
29  % McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and
30  % the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127,
31  % ISBN 978-0-646-55621-5.
32  %
33  %==========================================================================
34  display('Derive Absolute Salinity and Conservative Temperature...');
35
36  load initial_data.mat
37
38  eval(['float' int2str(flt) '(1,1).SA=NaN;']);              % Creates a new
        empty variable
39  eval(['float' int2str(flt) '(1,1).CT=NaN;']);              % Creates a new
        empty variable
40  eval(['[c profiles]=size(float' int2str(flt) ');']);       % Looks how many
        profiles there are for each float
41  for i=1:profiles;                                          % For each
        profile
42      P=eval(['float' int2str(flt) '(i).Pctd']);             % Define all
            variables
43      S=eval(['float' int2str(flt) '(i).S']);
44      T=eval(['float' int2str(flt) '(i).T']);
45      lon=eval(['float' int2str(flt) '(i).lon']);
46      lat=eval(['float' int2str(flt) '(i).lat']);
47      % Derive absolute salinity SA
48      SA=gsw_SA_from_SP(S,P,lon,lat);
49      % Derive conseravtive temperature
50      CT=gsw_CT_from_t(SA,T,P);
51      % Add new variables to dataset
```

```
52      eval(['float',int2str(flt),'(i).SA=[SA''];'])
53      eval(['float',int2str(flt),'(i).CT=[CT''];'])
54      clear P S T lon lat SA CT
55  end
56
57  save('initial_data.mat','-regexp','^float','^flt');
```

B.5.2 mx_derive_potential_density_anomaly.m

```
1   function [] = mx_derive_potential_density_anomaly
2
3   % mx_derive_potential_density_anomaly       Derive potential density anomaly
4   %==========================================================================
5   %
6   % USAGE:
7   %  [] = mx_derive_potential_density_anomaly
8   %
9   % DESCRIPTION:
10  %  Calculates potential density from Absolute Salinity and Conservative
11  %   Temperature, using the computationally-efficient 48-term expression for
12  %   density in terms of SA, CT and p (McDougall et al., 2011).
13  %
14  % INPUT:
15  %  SA  =  Absolute Salinity                                        [ g/kg ]
16  %  CT  =  Conservative Temperature (ITS-90)                       [ deg C ]
17  %  p   =  sea pressure                                             [ dbar ]
18  %
19  % OUTPUT:
20  %  sigma_0  =  potential density anomaly referenced to sea level [ kg/m^3 ]
21  %
22  % AUTHOR:
23  %  Amelie MEYER
24  %
25  % VERSION NUMBER: 1.0 (19th June, 2014)
26  %
27  % REFERENCES:
28  % McDougall, T.J. and P.M. Barker, 2011: Getting started with TEOS-10 and
29  % the Gibbs Seawater (GSW) Oceanographic Toolbox, 28pp., SCOR/IAPSO WG127,
30  % ISBN 978-0-646-55621-5.
31  %
32  %==========================================================================
33  display('Derive potential density anomaly referred to the surface...');
34
35  load initial_data.mat
36
37  eval(['float' int2str(flt) '(1,1).sigma_0=NaN;']);          % Creates a new
        empty variable
38  eval(['[c profiles]=size(float' int2str(flt) ');']);        % Looks how many
        profiles there are for each float
39  for i=1:profiles;                                           % For each
        profile
40      SA=eval(['float' int2str(flt) '(i).SA']);               % Define all
            variables
41      CT=eval(['float' int2str(flt) '(i).CT']);
42      % Derive potential density anomaly
43      sigma_0=gsw_rho(SA,CT,0)-1000; % where 0 is the reference pressure point
            (sea level)
44      % Add new variables to dataset
45      eval(['float',int2str(flt),'(i).sigma_0=[sigma_0''];'])
46      clear SA CT
47  end
48
49      save('initial_data.mat','-regexp','^float','^flt');
```

B.5.3 mx_derive_N2.m

```
1   function [] = mx_derive_N2(dzN2)
2
```

```
% mx_derive_N2                                    Derive the buoyancy frequency N2
%=========================================================================
%
% USAGE:
%   [] = mx_derive_N2(dzN2)
%
% DESCRIPTION:
%    Calculates the buoyancy frequency squared (N^2)(i.e. the Brunt-Vaisala
%    frequency squared) at the mid pressure from the equation,
%
%                2       2     d(rho_local)
%           N   =  g    x  ----------------
%                                  dP
%
% INPUT:
%  dz        = differential length (1 value)                          [dbar]
%
% OUTPUT:
%  N2        =  Brunt-Vaisala Frequency squared from Polzin code    [ 1/s^2 ]
%
% AUTHOR:
% Amelie MEYER Based on Kurt Polzin original nsq_mod.m code
%
% VERSION NUMBER: 1.0 (19th June, 2014)
%
%=========================================================================
display('Derive the buoyancy frequency (N2)...');
warning('off','MATLAB:interp1:NaNinY')

load initial_data.mat

eval(['float' int2str(flt) '(1,1).N2=NaN;']);            % Creates a new empty
    variable
eval(['[c profiles]=size(float' int2str(flt) ');']);% Looks how many profiles
    there are for each float
for i=1:profiles;                                       % For each profile
    s=eval(['float' int2str(flt) '(i).SA']);             % defines all
        variables for the nsq_modbs function
    t=eval(['float' int2str(flt) '(i).CT']);
    p=eval(['float' int2str(flt) '(i).Pctd']);
    sea_pressure=p-10.1325;
    lat=eval(['float' int2str(flt) '(i).lat']);
    % Derive N2 with nsq_modbs.m
    [nn,temp]=nsq_modbs(s,t,p,dzN2,lat);
    eval(['float',int2str(flt),'(i).N2=[nn];'])
end

save('initial_data.mat','-regexp','^float','^flt');
```

B.5.4 mx_derive_mixed_layer_depth.m

```
function [] = mx_derive_mixed_layer_depth(drho)

% mx_derive_mixed_layer_depth                    Derive the mixed layer depth
%=========================================================================
%
% USAGE:
%   [] = mx_derive_mixed_layer_depth(drho)
%
% DESCRIPTION:
%    Derives mixed layer depth and adds variable MLD [db]. MLD is the depth
% at which the potential density changes by a given threshold value drho =
% 0.03km/m^3 relative to the one at a reference depth refdep=10m and
% adding an extra 10m to be conservative.
%
% INPUT:
%  drho        = dendity change threshold                                [-]
%
% OUTPUT:
```

```
% MLD        =  Mixed layer depth                                    [ m ]
%
% AUTHOR:
%  Amelie MEYER
%
% VERSION NUMBER: 1.0 (19th June, 2014)
%
% REFERENCES:
% deBoyer Montegut et al., 2004 JGR-Oceans vol 109,doi:10.1029/2004JC002378
%
%=========================================================================
display('Derive the mixed layer depth (MLD)...');
warning('off','MATLAB:interp1:NaNinY')
load initial_data.mat

% Constants
dT=0.2;
refdep=10;                                                  % Minimum depth for
    MLD

eval(['float' int2str(flt) '(1,1).MLD=NaN;']);              % Creates a new empty
     variable
eval(['[c profiles]=size(float' int2str(flt) ');']);% Looks how many profiles
     there are for each float
for i=1:profiles;                                           % For each profile
    P=eval(['float' int2str(flt) '(i).Pctd']);              % defines all
        variables for the mixedlayerdepth function
    T=eval(['float' int2str(flt) '(i).CT']);
    sigma_0=eval(['float' int2str(flt) '(i).sigma_0+1000']);
    % Derive MLD
    [I_10]=max(find(P<=refdep));                        % Index value of closest
        data point to refdep (10m)
    if P(1,1)>=refdep;                                      % Test to check that
        first value of press is within refdep
    else
        if ~isempty(I_10)                                   % If (I_10) is a
            value...
            ref_ptemp=T(I_10);                              % Temperatue at
                refdep (10dbar)
            ref_pdens=sigma_0(I_10);                        % Potential density
                at refdep
            dT_profile=abs(T-ref_ptemp)-dT;             % Profile of anomaly
                potential temp minus dT
            ttt=find(dT_profile>0,1);                       % Finds index of
                first big spike in potential T
            drho_profile=abs(sigma_0-ref_pdens)-drho;% Profile of anomaly
                potential density minus dT
            if isempty(ttt)
                MLD=NaN;
            else
                MLD=P(max(find(drho_profile<0)));
            end
        end
    end
    eval(['float',int2str(flt),'(i).MLD=MLD+10;'])
end

save('initial_data.mat','-regexp','^float','^flt');
```

B.5.5 mx_derive_current_speed.m

```
function [] = mx_derive_current_speed

% mx_derive_current_speed                              Derive the current_speed
%=========================================================================
%
% USAGE:
%  [] = mx_derive_current_speed
%
```

```
% DESCRIPTION:
%   Derives the current speed as sqrt(U.^2+V.^2)
%
% OUTPUT:
%  speed      =  Current Speed                                    [ m/s ]
%
% AUTHOR:
%  Amelie MEYER
%
% VERSION NUMBER: 1.0 (19th June, 2014)
%
%=========================================================================
display('Derive the current speed [m/s]...');

load initial_data.mat

eval(['float' int2str(flt) '(1,1).speed=NaN;']);% Creates a new empty
    variable
eval(['[c profiles]=size(float' int2str(flt) ');']);% Looks how many profiles
     there are for each float
for i=1:profiles;                                % For each profile
    U=eval(['float' int2str(flt) '(i).U']);      % defines all variables
    V=eval(['float' int2str(flt) '(i).V']);
    % Derive current speed
    speed=sqrt(U.^2+V.^2);
    eval(['float',int2str(flt),'(i).speed=speed;'])
end

save('initial_data.mat','-regexp','^float','^flt');
```

B.6 Plot initial variables

B.6.1 mx_grid_all_initial_data.m

```
function [] = mx_grid_all_initial_data

% mx_grid_all_initial_data                          Grid the initial data file
%=========================================================================
%
% USAGE:
%   [] = mx_grid_all_initial_data
%
% DESCRIPTION:
%  The initial variables are changed from a structure file into a matrix
% format ?initial_data_gridded.mat?.
%
% INPUT:
%   initial_data.mat
%
% OUTPUT:
%   initial_data_gridded.mat
%
% AUTHOR:
%   Amelie MEYER
%
% VERSION NUMBER: 1.0 (23rd June, 2014)
%
% RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
%           Bindoff. Mixing variability in the Southern Ocean. Journal of
%           Physical Oceanography, 45,966-987, 2015.
%=========================================================================
display('Grid the initial data file into matrix format...');

load initial_data.mat

eval(['profilet=length(float' int2str(flt) ');']); % number of profiles for
    this float
```

```
33 eval(['profile_ctd=length(float' int2str(flt) '(1).Pctd);']); % length of
       each profile
34 eval(['profile_ef=length(float' int2str(flt) '(1).Pef);']); % length of each
       profile
35
36 %% Create empty matrices for all relevant variables
37 fltid=NaN(1,profilet);
38 profile_number=zeros(1,profilet);
39 lon=zeros(1,profilet);
40 lat=zeros(1,profilet);
41 surface_mlt=zeros(1,profilet);
42 MLD=NaN(1,profilet);
43 fallrate=zeros(profile_ef,profilet);
44 Pctd=NaN(profile_ctd,profilet);
45 SA=NaN(profile_ctd,profilet);
46 CT=NaN(profile_ctd,profilet);
47 sigma_0=NaN(profile_ctd,profilet);
48 N2=NaN(profile_ctd,profilet);
49 Pef=NaN(profile_ef,profilet);
50 U=NaN(profile_ef,profilet);
51 V=NaN(profile_ef,profilet);
52 speed=NaN(profile_ef,profilet);
53
54
55 %% Fill up the empty matrix with data using a loop through all flt all
       profiles;
56 for ii=1:profilet                                    % for each profile
57     fltid(1,ii)=eval(['str2num(float' int2str(flt) '(ii).fltid);']);
58     profile_number(1,ii)=eval(['float' int2str(flt) '(ii).profile_number;']);
59     lon(1,ii)=eval(['float' int2str(flt) '(ii).lon;']);
60     lat(1,ii)=eval(['float' int2str(flt) '(ii).lat;']);
61     surface_mlt(1,ii)=eval(['float' int2str(flt) '(ii).surface_mlt;']);
62     MLD(1,ii)=eval(['float' int2str(flt) '(ii).MLD;']);
63     eval(['fallrate(:,ii)=float' int2str(flt) '(ii).fallrate;']);
64     eval(['Pctd(:,ii)=float' int2str(flt) '(ii).Pctd;']);
65     eval(['SA(:,ii)=float' int2str(flt) '(ii).SA;']);
66     eval(['CT(:,ii)=float' int2str(flt) '(ii).CT;']);
67     eval(['sigma_0(:,ii)=float' int2str(flt) '(ii).sigma_0;']);
68     eval(['N2(:,ii)=float' int2str(flt) '(ii).N2;']);
69     eval(['Pef(:,ii)=float' int2str(flt) '(ii).Pef;']);
70     eval(['U(:,ii)=float' int2str(flt) '(ii).U;']);
71     eval(['V(:,ii)=float' int2str(flt) '(ii).V;']);
72     eval(['speed(:,ii)=float' int2str(flt) '(ii).speed;']);
73 end
74
75
76 %% NEW STRUCTURE
77 % Single values
78 initial_data_gridded.fltid=fltid;
79 initial_data_gridded.profile_number=profile_number;
80 initial_data_gridded.lon=lon;
81 initial_data_gridded.lat=lat;
82 initial_data_gridded.surface_mlt=surface_mlt;
83 initial_data_gridded.MLD=MLD;
84 % Many profiles
85 initial_data_gridded.fallrate=fallrate;
86 initial_data_gridded.Pctd=Pctd;
87 initial_data_gridded.SA=SA;
88 initial_data_gridded.CT=CT;
89 initial_data_gridded.sigma_0=sigma_0;
90 initial_data_gridded.N2=N2;
91 initial_data_gridded.Pef=Pef;
92 initial_data_gridded.U=U;
93 initial_data_gridded.V=V;
94 initial_data_gridded.speed=speed;
95
96 %% Save structure
97 clear fltid
98 save('initial_data_gridded.mat','-regexp','^initial_data_gridded','^flt');
```

B.6.2 mx_plot_temperature.m

```
function [] = mx_plot_temperature(fig,directory)

% mx_plot_temperature                                          Plot temperature
%=========================================================================
%
% USAGE:
%  [] = mx_plot_temperature(float_data)
%
% DESCRIPTION:
%  Plot all the conservative temperature profiles.
%
% INPUT:
%  fig              = either 'on' or 'off' to turn fig display on and off
%  directory        = path of directory where figure is saved
%
% OUTPUT:
%  Figure:  /temperature.eps
%
% AUTHOR:
%  Amelie MEYER
%
% VERSION NUMBER: 1.0 (23rd June, 2014)
%
% RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
%           Bindoff. Mixing variability in the Southern Ocean. Journal of
%           Physical Oceanography, 45,966-987, 2015.
%=========================================================================
disp('Plot the temperature...')

load initial_data_gridded.mat
load colormapbicolor

%% Plot of temperature along profile numbers
nb_profiles=length(initial_data_gridded.fltid);
mini=min(min(initial_data_gridded.CT));
maxi=max(max(initial_data_gridded.CT));

h2=figure(2);
clf
set(2,'Position',[30 50 476 245]); % Where [horiz ver width height] of the
    figure width=1000 for first plot
eval(['set(2,''visible'','''' fig '''');'])
imagesc(1:nb_profiles,initial_data_gridded.Pctd(:,1),initial_data_gridded.CT)
    ;
hold on
axis ij
set(gca,'Position',[0.088 0.13 0.87 0.77])    % position of plot in figure
colormap(colormapbicolor);
h=colorbar;
set(get(h,'ylabel'),'string','Conservative temperature [^oC]','fontsize',10);
caxis([mini maxi])
xlim([0 length(initial_data_gridded.profile_number)])
ylim([0 1600]);
xlabel('Cumulative profile number','FontSize',10);
ylabel('Pressure [dbar]','FontSize',10)

% Density contour
sigma_0=initial_data_gridded.sigma_0;
sigma_02=initial_data_gridded.sigma_0;
sigma_02(:,616:end)=NaN;
[C,h]=contour(1:length(initial_data_gridded.profile_number),
    initial_data_gridded.Pctd(:,1),sigma_0,[27:0.1:29],'color',[0.6 0.6 0.6],'
    linewidth',1);

title(['Float ' int2str(flt) ''])

% Saving figure options
set(gcf,'PaperPositionMode','auto')
set(gcf,'renderer','painters')
```

```
66 signature('Initial analysis','')
67 print(h2,'-depsc2',['' directory '/figures/temperature.eps']);
```

B.6.3 mx_plot_salinity.m

```
1  function [] = mx_plot_salinity(fig,directory)
2
3  % mx_plot_salinity                                          Plot salinity
4  %=========================================================================
5  %
6  % USAGE:
7  %  [] = mx_plot_salinity(fig,directory)
8  %
9  % DESCRIPTION:
10 %  Plot all the absolute salinity profiles
11 %
12 % INPUT:
13 %  fig              = either 'on' or 'off' to utrn fig display on and off
14 %  directory        = path of directory where figure is saved
15 %
16 % OUTPUT:
17 %  Figure:  /salinity.eps
18 %
19 % AUTHOR:
20 %  Amelie MEYER
21 %
22 % VERSION NUMBER: 1.0 (23rd June, 2014)
23 %
24 % RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
25 %           Bindoff. Mixing variability in the Southern Ocean. Journal of
26 %           Physical Oceanography, 45,966-987, 2015.
27 %=========================================================================
28 disp('Plot the salinity...')
29
30 load initial_data_gridded.mat
31 load colormapbluegreen
32
33 %% Plot of temperature along profile numbers
34 nb_profiles=length(initial_data_gridded.fltid);
35 mini=min(min(initial_data_gridded.SA));
36 maxi=max(max(initial_data_gridded.SA));
37
38 h2=figure(2);
39 clf
40 set(2,'Position',[30 50 476 245]); % Where [horiz ver width height] of the
       figure width=1000 for first plot
41 eval(['set(2,''visible'',''' fig ''');'])
42 imagesc(1:nb_profiles,initial_data_gridded.Pctd(:,1),initial_data_gridded.SA)
       ;
43 hold on
44 axis ij
45 set(gca,'Position',[0.088 0.13 0.87 0.77])    % position of plot in figure
46 colormap(colormap3);
47 h=colorbar;
48 set(get(h,'ylabel'),'string','Absolute salinity [g/kg]','fontsize',10);
49 caxis([mini maxi])
50 xlim([0 length(initial_data_gridded.profile_number)])
51 ylim([0 1600]);
52 xlabel('Cumulative profile number','FontSize',10);
53 ylabel('Pressure [dbar]','FontSize',10)
54
55 % Density contour
56 sigma_0=initial_data_gridded.sigma_0;
57 sigma_02=initial_data_gridded.sigma_0;
58 sigma_02(:,616:end)=NaN;
59 [C,h]=contour(1:length(initial_data_gridded.profile_number),
       initial_data_gridded.Pctd(:,1),sigma_0,[27:0.1:29],'color',[0.6 0.6 0.6],'
       linewidth',1);
60
```

```
61 title(['Float ' int2str(flt) ''])
62
63 % Saving figure options
64 set(gcf,'PaperPositionMode','auto')
65 set(gcf,'renderer','painters')
66 signature('')
67 print(h2,'-depsc2',['' directory '/figures/salinity.eps']);
```

B.6.4 mx_plot_mixed_layer_depth.m

```
1  function [] = mx_plot_mixed_layer_depth(fig,directory)
2
3  % mx_plot_mixed_layer_depth                    Plot the mixed layer depth
4  %=========================================================================
5  %
6  % USAGE:
7  %  [] = mx_plot_mixed_layer_depth(fig,directory)
8  %
9  % DESCRIPTION:
10 %  Plot all the mixed layer depth over the potential density profiles.
11 %
12 % INPUT:
13 %  fig              = either 'on' or 'off' to utrn fig display on and off
14 %  directory        = path of directory where figure is saved
15 %
16 % OUTPUT:
17 %  Figure:  /mixed_layer_depth.eps
18 %
19 % AUTHOR:
20 %  Amelie MEYER
21 %
22 % VERSION NUMBER: 1.0 (23rd June, 2014)
23 %
24 % RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
25 %           Bindoff. Mixing variability in the Southern Ocean. Journal of
26 %           Physical Oceanography, 45,966-987, 2015.
27 %=========================================================================
28 disp('Plot the mixed layer depth (MLD)...')
29
30 load initial_data_gridded.mat
31 load colormapbluegreen
32
33 %% Plot
34 % Find index of 500 dbar
35 depth_ind=find(round(initial_data_gridded.Pctd(:,2))==500);
36 sigma_0=initial_data_gridded.sigma_0(1:depth_ind,:);
37 nb_profiles=length(initial_data_gridded.fltid);
38 mini=min(min(sigma_0));
39 maxi=max(max(sigma_0));
40
41 h2=figure(2);
42 clf
43 set(2,'Position',[30 50 476 245]); % Where [horiz ver width height] of the
       figure width=1000 for first plot
44 eval(['set(2,''visible'','''' fig ''');'])
45 imagesc(1:nb_profiles,initial_data_gridded.Pctd(:,1),initial_data_gridded.
       sigma_0);
46 hold on
47 axis ij
48 set(gca,'Position',[0.088 0.13 0.87 0.77])    % position of plot in figure
49 colormap(flipud(fliplr(hot)));
50 h=colorbar;
51 set(get(h,'ylabel'),'string','\sigma_0 [kg m^{-3}]','fontsize',10);
52 caxis([mini maxi])
53 xlim([0 length(initial_data_gridded.profile_number)])
54 ylim([0 500]);
55 xlabel('Cumulative profile number','FontSize',10);
56 ylabel('Pressure [dbar]','FontSize',10)
57
```

```
% contour plot of the mixed layer depth
plot(initial_data_gridded.MLD,'color','k','linewidth',1)

title(['Float ' int2str(flt) ''])

% Saving figure options
set(gcf,'PaperPositionMode','auto')
set(gcf,'renderer','painters')
signature('')
print(h2,'-depsc2',['' directory '/figures/mixed_layer_depth.eps']);
```

B.6.5 mx_plot_current_speed.m

```
function [] = mx_plot_current_speed(fig,directory)

% mx_plot_current_speed                                    Plot the current speed
%=========================================================================
%
% USAGE:
%  [] = mx_plot_current_speed(fig,directory)
%
% DESCRIPTION:
%  Plot all the ccurrent speed profiles.
%
% INPUT:
%  fig              = either 'on' or 'off' to utrn fig display on and off
%  directory        = path of directory where figure is saved
%
% OUTPUT:
%  Figure:  /current_speed.eps
%
% AUTHOR:
%  Amelie MEYER
%
% VERSION NUMBER: 1.0 (23rd June, 2014)
%
% RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
%           Bindoff. Mixing variability in the Southern Ocean. Journal of
%           Physical Oceanography, 45,966-987, 2015.
%=========================================================================
disp('Plot the current speed...')

load initial_data_gridded.mat

%% Plot
nb_profiles=length(initial_data_gridded.fltid);
mini=min(min(initial_data_gridded.speed));
maxi=max(max(initial_data_gridded.speed));

h2=figure(2);
clf
set(2,'Position',[30 50 476 245]); % Where [horiz ver width height] of the
    figure width=1000 for first plot
eval(['set(2,''visible'','''' fig '''');'])
imagesc(1:nb_profiles,initial_data_gridded.Pctd(:,1),initial_data_gridded.
    speed);
hold on
axis ij
set(gca,'Position',[0.088 0.13 0.87 0.77])    % position of plot in figure
colormap(othercolor('Bu_10',200));
h=colorbar;
set(get(h,'ylabel'),'string','Horizontal velocity [m s^{-1}]','fontsize',10);
caxis([mini maxi/4])
xlim([0 length(initial_data_gridded.profile_number)])
ylim([0 1600]);
xlabel('Cumulative profile number','FontSize',10);
ylabel('Pressure [dbar]','FontSize',10)

% Density contour
```

```
55 sigma_0=initial_data_gridded.sigma_0;
56 sigma_02=initial_data_gridded.sigma_0;
57 sigma_02(:,616:end)=NaN;
58 [C,h]=contour(1:length(initial_data_gridded.profile_number),
       initial_data_gridded.Pctd(:,1),sigma_0,[27:0.1:29],'color',[0.6 0.6 0.6],'
       linewidth',1);
59
60 title(['Float ' int2str(flt) ''])
61
62 % Saving figure options
63 set(gcf,'PaperPositionMode','auto')
64 set(gcf,'renderer','painters')
65 signature('')
66 print(h2,'-depsc2',['' directory '/figures/current_speed.eps']);
```

B.7 Derive mixing variables

B.7.1 mx_derive_N2_ref.m

```
1  function [] = mx_derive_N2_ref(moving_window,dzN2ref)
2
3  % mx_derive_N2_ref                Derive the reference buoyancy frequency N2
4  %=========================================================================
5  %
6  % USAGE:
7  %   [] = mx_derive_N2_ref(moving_window,dzN2ref)
8  %
9  % DESCRIPTION:
10 % Derives a reference N2 (N2_ref) profile based on a x (moving_window
11 % eg 20) profiles moving average of each individual N2 profiles. First x/2
12 % N2_ref are based on profiles 1-x and last x+1 profiles are mean of
13 % profile end-(x-1):end. Since profiles have different length, interps the
14 % bottom of the N2_ref profile from a certain depth onwards based on how
15 % many values go into each point.
16 %
17 % INPUT:
18 %    dzN2ref          = bin depth for N2 computation
19 %    moving_window    = for eg 40
20 %
21 % OUTPUT:
22 %    N2_ref           =  Brunt-Vaisala Frequency squared from Polzin code    [
       1/s^2 ]
23 %
24 % AUTHOR:
25 %    Amelie MEYER based on Kurt Polzin original code.
26 %
27 % VERSION NUMBER: 1.0 (24th June, 2014)
28 %=========================================================================
29 display('Derive the reference buoyancy frequency (N2_ref)...');
30
31 warning('off','MATLAB:polyfit:RepeatedPointsOrRescale')
32
33 load initial_data_gridded.mat
34 n=moving_window;
35
36 %% DERIVE N2 PROFILE
37 s=initial_data_gridded.SA;
38 t=initial_data_gridded.CT;
39 p=initial_data_gridded.Pctd;
40 N2=initial_data_gridded.N2;
41 lat=initial_data_gridded.lat;
42 PID=initial_data_gridded.profile_number;
43 % Looks how many profiles there are:
44 profiles=length(initial_data_gridded.fltid);
45 % Creates a new empty variable N2_ref:
46 N2_ref(1:length(s(:,1)),profiles)=NaN;
47 % Derive N2:
```

```
48  n2(1:length(s(:,1)),profiles)=NaN;
49  i_min=firstgood2(s);     % finds 1st non NaN value in S_ref
50  i_max=lastgood2(s);      % finds last non NaN value in S_ref
51  n2(i_min:i_max,:)=nsq_modbs(s(i_min:i_max,:),t(i_min:i_max,:),p(i_min:i_max
        ,:),dzN2ref,lat);
52
53  %%  DERIVE RUNNING MEAN N2 PROFILE (N2_REF)
54  for i=1:profiles
55      % Find indexes to make mean N2 profile
56      a=i-(n/2);       % index of 1st profile going in the mean
57      b=i+(n/2)-1;     % index of last profile going in the mean
58
59      % For 1st n/2+1 profiles, uses the 1st n profiles:
60      if i<(n/2)+1;
61          a=1; b=n;
62      end
63      if i>(profiles-n/2)
64          a=profiles-n+1;
65          b=profiles;
66      end
67      % Calculate n2_ref based on mean of all N2 6db chosen
68      n2_ref1(:,i)=nanmean(n2(:,a:b),2);
69
70      % Check how far down the watercolumn all of the profiles that
71      % have data going into the n2_ref value go:
72      for ii=1:length(s(:,i));
73          t=sum(isfinite(s(end-(ii-1),:)));
74          if t>n
75              % ip_max is index of max depth at which all of profiles still go
76              ip_max=length(s(:,1))-(ii-1);
77              break
78          end
79      end; clear ii t
80
81      % Replace those bottom values with NaN where less than 90% profiles had
            data
82      n2_ref2=n2_ref1;
83      n2_ref2(ip_max+1:end,i)=NaN;
84
85      % Interp the bottom of the mean profile that was cut off:
86      for iii=1:length(s(:,1));
87          t=sum(isfinite(s(end-(iii-1),:)));
88          if t>0;
89              % works out whats the deepest data point out of all profiles
90              % which is going to be the deepest N2ref we calculate
91              i_mmax=length(s(:,1))-(iii-1);
92              break
93          end
94      end; clear iii t
95
96      fit=polyfit(p(600:ip_max,1),n2_ref2(600:ip_max,i),3);
97      fit2(1:length(n2_ref1))=NaN; % create a NaN variable
98      fit2(600:i_mmax)=polyval(fit,p(600:i_mmax,1));
99      n2_ref=n2_ref2;
100     % Add the interpolation bit to the bottom of the profile:
101     n2_ref(ip_max+1:i_mmax,i)=fit2(ip_max+1:i_mmax);
102
103     % Finds last non NaN value in N2 (local profile):
104     ii_max=max(find(isnan(N2(:,i))==0));
105     % Cut off N2_ref profile underneath ii_max:
106     n2_ref(ii_max+1:end,i)=NaN;
107 end;
108
109 %% NEW STRUCTURE
110 mixing_data.N2_ref(1:length(s(:,1)),1:profiles)=NaN;
111 % Single values
112 mixing_data.fltid=initial_data_gridded.fltid;
113 mixing_data.profile_number=initial_data_gridded.profile_number;
114 mixing_data.lon=initial_data_gridded.lon;
115 mixing_data.lat=initial_data_gridded.lat;
```

```
116 mixing_data.surface_mlt=initial_data_gridded.surface_mlt;
117 mixing_data.MLD=initial_data_gridded.MLD;
118 % Many profiles
119 mixing_data.Pctd=initial_data_gridded.Pctd;
120 mixing_data.SA=initial_data_gridded.SA;
121 mixing_data.CT=initial_data_gridded.CT;
122 mixing_data.sigma_0=initial_data_gridded.sigma_0;
123 mixing_data.N2=initial_data_gridded.N2;
124 mixing_data.N2_ref=n2_ref;
125 mixing_data.Pef=initial_data_gridded.Pef;
126 mixing_data.fallrate=initial_data_gridded.fallrate;
127 mixing_data.U=initial_data_gridded.U;
128 mixing_data.V=initial_data_gridded.V;
129 mixing_data.speed=initial_data_gridded.speed;
130
131 save('mixing_data.mat','-regexp','^mixing_data','^flt');
```

B.7.2 mx_derive_N2_100m.m

```
1  function [] = mx_derive_N2_100m
2
3  % mx_derive_N2_100m          Derive the reference buoyancy frequency N2_100m
4  %=========================================================================
5  %
6  % USAGE:
7  %   [] = mx_derive_N2_100m
8  %
9  % DESCRIPTION:
10 % Derives N2_100m and adds variable N2_100m [1/s^2]. N2_100m is the
11 % buoyancy frequency derived over a vertical window of 100m.
12 %
13 % INPUT
14 %
15 % OUTPUT:
16 %    N2_100m      =  Brunt-Vaisala Frequency squared                [ 1/s^2 ]
17 %
18 % AUTHOR:
19 %    Amelie MEYER based on Kurt Polzin original nsq_mod.m code
20 %
21 % VERSION NUMBER: 1.0 (24th June, 2014)
22 %=========================================================================
23 display('f_derive_N2_100m');
24
25 load mixing_data.mat
26 s=mixing_data.SA;
27 t=mixing_data.CT;
28 p=mixing_data.Pctd;
29 lat=mixing_data.lat;
30 % Function nsq_modbs calculates N2_100m:
31 [nn,temp]=nsq_modbs(s,t,p,100,lat);
32 mixing_data.N2_100m=nn;
33
34 save('mixing_data.mat','-regexp','^mixing_data','^flt');
```

B.7.3 mx_derive_strain.m

```
1  function [] = mx_derive_strain
2
3  % mx_derive_strain                                        Derive the strain
4  %=========================================================================
5  %
6  % USAGE:
7  %   [] = mx_derive_strain
8  %
9  % DESCRIPTION:
10 % Derives strain=N^2-N^2_ref/N^2_ref using the buoyancy frequency N2_ref.
11 %
12 % INPUT
```

```
%
% OUTPUT:
%   strain
%
% AUTHOR:
%   Amelie MEYER
%
% VERSION NUMBER: 1.0 (24th June, 2014)
%=========================================================================
display('Derive the strain...');

load mixing_data.mat
mixing_data.strain=(mixing_data.N2-mixing_data.N2_ref)./mixing_data.N2_ref;

save('mixing_data.mat','-regexp','^mixing_data','^flt');
```

B.7.4 mx_derive_shear.m

```
function [] = mx_derive_shear(dz,dzs)

% mx_derive_shear                                              Derive the shear
%=========================================================================
%
% USAGE:
%  [] = mx_derive_shear(dz,dzs)
%
% DESCRIPTION:
%   Derives shear where shear= complex(du,dv)/dp that is shear = dU/dP.
%
% INPUT:
%   dz          = vertical resolution of dataset [m]
%   dzs         = vertical scale over which shear is derived;
%                 dzs must be a multiple of dz (3).
%
% OUTPUT:
%   shear
%
% AUTHOR:
%   Amelie MEYER
%
% VERSION NUMBER: 1.0 (24th June, 2014)
%
% RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
%           Bindoff. Mixing variability in the Southern Ocean. Journal of
%           Physical Oceanography, 45,966-987, 2015.
%=========================================================================
display('Derive the shear...');

load mixing_data.mat

d=dzs/dz; % equivalent of dzs in indexes
profiles=length(mixing_data.fltid);

for i=1:profiles;
    % l is the number of bin depth in the profile:
    l=length(mixing_data.Pef);
    du=NaN(l,1);
    dv=NaN(l,1);
    dp=NaN(l,1);
    shear=NaN(l,1);

    % For most bin depth
    for ii=round(d/2)+1:l-round(d/2);
        du(ii,1)=mixing_data.U(ii+round(d/2),i)-mixing_data.U(ii-round(d/2),i
            );
        dv(ii,1)=mixing_data.V(ii+round(d/2),i)-mixing_data.V(ii-round(d/2),i
            );
        dp(ii,1)=mixing_data.Pef(ii+round(d/2),i)-mixing_data.Pef(ii-round(d
            /2),i);
```

```
49      end;
50      if isnan(du)==1; % if there is no vel data
51          % Feels the empty variable with NaNs:
52          mixing_data.shear(1:l,i)=NaN;
53      else
54      % Calculate shear:
55      shear=complex(du,dv)./dp;
56      % Copy shear into structure
57      mixing_data.shear(1:l,i)=NaN;
58      mixing_data.shear(:,i)=shear;
59      end
60  end
61
62  save('mixing_data.mat','-regexp','^mixing_data','^flt');
```

B.8 Derive mixing

B.8.1 mx_grid_mixingdata.m

```
1   function [] = mx_grid_mixingdata(dz)
2
3   % mx_grid_mixingdata                                    Grid the mixing data file
4   %=========================================================================
5   %
6   % USAGE:
7   %   [] = mx_grid_mixingdata(dz)
8   %
9   % DESCRIPTION:
10  %   Gridds both EM and CTD data from mixing_data.mat on same pressure grid.
11  %
12  % INPUT:
13  %   initial_data.mat
14  %   dz              = vertical resolution of dataset [m]
15  %
16  % OUTPUT:
17  %   mixing_data_gridded.mat
18  %
19  % AUTHOR:
20  %   Amelie MEYER
21  %
22  % VERSION NUMBER: 1.0 (25th June, 2014)
23  %=========================================================================
24  display('Grid the mixing data onto one pressure grid...');
25
26  warning('off','MATLAB:interp1:NaNinY');
27
28  load mixing_data.mat
29  profiles=length(mixing_data.fltid);
30
31  %% Copy single variables in new structure
32  mixing_data_gridded.fltid=mixing_data.fltid;
33  mixing_data_gridded.profile_number=mixing_data.profile_number;
34  mixing_data_gridded.lon=mixing_data.lon;
35  mixing_data_gridded.lat=mixing_data.lat;
36  mixing_data_gridded.surface_mlt=mixing_data.surface_mlt;
37  mixing_data_gridded.MLD=mixing_data.MLD;
38
39  %% Grab CTD & EM variables and interp on a P gridd
40  % Create new 3db pressure grid
41  P=(1:dz:1650)';
42  % For each profile
43  for i=1:profiles;
44      mixing_data_gridded.P(:,i)=P;
45      mixing_data_gridded.SA(:,i)=interp1(mixing_data.Pctd(:,i),mixing_data.SA
            (:,i),P);
46      mixing_data_gridded.CT(:,i)=interp1(mixing_data.Pctd(:,i),mixing_data.CT
            (:,i),P);
```

```
47      mixing_data_gridded.sigma_0(:,i)=interp1(mixing_data.Pctd(:,i),
            mixing_data.sigma_0(:,i),P);
48      mixing_data_gridded.N2(:,i)=interp1(mixing_data.Pctd(:,i),mixing_data.N2
            (:,i),P);
49      mixing_data_gridded.N2_ref(:,i)=interp1(mixing_data.Pctd(:,i),mixing_data
            .N2_ref(:,i),P);
50      mixing_data_gridded.N2_100m(:,i)=interp1(mixing_data.Pctd(:,i),
            mixing_data.N2_100m(:,i),P);
51      mixing_data_gridded.strain(:,i)=interp1(mixing_data.Pctd(:,i),mixing_data
            .strain(:,i),P);
52      mixing_data_gridded.fallrate(:,i)=interp1(mixing_data.Pef(:,i),
            mixing_data.fallrate(:,i),P);
53      mixing_data_gridded.U(:,i)=interp1(mixing_data.Pef(:,i),mixing_data.U(:,i
            ),P);
54      mixing_data_gridded.V(:,i)=interp1(mixing_data.Pef(:,i),mixing_data.V(:,i
            ),P);
55      mixing_data_gridded.speed(:,i)=interp1(mixing_data.Pef(:,i),mixing_data.
            speed(:,i),P);
56      mixing_data_gridded.shear(:,i)=interp1(mixing_data.Pef(:,i),mixing_data.
            shear(:,i),P);
57  end
58
59  save('mixing_data_gridded.mat','-regexp','^mixing_data_gridded','^flt');
```

B.8.2 mx_derive_mixing.m

```
1   function [mixing_data_gridded] = mx_derive_mixing(dz,fftpt,lzmin_fixed,
        lzmax_fixed,mx_parameters,run)
2
3   % mx_derive_mixing                                              Derive the mixing
4   %=========================================================================
5   %
6   % USAGE:
7   %   [] = mx_derive_mixing(dz,fftpt,lzmin_fixed,lzmax_fixed,mx_parameters,run)
8   %
9   % DESCRIPTION:
10
11  %
12  % INPUT:
13  %       dz              = pressure grid interval for main gridding [m]
14  %       fftpt           = Number of points for the fast fourier transform eg
        128
15  %       lzmin_fixed     = mini wavelength integration for shear/strain
        spectra  [m]
16  %       lzmax_fixed     = maxi wavelength integration for shear/strain
        spectra  [m]
17  %
18  % OUTPUT:
19  %   mixing_data_gridded.mat
20  %
21  % AUTHOR:
22  %   Amelie MEYER
23  %
24  % VERSION NUMBER: 1.0 (25th June, 2014)
25  %
26  %=========================================================================
27  display('Derive mixing...');
28
29  %% SET PARAMETERS
30  % Load the parameters
31  eval(['load ' mx_parameters '.mat']);
32  % Difference in pressure between 2 adjacent bins:
33  dzfd=dz;
34  % VERTICAL RESOLUTION
35  % vertical resoultion of CTD/strain profile:
36  dzg=dz;
37  % fftpt: set to 128 but could be 32,64,128... or any power of 2
38  fftpt;
39  % A function of how many points you want in each spectral calculation
```

```
%(fftpt=vertical_bin_size/dzg)
 vertical_bin_size=dzg*fftpt;
% The size of the overlap of vertical bins you are calculating spectra for
% where 1/2 is equivalent to 50% overlapping bins.
 dp_overlap=vertical_bin_size/2;
% (SW /2) vertical resolution of output (epsilon,kappa etc).The size of
% this relative to dp_overlap will determine how much overlap/vertical
% averaging goes into the estimate.Here set to segments that are 1/2
%overlapping.
 dp_resolution=dp_overlap/8;

%% Estimate dissipation using shear parameterization
% Computes the dissipation rate (epsilon) and diapycnal
% turbulent eddy diffusivity (kappa) using Henyey, Wright and Flatte
% model fine scale parameterization of turbulent dissipation using shear
% information only. This is computed in depth bins starting at the surface
% then down.  Implication:
% the bottom-most depth bin i snot full depth and the spectral
% estimate in the bottom-most depth bin is maybe compromised.
 mx_derive_mixing_shear

%% Estimate dissipation using strain parameterization
% Computes the dissipation rate (epsilon) and diapycnal
% turbulent eddy diffusivity (kappa) using Henyey, Wright and Flatte
% model fine scale parameterization of turbulent dissipation using strain
% information only. This is computed in depth bins starting at the surface
% then down.  Implication:
% the bottom-most depth bin i snot full depth and the spectral
% estimate in the bottom-most depth bin is maybe compromised.
 mx_derive_mixing_strain

%% Finescale shear strain parameterization of dissipation
% Combines estimates of dissipation from shear and dissipation from strain
% by computing the correction factor dependent on the observed (as opposed
% to assumed) shear to strain ratio. We use the "straight-up" shear to
% strain ratio and assume that the shear variance integral and strain
% variance integral have been calculated using the same wavelength range
% of integration.
 mx_derive_mixing_shearstrain
```

B.8.3 mx_derive_mixing_shear.m

```
%=========================================================================
% mx_derive_mixing_shear
%
% DESCRIPTION:
% Computes the dissipation rate (epsilon) and diapycnal
% turbulent eddy diffusivity (kappa) using Henyey, Wright and Flatte
% model fine scale parameterization of turbulent dissipation using shear
% information only. This is computed in depth bins starting at the surface
% then down.  Implication:
% the bottom-most depth bin i snot full depth and the spectral
% estimate in the bottom-most depth bin is maybe compromised.
%
% INPUT:
%   Vertical profile of pressure and vertical shear (complex).
%
% OUTPUT:
%   Shear spectra for various depth segments of the profile and
% estimates of dissipation and diffusivity for each depth segment.
%
% AUTHOR:                 Alberto NAVEIRA GARABATO
%
% Revisited by: Stephanie WATERMAN 2010
%                                 Amelie MEYER June 2014
%
% VERSION NUMBER: 1.0 (25th June, 2014)
%
% REFERENCES:
```

```
% Henyey, F. S., Wright, J., and Flatte, S.M., 1986: Energy and action
% flow through the internal wave field: an eikonal approach.
% Journal of Geophysical Research, 91, 8487-8496
%
%=========================================================================

load mixing_data_gridded.mat
profiles=length(mixing_data_gridded.fltid);

%% FOR EACH PROFILE
for i=1:profiles;
    % List of indices at which we have shear data
    ii=find(isnan(mixing_data_gridded.shear(:,i))==0);
    if isempty(ii)==1;
        % If there is no shear data in that pofile, add empty (NaN)
        % variables to structure:
        li=mixing_data_gridded.P(:,i);
        mixing_data_gridded.epsilon_shear(1:li,i)=NaN;
        mixing_data_gridded.Kz_shear(1:li,i)=NaN;
        mixing_data_gridded.P_m_shear(1:li,i)=NaN'';
        mixing_data_gridded.shear_variance(1:li,i)=NaN'';
        mixing_data_gridded.CW_S_variance(1:li,i)=NaN;
        mixing_data_gridded.CCW_S_variance(1:li,i)=NaN;
        mixing_data_gridded.Mc_shear(1:li,i)=NaN;

    else % if there are shear data, derive variables:
        i_min=min(ii);                          % smallest indice
        p_min=mixing_data_gridded.P(i_min,i);   % corresponding depth

        % Find maximum pressure with velocity measurement:
        i_max=max(ii);
        p_max=mixing_data_gridded.P(i_max,i); % corresponding depth

        % Define dp centre scale (centre points of depth segments)
        % top-down for this station; calculation will result in
        % epsilon/kappa values at dp centre:
        dp_centre_here=[(p_min+dp_overlap):dp_resolution:p_max-dp_overlap];

        % Make some variables to store the spectra from each depth bin
        % in so you can go back and look at them later:
        spectra=ones(length(dp_centre_here),fftpt+1)*NaN;
        spectra_mod=ones(length(dp_centre_here),fftpt+1)*NaN;
        spectra_CCW=ones(length(dp_centre_here),fftpt+1)*NaN;
        spectra_CCW_mod=ones(length(dp_centre_here),fftpt+1)*NaN;
        spectra_CW=ones(length(dp_centre_here),fftpt+1)*NaN;
        spectra_CW_mod=ones(length(dp_centre_here),fftpt+1)*NaN;
        spectra_GM=ones(length(dp_centre_here),fftpt+1)*NaN;
        freq_scale=ones(length(dp_centre_here),fftpt+1)*NaN;
        kzax_scale=ones(length(dp_centre_here),fftpt+1)*NaN;


        %% FOR EACH DEPTH BIN
        for idp=1:length(dp_centre_here)

            % Select pressure interval for this depth bin
            % Upper bound of depth segment:
            imin=max(1,(min(find(mixing_data_gridded.P(:,i)>=(dp_centre_here(
                idp)-dp_overlap)))));
            pressure_min=mixing_data_gridded.P(imin,i);
            % Lower bound of depth segment
            imax=min((max(find(mixing_data_gridded.P(:,i)<(dp_centre_here(idp
                )+dp_overlap)))),size(mixing_data_gridded.P(:,1),1));
            pressure_max=mixing_data_gridded.P(imax,i);

            %% NORMALISE BY N2
            % Normalise vertical shear segment by mean N
            % Calculate mean N over pressure interval:
            [yy imin_CTD]=min(abs(mixing_data_gridded.P(:,i)-pressure_min));
            [yy imax_CTD]=min(abs(mixing_data_gridded.P(:,i)-pressure_max));

```

```
96              if imax_CTD<length(mixing_data_gridded.P(:,i))
97                  imax_CTD=imax_CTD+1;
98              end
99
100             pressure_min_CTD=mixing_data_gridded.P(imin_CTD,i);
101             pressure_max_CTD=mixing_data_gridded.P(imax_CTD,i);
102
103             % Compute mean N^2 for this depth bin:
104             n_mean=sqrt(nanmean(mixing_data_gridded.N2(imin_CTD:imax_CTD,i)))
                    ;
105             n_ref_mean=sqrt(nanmean(mixing_data_gridded.N2_ref(imin_CTD:
                    imax_CTD,i)));
106
107             % Normalize shearfor selected segment
108             shear=NaN*ones(length(imin:imax));
109             shear=mixing_data_gridded.shear(imin:imax,i)./n_ref_mean;
110
111             %% DERIVE THE SHEAR SPECTRA
112             % As well as the cospectra and rotary spectra for segment
113
114             shearttemp = shear(~isnan(shear(:)));
115             N = length(shearttemp);
116             ratio = N/length(shear);
117             x=shearttemp;
118
119             if isempty(x)==0
120                 p_grid(idp)=dp_centre_here(idp);
121                 pressure_min_segment_CTD(idp)=pressure_min_CTD;
122                 pressure_max_segment_CTD(idp)=pressure_max_CTD;
123                 n_mean_segment(idp)=n_mean;
124                 n_ref_mean_segment(idp)=n_ref_mean;
125                 pressure_min_segment(idp)=pressure_min;
126                 pressure_max_segment(idp)=pressure_max;
127                 % Number of points in spectra calculation not to be
128                 % confused with the bouyancy frequency:
129                 N_segment(idp)=N;
130                 ratio_segment(idp)=ratio;
131                 % dz2 assumes depthbin uniform over depthrange of transform
132                 dz2 = sw_dpth(nanmean(diff(mixing_data_gridded.P(imin:imax,i)
                        )),mixing_data_gridded.lat(i));
133                 Fs = 1/dz2;
134                 x=real(shearttemp);
135                 y=imag(shearttemp);
136                 % We use Tycho2_cospectra.m to derive power and variance
137                 % spectrum. Kzax, one of the output is  the variance of
138                 % the shear.
139                 eval(parameters.cospectral_method)
140
141                 %% CORRECTIONS
142                 % Sampled data to improve the high wavenumber part of the
143                 % spectra. See Polzin et al. (2002) for details.
144
145                 % 1) first-differencing:
146                 if parameters.switch_fd==1
147                     Tfd = sinc(kzax*dzfd/(2*pi)).^2;
148                 else
149                     Tfd = 1;
150                 end
151
152                 % 2) Voltmeter correction
153                 % kzax=m (wave number), W=fall rate (ms-1), 50s represents
154                 % 50 points of 1s data (fitting interval)
155                 W_mean=nanmean(mixing_data_gridded.fallrate(imin_CTD:imax_CTD
                        ,i));
156                 Tvolt=1./[sinc(kzax*50*W_mean/(2*pi))];
157
158                 % Calculate model transfer function
159                 Tmod=Tvolt.*Tfd;
160
161                 % Correct the power spectral density
```

```
162                 spec_mod=spec./Tmod';
163                 spec_CCW_mod=spec_CCW./Tmod';
164                 spec_CW_mod=spec_CW./Tmod';
165
166
167                 %% CRITICAL WAVENUMBER
168                 % Using the info we have of local N2, we work out critical
169                 % wavenumber mc which gives us better estimate of the
170                 % minimum vertical wavelength of integration.
171
172                 shearvariance=kzax(2)*cumsum(spec_mod);
173                 [nnn,index]=min(abs(shearvariance-0.7));
174
175                 if shearvariance(end)>0.7
176                     index=find(shearvariance-0.7>0,1);
177                     if index==1
178                         index=round(length(shearvariance)/2);
179                         criticalm(idp)=kzax(index)*(0.7/shearvariance(index))
                                /(2*pi);
180                         minwavelength(idp)=1/criticalm(idp);
181                     else
182                     fraction=(0.7-shearvariance(index-1))/(shearvariance(
                            index)-shearvariance(index-1));
183                     criticalm(idp)=(kzax(index)+kzax(2)*(0.5+fraction))/(2*pi
                            );
184                     minwavelength(idp)=1/criticalm(idp);
185                     end
186                 else index=round(length(shearvariance)/2);
187                     criticalm(idp)=kzax(index)*(0.7/shearvariance(index))/(2*
                            pi);
188                     minwavelength(idp)=1/criticalm(idp);
189                 end
190
191                 ind(idp)=index;
192
193
194                 %% LINEAR INTERPOLATION
195                 % Linearly interpolate onto finer frequency axis makes
196                 % integrating between certain wavelengths more accurate.
197                 lambda_max=1/freq(2);
198                 lambda_min=1/freq(end);
199                 wavelengths=lambda_min:1:lambda_max;
200                 freq2=1./wavelengths;
201
202                 spec2 = interp1(freq,spec,freq2);
203                 spec_mod2 = interp1(freq,spec_mod,freq2);
204                 spec_CCW2 = interp1(freq,spec_CCW,freq2);
205                 spec_CCW_mod2 = interp1(freq,spec_CCW_mod,freq2);
206                 spec_CW2 = interp1(freq,spec,freq2);
207                 spec_CW_mod2 = interp1(freq,spec_CW_mod,freq2);
208                 kzax2 = interp1(freq,kzax,freq2);
209
210                 spec=spec2;
211                 spec_mod=spec_mod2;
212                 spec_CCW=spec_CCW2;
213                 spec_CCW_mod=spec_CCW_mod2;
214                 spec_CW=spec_CW2;
215                 spec_CW_mod=spec_CW_mod2;
216                 kzax=kzax2;
217                 freq=freq2;
218
219                 % Store spectra for this depth bin
220                 for ii=1:length(spec)
221                     spectra(idp,ii)=spec(ii);
222                     spectra_mod(idp,ii)=spec_mod(ii);
223                     spectra_mod2(idp,ii)=spec_mod2(ii); % new scale spectra
224
225                     spectra_CCW(idp,ii)=spec_CCW(ii);
226                     spectra_CCW_mod(idp,ii)=spec_CCW_mod(ii);
227                     spectra_CCW_mod2(idp,ii)=spec_CCW_mod2(ii);
```

```
228
229                     spectra_CW(idp,ii)=spec_CW(ii);
230                     spectra_CW_mod(idp,ii)=spec_CW_mod(ii);
231                     spectra_CW_mod2(idp,ii)=spec_CW_mod2(ii);
232
233                     freq_scale(idp,ii)=freq(ii);
234                     kzax_scale(idp,ii)=kzax(ii);
235                     freq_scale2(idp,ii)=freq2(ii);
236                     kzax_scale2(idp,ii)=kzax2(ii);
237                 end
238
239
240                 %% INTEGRATED SHEAR VARIANCE
241                 % Consider the integrated shear variance between two
242                 % wavelengths of interest. It is the important part of the
243                 % fine structure estimate of epsilon/kappa.
244
245                 % Define lzmax and calculate lzmin, the minimum vertical
246                 % wavelength of integration:
247                 lzmin(idp)=minwavelength(idp);
248                 lzmax(idp)=(2*pi/min(kzax))-1;
249                 % Could also chose to use the fixed values:
250                     % lzmin(idp)=lzmin_fixed;
251                     % lzmax(idp)=lzmax_fixed;
252
253                 % Integrate power spectral density between lzmin and lzmax:
254                 imax_int=min(find(kzax<(2*pi/lzmax(idp))));
255                 imin_int=max(find(kzax>(2*pi/lzmin(idp))));
256
257                 if isempty(imin_int)==0 & isempty(imax_int)==0
258                     [rows columns]=size(kzax);
259                     if rows~=1
260                         kzax=kzax';
261                     end
262                     vector1=cat(2,kzax(imin_int),kzax(imin_int:imax_int),kzax
                            (imax_int),kzax(imin_int));
263
264                     % Shear variance:
265                     temp=spec_mod(imin_int:imax_int);
266                     [rows columns]=size(spec_mod);
267                     if rows~=1
268                         temp=temp';
269                     end
270                     vector2=cat(2,0,temp,0,0);
271                     variance_int=polyarea(vector1,vector2);
272
273                     % Variance CCW shear:
274                     temp=spec_CCW_mod(imin_int:imax_int);
275                     [rows columns]=size(spec_mod);
276                     if rows~=1
277                         temp=temp';
278                     end
279                     vector2=cat(2,0,temp,0,0);
280                     variance_int_CCW=polyarea(vector1,vector2);
281
282                     % Variance CW shear:
283                     temp=spec_CW_mod(imin_int:imax_int);
284                     [rows columns]=size(spec_mod);
285                     if rows~=1
286                         temp=temp';
287                     end
288                     vector2=cat(2,0,temp,0,0);
289                     variance_int_CW=polyarea(vector1,vector2);
290                     variance_integral(idp)=variance_int;
291                     variance_integral_CCW(idp)=variance_int_CCW;
292                     variance_integral_CW(idp)=variance_int_CW;
293
294                     kmin_integral(idp)=kzax(imin_int);
295                     kmax_integral(idp)=kzax(imax_int);
296                     no_pts_integral(idp)=imax_int-imin_int+1;
```

```
297
298
299                     %% DERIVE DISSIPATION AND DIFFUSIVITY
300                     % For normalization purposes we compute power spectral
301                     % density of normalized vertical shear for the GM76 model
302                     N2mean = n_ref_mean^2;
303
304                     % Power spectral density of vertical shear normalised
305                     % by N for the GM76 model:
306                     betastar=pi*parameters.jstar/parameters.b*sqrt(N2mean)/
                            parameters.N0;
307
308                     % Power spectral density of horizontal velocity:
309                     phi_u = 3*parameters.E*parameters.b^3*parameters.N0^2/(2*
                            parameters.jstar*pi) ./(1+kzax/betastar).^2;
310                     % Power spectral density of vertical shear normalised
311                     phi_sn = kzax.^2.*phi_u/N2mean;
312
313                     for ii=1:length(phi_sn)
314                         spectra_GM(idp,ii)=phi_sn(ii);
315                     end
316
317                     % Integrate GM76 power spectral density up to cutoff
318                     % wavelength
319                     if isempty(imin_int)==0 & isempty(imax_int)==0
320
321                         vector1=cat(2,kzax(imin_int),kzax(imin_int:imax_int),
                                kzax(imax_int),kzax(imin_int));
322
323                         temp=phi_sn(imin_int:imax_int);
324                         [rows columns]=size(phi_sn);
325                         if rows~=1
326                             temp=temp';
327                         end
328                         vector2=cat(2,0,temp,0,0);
329
330                         GM_variance_integral(idp)=polyarea(vector1,vector2);
331
332                     else
333                         GM_variance_integral(idp)=NaN;
334                     end
335
336                     % Compute dissipation rate (epsilon) from
337                     % the integrated strain variance:
338                     epsilon(idp)=parameters.epsilon0*N2mean/parameters.N0^2*
                            variance_integral(idp)^2/GM_variance_integral(idp)^2;
                            % parameterization uncertain to within a factor of 2
339
340                     % Modify esimate of epsilon by a few factors (see
341                     % Kurt's papers for details); First a correction factor
342                     % that depends on latitude. Second a correction factor
343                     % that depends on the shear to strain ratio.
344                     fmean=abs(sw_f(mixing_data_gridded.lat(i)));
345                     extra1 = (fmean/parameters.f0) * acosh(sqrt(N2mean)/fmean
                            )/acosh(parameters.N0/parameters.f0);
346                     epsilon(idp) = epsilon(idp)*extra1;
347                     extra2=(3*(parameters.R+1))./(2*sqrt(2).*parameters.R.*(
                            abs(parameters.R-1)).^(1/2));
348
349                     epsilon(idp)=epsilon(idp)*extra2;
350
351                     % Finally from epsilon calculate diapycnal turbulent
352                     % eddy diffusivity assuming the Osborn relation:
353                     kappa(idp)=parameters.gamma*epsilon(idp)/N2mean;
354
355                     % Add variables to float structure:
356                     mixing_data_gridded.epsilon_shear(idp,i)=epsilon(idp);
357                     mixing_data_gridded.Kz_shear(idp,i)=kappa(idp);
358                     mixing_data_gridded.P_m_shear(idp,i)=p_grid(idp);
```

```
359                         mixing_data_gridded.shear_variance(idp,i)=
                                variance_integral(idp);
360                         mixing_data_gridded.CW_S_variance(idp,i)=
                                variance_integral_CW(idp);
361                         mixing_data_gridded.CCW_S_variance(idp,i)=
                                variance_integral_CCW(idp);
362                         mixing_data_gridded.Mc_shear(idp,i)=criticalm(idp);
363
364                     end
365                 end
366             end
367         end
368 end
369
370 save('mixing_data_gridded.mat','-regexp','^mixing_data_gridded','^flt');
```

B.8.4 mx_derive_mixing_strain.m

```
1  %=========================================================================
2  % mx_derive_mixing_strain
3  %
4  % DESCRIPTION:
5  % Computes the dissipation rate (epsilon) and diapycnal
6  % turbulent eddy diffusivity (kappa) using Henyey, Wright and Flatte
7  % model fine scale parameterization of turbulent dissipation using strain
8  % information only. This is computed in depth bins starting at the surface
9  % then down.  Implication:
10 % the bottom-most depth bin i snot full depth and the spectral
11 % estimate in the bottom-most depth bin is maybe compromised.
12 %
13 % INPUT:
14 %   Vertical profile of pressure, N^2, N^2 reference and strain =
15 % (N^2-N^2_ref)/N^2_ref.
16 %
17 % OUTPUT:
18 %   Strain spectra for various depth segments of the profile and
19 % estimates of dissipation and diffusivity for each depth segment.
20 %
21 % AUTHOR:                 Alberto NAVEIRA GARABATO
22 %
23 % Revisited by: Stephanie WATERMAN 2010
24 %                                 Amelie MEYER June 2014
25 %
26 % VERSION NUMBER: 1.0 (26th June, 2014)
27 %
28 % REFERENCES:
29 % Henyey, F. S., Wright, J., and Flatte, S.M., 1986: Energy and action
30 % flow through the internal wave field: an eikonal approach.
31 % Journal of Geophysical Research, 91, 8487-8496
32 %
33 %=========================================================================
34
35 load mixing_data_gridded.mat
36 profiles=length(mixing_data_gridded.fltid);
37
38 %% FOR EACH PROFILE
39 for i=1:profiles;
40     % Find minimum pressure of this profile to use as starting point of
41     % first depth segment
42     % List of indices at which we have strain:
43     ii=find(isnan(mixing_data_gridded.strain(:,i))==0);
44     i_min=min(ii);                              % smallest indice
45     i_max=max(ii);                              % smallest indice
46     p_min=mixing_data_gridded.P(i_min,i);       % coprresponding depth
47     p_max=mixing_data_gridded.P(i_max,i);       % coprresponding depth
48
49     % Define dp centre scale (centre points of depth segments) top-down
50     % for this station; calculation will result in epsilon/kappa values at
51     % dp centre:
```

```
52      dp_centre_here=[(p_min+dp_overlap):dp_resolution:6000];
53
54      % Make some variables to store the spectra from each depth bin in so
55      % you can go back and look at them later:
56      spectra=ones(length(dp_centre_here),fftpt+1)*NaN;
57      spectra_mod=ones(length(dp_centre_here),fftpt+1)*NaN;    % GM corrected
58      spectra_GM=ones(length(dp_centre_here),fftpt+1)*NaN;     % GM corrected
59      freq_scale=ones(length(dp_centre_here),fftpt+1)*NaN;
60      kzax_scale=ones(length(dp_centre_here),fftpt+1)*NaN;
61
62
63  %% FOR EACH DEPTH BIN
64  for idp=1:length(dp_centre_here)
65
66      % Select pressure interval for this depth bin
67      % Upper bound of depth segment:
68      imin=max(1,(min(find(mixing_data_gridded.P(:,i)>=(dp_centre_here(idp)-
            dp_overlap)))));
69      % Lower bound of depth segment:
70      imax=min((max(find(mixing_data_gridded.P(:,i)<(dp_centre_here(idp)+
            dp_overlap)))),size(mixing_data_gridded.P(:,i),1));
71
72      pressure_min=mixing_data_gridded.P(imin,i);
73      pressure_max=mixing_data_gridded.P(imax,i);
74
75      % Compute mean N^2 for this depth bin:
76      n_mean=sqrt(nanmean(mixing_data_gridded.N2(imin:imax,i)));
77      n_ref_mean=sqrt(nanmean(mixing_data_gridded.N2_ref(imin:imax,i)));
78
79      %% DERIVE THE STRAIN SPECTRA
80      % Isolate desired depth range and calculate vertical strain spectrum
81      strain=mixing_data_gridded.strain(imin:imax,i);
82
83      straintemp = strain(~isnan(strain(:)));
84      N = length(straintemp);
85      ratio=N/length(strain);
86
87      if length(straintemp)>1
88          p_grid(idp)=dp_centre_here(idp);
89          pressure_min_segment(idp)=pressure_min;
90          pressure_max_segment(idp)=pressure_max;
91          n_mean_segment(idp)=n_mean;
92          n_ref_mean_segment(idp)=n_ref_mean;
93          N_segment(idp)=N;
94          ratio_segment(idp)=ratio;
95
96          dz =sw_dpth(nanmean(diff(mixing_data_gridded.P(imin:imax,i))),
                mixing_data_gridded.lat(i)); % assumes depth bin is uniform over
                depth range of transform
97          Fs = 1/dz;
98
99          x=straintemp;
100         eval(parameters.spectral_method)   % Method to derive spectrum
101
102         %% SPECTRAL CORRECTIONS
103         % First-differencing
104         if parameters.switch_fd==1
105             Tfd = sinc(kzax*dzfd/(2*pi)).^2;
106         else
107             Tfd = 1;
108         end
109
110         % Calculate model transfer function
111         Tmod = Tfd;
112
113         % Calculate corrected power spectral density
114         spec_mod=spec./Tmod';
115
116         %% CRITICAL WAVENUMBER
117         % Using the info we have of local N2, we work out critical
```

```
118         % wavenumber (mc) which gives us better estimate of the
119         % minimum vertical wavelength of integration:
120
121         shearvariance=kzax(2)*cumsum(spec);
122         [nnn,index]=min(abs(shearvariance-0.7));
123         ind(idp)=index;
124         criticalm(idp)=kzax(index);
125         minwavelength(idp)=1/criticalm(idp)*2*pi;
126         if minwavelength(idp)<12;
127             minwavelength(idp)=12;
128         end
129
130         %% LINEAR INTERPOLATION
131         % Linearly interpolate onto finer frequency axis (makes integrating
132         % between certain wavelengths more accurate)
133
134         lambda_max=1/freq(2);
135         lambda_min=1/freq(end);
136         wavelengths=lambda_min:1:lambda_max;
137         freq2=1./wavelengths;
138
139         spec2 = interp1(freq,spec,freq2);
140         spec_mod2 = interp1(freq,spec_mod,freq2);
141         kzax2 = interp1(freq,kzax,freq2);
142
143         % Store spectra for this depth bin so you can look at it later:
144         for ii=1:length(spec2)
145             spectra(idp,ii)=spec2(ii);
146             spectra_mod(idp,ii)=spec_mod2(ii);
147             freq_scale(idp,ii)=freq2(ii);
148             kzax_scale(idp,ii)=kzax2(ii);
149         end
150
151         spec=spec2;
152         spec_mod=spec_mod2;
153         kzax=kzax2;
154         freq=freq2;
155
156         %% INTEGRATED SHEAR VARIANCE
157         % Consider the integrated strain variance between the two
158         % wavelengths of interest - this is the meat in the fine structure
159         % estimate of mixing
160
161         % Define lzmax and lzmin, the range of vertical
162         % wavelengths of integration:
163         lzmin(idp)=minwavelength(idp);
164         lzmax(idp)=(2*pi/min(kzax))-1;
165         % Or could have used the set wavelengths of integration:
166             % lzmin(idp)=lzmin_fixed;
167             % lzmax(idp)=lzmax_fixed;
168
169         % Integrate power spectral density between lzmin and lzmax:
170         imax_int=min(find(kzax<(2*pi/lzmax(idp))));
171         imin_int=max(find(kzax>(2*pi/lzmin(idp))));
172
173         if isempty(imin_int)==0 & isempty(imax_int)==0
174
175             [rows columns]=size(kzax);
176             if rows~=1
177                 kzax=kzax';
178             end
179             vector1=cat(2,kzax(imin_int),kzax(imin_int:imax_int),kzax(
                    imax_int),kzax(imin_int));
180
181             [rows columns]=size(spec_mod);
182             if rows~=1
183                 spec_mod=spec_mod';
184             end
185
186             vector2=cat(2,0,spec_mod(imin_int:imax_int),0,0);
```

```
187
188             % Integrated variance:
189             variance_int=polyarea(vector1,vector2);
190             variance_integral(idp)=variance_int;
191
192             kmin_integral(idp)=kzax(imin_int);
193             kmax_integral(idp)=kzax(imax_int);
194             no_pts_integral(idp)=imax_int-imin_int+1;
195
196             %% DERIVE DISSIPATION AND DIFFUSIVITY
197             % For normalization purposes compute power spectral density of
198             % vertical strain for the GM76 model
199
200             N2mean = n_mean.^2;
201             N2refmean = n_ref_mean.^2;
202
203             fmean=abs(sw_f(mixing_data_gridded.lat(i)));
204
205             betastar=pi*parameters.jstar/parameters.b*sqrt(N2refmean)/
                    parameters.N0;
206
207             % Power spectral density of vertical displacement:
208             phi_zeta = parameters.E*parameters.b^3*parameters.N0^2/(2*
                    parameters.jstar*pi*N2refmean) ./(1+kzax/betastar).^2;
209             % Power spectral density of vertical strain:
210             phi_eta = kzax.^2.*phi_zeta;
211
212             for ii=1:length(phi_eta)
213                 spectra_GM(idp,ii)=phi_eta(ii);
214             end
215
216             %% Integrate GM76 power spectral density
217             if isempty(imin_int)==0 & isempty(imax_int)==0
218                 vector1=cat(2,kzax(imin_int),kzax(imin_int:imax_int),kzax(
                        imax_int),kzax(imin_int));
219                 [rows columns]=size(phi_eta);
220
221                 if rows~=1
222                     phi_eta=phi_eta';
223                 end
224
225                 vector2=cat(2,0,phi_eta(imin_int:imax_int),0,0);
226                 GM_variance_int=polyarea(vector1,vector2);
227                 GM_variance_integral(idp)=GM_variance_int;
228             else
229                 GM_variance_integral(idp)=NaN;
230             end
231
232             % Save the N2refmean as new variable: N2b (for background N2)
233             N2b(idp)=N2refmean;
234
235             % Compute dissipation rate (epsilon) from the integrated
236             % strain variance
237             % strain variance normalized to GM76:
238             norm_strain_variance(idp)=variance_integral(idp)^2/
                    GM_variance_integral(idp)^2;
239             epsilon(idp)=parameters.epsilon0*N2refmean/parameters.N0^2*
                    norm_strain_variance(idp);
240
241             % Modify esimate of epsilon by a few factors (see Kurt's papers
242             % for details);  First a correction factor that depends on
243             % latitude.  Second a correction factor that depends on the
244             % shear to strain ratio.
245
246             extra1 = (fmean/parameters.f0) * acosh(sqrt(N2refmean)/fmean)/
                    acosh(parameters.N0/parameters.f0);
247             epsilon(idp) = epsilon(idp)*extra1;
248
249             extra2 = 1/(6*sqrt(2))*(parameters.R*(parameters.R+1))/(sqrt(abs(
                    parameters.R-1)));
```

```
250             epsilon(idp)=epsilon(idp)*extra2;
251
252             % Finally from epsilon calculate diapycnal diffusivity
253             % (kappa) assuming the Osborn relation:
254             kappa(idp)=parameters.gamma*epsilon(idp)/N2refmean;
255
256             % Add variables to float structure:
257             mixing_data_gridded.epsilon_strain(idp,i)=epsilon(idp);
258             mixing_data_gridded.Kz_strain(idp,i)=kappa(idp);
259             mixing_data_gridded.P_m_strain(idp,i)=p_grid(idp);
260             mixing_data_gridded.strain_variance(idp,i)=variance_integral(idp)
                    ;
261             mixing_data_gridded.N2b(idp,i)=N2b(idp)'';
262             mixing_data_gridded.critiWave_strain(idp,i)=minwavelength(idp);
263
264         end
265     end
266 end
267
268     % Index down to which mixing value that can be trusted:
269     ii_max=max(find(p_grid<(p_max-dp_overlap)));
270     mixing_data_gridded.epsilon_strain(ii_max+1:end,i)=NaN;
271     mixing_data_gridded.Kz_strain(ii_max+1:end,i)=NaN;
272     mixing_data_gridded.P_m_strain(ii_max+1:end,i)=NaN;
273     mixing_data_gridded.N2b(ii_max+1:end,i)=NaN;
274
275 end
276
277 save('mixing_data_gridded.mat','-regexp','^mixing_data_gridded','^flt');
```

B.8.5 mx_derive_mixing_shearstrain.m

```
1  %=========================================================================
2  % mx_derive_mixing_shearstrain
3  %
4  % DESCRIPTION:
5  % Combines estimates of finestructure epsilon from shear
6  % and strain by computing the correction factor dependent on the observed
7  % (as opposed to assumed) shear to strain ratio.
8  % Here uses the "straight-up" shear to strain ratio and assumes that the
9  % shear variance integral and strain variance integral have been calculated
10 % using the same wavelength range of integration.
11 %
12 % INPUT:
13 %
14 % OUTPUT:
15 %   N2_ref      =  Brunt-Vaisala Frequency squared from Polzin code   [ 1/s^2
       ]
16 %
17 % AUTHOR:                 Alberto NAVEIRA GARABATO
18 %
19 % Revisited by: Stephanie WATERMAN 2010
20 %                                Amelie MEYER June 2014
21 %
22 % VERSION NUMBER: 1.0 (25th June, 2014)
23 %
24 %=========================================================================
25
26 profiles=length(mixing_data_gridded.fltid);
27
28 for i=1:profiles;                                          % For each profile
29     common_p_scale=1:3:1648;
30     N2=mixing_data_gridded.N2(:,i);
31
32     %% SHEAR VARIANCE:
33     shear_variance=ones(length(common_p_scale),1)*NaN;
34     epsilon_shear=ones(length(common_p_scale),1)*NaN;
35
36     % Check if there is any shear data for that profile...
```

```
37      check=find(isnan(mixing_data_gridded.Kz_shear)==0);
38      if isempty(check)==1;
39          % There was no shear data then add empty (NaN) variables
40          li=length(mixing_data_gridded.P(:,i));
41          mixing_data_gridded.epsilon_SS(1:li,i)=NaN;
42          mixing_data_gridded.Kz_SS(1:li,i)=NaN;
43          mixing_data_gridded.P_m_SS(1:li,i)=NaN;
44          mixing_data_gridded.Rw(1:li,i)=NaN;
45
46      % If there are shear data then we derive variables:
47      else
48          for idp=1:length(mixing_data_gridded.P_m_shear(:,i));
49              clear ii jj
50              ii=find(common_p_scale==mixing_data_gridded.P_m_shear(idp,i)-(
                    dp_resolution/2));
51              jj=find(common_p_scale==mixing_data_gridded.P_m_shear(idp,i)+(
                    dp_resolution/2));
52              for idp2=ii:jj
53                  % Copy variables from mixing_data
54                  shear_variance(idp2)=mixing_data_gridded.shear_variance(idp,i
                        );
55                  epsilon_shear(idp2)=mixing_data_gridded.epsilon_shear(idp,i);
56              end
57          end
58
59          %% STRAIN
60          % Pre-allocate some variables
61          strain_variance=ones(length(common_p_scale),1)*NaN;
62          epsilon_strain=ones(length(common_p_scale),1)*NaN;
63          N2b=ones(length(common_p_scale),1)*NaN;
64          mixing_data_gridded.P_m_shear(isnan(mixing_data_gridded.P_m_shear))
                =0;
65          % Works out hom many d index apart shear and strain are:
66          d=mixing_data_gridded.P_m_shear(1,i)-mixing_data_gridded.P_m_strain
                (1,i)/3;
67          if d>3
68              d=0;
69          end
70          if d<0
71              d=0;
72          end
73
74          for idp=1:length(mixing_data_gridded.P_m_strain(:,i));
75              clear ii2 jj2
76              ii2=find(common_p_scale==mixing_data_gridded.P_m_strain(idp,i)-(
                    dp_resolution/2));
77              jj2=find(common_p_scale==mixing_data_gridded.P_m_strain(idp,i)+(
                    dp_resolution/2));
78              % Moving all epsi strain by 1 index down to match epsi
79              % shear scale:
80              for idp2=ii2:jj2
81                  % Copy variables from mixing_data
82                  strain_variance(idp2+d)=mixing_data_gridded.strain_variance(
                        idp,i);
83                  epsilon_strain(idp2+d)=mixing_data_gridded.epsilon_strain(idp
                        ,i);
84                  N2b(idp2+d)=mixing_data_gridded.N2b(idp,i);
85              end
86          end
87
88          %% FIND POINT OF MAX N2 IN WATER COLUMN
89          [crap ind]=max(N2);
90          depth(i)=common_p_scale(ind);
91          treshhold(i)=depth(i)+dp_overlap;
92          [crap index_treshhold(i)]=min(abs(treshhold(i)-common_p_scale));
93
94          %% DERIVE SHEAR STRAIN RATIO
95          Rw=shear_variance./strain_variance;
96          % Replace Rw values in surface waters with a mean Rw
97          Rw(1:index_treshhold(i))=5; % 5 is the mean Rw for the data set
```

```
98
99          % Apply extra factor multiplying epsilon depending
100         % on shear to strain ratio using actual shear to strain ratio:
101         h=real((3*(Rw+1))./(2*sqrt(2).*Rw.*((Rw-1)).^(1/2)));
102
103         % Remove constant Rw applied:
104         epsilon=epsilon_shear;
105         extra2=(3*(parameters.R+1))./(2*sqrt(2).*parameters.R.*((parameters.R
                -1)).^(1/2));
106         epsilon2=epsilon/extra2;
107
108         % Apply variable Rw:
109         epsilon_shear_and_strain=epsilon2.*h;
110
111         % Finally from epsilon calculate diapycnal diffusivity
112         % (kappa) assuming the Osborn relation:
113         kappa=parameters.gamma*epsilon_shear_and_strain./N2b;
114
115         % Add variables to float structure:
116         mixing_data_gridded.epsilon(:,i)=epsilon_shear_and_strain;
117         mixing_data_gridded.Kz(:,i)=kappa;
118         mixing_data_gridded.P_m(:,i)=common_p_scale;
119         mixing_data_gridded.Rw(:,i)=Rw;
120     end
121 end
122
123 % Remove a variable from the float structure
124 mixing_data_gridded=rmfield(mixing_data_gridded,'N2b');
125
126 eval(['save(''mixing_data_gridded_' run '.mat'',''-regexp'',''^
        mixing_data_gridded'',''^flt'')']);
```

B.9 Plot mixing variables

B.9.1 mx_plot_dissipation_rate.m

```
1  function [] = mx_plot_dissipation_rate(fig,directory,run)
2
3  % mx_plot_dissipation_rate                          Plot the dissipation rate
4  %=========================================================================
5  %
6  % USAGE:
7  %   [] = mx_plot_dissipation_rate(fig,directory,run)
8  %
9  % DESCRIPTION:
10 %   Plot all the dissipation rate profiles.
11 %
12 % INPUT:
13 %   fig             = either 'on' or 'off' to utrn fig display on and off
14 %   directory       = path of directory where figure is saved
15 %
16 % OUTPUT:
17 %   Figure:  /dissipation_rate.eps
18 %
19 % AUTHOR:
20 %   Amelie MEYER
21 %
22 % VERSION NUMBER: 1.0 (26rd June, 2014)
23 %
24 % RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
25 %           Bindoff. Mixing variability in the Southern Ocean. Journal of
26 %           Physical Oceanography, 45,966-987, 2015.
27 %=========================================================================
28 disp('Plot the dissipation rate (epsilon)...')
29
30 eval(['load mixing_data_gridded_' run '.mat']);
31
```

```
%% Plot of dissipation along profile numbers
nb_profiles=length(mixing_data_gridded.fltid);
mini=min(min(mixing_data_gridded.epsilon));
maxi=max(max(mixing_data_gridded.epsilon));
epsilon=mixing_data_gridded.epsilon;
epsilon(find(epsilon==0))=NaN;

h2=figure(3);
clf
set(3,'Position',[30 50 476 245]); % Where [horiz ver width height] of the
    figure width=1000 for first plot
eval(['set(3,''visible'',''' fig ''');'])
imagesc(1:nb_profiles,mixing_data_gridded.P(:,1),log10(epsilon));
hold on
axis ij
set(gca,'Position',[0.088 0.13 0.87 0.77])    % position of plot in figure
colormap(rot90(hot,2));
h=colorbar;
set(get(h,'ylabel'),'string','log_{10}(\epsilon) [m^{2}s^{-3}]','fontsize'
    ,10);
caxis([log10(mini) log10(maxi)])
xlim([0 length(mixing_data_gridded.profile_number)])
ylim([0 1600]);
xlabel('Cumulative profile number','FontSize',10);
ylabel('Pressure [dbar]','FontSize',10)

%% Density contour
sigma_0=mixing_data_gridded.sigma_0;
[C,h]=contour(1:length(mixing_data_gridded.profile_number),
    mixing_data_gridded.P(:,1),sigma_0,[27:0.1:29],'color',[0.6 0.6 0.6],'
    linewidth',1);

title(['Float ' int2str(flt) ''])

%% Saving figure options
set(gcf,'PaperPositionMode','auto')
set(gcf,'renderer','painters')
signature('Initial analysis','')
print(h2,'-depsc2',['' directory '/figures/dissipation_rate.eps']);
```

B.9.2 mx_plot_diffusivity.m

```
function [] = mx_plot_diffusivity(fig,directory,run)

% mx_plot_diffusivity                                    Plot the diffusivity
%=========================================================================
%
% USAGE:
%  [] = mx_plot_diffusivity(fig,directory,run)
%
% DESCRIPTION:
%  Plot all the diffusivity profiles
%
% INPUT:
%  fig              = either 'on' or 'off' to utrn fig display on and off
%  directory        = path of directory where figure is saved
%
% OUTPUT:
%  Figure:  /diffusivity.eps
%
% AUTHOR:
%  Amelie MEYER
%
% VERSION NUMBER: 1.0 (26rd June, 2014)
%
% RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
%           Bindoff. Mixing variability in the Southern Ocean. Journal of
%           Physical Oceanography, 45,966-987, 2015.
%=========================================================================
```

```
28 disp('Plot the diffusivity (kappa)...')
29
30 eval(['load mixing_data_gridded_' run '.mat']);
31 load private1/othercolor/colorData.mat Msunsetcolors
32
33 %% Plot of dissipation along profile numbers
34 nb_profiles=length(mixing_data_gridded.fltid);
35 mini=min(min(mixing_data_gridded.Kz));
36 maxi=max(max(mixing_data_gridded.Kz));
37 diffusivity=mixing_data_gridded.Kz;
38 diffusivity(find(diffusivity==0))=NaN;
39
40 h2=figure(3);
41 clf
42 set(3,'Position',[30 50 476 245]); % Where [horiz ver width height] of the
       figure width=1000 for first plot
43 eval(['set(3,''visible'',''' fig ''');'])
44 imagesc(1:nb_profiles,mixing_data_gridded.P(:,1),log10(diffusivity));
45 hold on
46 axis ij
47 set(gca,'Position',[0.088 0.13 0.87 0.77])    % position of plot in figure
48 colormap(flipud(Msunsetcolors));
49 h=colorbar;
50 set(get(h,'ylabel'),'string','log_{10}(K_{z}) [m^{2}s^{-1}]','fontsize',10);
51 caxis([log10(mini) log10(maxi)])
52 xlim([0 length(mixing_data_gridded.profile_number)])
53 ylim([0 1600]);
54 xlabel('Cumulative profile number','FontSize',10);
55 ylabel('Pressure [dbar]','FontSize',10)
56
57 %% Density contour
58 sigma_0=mixing_data_gridded.sigma_0;
59 [C,h]=contour(1:length(mixing_data_gridded.profile_number),
       mixing_data_gridded.P(:,1),sigma_0,[27:0.1:29],'color',[0.6 0.6 0.6],'
       linewidth',1);
60
61 title(['Float ' int2str(flt) ''])
62
63 %% Saving figure options
64 set(gcf,'PaperPositionMode','auto')
65 set(gcf,'renderer','painters')
66 signature('Initial analysis','')
67 print(h2,'-depsc2',['' directory '/figures/diffusivity.eps']);
```

B.9.3 mx_plot_CCW_CW.m

```
1  function [] = mx_plot_CCW_CW(fig,directory,run)
2
3  % mx_plot_CCW_CW                                          Plot the CCW / CW ratio
4  %=========================================================================
5  %
6  % USAGE:
7  %   [] = mx_plot_CCW_CW(fig,directory,run)
8  %
9  % DESCRIPTION:
10 %   Plot all the ratio of rotary shear variance profiles.
11 %
12 % INPUT:
13 %   fig              = either 'on' or 'off' to utrn fig display on and off
14 %   directory        = path of directory where figure is saved
15 %
16 % OUTPUT:
17 %   Figure:  /CCW_CW.eps
18 %
19 % AUTHOR:
20 %   Amelie MEYER
21 %
22 % VERSION NUMBER: 1.0 (26rd June, 2014)
23 %
```

```
% RERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
%           Bindoff. Mixing variability in the Southern Ocean. Journal of
%           Physical Oceanography, 45,966-987, 2015.
%=========================================================================
disp('Plot the CCW / CW ratio...')

eval(['load mixing_data_gridded_' run '.mat']);

%% Plot
nb_profiles=length(mixing_data_gridded.fltid);
CCW=mixing_data_gridded.CCW_S_variance;
CW=mixing_data_gridded.CW_S_variance;

h2=figure(3);
clf
set(3,'Position',[30 50 476 245]); % Where [horiz ver width height] of the
    figure width=1000 for first plot
eval(['set(3,''visible'',''' fig ''');'])
imagesc(1:nb_profiles,mixing_data_gridded.P(:,1),log10(CCW./CW));
hold on
axis ij
set(gca,'Position',[0.088 0.13 0.87 0.77])    % position of plot in figure
colormap(flipud(hot));
h=colorbar;
set(get(h,'ylabel'),'string','log_{10} CCW/CW shear variance ratio','fontsize
    ',10);
caxis([-1 1])
xlim([0 length(mixing_data_gridded.profile_number)])
ylim([0 1600]);
xlabel('Cumulative profile number','FontSize',10);
ylabel('Pressure [dbar]','FontSize',10)

%% Density contour
sigma_0=mixing_data_gridded.sigma_0;
[C,h]=contour(1:length(mixing_data_gridded.profile_number),
    mixing_data_gridded.P(:,1),sigma_0,[27:0.1:29],'color',[0.6 0.6 0.6],'
    linewidth',1);

title(['Float ' int2str(flt) ''])

%% Saving figure options
set(gcf,'PaperPositionMode','auto')
set(gcf,'renderer','painters')
signature('Initial analysis','')
print(h2,'-depsc2',['' directory '/figures/CCW_CW.eps']);
```

B.9.4 mx_plot_Rw.m

```
function [] = mx_plot_Rw(fig,directory,run)

% mx_plot_Rw                                  Plot the shear/strain ratio (Rw)
%=========================================================================
%
% USAGE:
%  [] = mx_plot_Rw(fig,directory,run)
%
% DESCRIPTION:
%  Plot all the shear-to-strain variance ratio profiles.
%
% INPUT:
%  fig              = either 'on' or 'off' to utrn fig display on and off
%  directory        = path of directory where figure is saved
%
% OUTPUT:
%  Figure:  /Rw.eps
%
% AUTHOR:
%  Amelie MEYER
%
```

```
% VERSION NUMBER: 1.0 (26rd June, 2014)
%
% REFERENCE:  A. Meyer, B.M. Sloyan, K.L. Polzin, H.E. Phillips, and N.L.
%             Bindoff. Mixing variability in the Southern Ocean. Journal of
%             Physical Oceanography, 45,966-987, 2015.
%=========================================================================
disp('Plot the shear/strain ratio (Rw)...')

eval(['load mixing_data_gridded_' run '.mat']);

%% Plot
nb_profiles=length(mixing_data_gridded.fltid);
Rw=mixing_data_gridded.Rw;
mini=min(min(mixing_data_gridded.Rw));
maxi=max(max(mixing_data_gridded.Rw));

h2=figure(3);
clf
set(3,'Position',[30 50 476 245]); % Where [horiz ver width height] of the
    figure width=1000 for first plot
eval(['set(3,''visible'',''' fig ''');'])
imagesc(1:nb_profiles,mixing_data_gridded.P(:,1),Rw);
hold on
axis ij
set(gca,'Position',[0.088 0.13 0.87 0.77])    % position of plot in figure
colormap(flipud(fliplr(hot)));
h=colorbar;
set(get(h,'ylabel'),'string','R_\omega','fontsize',10);
caxis([mini maxi])
xlim([0 length(mixing_data_gridded.profile_number)])
ylim([0 1600]);
xlabel('Cumulative profile number','FontSize',10);
ylabel('Pressure [dbar]','FontSize',10)

%% Density contour
sigma_0=mixing_data_gridded.sigma_0;
[C,h]=contour(1:length(mixing_data_gridded.profile_number),
    mixing_data_gridded.P(:,1),sigma_0,[27:0.1:29],'color',[0.6 0.6 0.6],'
    linewidth',1);

title(['Float ' int2str(flt) ''])

%% Saving figure options
set(gcf,'PaperPositionMode','auto')
set(gcf,'renderer','painters')
signature('Initial analysis','')
print(h2,'-depsc2',['' directory '/figures/Rw.eps']);
```

B.10 Function to check the installation of the library

B.10.1 mx_check_functions.m

```

if isempty(which('lastgood2.m'))
    fprintf(2,'You need to add the MX "private1" subdirectory to your path. \
        n');
end

if isempty(which('lastgood2.m'))
    error('You have not added the MX subdirectories to you MATLAB Path')
end

float_data='float_data_vmx';
run='test';
fig='off';
dummy = which('lastgood2.m');
directory = [dummy(1:end-20)];
```

```
15 mx_parameters='mx_parameters';
16
17 fprintf(1,' \n');
18 fprintf(1,'This function is running 36 stored vertical profiles of
       temperature, \n');
19 fprintf(1,'salinity and horizontal velocity from the EM-APEX float 3951
       through\n');
20 fprintf(1,'all the functions in the MX Oceanographic Toolbox.\n');
21 fprintf(1,' \n');
22
23 mx_cf.mx_chks = 1;
24
25
26 %% Check value of constants and parameters in mx_parameters.mat
27 load(mx_parameters);
28
29 fprintf(1,' \n');
30 fprintf(1,'Check value of constants and parameters \n');
31 fprintf(1,'in the parameter file mx_parameters.mat \n');
32 fprintf(1,' \n');
33
34 U = parameters.U;
35 if eval(['' U '-1']) > 1e-13
36     fprintf(2,'Parameter U:    Failed\n');
37     mx_cf.mx_chks = 0;
38 end
39
40 V = parameters.V;
41 if eval(['' V '-1']) > 1e-13
42     fprintf(2,'Parameter V:    Failed\n');
43     mx_cf.mx_chks = 0;
44 end
45
46 if abs(parameters.moving_window - 20) > 1e-13
47     fprintf(2,'Parameter moving_window:    Failed\n');
48     mx_cf.mx_chks = 0;
49 end
50
51 if abs(parameters.dzN2ref - 24) > 1e-13
52     fprintf(2,'Parameter dzN2ref:    Failed\n');
53     mx_cf.mx_chks = 0;
54 end
55
56 if abs(parameters.dzN2 - 6) > 1e-13
57     fprintf(2,'Parameter dzN2:    Failed\n');
58     mx_cf.mx_chks = 0;
59 end
60
61 if abs(parameters.drho - 0.03) > 1e-13
62     fprintf(2,'Parameter drho:    Failed\n');
63     mx_cf.mx_chks = 0;
64 end
65
66 if abs(parameters.dz - 3) > 1e-13
67     fprintf(2,'Parameter dz:    Failed\n');
68     mx_cf.mx_chks = 0;
69 end
70
71 if abs(parameters.dzs - 6) > 1e-13
72     fprintf(2,'Parameter dzs:    Failed\n');
73     mx_cf.mx_chks = 0;
74 end
75
76 if abs(parameters.fftpt - 128) > 1e-13
77     fprintf(2,'Parameter fftpt:    Failed\n');
78     mx_cf.mx_chks = 0;
79 end
80
81 if abs(parameters.lzmin_fixed - 50) > 1e-13
82     fprintf(2,'Parameter lzmin_fixed:    Failed\n');
```

```
83      mx_cf.mx_chks = 0;
84  end
85
86  if abs(parameters.lzmax_fixed - 300) > 1e-13
87      fprintf(2,'Parameter lzmax_fixed:    Failed\n');
88      mx_cf.mx_chks = 0;
89  end
90
91  if abs(parameters.R - 5) > 1e-13
92      fprintf(2,'Parameter R:    Failed\n');
93      mx_cf.mx_chks = 0;
94  end
95
96  if abs(parameters.gamma - 0.2) > 1e-13
97      fprintf(2,'Parameter gamma:    Failed\n');
98      mx_cf.mx_chks = 0;
99  end
100
101 if abs(parameters.epsilon0 - 8e-10) > 1e-13
102     fprintf(2,'Parameter epsilon0:    Failed\n');
103     mx_cf.mx_chks = 0;
104 end
105
106 if abs(parameters.N0 - 0.0052) > 1e-4
107     fprintf(2,'Parameter N0:    Failed\n');
108     mx_cf.mx_chks = 0;
109 end
110
111 if abs(parameters.f0 - 7.8360e-5) > 1e-8
112     fprintf(2,'Parameter f0:    Failed\n');
113     mx_cf.mx_chks = 0;
114 end
115
116 if abs(parameters.E - 6.3e-5) ~= 0
117     fprintf(2,'Parameter E:    Failed\n');
118     mx_cf.mx_chks = 0;
119 end
120
121 if abs(parameters.b - 1300) > 1e-13
122     fprintf(2,'Parameter b:    Failed\n');
123     mx_cf.mx_chks = 0;
124 end
125
126 if abs(parameters.jstar - 3) > 1e-13
127     fprintf(2,'Parameter jstar:    Failed\n');
128     mx_cf.mx_chks = 0;
129 end
130
131 spectralmethod = parameters.spectral_method;
132 string='Tycho2';
133 if eval(' spectralmethod ') ~= eval(' string ')
134     fprintf(2,'Parameter spectral_method:    Failed\n');
135     mx_cf.mx_chks = 0;
136 end
137
138 cospectralmethod = parameters.cospectral_method;
139 string='Tycho2_cospectra';
140 if eval(' cospectralmethod ') ~= eval(' string ')
141     fprintf(2,'Parameter cospectral_method:    Failed\n');
142     mx_cf.mx_chks = 0;
143 end
144
145 if abs(parameters.switch_fd - 1) > 1e-13
146     fprintf(2,'Parameter switch_fd:    Failed\n');
147     mx_cf.mx_chks = 0;
148 end
149
150 if isnan(parameters.crit_rat) ~= 1
151     fprintf(2,'Parameter crit_rat:    Failed\n');
152     mx_cf.mx_chks = 0;
```

```
153 end
154
155 if isnan(parameters.lzmin_threshold) ~= 1
156     fprintf(2,'Parameter lzmin_threshold:    Failed\n');
157     mx_cf.mx_chks = 0;
158 end
159
160 clear E N0 R U V b cospectralmethod critrat drho dz dzN2ref dzN2 dzs epsilon0
        f0 fftpt gamma jstar lzmaxfixed lzminfixed lzminthreshold movingwindow
    spectralmethod string switchfd
161
162 %% Load and build datasets
163 if exist('parameters','var')==0;
164     fprintf(2,'load parameters:    Failed\n');
165     mx_cf.mx_chks = 0;
166 end
167
168 mx_build_initial_data(float_data)
169 if exist('initial_data.mat','file')==0;
170     fprintf(2,'mx_build_initial_data:    Failed\n');
171     mx_cf.mx_chks = 0;
172 end
173
174 %% Derive fallrate of the EM-APEX floats
175 mx_derive_fallrate
176 load initial_data.mat
177 if isempty(['Flt' int2str(flt) '(1).fallrate)']);
178     fprintf(2,'mx_derive_fallrate:    Failed\n');
179     mx_cf.mx_chks = 0;
180 end
181 clear(['Flt' int2str(flt) ''])
182 clear flt
183
184 %% Grid CTD data onto 2.2dbar and EM on 3dbar
185 mx_grid_initialdata(parameters.U,parameters.V)
186
187 %% Derive initial variables
188 mx_derive_abs_T_S
189 mx_derive_potential_density_anomaly
190 mx_derive_N2(parameters.dzN2)
191 mx_derive_mixed_layer_depth(parameters.drho)
192 mx_derive_current_speed
193
194 load initial_data.mat
195
196 if isempty(['float' int2str(flt) '(1).SA)'])==1;
197     fprintf(2,'mx_derive_abs_T_S:    Failed\n');
198     mx_cf.mx_chks = 0;
199 end
200 if isempty(['float' int2str(flt) '(1).CT)'])==1;
201     fprintf(2,'mx_derive_abs_T_S:    Failed\n');
202     mx_cf.mx_chks = 0;
203 end
204 if isempty(['float' int2str(flt) '(1).sigma_0)'])==1;
205     fprintf(2,'mx_derive_potential_density_anomaly:    Failed\n');
206     mx_cf.mx_chks = 0;
207 end
208 if isempty(['float' int2str(flt) '(1).N2)'])==1;
209     fprintf(2,'mx_derive_N2:    Failed\n');
210     mx_cf.mx_chks = 0;
211 end
212 if isempty(['float' int2str(flt) '(1).MLD)'])==1;
213     fprintf(2,'mx_derive_mixed_layer_depth:    Failed\n');
214     mx_cf.mx_chks = 0;
215 end
216 if isempty(['float' int2str(flt) '(1).speed)'])==1;
217     fprintf(2,'mx_derive_current_speed:    Failed\n');
218     mx_cf.mx_chks = 0;
219 end
220
```

```
221 clear(['float' int2str(flt) ''])
222 clear flt
223
224 %% Plot main variables
225 fprintf(1,' \n');
226 fprintf(1,'The default set up for the MX analysis is that output \n');
227 fprintf(1,'figures are turned off (do not appear on the screen)\n');
228 fprintf(1,'and are saved instead in the ''figures'' folder.\n');
229 fprintf(1,' \n');
230
231 mx_grid_all_initial_data
232 if exist('initial_data_gridded.mat','file')==0;
233     fprintf(2,'mx_grid_all_initial_data:   Failed\n');
234     mx_cf.mx_chks = 0;
235 end
236
237 mx_plot_temperature(fig,directory)
238 mx_plot_salinity(fig,directory)
239 mx_plot_N2(fig,directory)
240 mx_plot_mixed_layer_depth(fig,directory)
241 mx_plot_current_speed(fig,directory)
242 if exist([directory,'figures',directory(end),'temperature.eps'],'file')==0;
243     fprintf(2,'mx_plot_temperature:   Failed\n');
244     mx_cf.mx_chks = 0;
245 end
246 if exist([directory,'figures',directory(end),'salinity.eps'],'file')==0;
247     fprintf(2,'mx_plot_salinity:   Failed\n');
248     mx_cf.mx_chks = 0;
249 end
250 if exist([directory,'figures',directory(end),'N2.eps'],'file')==0;
251     fprintf(2,'mx_plot_N2:   Failed\n');
252     mx_cf.mx_chks = 0;
253 end
254 if exist([directory,'figures',directory(end),'mixed_layer_depth.eps'],'file')
    ==0;
255     fprintf(2,'mx_plot_mixed_layer_depth:   Failed\n');
256     mx_cf.mx_chks = 0;
257 end
258 if exist([directory,'figures',directory(end),'current_speed.eps'],'file')==0;
259     fprintf(2,'mx_plot_current_speed:   Failed\n');
260     mx_cf.mx_chks = 0;
261 end
262
263 %% Derive mixing variables
264 mx_derive_N2_ref(parameters.moving_window,parameters.dzN2ref)
265 mx_derive_N2_100m
266 mx_derive_strain
267 mx_derive_shear(parameters.dz,parameters.dzs)
268
269 load mixing_data.mat
270
271 if isempty(mixing_data(1).N2_ref)==1;
272     fprintf(2,'mx_derive_N2_ref:   Failed\n');
273     mx_cf.mx_chks = 0;
274 end
275 if isempty(mixing_data(1).N2_100m)==1;
276     fprintf(2,'mx_derive_N2_100m:   Failed\n');
277     mx_cf.mx_chks = 0;
278 end
279 if isempty(mixing_data(1).strain)==1;
280     fprintf(2,'mx_derive_strain:   Failed\n');
281     mx_cf.mx_chks = 0;
282 end
283 if isempty(mixing_data(1).shear)==1;
284     fprintf(2,'mx_derive_shear:   Failed\n');
285     mx_cf.mx_chks = 0;
286 end
287
288 clear mixing_data flt
289
```

```
290 %% Derive mixing
291 mx_grid_mixingdata(parameters.dz)
292 if exist('mixing_data_gridded.mat','file')==0;
293     fprintf(2,'mx_grid_mixingdata:   Failed\n');
294     mx_cf.mx_chks = 0;
295 end
296
297 mixing_data_gridded=mx_derive_mixing(parameters.dz,parameters.fftpt,
        parameters.lzmin_fixed,parameters.lzmax_fixed,mx_parameters,run);
298
299 load(['mixing_data_gridded_' run '.mat'])
300 if isempty(mixing_data_gridded.epsilon)==1;
301     fprintf(2,'mx_derive_mixing:   Failed\n');
302     mx_cf.mx_chks = 0;
303 end
304 if isempty(mixing_data_gridded.Kz)==1;
305     fprintf(2,'mx_derive_mixing:   Failed\n');
306     mx_cf.mx_chks = 0;
307 end
308 if isempty(mixing_data_gridded.Rw)==1;
309     fprintf(2,'mx_derive_mixing:   Failed\n');
310     mx_cf.mx_chks = 0;
311 end
312 clear mixing_data_gridded flt
313
314 if exist(['mixing_data_gridded_' run '.mat'],'file')==0;
315     fprintf(2,'mx_derive_mixing:   Failed\n');
316     mx_cf.mx_chks = 0;
317 end
318
319 %% Plot main mixing variables
320 fprintf(1,' \n');
321 fprintf(1,'The default set up for the MX analysis is that output \n');
322 fprintf(1,'figures are turned off (do not appear on the screen)\n');
323 fprintf(1,'and are saved instead in the ''figures'' folder.\n');
324 fprintf(1,' \n');
325
326 mx_plot_dissipation_rate(fig,directory,run)
327 mx_plot_diffusivity(fig,directory,run)
328 mx_plot_CCW_CW(fig,directory,run)
329 mx_plot_Rw(fig,directory,run)
330
331 if exist([directory,'figures',directory(end),'dissipation_rate.eps'],'file')
        ==0;
332     fprintf(2,'mx_plot_dissipation_rate:   Failed\n');
333     mx_cf.mx_chks = 0;
334 end
335 if exist([directory,'figures',directory(end),'diffusivity.eps'],'file')==0;
336     fprintf(2,'mx_plot_diffusivity:   Failed\n');
337     mx_cf.mx_chks = 0;
338 end
339 if exist([directory,'figures',directory(end),'CCW_CW.eps'],'file')==0;
340     fprintf(2,'mx_plot_CCW_CW:   Failed\n');
341     mx_cf.mx_chks = 0;
342 end
343 if exist([directory,'figures',directory(end),'Rw.eps'],'file')==0;
344     fprintf(2,'mx_plot_Rw:   Failed\n');
345     mx_cf.mx_chks = 0;
346 end
347
348 %% Diagnostic results
349 if mx_cf.mx_chks == 1 ;
350     fprintf(1,' Finished.\n');
351     fprintf(1,'\n');
352 end
353
354 if mx_cf.mx_chks == 0
355     fprintf(2,'Your installation of the Mixing (MX) Oceanographic Toolbox has
            errors !\n');
356 else
```

```
357       fprintf(1,'Well done! The mx_check_fuctions confirms that the \n');
358       fprintf(1,'Mixing (MX) Oceanographic Toolbox is installed correctly.\n');
359       fprintf(1,' \n');
360       fprintf(1,'You will find output figures saved in the ''figure'' \n');
361       fprintf(1,'folder in the MX library.\n');
362       fprintf(1,'\n');
363       clear mx_cf mx_cv float_data mixing_data fig parameters run directory
              mx_parameters
364   end
```