OceanVar
User Manual

Version 2.0

Mario Adani, Ali Aydogdu, Andrea Cipollone, Eric Jansen, Francesco Carrere, Paolo Oddo

March 10, 2025

Contents

Introduction

OceanVar is a stand-alone fortran software that can be used to minimize the misfits between ocean general circulation models and observations.

It is based on a three-dimensional variational scheme [DP08].

In the following sections the code is described starting from the model basics, the code structure, its implementation and referred namelist.

Part I

Model Basics

1 Formulation of the 3D Variational Assimilation Scheme

Using the the well-known theorem of Bayes we can formulate the probability density function (PDF) for a model state $\mathbf{x}$, given a set of observations, $\mathbf{y}$ as follows:

$$\mathcal{P}(\mathbf{x}|\mathbf{y}) = \frac{\mathcal{P}(\mathbf{y}|\mathbf{x})\,\mathcal{P}(\mathbf{x})}{\mathcal{P}(\mathbf{y})} \tag{1}$$

Assuming Gaussian model and observational errors, the PDFs can be written as:

$$\mathcal{P}(\mathbf{x}) \propto \exp\left[-\frac{1}{2}(\mathbf{x}-\mathbf{x}_b)^{\mathsf{T}}\mathbf{B}^{-1}(\mathbf{x}-\mathbf{x}_b)\right] \tag{2}$$

$$\mathcal{P}(\mathbf{y}|\mathbf{x}) \propto \exp\left[-\frac{1}{2}(\mathbf{y}-H(\mathbf{x}))^{\mathsf{T}}\mathbf{R}^{-1}(\mathbf{y}-H(\mathbf{x}))\right] \tag{3}$$

with $\mathbf{x}_b$ and $\mathbf{B}$ the model background state and covariance matrix, $\mathbf{R}$ the observational covariance matrix and $H(\mathbf{x})$ an observation operator that maps the background state $\mathbf{x}$ onto the observations $\mathbf{y}$.

The analysis follows as the state $\mathbf{x}$ that maximises the $\mathcal{P}(\mathbf{x}|\mathbf{y})$, i.e. the state that minimises the negative log-likelihood:

$$-\ln\left[\mathcal{P}(\mathbf{x}|\mathbf{y})\right] \propto J(\mathbf{x}) \equiv \frac{1}{2}(\mathbf{x}-\mathbf{x}_b)^{\mathsf{T}}\mathbf{B}^{-1}(\mathbf{x}-\mathbf{x}_b) + \frac{1}{2}(\mathbf{y}-H(\mathbf{x}))^{\mathsf{T}}\mathbf{R}^{-1}(\mathbf{y}-H(\mathbf{x})) \tag{4}$$

This likelihood is proportional to the cost function $J(\mathbf{x})$. The term $\mathcal{P}(\mathbf{y})$ can be ignored as it is a constant that does not depend on $\mathbf{x}$.

The gradient of the cost function, ∇J and its linearisation $\nabla \mathcal{J}$, then follow from this expression as:

$$\nabla J(\mathbf{x}) \approx \nabla \mathcal{J}(x) = \frac{\mathrm{d}}{\mathrm{d}\mathbf{x}} \mathcal{J}(\mathbf{x}) = \mathbf{B}^{-1}(\mathbf{x} - \mathbf{x}_b) - \mathbf{H}^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{y} - \mathbf{H}) \tag{5}$$

where $\mathbf{H}$ is the Jacobian matrix of the observation operator $H(\mathbf{x})$, as seen in the Taylor expansion of $H(\mathbf{x})$ around $\mathbf{x} = \mathbf{x}_b$:

$$H(\mathbf{x}) = H(\mathbf{x}_b) + \mathbf{H}(\mathbf{x} - \mathbf{x}_b) + \mathcal{O}\left((\mathbf{x} - \mathbf{x}_b)^2\right) \tag{6}$$

The final analysis is given by the solution of

$$\nabla \mathcal{J}(\mathbf{x}_a) = 0 \tag{7}$$

with $\mathbf{x}_a = \mathbf{x}_b + \delta\mathbf{x}$

The time-dependence of the model state $\mathbf{x}$ is not explicitly included in 3DVar, instead the algorithm is applied in so-called assimilation windows. Such a window spans a certain period around the time of the model state $\mathbf{x}_b$, and the observations within the window are considered to be coincidental with $\mathbf{x}_b$. The analysis state $\mathbf{x}_a$ obtained from the 3DVar minimisation is then used as the new initial state for advancing the model to the next assimilation window.

1.1 FGAT approach

The time-dependence can be made explicit by extending the cost function in the following way:

$$\mathcal{J}(\mathbf{x}_0) = \frac{1}{2}(\mathbf{x}_0 - \mathbf{x}_b)^{\mathrm{T}} \mathbf{B}^{-1}(\mathbf{x}_0 - \mathbf{x}_b) + \frac{1}{2}\sum_{i=0}^{N}(\mathbf{y}_i - H_i(\mathbf{x}_i))^{\mathrm{T}} \mathbf{R}_i^{-1}(\mathbf{y}_i - H_i(\mathbf{x}_i)) \tag{8}$$

with $\mathbf{x}_0$ and $\mathbf{x}_b$ the model state and background state at the start of the the assimilation window and the $\mathbf{x}_i, y_i, R_i$ the model state, observation vector, and covariance matrix at various timesteps i within the window. Minimising this cost function requires the formulation of a tangent linear model operator and an adjoint model operator, which would correspond to the 4DVar scheme.

The 3DVar FGAT (First Guess at Appropriate Time) scheme can be used to obtain from some of the benefits of 4DVar, but without the additional complexity of the tangent linear and adjoint model operators. In 3DVar FGAT the innovation $\mathbf{d}_i$ is approximated as follows:

$$\mathbf{y}_i - H_i(\mathbf{x}_i) \approx \mathbf{y}_i - H_i(\mathbf{x}_{b,i}) - \mathbf{H}_i(\mathbf{x}_0 - \mathbf{x}_b) = \mathbf{d}_i - \mathbf{H}_i(\mathbf{x}_0 - \mathbf{x}_b) \tag{9}$$

with

$$\mathbf{d}_i = \mathbf{y}_i - H_i(\mathbf{x}_{b,i}) \tag{10}$$

the innovation. Only the initial comparison between the observations and the background is performed at the correct time *i* and the minimisation itself is identical to 3DVar. The background values at the correct time $\mathbf{x}_{b,i}$ can be straightforwardly obtained from the model by advancing it once through the assimilation window to obtain the innovations $\mathbf{d}_i$.

In practice the innovations $\mathbf{d}_i$ are often used as the input for assimilation software rather than the observations $\mathbf{y}_i$ themselves, this enables a great level of flexibility in how the innovations are calculated and separates the often complicated processing of observational data from the main assimilation code. In that case we can consider:

$$\mathbf{d} = (\mathbf{d}_0, \mathbf{d}_1, \dots, \mathbf{d}_N) \tag{11}$$

and drop the subscript *i*, so that the cost function becomes:

$$\mathcal{J}(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}_b)^{\mathsf{T}} \mathbf{B}^{-1}(\mathbf{x} - \mathbf{x}_b) + \frac{1}{2}(\mathbf{d} - \mathbf{H}(\mathbf{x} - \mathbf{x}_b))^{\mathsf{T}} \mathbf{R}^{-1}(\mathbf{d} - \mathbf{H}(\mathbf{x} - \mathbf{x}_b)) \tag{12}$$

Also in the case of OceanVar, the input data are the innovations.

2 Background Error Covariance Matrix

In order to avoid calculating the full covariance matrix **B**, OceanVar shapes **B** using a control vector transformation CVT (**V**) that moves the minimization in a space where the new variables are uncorrelated and with unitary variance, as:

$$\mathbf{B} = \mathbf{V}\mathbf{V}^{\mathsf{T}} \tag{13}$$

As the covariance matrix is positive definite matrix, such a V, with rank equal to the rank of B exists. In OceanVar the transformation from $\delta\mathbf{x}$ in physical space to $\mathbf{v}$ in control space reads:

$$\delta\mathbf{x} = \mathbf{x} - \mathbf{x}_b = \mathbf{V}\mathbf{v} \qquad \mathbf{v} = \mathbf{V}^{+}\delta\mathbf{x} \tag{14}$$

where the superscript "+" indicates the generalized inverse. This transformation simplifies the cost function of Eq. (12) and its gradient to:

$$\mathcal{J}(\mathbf{v}) = \frac{1}{2}\mathbf{v}^{\mathsf{T}}\mathbf{v} + \frac{1}{2}\left(\mathbf{d} - \mathbf{H}\mathbf{V}\mathbf{v}\right)^{\mathsf{T}}\mathbf{R}^{-1}\left(\mathbf{d} - \mathbf{H}\mathbf{V}\mathbf{v}\right) \tag{15}$$

$$\nabla\mathcal{J}(\mathbf{v}) = \mathbf{v} - \mathbf{V}^{\mathsf{T}}\mathbf{H}^{\mathsf{T}}\mathbf{R}^{-1}\left(\mathbf{d} - \mathbf{H}\mathbf{V}\mathbf{v}\right) \tag{16}$$

In this new space only the definition of $\mathbf{V}$ is required. However, the background covariance matrix B may be hard to determine and extremely large, $\mathbf{V}$ is therefore approximated [WVA03]. In OceanVar the matrix $\mathbf{V}$ is approximated as [DP08]:

$$\mathbf{V} \approx \mathbf{V}_D\mathbf{V}_{(u,v)}\mathbf{V}_\eta\mathbf{V}_H\mathbf{V}_V. \tag{17}$$

From right to left $\mathbf{V}_V$ defines the vertical covariance, $\mathbf{V}_H$ projects the vertical increments onto the horizontal space, $\mathbf{V}_\eta$ is the balance operator, and $\mathbf{V}_{(u,v)}$ forces a geostrophic balance between temperature, salinity and the velocity components. Finally, $\mathbf{V}_D$ is a divergence-damping operator avoiding spurious currents close to the coast in the presence of complex coastlines.

2.1 Vertical Covariances

The vertical transformation operator $\mathbf{V}_v$ has the form:

$$\mathbf{V}_V = \mathbf{S}_c\mathbf{\Lambda}_c^{1/2}, \tag{18}$$

where columns of $\mathbf{S}_c$ contain eigenvectors and $\mathbf{\Lambda}_c$ is a diagonal matrix with eigenvalues of multivariate Empirical Orthogonal Functions (EOFs).

2.1.1 How to construct EOFs

Considering a model with l vertical levels, the state vector $\mathbf{x}$ for a water column containing temperature T, salinity S and sea surface height η, has a length of $n = 2l + 1$ and can be expressed as:

$$\mathbf{x} = (T_0, \ldots T_l, S_0, \ldots, S_l, \eta). \tag{19}$$

Using state vectors at the same location but at m different times (e.g. from a long model integration), it is possible to construct a matrix:

$$\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_m) \tag{20}$$

and its anomaly:

$$\mathbf{X}' = \mathbf{X} - \mathbb{I}\frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i \tag{21}$$

Using the anomaly matrix, the covariance of $\mathbf{x}$ can be expressed as:

$$\mathrm{cov}(\mathbf{x},\mathbf{x}) = \frac{1}{m}\mathbf{X}'^{\mathrm{T}}\mathbf{X}' \tag{22}$$

Because of the relationship between eigenvalue and singular value decomposition (SVD), the eigenvalues of $\mathbf{X}'^{\mathrm{T}}\mathbf{X}'$ are the square of the singular values of $\mathbf{X}'$. So the eigenvalue decomposition of the covariance matrix can be computed using:

$$\underset{[m\times n]}{\mathbf{X}'} = \underset{[m\times k]}{\mathbf{U}}\ \underset{[k\times k]}{\mathbf{S}}\ \underset{[k\times n]}{\mathbf{W}^{\mathrm{T}}} \tag{23}$$

with $\mathbf{U}$ and $\mathbf{W}$ unitary matrices, $\mathbf{S} = \mathbf{S}^{\mathrm{T}}$ a diagonal matrix with the singular values of $\mathbf{X}'$ and $k \leq n$ the number of retained values. As the singular values are ordered from most to least significant, the result can be straightforwardly filtered by truncating after k modes.

The covariance of Eq. (22) can then be written as:

$$\begin{aligned}\mathrm{cov}(\mathbf{x},\mathbf{x}) &= \frac{1}{m}\mathbf{WS}\left(\mathbf{U}^{\mathrm{T}}\mathbf{U}\right)\mathbf{SW}^{\mathrm{T}} \\ &= \frac{1}{m}\left(\mathbf{WS}\right)\left(\mathbf{WS}\right)^{\mathrm{T}}.\end{aligned} \tag{24}$$

For $k = n$ the eigenvectors of the covariance are equal to $\mathbf{W}/\sqrt{m}$ and the corresponding eigenvalues are $\mathrm{diag}(\mathbf{S})$. The vertical covariance operator $\mathbf{V}_V$ of Eq. (17) is then given by:

$$\mathbf{V}_V = \frac{1}{\sqrt{m}}(\mathbf{WS})^{\mathrm{T}} \tag{25}$$

The implementation in OceanVar uses an internal mapping from the model grid to the EOFs, allowing to use independent EOFs for each location or by region. Typically the EOFs are also conditioned by season.

2.2 Horizontal Covariances

To project the vertical increments onto the horizontal space, $\mathbf{V}_H$ has the form of a diffusive operator:

$$\mathbf{V}_H = \nabla_H(k_c \nabla_H C) \tag{26}$$

Where ∇_H is the horizontal differential operator, k_c is the spatially variable diffusivity coefficient corresponding to horizontal correlfation lengths, and C is a generic variable being assimilated. The operator is implemented in an implicit form. Assuming a gaussian solution ([WC01]), the relation between k_c and the horizontal correlation radius is:

$$R_H = \sqrt{(2k_c \Delta t)}, \tag{27}$$

where R_H, in meters, is the horizontal correlation radius, and Δt is the time-step using to integrate the diffusion equation. Particular care has to be taken to avoid connecting different regions across land. In OceanVar this is implemented by an ad-hoc replacement of any domain boundaries and land points by a series of virtual points that correspond to several correlation lengths. This buffer of virtual points ensures that sea points are not affected by the presence of land or domain boundaries. The virtual points are removed after every iteration. The present formulation of OceanVar allows the possibility to compute correction also for the velocity field.

The $\mathbf{V}_{(u,v)}$ operator calculates the baroclinic parts of the velocity components from the usual dynamic height formula, assuming geostrophic balance (see [DP08] for details). The advantage of the geostrophic assumption is that it requires only a small computational effort, but the disadvantage is that it is not valid at the Equator and may produce velocity vectors orthogonal to the coast. Enforcing the zero-boundary condition for the velocity component perpendicular to the coast the divergence component of the velocity field may become unrealistically large. Therefore, the divergence damping operator $\mathbf{V}_D$ is implemented to damp velocity divergence near coasts, while maintaining the vorticity part unchanged. Detail on the implementation of the divergence damping operator are provided in [DP08].

2.3 Balance Operators

2.3.1 Dynamic Height $\mathbf{V}_{DH}$

The simplest operator that covaries temperature and salinity with SLA is the dynamic height operator $\mathbf{V}_{DH}$ via the Local Hydrostatic Adjustment (LHA) approximation described in [Sto+10]. This scheme uses the vertical integration of density increments $\delta\rho(T,S)$ to calculate the increment in the steric sea level $\delta\eta$:

$$\rho_0 g\delta\eta + \int_{-h_b}^{0} g\delta\rho(T,S)\mathrm{d}z = \delta p_b \tag{28}$$

with ρ_0 the reference density at the surface, g the gravitational acceleration and δp_b the pressure increment at the bottom. Rather than using the bottom depth h_b, the integral can be performed starting from some depth $h^* < h_b$ using the assumption that horizontal currents at h^* are practically zero and therefore δp_{h^*} vanishes. This leads to a linearised observation operator:

$$\mathbf{H}(\mathbf{x}-\mathbf{x}_b) = \frac{1}{\rho_0}\int_{-h^*}^{0}\delta\rho(\mathbf{x}-\mathbf{x}_b)\mathrm{d}z \tag{29}$$

2.3.2 Barotropic Operator $\mathbf{V}_{\eta}$

To relax the geostrophic assumption, the OceanVar allows the application of a more complex linear $\mathbf{V}_{\eta}$ which gives the steady state results of a barotropic model forced by buoyancy anomalies induced by the temperature and salinity increments ([DP08]). The barotropic model equations, discretized in time by the semi-implicit scheme ([KR71]), are

$$\frac{U^{n+1}-U^{n-1}}{2\Delta t} - fV^n = -gH\frac{\partial\eta^*}{\partial x} - \int_{-H}^{0}\left[\int_{-z}^{0}\frac{\partial(\delta b)}{\partial x}dz'\right]dz + \gamma\nabla^2 U^* \tag{30}$$

$$\frac{V^{n+1}-V^{n-1}}{2\Delta t} - fU^n = -gH\frac{\partial\eta^*}{\partial y} - \int_{-H}^{0}\left[\int_{-z}^{0}\frac{\partial(\delta b)}{\partial y}dz'\right]dz + \gamma\nabla^2 V^* \tag{31}$$

$$\frac{\eta^{n+1}-\eta^{n1}}{2\Delta t} + \left(\frac{\partial U^*}{\partial x} + \frac{\partial V^*}{\partial y}\right) = 0 \tag{32}$$

where U and V are vertically integrated velocity components, f is the Coriolis parameter, g acceleration due to gravity, H the bottom depth,η the surface elevation, δb the buoyancy anomaly, and γ is the coefficient of horizontal viscosity.

The superscripts indicate the time step relative to *n*, and the superscript "*" indicates the weighted average between two timesteps. A more detailed description of the barotropic model and its discretization can be found in [DP08]. In the present version the barotropic model assumes closed lateral boundaries, thus the possibility to be used as balance operator needs to be carefully evaluated. In the future the barotropic model will be developed also with open boundary conditions. The buoyancy forcing term is defined as:

$$\delta b = g\left(\frac{\delta\rho}{\rho_0}\right) \tag{33}$$

and the density perturbation $\delta\rho$ is estimated by the linear equation:

$$\delta\rho = \alpha\delta T - \beta\delta S \tag{34}$$

Expansion and contraction coefficients (α and β)can be assumed space independent and prescribed through namelist or spatially variable and estimated linearizing the equation of state around a user defined background temperature (T_b) and salinity field (S_b):

$$\alpha = \left.\frac{\partial\rho}{\partial T}\right|_{T_b,S_b} \tag{35}$$

$$\beta = \left.\frac{\partial\rho}{\partial S}\right|_{T_b,S_b} \tag{36}$$

In the latter case the coefficients are read from an external input file. The aforementioned sea-level operators can replace or complement the cross-covariance between temperature, salinity and sea-level provided by the EOFs. The choice of sea level operator has consequences on the velocity operator. $\mathbf{V}_{(u,v)}$ computes the velocity correction assuming geostrophic balance under Boussinesq and incompressible approximation:

$$fu_g(z) = -\frac{1}{\rho_0}\frac{\partial p}{\partial y} \tag{37}$$

$$fv_g(z) = -\frac{1}{\rho_0}\frac{\partial p}{\partial x} \tag{38}$$

Decomposing the pressure p at any level z as:

$$p(z) = p_{atm} + g\,\rho\left(\eta + \int_{-z}^{0}\frac{\delta\rho}{\rho_0}dz\right) \tag{39}$$

where p_{atm} is the atmospheric pressure, g is the effective gravity, η is the free surface elevation and $\delta\rho$ is the density departure from a reference state ρ_0. Neglecting the atmospheric pressure, and rewriting the hydrostatic term as buoyancy term, the geostrophic velocities become:

$$v_{gs}(z) = \frac{g}{f}\frac{\partial\zeta}{\partial x} + \frac{g}{f\rho_0}\int_{-z}^{0}\frac{\partial\delta\rho(z)}{\partial x}dz \tag{40}$$

$$u_{gs}(z) = -\frac{g}{f}\frac{\partial\zeta}{\partial y} - \frac{g}{f\rho_0}\int_{-z}^{0}\frac{\partial\delta\rho(z)}{\partial y}dz \tag{41}$$

The first term on the r.h.s is vertically homogeneous and accounts for the horizontal gradient of the sea surface elevation. The second term of the r.h.s is vertically varying and represents the horizontal pressure gradients due to density variations. When adopting the barotropic model $\mathbf{V}_\eta$, no assumptions are needed except the linearization, thus velocity corrections can be computed from surface to the ocean bottom. The lower integral limit varies in space and reaches -H(x,y), the ocean floor. On the other hand, when $\mathbf{V}_{DH}$ is used: it is assumed that the horizontal pressure-gradient force vanishes at a certain depth (“level of no motion”). In the equations for the geostrophic correction η is then substituted with η^D: the lower limit of the buoyancy vertical integral is spatially constant and set equal to the level of no motion. While in case of the barotropic model, velocity corrections are provided for the entire water column, using dynamic height, the velocity corrections are provided from surface to the level of no motion.

3 Observation Error Covariance Matrix

The observation error is usually defined as the difference between the observation array $\mathbf{y}^o$ and the corresponding model counterpart calculated over the (unknown) *true* ocean state $\mathbf{x}^t$:

$$\mathbf{y}^o = \mathcal{H}(\mathbf{x}^t) + \epsilon^o \longrightarrow \epsilon^o = \underbrace{(\mathbf{y}^o - \mathbf{y}^t)}_{\epsilon^m} - \underbrace{(\mathcal{H}(\mathbf{x}^t) - \mathbf{y}^t)}_{\epsilon^r} \tag{42}$$

where $\mathcal{H}$ is the non linear observation operator, ϵ^m labels the instrumental or measurement error (distance of the actual value from the true state) while ϵ^r gathers the different components of the representativeness errors, from inaccurancies in the $\mathcal{H}$ to physical processes that are not considered in the model or that are not resolved due to the grid spacing. Under the assumption of error

unbiased $\langle \epsilon^0 \rangle = 0$ and of error independency between ϵ^m and ϵ^r, the error covariance matrix **R** can be constructed as the sum of two terms that can be estimated independently :

$$\mathbf{R} = \langle \epsilon^o \epsilon^o \rangle \approx \langle \epsilon^m \epsilon^m \rangle + \langle \epsilon^r \epsilon^r \rangle = \mathbf{R}^m + \mathbf{R}^r$$

Assuming that the errors associated to different observations are uncorrelated, the structure of **R** greatly simplifies in a diagonal matrix,

$$\mathbf{R} = \mathrm{DIAG}\{(\sigma_1^m)^2 + (\sigma_1^r)^2; (\sigma_2^m)^2 + (\sigma_2^r)^2; ...; (\sigma_N^m)^2 + (\sigma_N^r)^2\} \tag{43}$$

where N is the total number of observations. This hypothesis is common to the most of current global/regional forecasting/reanalysis systems and it is generally correct when observations are sampled relatively far in time (say few hours to avoid cross-correlation term in $\mathbf{R}^m$) or sparse with respect grid resolution (to not include off-diagonal elements in $\mathbf{R}^r$). The approximation in Eq. (43) allows to represent the **R** matrix simply as a multiplicative factor for each misfit thus avoiding the need to invert the full matrix. The observation penalty term can be written as:

$$J_{\mathbf{R}} \approx \sum_{ij} (\mathbf{HVv} - \mathbf{d})_i \mathbf{R}_{ij}^{-1} (\mathbf{HVv} - \mathbf{d})_j \rightarrow \sum_i \underbrace{[(\mathbf{HVv} - \mathbf{d})_i / (\mathbf{R}_{ii}^{1/2})]^2}_{\texttt{obs\%amo}} \tag{44}$$

where $J_{\mathbf{R}}$ is simply obtained once the vector `obs%amo` is evaluated for each observation, being simply the sum of the squared elements of `obs%amo` that is calculated in costf.F90, there is no distinction among types of observations that are gathered together in `ctl%f_o`, the first condition consider the Huber-norm extension, if not, the standard formulation is used:

Listing 1: costf.F90

```
 IF ( (ctl%isave(30) .GT. huberqc%iter) .AND.  &
   (huberqc%any) ) THEN
   CALL huber_costf(ctl%f_o)
ELSE
       ctl%f_o = 0.5_r8 * DOT_PRODUCT( obs%amo, obs%amo )
ENDIF
```

The vector `obs%amo` is first defined in resid.F90. All the observation-related information is stored in `obs%` datastructures using the fixed order: sea-level-anomaly (SLA) observations, temperature and salinity by argo, temperature by xbt, drifters' position, velocity observations by drifters, velocity observation by gliders, sst observations. For example, for SLA observations, `obs%amo` reads:

Listing 2: resid.F90

```
        do i=1,sla%no
        !! satellite observations of sla,
            if(sla%flc(i).eq.1)then
                k = k + 1
                obs%inc(k) = sla%inc(i)    !! (HVv)
                obs%amo(k) = ( obs%inc(k) - obs%res(k) ) / obs%err(k) !! (HVv-d)/sqrt(R
                    )
            endif
        enddo
```

where *i* runs over the number of sla observations and `sla%flc` is the quality flag of each observation, i.e. the operation is applied only to good ones. The diagonal term $\mathbf{R}_{ii}^{1/2}$ corresponds to the standard deviation that is represented by the observation error `obs%err`. The array `obs%inc` considers the increments at the observation location:

$$\texttt{obs\%amo} = (\underbrace{[\mathbf{HVv}]}_{\texttt{obs\%inc}} - \mathbf{d})_i / (\mathbf{R}_{ii}^{1/2}) \tag{45}$$

and are generated obtained by applying the observation operator **H** to the increments (**Vv**) in the model space in Eq. (45).

3.1 Observation operator H

The approximations to the linearised observation operator **H** are defined in obsop.F90 and can differ according to the observation type and physical hypothesis:

Listing 3: obsop.F90

```
!  Satellite observations of SLA
   CALL obs_sla

! ---
! Observations by ARGO floats
   CALL obs_arg

! ---
! Observations by XBT profiles
   CALL obs_xbt

! ---
! Observations by gliders
   CALL obs_gld

[...]
```

The order of application follows the order of the data in the `obs` type. For most of the observation types **H** is approximated by an interpolation algorithm that calculates the model fields at the observation location. For insitu data, this approximation to **H** reads:

Listing 4: obs_argo.F90

```
DO kk = 1,arg%no
    IF ( ( arg%flc(kk) .EQ. 1 .OR. arg%flg(kk) .EQ. 1 ) .AND. arg%par(kk) .EQ. 1 )
        THEN
       i = arg%ib(kk)
       j = arg%jb(kk)
       k = arg%kb(kk)
       arg%inc(kk) = arg%pq1(kk) * grd%tem(i  ,j  ,k  ) +       &
                     arg%pq2(kk) * grd%tem(i+1,j  ,k  ) +       &
                     arg%pq3(kk) * grd%tem(i  ,j+1,k  ) +       &
                     arg%pq4(kk) * grd%tem(i+1,j+1,k  ) +       &
                     arg%pq5(kk) * grd%tem(i  ,j  ,k+1) +       &
                     arg%pq6(kk) * grd%tem(i+1,j  ,k+1) +       &
                     arg%pq7(kk) * grd%tem(i  ,j+1,k+1) +       &
                     arg%pq8(kk) * grd%tem(i+1,j+1,k+1)
    ELSEIF ( (arg%flc(kk) .EQ. 1 .OR. arg%flg(kk) .EQ. 1) .AND. arg%par(kk) .EQ. 2 )
        THEN
       i = arg%ib(kk)
       j = arg%jb(kk)
       k = arg%kb(kk)
       arg%inc(kk) = arg%pq1(kk) * grd%sal(i  ,j  ,k  ) +       &
                     arg%pq2(kk) * grd%sal(i+1,j  ,k  ) +       &
                     arg%pq3(kk) * grd%sal(i  ,j+1,k  ) +       &
                     arg%pq4(kk) * grd%sal(i+1,j+1,k  ) +       &
                     arg%pq5(kk) * grd%sal(i  ,j  ,k+1) +       &
                     arg%pq6(kk) * grd%sal(i+1,j  ,k+1) +       &
                     arg%pq7(kk) * grd%sal(i  ,j+1,k+1) +       &
                     arg%pq8(kk) * grd%sal(i+1,j+1,k+1)
    ENDIF
 ENDDO
```

where `arg%flg, arg%flc` take into account whether the single measurement *kk* is correct and if it refers to the correct grid respectively, while `arg%par` distinguishes temperature measurements from salinity ones. Its adjoint is simply the transpose of the interpolation. The approximation of the **H** operator can differ when the assimilation of trajectories is involved. Temperature and salinity increments (`grd%tem`, `grd%sal`) are interpolated at the observation location (`arg%inc`) through a horizontal bilinear interpolation plus a linear vertical one with *k* = `arg%kb` corresponding to the shallowest vertical level. In these expressions, `arg%ib`, `arg%jb` label the grid indexes *i*, *j* of the observation position while `arg%pq` contains the horizontal interpolation weights. In the present version the SLA case is treated exactly the same as the T S counterparts, the **H**

operator is approximated by using a simple interpolator and the balance/unbalance treatment is performed in the **B** matrix. Alternatively, the balance operator is included in the approximation to **H** to ease the minimisation, in case the final SLA increments are not used [Sto+10].

3.1.1 Interpolation operator

The calculation of the interpolation weights is performed in the int_part.F90, following the order of observation types included in the obs structure

Listing 5: int_part.F90

```
! ----
! Load SLA observations
   CALL int_par_sla

! ----
! Load ARGO observations
   CALL int_par_arg

! ----
! Load XBT observations
   CALL int_par_xbt

! ----
! Load glider observations
   CALL int_par_gld
   [...]
```

Calculation of the interpolation parameters is performed in the corresponding get_obs_* files, and are stored in the

Listing 6: get_obs_arg.F90

```
 ---
! Horizontal interpolation PARAMETERs
      CALL int_obs_hor ( arg%no, arg%lat, arg%lon, arg%flc, arg%eve, arg%ib, arg%jb,
          arg%pb, arg%qb)
! ---
! Undefine masked for multigrid
      DO k = 1,arg%no
         IF ( arg%flc(k) .EQ. 1 ) THEN
            i1 = arg%ib(k)
            j1 = arg%jb(k)
            idep = arg%kb(k)+1
            msk4 = grd%msk(i1,j1,idep) + grd%msk(i1+1,j1,idep) + grd%msk(i1,j1+1,idep)
                + grd%msk(i1+1,j1+1,idep)
            IF ( msk4 .LT. 1. ) THEN
               arg%flc(k) = 0
               arg%eve(k) = 3
```

```
            ENDIF
         ENDIF
      ENDDO

! ---
! Horizontal interpolation parameters for each masked grid
      DO k = 1,arg%no
         IF ( arg%flc(k) .EQ. 1 ) THEN

            klev = arg%kb(k)
            CALL int_obs_pq( arg%ib(k), arg%jb(k), klev, arg%pb(k), arg%qb(k),  &
                             arg%pq1(k), arg%pq2(k), arg%pq3(k), arg%pq4(k))
            klev = arg%kb(k) + 1
            CALL int_obs_pq( arg%ib(k), arg%jb(k), klev, arg%pb(k), arg%qb(k),  &
                             arg%pq5(k), arg%pq6(k), arg%pq7(k), arg%pq8(k))
            r1=arg%rb(k)
            arg%pq1(k) = (1.-r1) * arg%pq1(k)
            arg%pq2(k) = (1.-r1) * arg%pq2(k)
            arg%pq3(k) = (1.-r1) * arg%pq3(k)
            arg%pq4(k) = (1.-r1) * arg%pq4(k)
            arg%pq5(k) =     r1  * arg%pq5(k)
            arg%pq6(k) =     r1  * arg%pq6(k)
            arg%pq7(k) =     r1  * arg%pq7(k)
            arg%pq8(k) =     r1  * arg%pq8(k)
         ELSE
            arg%pq1(k) = 0.0_r8
            arg%pq2(k) = 0.0_r8
            arg%pq3(k) = 0.0_r8
            arg%pq4(k) = 0.0_r8
            arg%pq5(k) = 0.0_r8
            arg%pq6(k) = 0.0_r8
            arg%pq7(k) = 0.0_r8
            arg%pq8(k) = 0.0_r8
         ENDIF
      ENDDO
```

3.2 Gradient computation of the observation cost function term

Starting from Eq. 44, the gradient of the observation penalty term can be written as :

$$\nabla J_R \approx \underbrace{\mathbf{V}^T\mathbf{H}^T\,(\sqrt{\mathbf{R}})^{-1}\,\underbrace{(\mathbf{HVv}-\mathbf{d})/\sqrt{\mathbf{R}}}_{obs\%amo}}_{obs\%gra} \tag{46}$$

in case of diagonal $\mathbf{R}$, where obs%gra corresponds to $\mathbf{R}^{-1}(\mathbf{HVv}-\mathbf{d})$ and is calculated dividing obs%amo by the standard deviation obs%err in :

Listing 7: res_inc.F90

```
!First compute without considering huber norm
```

```
   obs%gra(:) = obs%amo(:) / obs%err(:)

! If necessary overwrite obs%gra considering huber norm

! SLA observations
   jstart = 1
   jend   = sla%nc
   IF ( huberqc%sla .AND. (ctl%isave(30) .GT. huberqc%iter) ) CALL huber_resinc(jstart,
       jend)

  [...]
```

All these information are store in the obs structure. In order to be consistent with the rest of the arrays, the functions associated to different observation types must be called in the original order. The calculation of ∇J proceeds with the adjoint of the observation operator $\mathbf{H}^T$The array obs%amo in Equation 46 is moved in the model space with the help of the adjoint of the observation operator $\mathbf{H}^T$ in obsop_ad.F90 whose structure is similar to obsop.F90 and reads:

Listing 8: obsop_ad.F90

```
 ! ---
   obs%k = 0

! ---
! Satellite observations of SLA
   CALL obs_sla_ad

! ---
! ARGO observations
   CALL obs_arg_ad

[...]
```

obs%k is set to zero in obsop_ad.F90. $\mathbf{H}^T$ is the adjoint of an interpolation routine that goes to populate the incremental variable in the model space, for example for each observation the grd%eta_ad (in the model space) is populated in the four closest points to the observation position, by interpolating obs%gra(obs%k).

Listing 9: obs_sla_ad.F90

```
 ! ---

   DO k=1,sla%no
      IF ( sla%flc(k) .EQ. 1 ) THEN
         obs%k = obs%k + 1
         i = sla%ib(k)
         j = sla%jb(k)
[...]
```

```
        grd%eta_ad(i  ,j  ) = grd%eta_ad(i  ,j  ) + sla%pq1(k) * obs%gra(obs%k)
         grd%eta_ad(i+1,j  ) = grd%eta_ad(i+1,j  ) + sla%pq2(k) * obs%gra(obs%k)
         grd%eta_ad(i  ,j+1) = grd%eta_ad(i  ,j+1) + sla%pq3(k) * obs%gra(obs%k)
         grd%eta_ad(i+1,j+1) = grd%eta_ad(i+1,j+1) + sla%pq4(k) * obs%gra(obs%k)
[...]
      ENDIF
ENDDO
```

At this stage the grd%*_ad array is different from zero only in the proximity of observation locations. $\mathbf{V}^T$ is then applied to grd%*_ad to generate the full increments in the control vector space. The expression of the observation penalty term and its gradient can be modified to reduce the impact of the large tails of the departure distribution in the observation penalty term(see section 3.3).

3.3 Huber-norm correction of observation cost function

OceanVar allows the use of a Huber-norm pdf in the cost function to better represent the observation penalty term. Under the gaussian hypothesis, large innovations (populating the tails of the gaussian distribution) are associated to large weights and the minimisation goes primarily in the direction of reducing such innovations rather than reducing the ones close to zero. However the initial misfits distribution does not follow a gaussian distribution,i.e. such tails can be largely populated, thus spoiling the minimization of the departures that are close to zero. Following [Sto16], the initial misfits distribution ca be approximated with a Huber-norm pdf to give less impact to such tails. The initial gaussian approximation in the cost function reads:

$$\left[-\frac{1}{2}(\mathbf{w}^2)\right] \qquad \text{with} \qquad \mathbf{w} = \mathbf{R}^{-1/2}(\mathbf{HVv} - \mathbf{d}) \tag{47}$$

where **w** is our obs%amo term, being the vector of innovations normalized by the observation error standard deviation (or simply normalized innovations). Equation 47 is substituted with a new one the reads:

$$\begin{aligned} \left[-\frac{1}{2}\mathbf{w}^2\right] \qquad & \text{if } |\mathbf{w}| \leq a \\ \left[\frac{1}{2}a^2 - a|\mathbf{w}|\right] \qquad & \text{if } a < |\mathbf{w}| \leq b \\ \left[-c|\mathbf{w}|^{1/2} + e\right] \qquad & |\mathbf{w}| > b \end{aligned} \tag{48}$$

Imposing the continuity of the corresponding PDF, i.e. $P(\mathbf{y}|\mathbf{x}) = k \exp[Eq.48]$ and its derivative yields $c = 2a\sqrt{b}$ and $e = (1/2)a^2 + ab$. k is the normalization factor

that reads $(2\pi)^{-N/2}|\mathbf{R}|^{-1/2}$, with N the number of observations and $|\mathbf{R}|$ the determinant. For the purpouse of minimization *k* is however a constant factor. In the following we show the structure used in the calculation of the cost function and corresponding correction apply to gradient too:

Listing 10: costf.F90

```
  ! ARGO observations
   jstart = sla%nc + 1
   jend   = sla%nc + arg%nc
   IF ( huberqc%arg ) THEN
      jo_arg = huberf(jstart,jend)
   ELSE
        [...]
    ENDIF
    [...]
 REAL (KIND=r8) FUNCTION huberf(jstart,jend)
       jo_cost = 0._r8

   DO jo = jstart,jend

      IF ( obs%amo(jo) .GT. ABS(obs%ahub(jo,1)) .AND.                          &
           obs%amo(jo) .LE. ABS(obs%ahub2(jo,1)) ) THEN

         zcst = obs%amo(jo)*obs%ahub(jo,1)-0.5_r8*obs%ahub(jo,1)*obs%ahub(jo,1)

      ELSEIF ( obs%amo(jo) .LT. -ABS(obs%ahub(jo,2)) .AND.                     &
               obs%amo(jo) .GE. -ABS(obs%ahub2(jo,2)) ) THEN
      [...]
        ENDIF

      jo_cost = jo_cost + zcst

   ENDDO

   huberf = jo_cost
END FUNCTION huberf
```

huberqc%arg is the flag that switch on/off the hubernorm correction for argo. The condition ctl%kiter *.gt*. iter_hubqc states that such corrections start after a number of iterations of *iter_hubqc* that is typically around 10 iterations, i.e. the first 10 iterations run without the huber-norm corrections. This strategy is used to avoid initial shocks and provide smooth corrections. Parameters a and b in 48 correspond to obs%ahub(jo,1) and obs%ahub(jo,2) that are usually calculated by fitting the initial mistits distribution with huber-norm function and stored in an external file. The corresponding gradient for Equation 48 is embedded in the file.

Listing 11: res_inc.F90

```
SUBROUTINE huber_resinc(jstart,jend)
```

```
[...]
 DO jo  =  jstart,jend

      IF ( obs%amo(jo)  .GT.  ABS(obs%ahub(jo,1)) .AND. obs%amo(jo)  .LE.  ABS(obs%
          ahub2(jo,1)) ) THEN

         obs%gra(jo) = obs%ahub(jo,1) / obs%err(jo)

      ELSEIF ( obs%amo(jo)  .LT.  -ABS(obs%ahub(jo,2)) .AND. obs%amo(jo)  .GE.  -ABS(
          obs%ahub2(jo,2)) ) THEN

         obs%gra(jo) = -obs%ahub(jo,2) / obs%err(jo)
[...]
```

4 Error definition

Observation errors ϵ_0 in Eq. 43 are gathered as function of the type of the observation in OceanVar. They can be explicitly constructed by summing up different components of the uncertainties corresponding to σ_m and/or σ_r or simply by constructing a unique *optimal* error $\epsilon^0 = \sigma^m_{\mathrm{Des}}$ evaluated using Deszroziers' relations (Deszroziers et al, 2005). The **R** matrix is populated soon after the reading of observations and interpolation parameter in oceanvar.F90 :

Listing 12: oceanvar.F90

```
! Get observations
         IF ( ktr .EQ. 1 .AND.  kts .EQ. 1 ) THEN
 [,..]
! ---
! Define interpolation parameters
         IF ( ktr .EQ. 1 .OR. drv%ratio(ktr) .NE. 1.0_r8 ) THEN
  [,..]
! ---
! Define observational errors
         IF ( ktr .EQ. 1 .AND.  .NOT. obserr%rd_err_ff ) THEN
            WRITE (drv%dia,*) ' --------------------------- '
            WRITE (drv%dia,*) ' Define observational errors '
            CALL obserrors
            CALL FLUSH(drv%dia)
         ENDIF
```

Subroutine obserrors assings the error at each observation, starting from SLA data then insitu, satellite sst and sss.

Listing 13: obserrors.F90

```
 IF ( sla%no .GT. 0 .AND. obs%sla .NE. 0 ) &
```

```
      CALL obserr_sla

   IF ( arg%no .GT. 0 .AND. obs%arg .NE. 0 ) &
      CALL obserr_arg

   IF ( xbt%no .GT. 0 .AND. obs%xbt .NE. 0 ) &
      CALL obserr_xbt
[...]
```

The overall error is finally multiplied by a time inflation coefficient. This coefficient increases the error in the case the observation is far from the center of the assimilation window. It considers the temporal distance (*td*) of the observation with respect to the center of the assimilation window and uses a gaussian weight with standard deviation being the assimilation window (ztime_weigth).

Listing 14: obserrors.F90

```
 REAL(KIND=r8) FUNCTION tim_dep_errfact(time)
[...]
  td=MIN(obserr%ztime_weigth,ABS(time))
   td2=td*td
   zt2=obserr%ztime_weigth*obserr%ztime_weigth
   tim_dep_errfact = 1._r8 / EXP( -td2 / zt2 )
```

4.1 Definition of sea level anomaly error

Sea level is assimilated through an anomaly formulation where global trends are removed by means of global or along-track bias removal. A proper correction of the global trend should include changes in the NEMO model that is not able to reproduce the global evolution of SSH: two third of global SSH trend come directly from effects that are not included in forced ocean simulation such as the melting of land-ice. The remaining one third comes from the global steric effect (warming and freshening of the water column) that however is not directly included in the SSH equation due to the Boussinesq approximation that tends to conserve volume rather than mass, and so do not properly represent global expansion or contraction.

The anomaly formulation of observed SSH $\delta\eta_o = \eta_o - \eta_{<o>}$ considers the difference of the observed sea level height and the observed mead dynamic topography $\eta_{<o>}$. The innovation in OceanVar can be therefore written as (Dobri-

cic,2005):

$$\begin{aligned} \delta\eta_o &= \mathcal{H}(\mathbf{x}) + \epsilon \\ \epsilon &= \delta\eta_o - \mathcal{H}(\mathbf{x}_b) - \mathbf{H}(\mathbf{x} - \mathbf{x}_b) = \underbrace{\delta\eta_o - \eta_f(\mathbf{x}_b) + \eta_{<f>}}_{\mathbf{d}} - \underbrace{\mathbf{H}(\mathbf{x} - \mathbf{x}_b)}_{\delta\eta_{balance}=\mathbf{HVv}} \end{aligned} \tag{49}$$

where $\eta_{<f>}$ is the model MDT while η_f labels the background sea level height and the total error in Eq. 49 is residual term multiplying the $\mathbf{R}^{-1}$. The corresponding variances in the matrix **R** include different sources of errors (error propagation) :

$$\sigma^2_{\mathrm{SLA}} = \sigma^2_m(\mathrm{SAT}) + \sigma^2_r(lon, lat) \tag{50}$$

where $\sigma^2_m(\mathrm{SAT})$ is the instrumental error that is specific for each satellite and is it assumed to be constant in time as the representativeness error $\sigma^2_r(lon, lat)$ that only varies in space. Representativeness error considers those processes that cannot be resolved by the model due to grid resolution or physics not modelled. Such errors are not uniform and depends mainly on the local Rossby deformation radius being larger in areas of strong eddy activity. Usually it is derived following the approach of [OS08; Sto+10] and can be tuned using a global inflation coefficient. In the present system it include also the error associated to the approximation of the observation operator (dynamic height in this case) and inaccuracies in the mean dynamic topography (spatially varying). Specifically the representativeness error can be activated with obserr%sla_hor_dep_err and read an external files in obserrors.F90:

Listing 15: obserrors.F90

```
   IF ( obserr%sla_hor_dep_err ) THEN

      ALLOCATE ( obserr%stde(1-grd%ias:grd%im+grd%iae,1-grd%jas:grd%jm+grd%jae), &
                          x2(1-grd%ias:grd%im+grd%iae,1-grd%jas:grd%jm+grd%jae) )

      stat = NF90_OPEN(drv%inpdir//'/'//obserr%sla_flname, NF90_NOWRITE, ncid)
      IF (stat /= NF90_NOERR) CALL netcdf_err(stat)
! Get dimensions
 [...]
    stat = nf90_inq_varid (ncid, 'stde', idvar)
      IF (stat /= NF90_NOERR) CALL netcdf_err(stat)
      stat = nf90_get_var (ncid,idvar,x2,start(1:2),count(1:2))
      obserr%stde(:,:) = DBLE(x2(:,:))
ENDIF
```

that is interpolated at the observation position in obs_err_sla.F90. The satellite dependent error, $\sigma^2_m(\mathrm{SAT})$, usually shared by the data provider, is added in obs_err_sla.F90 :

Listing 16: obs_err_sla.F90

```
   IF ( obserr%sla_sat_dep_err ) THEN
      DO i = 1,obserr%sla_sat_nu
         WHERE (sla%ksat .EQ. i ) sla%err = obserr%sla_sat_err(i)
      ENDDO
   ELSE
      sla%err(:) = obserr%sla_con_err
   ENDIF
```

In regional domain, the different contributions to the SLA error can be roughly considered to be spatially independent in the first approximation and a static unique term is used .

4.2 Definition of insitu error

The treatment of insitu error is constructed as the multiplication of an instrumental error and a spatial-varying factor representing the representativeness error. The instrumental uncertainty depends on the variable measured (Temperature or Salinity) and the depth of the observation. The depth-varying error is employed by switching on obserr%ts_ver_dep_err and providing a file with a vector of errors according to the depths

Listing 17: obserrors.F90

```
   IF (obserr%ts_ver_dep_err) THEN

      stat = NF90_OPEN(drv%inpdir//'/'//obserr%ts_flname, NF90_NOWRITE, ncid)
      IF (stat /= NF90_NOERR) CALL netcdf_err(stat)
[...]
      stat = nf90_inq_varid (ncid, 'tem', idvar)
      IF (stat /= NF90_NOERR) CALL netcdf_err(stat)
      stat = nf90_get_var (ncid, idvar, x1)
      IF (stat /= NF90_NOERR) CALL netcdf_err(stat)
      obserr%tem(:) = DBLE(x1(:))
      stat = nf90_inq_varid (ncid, 'sal', idvar)
      IF (stat /= NF90_NOERR) CALL netcdf_err(stat)
      stat = nf90_get_var (ncid, idvar, x1)
      IF (stat /= NF90_NOERR) CALL netcdf_err(stat)
      obserr%sal(:) = DBLE(x1(:))
      stat = nf90_inq_varid (ncid, 'depth', idvar)
      IF (stat /= NF90_NOERR) CALL netcdf_err(stat)
      stat = nf90_get_var (ncid, idvar, x1)
```

A multiplicative, spatially-varying factor (representativeness contribution) can be eventually provided

Listing 18: obserrors.F90

```
   stat = nf90_inq_varid (ncid, 'tem_fc', idvar)
     IF (stat /= NF90_NOERR) THEN
        WRITE (drv%dia,*)' Multiplication factor for t vertical error set to 1'
        obserr%tem_fc(:,:) = 1._r8
     ELSE
        stat = nf90_get_var (ncid,idvar,x2,start(1:2),count(1:2))
        obserr%tem_fc(:,:) = DBLE(x2(:,:))
     ENDIF
     stat = nf90_inq_varid (ncid, 'sal_fc', idvar)
     IF (stat /= NF90_NOERR) THEN
        WRITE (drv%dia,*)' Multiplication factor for s vertical error set to 1'
        obserr%sal_fc(:,:) = 1._r8
     ELSE
        stat = nf90_get_var (ncid,idvar,x2,start(1:2),count(1:2))
        obserr%sal_fc(:,:) = DBLE(x2(:,:))
     ENDIF
  [...]
```

exact interpolation in the observation position is performed in the associated obs_err_arg.F90 file. Instrumental error profile is nitially taken from Ingleby and Huddleston (2007).For temperature the maxima of the errors peaked at the mixed layer depth, ranging from 0.8 to 1.2 degrees Celsius. Usually salinity errors are located at the sea surface, ranging from 0.27 and 0.39 psu. In order to account for the representativeness error is scaled toby a multiplicative factor *insfac* ranging from 1 to 1.5 and depending on the type of instruments too

An approach that is generally used is to generate an optimal observation error $\sigma^2_{\mathrm{Des}}(ins, type, dept, time)$ based on the Desroziers' relation that is used to provide a better extimation of the error based on the *a-posteriori* statistics and depending also on time in order to have an error that correctly follow the seasonal variability of the mixed layer depth.

Part II

Getting Started

The code can be download using the command:

```
$ git clone -b main git@github.com:CMCC-Foundation/3DVAR_v0.1
↪  Oceanvar
```

The OceanVar directory is composed of sub-folder:

- *src*: where the source code is located
- *namelist*: where the namelist (list of options) is located
- *bin*: where the executable will be locate after the compilation of the source code
- *html*: where the reference manual is located
- *prep*: where some python pre-processing file are located
- *test_case*: where simple test case can be executed
- *test_suite*: where a test suite is located which may test automatically for bitwise reproducibility across MPI processes (Section **??**

Enter in the directory *src* with the command:

```
$ cd  src
```

Open the file *configuration.mk*:

```
$ vi configuration.mk
```

Change compiler and compilation options according to your need. Alternatively, one can specify a predefined architecture (which are defined in “src/Makefile") but for a general user this is not the case.

One should have the NetCDF library installed and include and link the libraries in the *EXTINC*, *EXTLIB*. One can find which paths to include/link by checking the commands `nc-config -all` and/or `nf-config -all`

Execute the file *link_in_work* with the command:

```
$ ./link_in_work
```

The command will create a new directory called *work* at the same level of the other directory listed before.

Enter in the work directory by typing:

```
$ cd ../work
```

Compile the code just typing:

```
$ make
```

At the end of the compilation procedure the message *../bin/var_3d is compiled*.

Enter in the *test_case* directory by tyoing:

```
$ cd ../test_case
```

Execute the code:

```
$ ./run_test.sh
```

OceanVar is installed, have fun!!!!

5 Prerequisites

The prerequisite for OceanVar2.0 are:

- `bash`
- `make`
- `git`
- A compiler able to compile Fortran77 and Fortran90
- `python3`
- Optionally, an MPI distribution if one wants to run in parallel e.g. `openMPI`, `MPICH` or `IntelMPI`
- Optionally, if one wants to use the testing suite to check for bitwise reproducibility across MPI processes (Section **??**)
  - `bash` version $>$ 4
  - a `python3` version which supports '*'/'star'-list unpacking (e.g. $>=$ 3.11.5)

6 Flowchart

The main driver of the oceanvar software is the file *oceanvar.F90*. In figure 1 the related flow chart is shown.

The cores of the software is the computation of the cost functions *(cost.F90)* and its minimization *(min_cfn.F90)*. In figure 2 and 3 the related flow chart are shown.

For further information the reader is referred to the documentation in the *html* directory

7 Namelist

The user may configure many part of the code by using the namelist that is necessary to the OceanVar to be executed. An example of namelist is found in the folder *namelist* and it is called *var_3d_nml*. The file is composed of several namelists and in the following section a detailed description of the namelist will be presented.

7.1 Define the general set-up

```
! Namelist runlst
! ---
!
!              Define the general set-up
!
!  flag_a    - character*128: flag for the analysis
!  sdat_f    - Starting date of the forecast
!  shou_f    - Starting hour of the forecast
!
! ---
&runlst
   flag_a    = "MFS_analysis"
   sdat_f    = YYYYMMDD
   shou_f    = HH
/
```

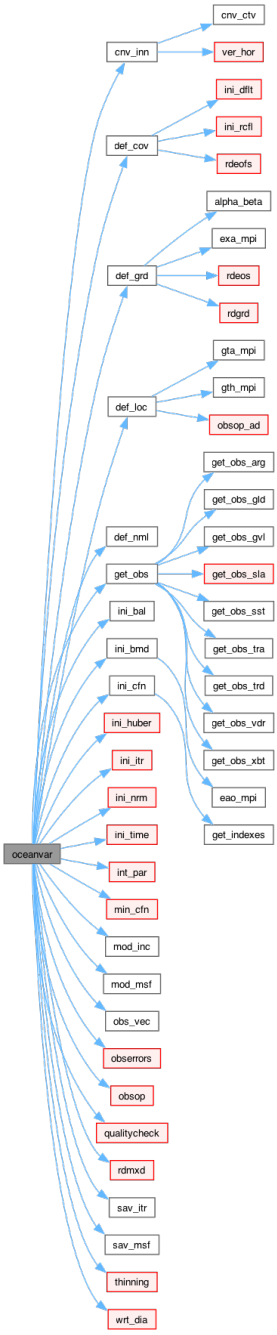

Figure 1: OceanVar flow chart.

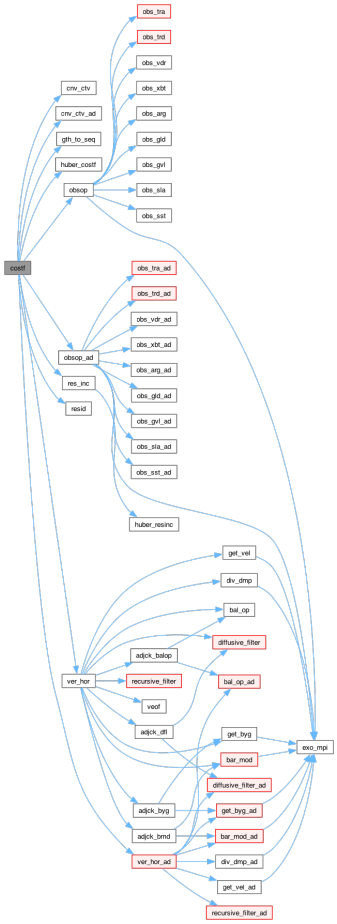

Figure 2: Cost function creation flow chart.

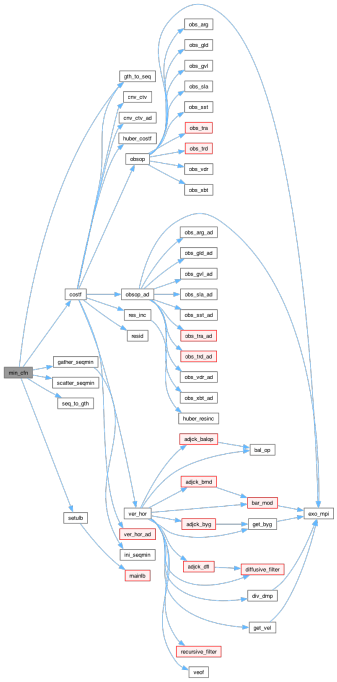

Figure 3: Minimization procedure flow chart.

- *flag_a* is the name of the experiment. It is define as character and it can not be longer than 128 characters.
- *sdat_f* is the starting date of the forecast after the analysis and it is define as integer. It is defined as year month day. First 4 digits are for the year, 2 for month and 2 for day.
- *shou_f* is the starting hour of the forecast after the analysis and it is define as a integer.

7.2 Outer loop set-up

```
! Namelist drvlst
! ---
!
!              Outer loop set-up
!
!  ntr      -  Number of outer iterations over grids
!  prj      -  0 = lat/lon regular grid
!  filter   -  Filter type:
```

```
!             1 = Recursive
!             2 = Diffusive
!             3 = None
!  grid    -  grid number in each iteration
!  ratio   -  Resolution ratio between the previous
!             and the current grid
!  mask    -  Mask type:
!              1 - no land
!              2 - 2d mask by coasts
!              3 - 3d mask by bottom topography
!  barmd   -  Run barotropic model on the grid
!  balmd   -  Simplified balance operator based on dynamic
 ↪  height
!  divda   -  Apply divergence damping on the grid
!  divdi   -  Initialise corrections by divergence damping
!  cntm    -  Maximum number of cost function evaluations
!  nneos   -  Equation of state:
!             1 - constant exp_coef_t, exp_coef_t defined from
 ↪  namelist
!             2 - alpha3d, beta3d compute from background
!             3 - tl density and its adjoint
!  eos_flname     - filename with T/S to compute
 ↪  Expantion/Contraction Coefficient (if nneos /= 1)
!  ssh_unbalanced - unbalance component of eta (EOFs)
!  exp_coef_t     - expansion coefficient for temperature
!  exp_coef_s     - contraction coefficient for salinity
!
! ---
&grdlst
   ntr             =  1,
   prj             =  0,
   filter          =  2,
   grid            =  1,
   ratio           =  1.,
   mask            =  3,
   barmd           =  1,
   balmd           =  0,
   divda           =  0,
   divdi           =  0,
```

```
cntm            =  100,
eos_flname      = 'background.nc'
nneos           =  1,
ssh_unbalanced = .F.,
exp_coef_t      = 0.24
exp_coef_s      = 0.74
/
```

- *ntr* is the number of iteration over the grid. **Suggested value is 1**
- *prj* is the type of grid:

  0 for lat/lon regular grid

  otherwise irregular grid
- *filter* is the type of horizontal filter. It can assume the following values:

  1. recursive filter is activated

  2. diffusive filter is activated

  3. no horizontal filter is activated
- *grid* is the number of grids over the same domain. **Suggested value is 1**
- *ratio* is the ratio between the grids in case *grid* parameter is different from 1. **If *grid* parameter is 1 *ratio* should be 1 as well.**
- *mask* is the type of mask. It can assume the following values:

  1: there is no land in the domain.

  2: Bi-dimensional horizontal mask.

  3: Three-dimensional mask, horizontally and vertically varying.
- *barmod* is the flag to activate the barotropic model. If it is 1 the barotropic model is activated. It converts temperature and salinity increments in surface elevation increments.
- *balmd* is the flag to activate the dynamic height. If it is 1 the dynamic height is activated. It converts temperature and salinity increments in surface elevation increments.
- *divda* is the flag to activate the divergence dumping. If is 1 the the divergence dumping is activated. It enforces the zero boundary condition for the total velocity component perpendicular to the coast.
- *divdi* is the flag to initialize the corrections with the divergence dumping. If

is 1 the corrections are initialized by the divergence.

- *cbtm* is the maximum number of cost function evaluations
- *nneos* is the type of thermal and haline expansion/contraction coefficient used to convert temperature and salinity in density. It can assume the following values:

  1. Values are constant and are defined by *exp_coef_t* for temperature and *exp_coef_s* for salinity

  2. Values are computed from temperature and salinity field using the non-linear equation before the minimization.

  3. Values are computed using the tangent linear and the adjoint version of the equation of state during the minimization procedure.
- *eos_flname* is the filename from where temperature and salinity field for the thermal and haline expansion/contraction computation are read.
- *ssh_unbalanced* flag to activate the sum of the corrections coming from dynamic height and EOFs for the surface elevation. If it is **False** the sea surface correction is determined only by the dynamic height operator. If is **True** the correction is the sum of the two corrections. If the barotropic model is activated (*barmod*=1 and *balmod*=0) this flag is ineffective.
- *exp_coef_t* is the value of the expansion coefficient for temperature.
- *exp_coef_s* is the value of the contraction coefficient for salinity.

barmod and *balmd* can not be 1 at the same time but they can be both 0. In this case the conversion between temperature and salinity increments is made by the EOFs(unbalance part). In the present code, the EOFs must always be trivariated, i.e. they must include covariances between temperature, salinity and SLA. It is presently not possible to use directly bivariate EOFs (covaring temperature and salinity only). Such configuration can be produced by artificially resetting the covariances between T,S with SLA.

Note: The present version of the code supports only *ntr*=1, *grd*=1 and *ratio*=1. However some part of the software were developed to support multi grid loops.

7.3 Observational data sets set-up

```
! Namelist obslst
!
!             Observational data sets set-up
!
! ---
!  obs_sla -  1-assimilate sla, 0-do not assimilate
!  obs_arg -  1-assimilate Argo, 0-do not assimilate
!  obs_xbt -  1-assimilate XBT, 0-do not assimilate
!  obs_gld -  1-assimilate glider, 0-do not assimilate
!  obs_tra -  1-assimilate Argo trajectories, 0-do not
↪  assimilate
!  obs_trd -  1-assimilate drifter trajectories, 0-do not
↪  assimilate
!  obs_vdr -  1-assimilate velocity from drifters,
!             0-do not assimilate
!  obs_gvl -  1-assimilate glider velocities, 0-do not
↪  assimilate
!  obs_sst -  1-assimilate SST, 0-do not assimilate
!
! ---
&obslst
   obs_sla = 1
   obs_arg = 1
   obs_xbt = 0
   obs_gld = 0
   obs_tra = 0
   obs_trd = 0
   obs_vdr = 0
   obs_gvl = 0
   obs_gvl = 0
/
```

- *obs_sla* is the flag to activate the assimilation of sea level anomaly.
- *obs_arg* is the flag to activate the assimilation Argo.
- *obs_xbt* is the flag to activate the assimilation XBT.
- *obs_gld* is the flag to activate the assimilation glider.
- *obs_tra* is the flag to activate the assimilation Argo trajectories.

- *obs_trd* is the flag to activate the assimilation drifter trajectories.
- *obs_vdr* is the flag to activate the assimilation drifter velocity.
- *obs_gvl* is the flag to activate the assimilation glider velocity.
- *obs_gvl* is the flag to activate the assimilation sea surface temperature.

In all cases **1** means assimilate, **0** do not assimilate.

7.4 Sea Level Anomaly assimilation set-up

```
!
! Namelist slalst
! ---
!
!          SLA assimilation set-up
!
!  sla_dep -  Maximum depth for assimilation of sla
!  sla_minobs - Minimum number of observation per track
!  sla_dsm -  Maximum distance between sla obs to be considered
 ↪  of the same track (km)
!  unbias   = T-remove bias,      F-do not remove bias
!  bias_at -  T-remove bias along track, F-remove global bias
 ↪  (ref=0) !if unbias = T
!  sla_minobspt - Minimum number of observation per track (only
 ↪  bias_at = T)
!
! ---
&slalst
   sla_dep      = 150.
   sla_minobspt = 4
   sla_dsm      = 40.
   unbias       = .T.
   bias_at      = .T.
/
```

- *sla_dep* is the value of the minimum ocean depth for which a sea level observation is considered valid. If the dynamic height is activated *balmd*=1 then it is also the reference depth to compute the dynamic height from temperature and salinity anomalies.

- *sla_minobspt* is the minimum number of observation per track for considering the track valid.
- *sla_dsm* is the maximum distance allowed between SLA observations in order to be considered observations belonging to the same track. The value is expressed in kilometers.
- *unbias* is the flag to activate bias removal of SLA observations. If it is True the bias removal is activated
- *bias_at* is the flag to activate the bias removal from track. If it is True the along track bias removal is activated, if it is False the global average is removed. It is effective only in case *unbias*=T.

7.5 Observational error set-up

```
! Namelist errlst
!
!             Observational error set-up
!
! ---
!  err_from_file   - T-read error from obs file, F-compute
↪  error
!  tim_dep_err     - T-apply temporal dependency of the error
↪  if err_from_file = F
!  ztime_weigth    - Temporal decay (in days) for obs errors if
↪  tim_dep_err = T
!  sla_sat_dep_err - T-apply sla satellite dependency of the
↪  error if err_from_file = F
!  sla_hor_dep_err - T-apply sla spatial   dependency of the
↪  error if err_from_file = F
!  sla_hor_dep_ecf - Corrective factor for horizontal error if
↪  sla_hor_dep_err = T
!  obserr_sla_flname - horizontal dependent error file name if
↪  sla_hor_dep_err = T
!  sla_sat_nu      - Number of SLA satellites if
↪  sla_sat_dep_err = T
!  sla_sat_na      - Name   of SLA satellites if
↪  sla_sat_dep_err = T
```

```
!  sla_sat_err     - Error  of SLA satellites if
↪  sla_sat_dep_err = T
!  sla_err         - Error  of SLA if sla_sat_dep_err = F and
↪  sla_hor_dep_err = F
!  ts_ver_dep_err  - T-apply vertical dependency of error to
↪  temperature and salinity
!  obserr_ts_flname- T/S vertical error file name if
↪  ts_ver_dep_err = T
!  tem_err         - Error  of Temperature if ts_ver_dep_err =
↪  F
!  sal_err         - Error  of Salinity    if ts_ver_dep_err =
↪  F
! ---
&errlst
   err_from_file     = .T.
   tim_dep_err       = .F.
   ztime_weigth      = 100.0
   sla_sat_dep_err   = .F.
   sla_hor_dep_err   = .F.
   sla_hor_dep_ecf   = 1.0
   obserr_sla_flname = 'SLA_repres.nc'
   sla_sat_nu        = 18
   sla_sat_na        = 'ERS1         ', 'ERS2         ',
    ↪  'ENVISAT      ', 'GFO          ', 'JASON1       ',
                         'JASON2       ', 'TP           ',
                          ↪  'CRYOSAT2     ', 'GEOSAT       ',
                          ↪  'ALTIKA       ',
                         'JASON2N      ', 'JASON3       ',
                          ↪  'SENTINEL3A   ', 'JASON2G      ',
                          ↪  'SENTINEL3B   '
                         'HY-2A        ', 'HY-2B        ',
                          ↪  'SENTINEL6A   '
   sla_sat_err=         0.03          ,  0.03          ,  0.03
    ↪  ,  0.03          ,  0.03          ,
                         0.03          ,  0.03          ,  0.03
                          ↪  ,  0.03          ,  0.03          ,
                         0.03          ,  0.03          ,  0.03
                          ↪  ,  0.03          ,  0.03
                         0.03          ,  0.03          ,  0.03
```

```
   sla_err            =  0.03
   ts_ver_dep_err     = .F.
   obserr_ts_flname   = 'insitu_errors.nc'
   tem_err            = 0.2
   sal_err            = 0.02
/
```

- *err_from_file* is the flag to activate the observational error computation. If it is True the error associated to the input misfit file (see chapter 8) is used as observational errors and all the other items in this namelist are meaningless. If it is False the values are still read but they are discharged and the computation of the observational errors is done accordingly to the item listed below.
- *tim_dep_err* is the flag to activate the time dependent error. If it is True the observational error is weighted by the time distance between the time of the observation retrieval and time at which the increments will be applied. The weighting function is an exponentially decaying function with characteristic time length equal to textitztime_weigth.
- *ztime_weigth* is the characteristic time length exponentially decaying function expressed in days.
- *sla_sat_dep_err* is the flag to activate the satellite dependency of the SLA error. If it True the errors associated to *sla_sat_err* are used. If it is false the error *sla_err* is used regardless of the type of satellite.
- *sla_hor_dep_err* is the flag to activate the horizontal dependency of the SLA error. If it True the code requires to read a variable called *stde* from the file defined in *obserr_sla_flname*. This variable of the dimension of the horizontal grid correspond to the SLA error. If it is False *obserr_sla_flname* and *sla_hor_dep_ecf* are meaningless.
- *sla_hor_dep_ecf* is the correction factor to be applied to the horizontal dependent SLA error, namely *stde* error.
- *obserr_sla_flname* is the filename where horizontal dependent error is read.
- *sla_sat_nu* is the number of SLA satellite considered in the assimilation. It is meaningfull only if *sla_sat_dep_err* = T.
- *sla_sat_na* is the name of SLA satellite considered in the assimilation. It is meaningfull only if *sla_sat_dep_err* = T.

- *sla_sat_err* is the value of SLA error associated to each satellite. It is meaningfull only if *sla_sat_dep_err* = T .
- *sla_err* is the value of SLA error if *sla_sat_dep_err* = F.
- *ts_ver_dep_err* is the flag to activate the vertically dependence of the temperature and salinity error. If it is False temperature and salinity errors are considered constant and they are defined by the value *tem_err* and *sal_err*, respectively. If the value is True the code requires to read several variables from the file defined in *obserr_ts_flname*. The variables are:
  1. *sal* vertical profile of salinity error
  2. *tem* vertical profile of temperature error
  3. *dep* depth at which the errors are defined
  4. *tem_fc* horizontal map of scaling factor of the error for temperature
  5. *sal_fc* horizontal map of scaling factor of the error for salinity

  The observational error is the product of *tem_fc* and *tem* for temperature and *sal_fc* and *sal* for salinity
- *obserr_ts_flname* filename of the temperature and salinity error.
- *tem_err* constant temperature error. It is meaningful only if *ts_ver_dep_err* = F.
- *sal_err* constant salinity error. It is meaningful only if *ts_ver_dep_err* = F.

All the errors are additive.

7.6 Coastal Rejection set-up

```
! Namelist crjlst
! ---
!
!            Coastal Rejection namelist
!
!  coastrej_arg    -  Apply coastal rejection to ARG
!  coastrej_gld    -  Apply coastal rejection to GLD
!  coastrej_gvl    -  Apply coastal rejection to GVL
!  coastrej_sla    -  Apply coastal rejection to SLA
!  coastrej_sst    -  Apply coastal rejection to SST
!  coastrej_tra    -  Apply coastal rejection to TRD
```

```
!  coastrej_trd    -  Apply coastal rejection to TRA
!  coastrej_vdr    -  Apply coastal rejection to VDR
!  coastrej_xbt    -  Apply coastal rejection to XBT
!  coastrej_km_arg - Km from the coast ( if coastrej_arg = T )
!  coastrej_km_gld - Km from the coast ( if coastrej_gld = T )
!  coastrej_km_gvl - Km from the coast ( if coastrej_gvl = T )
!  coastrej_km_sla - Km from the coast ( if coastrej_sla = T )
!  coastrej_km_sst - Km from the coast ( if coastrej_sst = T )
!  coastrej_km_tra - Km from the coast ( if coastrej_tra = T )
!  coastrej_km_trd - Km from the coast ( if coastrej_trd = T )
!  coastrej_km_vdr - Km from the coast ( if coastrej_vdr = T )
!  coastrej_km_xbt - Km from the coast ( if coastrej_xbt = T )
!
!
! ---
&crjlst
  coastrej_arg    = .F.
  coastrej_gld    = .F.
  coastrej_gvl    = .F.
  coastrej_sla    = .F.
  coastrej_sst    = .F.
  coastrej_tra    = .F.
  coastrej_trd    = .F.
  coastrej_vdr    = .F.
  coastrej_xbt    = .F.
  coastrej_km_arg = 15.
  coastrej_km_gld = 15.
  coastrej_km_gvl = 15.
  coastrej_km_sla = 15.
  coastrej_km_sst = 15.
  coastrej_km_tra = 15.
  coastrej_km_trd = 15.
  coastrej_km_vdr = 15.
  coastrej_km_xbt = 15.
/
```

- *coastrej_arg* is the flag to activate the coastal rejection for Argo.
- *coastrej_gld* is the flag to activate the coastal rejection for glider.
- *coastrej_gvl* is the flag to activate the coastal rejection for glider velocity.

- *coastrej_sla* is the flag to activate the coastal rejection for SLA.
- *coastrej_sst* is the flag to activate the coastal rejection for SST.
- *coastrej_tra* is the flag to activate the coastal rejection for Argo trajectory.
- *coastrej_trd* is the flag to activate the coastal rejection for drifter trajectories.
- *coastrej_vdr* is the flag to activate the coastal rejection for drifter velocity.
- *coastrej_xbt* is the flag to activate the coastal rejection for XBT.
- *coastrej_km_arg* value of distance from the coast to reject the data. Meaningful only if *coastrej_arg* = T.
- *coastrej_km_gld* value of distance from the coast to reject the data. Meaningful only if *coastrej_gld* = T.
- *coastrej_km_gvl* value of distance from the coast to reject the data. Meaningful only if *coastrej_gvl* = T.
- *coastrej_km_sla* value of distance from the coast to reject the data. Meaningful only if *coastrej_sla* = T.
- *coastrej_km_sst* value of distance from the coast to reject the data. Meaningful only if *coastrej_sst* = T.
- *coastrej_km_tra* value of distance from the coast to reject the data. Meaningful only if *coastrej_tra* = T.
- *coastrej_km_trd* value of distance from the coast to reject the data. Meaningful only if *coastrej_trd* = T.
- *coastrej_km_vdr* value of distance from the coast to reject the data. Meaningful only if *coastrej_vdr* = T.
- *coastrej_km_xbt* value of distance from the coast to reject the data. Meaningful only if *coastrej_xbt* = T.

7.7 Quality check set up

```
! Namelist qcklst
!
!            Quality check
!
!  qc_res          ! Absolute misfits quality control
```

```
!  qc_conbgr       ! Background ratio quality control
!  qc_eofbgr       ! Background ratio quality control based on
↪  EOFs
!  qc_clm          ! Climatological   quality control
!  qc_vert         ! Reject measurements if some observations
↪  above within the same profile were rejected
!  qc_res_sla      ! Max. residual allowed for SLA          (if
↪  qc_res = T)
!  qc_res_tem      ! Max. residual allowed for Temperature  (if
↪  qc_res = T)
!  qc_res_sal      ! Max. residual allowed for Salinity     (if
↪  qc_res = T)
!  qc_res_vel      ! Max. residual allowed for U/V velocity (if
↪  qc_res = T)
!  qc_res_sst      ! Max. residual allowed for SST          (if
↪  qc_res = T)
!  qc_res_dis      ! Max. residual allowed for Distance (Km)(if
↪  qc_res = T)
!  qc_bgr_sla      ! Background error and Threshold ratio for
↪  SLA          (if qc_conbgr = T)
!  qc_bgr_tem      ! Background error and Threshold ratio for
↪  Temperature  (if qc_conbgr = T)
!  qc_bgr_sal      ! Background error and Threshold ratiofor
↪  Salinity      (if qc_conbgr = T)
!  qc_bgr_vel      ! Background error and Threshold ratio for
↪  U/V velocity (if qc_conbgr = T)
!  qc_bgr_sst      ! Background error and Threshold ratio for
↪  SST           (if qc_conbgr = T)
!  qc_bgr_dis      ! Background error and Threshold ratio for
↪  Distance (Km) (if qc_conbgr = T)
!  qc_clm_limT     ! Max Treshold allowed between observation
↪  and climatology for Temperature (if qc_clm =T)
!  qc_clm_limS     ! Max Treshold allowed between observation
↪  and climatology for Salinity    (if qc_clm =T)
!  qc_clm_flname   ! Filename for climatology used also for
↪  qc_eofbgr
! ---
&qcklst
   qc_res      = .F.
```

```
qc_clm      = .F.
qc_vert     = .F.
qc_conbgr   = .F.
qc_eofbgr   = .F.
qc_res_sla  =  0.3
qc_res_tem  =  5.0
qc_res_sal  =  2.0
qc_res_vel  =  1.0
qc_res_sst  =  0.4
qc_res_dis  =  10.0
qc_bgr_sla  =  0.1, 0.64
qc_bgr_tem  =  1.0, 25.0
qc_bgr_sal  =  0.2, 25.0
qc_bgr_vel  =  0.2, 25.0
qc_bgr_sst  =  0.5, 11.0
qc_bgr_dis  =  2.0, 25.0
qc_clm_limT =  9.0
qc_clm_limS =  3.0
qc_clm_flname = 'climatology.nc'
```

- *qc_res* logical flag to activate the residual check. If it is True observations that have too large absolute difference with respect to the background are rejected. The threshold limits depend from the variable taken into account and are defined by the user with the variables *qc_res_xxx* defined below.
- *qc_clm* logical flag to activate the climatological check. If it is True observations that have too large absolute difference with respect to the climatology are rejected. The threshold limits depend from the variable taken into account and are defined by the user with the variables *qc_clm_limx* defined below.
- *qc_vert* logical flag to activate the vertical check quality control. If it is True a profile that has only valid sub-surface observations is rejected.
- *qc_conbgr* logical flag to activate the background check based on background field. If it is True $[y - H(x_b)]^2/(\sigma_b^2 + \sigma_o^2) > threshold$ are rejected. The threshold limits depend from the variable taken into account and are defined by the user with the variables *qc_bgr_xxx* defined below.
- *qc_eofbgr* logical flag to activate the background check based on eof field. If it is True $[y - H(x_b)]^2/(\sigma_b^2 + \sigma_o^2) > threshold$ are rejected. The threshold

limits depend from the variable taken into account and are defined by the user with the variables *qc_bgr_xxx* defined below.

- *qc_res_sla* Threshold limit for residual check for SLA variable.
- *qc_res_tem* Threshold limit for residual check for temperature variable.
- *qc_res_sal* Threshold limit for residual check for salinity variable.
- *qc_res_vel* Threshold limit for residual check for velocity variable.
- *qc_res_sst* Threshold limit for residual check for SST variable.
- *qc_res_dis* Threshold limit for residual check for distance variable (applyed to trajectories).
- *qc_bgr_sla* Threshold limit for background check for SLA variable.
- *qc_bgr_tem* Threshold limit for background check for temperature variable.
- *qc_bgr_sal* Threshold limit for background check for salinity variable.
- *qc_bgr_vel* Threshold limit for background check for velocity variable.
- *qc_bgr_sst* Threshold limit for background check for SST variable.
- *qc_bgr_dis* Threshold limit for background check for distance variable (applyed to trajectories).
- *qc_clm_limT* Threshold limit for climatological check for temperature variable.
- *qc_clm_limS* Threshold limit for climatological check for salinity variable.
- *qc_clm_flname* Filename for climatology used also for *qc_eofbgr*

7.8 Huber norm set-up

```
! Namelist hublst
! ---
!
!             Huber norm quality check
!
!  huberqc_arg    -  Apply Huber Norm QC to ARG
!  huberqc_gld    -  Apply Huber Norm QC to GLD
!  huberqc_gvl    -  Apply Huber Norm QC to GVL
!  huberqc_sla    -  Apply Huber Norm QC to SLA
!  huberqc_sst    -  Apply Huber Norm QC to SST
```

```
!  huberqc_tra   -  Apply Huber Norm QC to TRD
!  huberqc_trd   -  Apply Huber Norm QC to TRA
!  huberqc_vdr   -  Apply Huber Norm QC to VDR
!  huberqc_xbt   -  Apply Huber Norm QC to XBT
!  huberqc_asymm -  T- Non-symmetric coefficients; F- Symmetric
↪  coefficients (if ANY of above)
!  huberqc_L05   -  T- L2-L1 + L05 Norm, F- only L05 Norm
↪  (http://dx.doi.org/10.1016/j.ocemod.2016.06.011)
!  huberqc_flname-  Coefficients' file name
!  huberqc_iter  -  No. of iteration at which variational
↪  quality control start
!
! ---
&hublst
   huberqc_arg   = .F.
   huberqc_gld   = .F.
   huberqc_gvl   = .F.
   huberqc_sla   = .F.
   huberqc_sst   = .F.
   huberqc_tra   = .F.
   huberqc_trd   = .F.
   huberqc_vdr   = .F.
   huberqc_xbt   = .F.
   huberqc_asymm = .F.
   huberqc_L05   = .F.
   huberqc_flname= 'hubernorm_limits_3dvar.nc'
   huberqc_iter  = 10
```

- *huberqc_arg* logical flag to activate the huber norm quality check for argo
- *huberqc_gld* logical flag to activate the huber norm quality check for glider
- *huberqc_gvl* logical flag to activate the huber norm quality check for glider velocity
- *huberqc_sla* logical flag to activate the huber norm quality check for sla
- *huberqc_sst* logical flag to activate the huber norm quality check for sst
- *huberqc_tra* logical flag to activate the huber norm quality check for argo trajectory
- *huberqc_trd* logical flag to activate the huber norm quality check for drifter

trajectory

- *huberqc_vdr* logical flag to activate the huber norm quality check for drifter velocity trajectory
- *huberqc_xbt* logical flag to activate the huber norm quality check for xbt
- *huberqc_asymm* logical flag to activate the asymmetric coefficients
- *huberqc_L05* logical flag to activate the type of norm. For more information please refers to http://dx.doi.org/10.1016/j.ocemod.2016.06.011
- *huberqc_flname* name of the huber norm limits
- *huberqc_iter* Number of iteration at which variational quality control starts

7.9 Thinning set-up

```
! Namelist thinning
!
!            Thinning
!  thin_arg   -  Apply thinning to  ARG
!  thin_gld   -  Apply thinning to  GLD
!  thin_gvl   -  Apply thinning to  GVL
!  thin_sla   -  Apply thinning to  SLA
!  thin_sst   -  Apply thinning to  SST
!  thin_tra   -  Apply thinning to  TRD
!  thin_trd   -  Apply thinning to  TRA
!  thin_vdr   -  Apply thinning to  VDR
!  thin_xbt   -  Apply thinning to  XBT
!  thin_tim   -  Time  window for averaging (sec.)
!  thin_spc   -  Space window for averagind ( km )
!
! ---
&thnlst
   thin_arg   = .F.
   thin_gld   = .F.
   thin_gvl   = .F.
   thin_sla   = .F.
   thin_sst   = .F.
   thin_tra   = .F.
   thin_trd   = .F.
```

```
   thin_vdr    = .F.
   thin_xbt    = .F.
   thin_tim    = 60.
   thin_spc    =  5.
/
```

- *thin_arg* logical flag to activate the thinning of Argo profiles
- *thin_gld* logical flag to activate the thinning of glider profiles
- *thin_gvl* logical flag to activate the thinning of glider velocity profiles
- *thin_sla* logical flag to activate the thinning of SLA
- *thin_sst* logical flag to activate the thinning of SST
- *thin_tra* logical flag to activate the thinning of Argo trajectories (**Not implemented**)
- *thin_trd* logical flag to activate the thinning of drifters trajectories (**Not implemented**)
- *thin_vdr* logical flag to activate the thinning of drifter velocity
- *thin_xbt* logical flag to activate the thinning of XBT
- *thin_tim* time window (**Not implemented**)
- *thin_spc* space window (**Not implemented**)

In the present version the thinning is performed only if more than one observation per type is whithin the same grid cell.

7.10 BFGS minimizer set up

```
! Namelist ctllst
! ---
!
!             BFGS minimizers set-up
!
!  ctl_m   -  Number of copies saved in the minimizer
!  ctl_tol -  Stopping criteria (absolute)
!  ctl_per -  Stopping criteria (relative)
!
! ---
&ctllst
```

```
   ctl_m     = 5
   ctl_tol   = 1.e-8
   ctl_per   = 1.e-2
/
```

- *ctl_m* number of copies saved in the minimizer
- *ctl_tol* absolute value for stopping criteria.
- *ctl_per* relative value for stopping criteria

The stopping criteria is define based on the gradient. In case of relative criteria the iterations are stopped when the ratio between the actual gradient and the gradient at the first iteration is below *ctl_per*

7.11 Covariance constants

```
!
! Namelist rcflst
! ---
!
!             Covariance constants
!
!  eof_flname - EOF filenema
!  neof       - Number of vertical EOFs
!  nreg       -  Number of regions
!  rcf_L      -  Horizontal correlation radius
!  rcf_loc    -  Horizontal radius of localization
!  mld        -  1 - mixed layer depth, 0 - no
! ---
&covlst
   eof_flname = 'eofs.nc'
   neof     = 1
   nreg     = 31065
   rcf_L    = 60000.
   rcf_loc  = 1.
   mld      = 0
/
```

- *eof_flname* filename containing Singular Values (Sv) and Empirical Orthogonal Functions (EOFs).

- *neof* number of vertical EOFs used in the minimization procedure
- *nreg* number of regions for which EOFs are computed.
- *rcf_L* horizontal correlation radius for recursive filter expressed in meters.
- *rcf_loc* horizontal localization radius expressed in meters. If *rcf_loc* > *rcf_L* the amplitude of the increments at large distance is damped.
- *mld* flag to read mix layer depth. If flag is activated no corrections are provided below the mixed layer depth.

7.12 Diffusive Filter set up

```
! Namelist diflst
!---
!
!       Diffusive Filter namelist
!
!  nt           = number of iteration for the diffusion filter
!  rd_corr      = read correlation radius from file
!  rx           = correlation radius in x  ( if not read from
↪  file )
!  ry           = correlation radius in y  ( if not read from
↪  file )
!  use_bc       = use boundary conditions
!  bc_type      = type of boundary condition (if use_bc = .T.
↪  then choose NEUMANN/DIRICHLET)
!  rd_wgh       = read weights from external file (if F compute
↪  analytically)
!  wgh_flname   = Normalization factor filename
!  crl_flname   = Horizontal correlation length filename
!  use_cst      = use coastal distance to reduce the
↪  correlation radius
!  cst_dis      = costal distance value ( if use_cst = .T. )
&diflst
   nt           = 5 !20
   rd_corr      = .F.
   rx           = 60000.0
   ry           = 60000.0
   use_bc       = .T.
```

```
   bc_type       = 'NEUMANN' !'DIRICHLET'
   rd_wgh        = .F.
   wgh_flname    = 'weights.nc'
   crl_flname    = 'hcorr.nc'
   use_cst       = .T.
   cst_dst       = 30000
/
```

- *nt* is the number of iterations applied to the diffusive fileter
- *rd_corr* is the flag to activate variable correlation radii. If it is False the correlation radius is expressed by *rx* and *ry* from namelist. If it is True the code requires to read a file defined in *crl_flname* that contains three-dimensional variables called *rx* and *ry* of the same dimension of the grid.
- *rx* is the correlation radius in x expressed in meters
- *ry* is the correlation radius in y expressed in meters
- *use_bc* flag to activate boundary condition on land.
- *bc_type* type of boundary condition. Possible value are *NEUMANN* or *DIRICHLET*. It is effective only if *use_bc*=T.
- *rd_wgh* flag to activate the read of external weights to convert correlation radius to diffusion coefficients. If it is True the code requires to read a file defined in *wgh_flname* that contains a three-dimensional variable called *coef* with the dimensions of the grid.
- *wgh_flname* filename containing the weights
- *crl_flname* filename containing the correlation radii
- *use_cst* flag to activate the decrease of correlation radius close to the coast. If it is True the distance is defined by *cst_dst* flag
- *cst_dst* distance from the coast to decrease the correlation radius. It is expressed in meters. In is meaningful only if *use_cst*=T.

7.13 Barotropic model set-up

```
! Namelist bmdlst
! ---
!
!          Barotropic model set-up
```

```
!
!  bmd_dt   -  Time step
!  bmd_ndy  -  Number of days to integrate the model
!  bmd_ady  -  Number of days for averaging
!  bmd_alp  -  Weight for the trapezoidal scheme
!  bmd_fc1  -  Horizontal friction for vorticity
!  bmd_fc2  -  Horizontal friction for divergence
!  bmd_ovr  -  Overrelaxation
!  bmd_resem - Stopping criteria
!  bmd_ncnt - Maximum number of successive corrections
!
! ---
&bmdlst
  bmd_dt          = 7200 !3600.
  bmd_ndy         = 3    !1.
  bmd_ady         = 1    !0.5
  bmd_alp         = 1.0
  bmd_fc1         = 0.1 ! 0.1
  bmd_fc2         = 0.2 ! 0.2
  bmd_ovr         = 1.9
  bmd_resem       = 5.e-2
  bmd_ncnt        = 201
/
```

- *bmd_dt* is the time step of the barotropic model in second. It correspond to $2\Delta T$
- *bmd_ndy* is the number of days to integrate the model
- *bmd_ady* is the number of days for averaging
- *bmd_alp* is the weight for the trapezoidal scheme
- *bmd_fc1* is the scaled horizontal friction at time step t - 1. The value is adimensional because is divided by the averaged area of the grid cells $\overline{dxdy}$ and multiplied by the $2\Delta T$. Its relationship with the actual friction coefficient is expressed by the formula γ = $fc1 * \overline{dxdy}/2\Delta T$
- *bmd_fc2* is the scaled horizontal friction at time step t - 1. The value is adimensional because is divided by the averaged area of the grid cells $\overline{dxdy}$ and multiplied by the $2\Delta T$. Its relationship with the actual friction coefficient is expressed by the formula γ = $fc2 * \overline{dxdy}/2\Delta T$

- *bmd_ovr* is the over-relaxation parameter
- *bmd_resem* is the stopping criteria for implicit scheme
- *bmd_ncnt* is the maximum number of successive corrections

7.14 Adjoint check set up

```
! Namelist adjck
! ---
!
!          Adjoint check
!
!  bal_ck  - check simplified balance operator adjoint
&adjcklst
  bal_ck  = .F.
  bmd_ck  = .F.
  byg_ck  = .F.
  dfl_ck  = .F.
/
```

- *bal_ck* flag to activate (True) or deactivate (False) the adjoint check of the dynamic height operator. It is effective only without domain decomposition.
- *bmd_ck* flag to activate (True) or deactivate (False) the adjoint check of the barotropic model operator. It is effective only without domain decomposition.
- *byg_ck* flag to activate (True) or deactivate (False) the adjoint check of the buoyancy computation. It is effective only without domain decomposition.
- *dfl_ck* flag to activate (True) or deactivate (False) the adjoint check of the diffusive filter operator. It is effective only without domain decomposition.

7.15 MPI tiles set up

```
! Namelist mpilst
! ---
!
!          Set-up for mpi tiles
!
```

```
!  mpi_irm -  number of tiles in the x direction
!  mpi_jrm -  number of tiles in the y direction
!  mpi_thx -  number of RF threads in x direction
!  mpi_thy -  number of RF threads in x direction
!  mpi_flgmin - 1 sequantial - 0 parallel
!
! ---
&mpilst
  mpi_irm    = @MPII@
  mpi_jrm    = @MPIJ@
  mpi_thx    = 10
  mpi_thy    = 10
  mpi_flgmin    = 1
/
```

- *mpi_irm* number of tiles in the x direction
- *mpi_jrm* number of tiles in the y direction
- *mpi_thx* number of recursive filter threads in x direction
- *mpi_thy* number of recursive filter threads in x direction
- *mpi_flgmin* flag to activate the parallel or sequential version of the minimizer. If *mpi_flgmin*=0 then parallel otherwise sequential.

7.16 I/O directories set up

```
! Namelist iolst
! ---
!
!          Directories for reading I/O
!
!  inp_dir - directory read input
!
! ---
&iolst
  inp_dir = "./"
/
```

- *inp_dir* directory where input and output files are read and write

8 Input/Output

In the present section the I/O structure will be presented in more detail. In the present version I/O files are a mixture of simple binary, NetCDF and ASCII files. In future releases the intent is to uniform all the I/O in NetCDF. All the variable and dimension name are strictly hard-coded in the sofware so the user should follow the templates shown below to compute its on input files.

8.1 Input files

All the input files are listed below. Not all of them are mandatory. The presence of some of them depends from the choices made in the namelist. Some filenames can be changed accordingly to the namelist other are hard-coded in the code.

8.1.1 Grid file

The grid file is mandatory and its filename should be ***grid#.nc*** where the character # indicates the number of the grid defined by the parameter *grid* in the *&grdlst* namelist. The suggested value for *grid* is 1 so the filename is grid1.nc. It basically contains all the information of the grid. Below it is shown an output of the ncdump command.

```
$ ncdump -h grid1.nc
    netcdf grid1 {
    dimensions:
        km = 141 ;
        im = 327 ;
        jm = 95 ;
    variables:
        float lon(jm, im) ;
            lon:standard_name = "longitude" ;
            lon:long_name = "longitude" ;
            lon:units = "degrees_east" ;
            lon:_CoordinateAxisType = "Lon" ;
        float lat(jm, im) ;
            lat:standard_name = "latitude" ;
            lat:long_name = "latitude" ;
            lat:units = "degrees_north" ;
            lat:_CoordinateAxisType = "Lat" ;
```

```
float dep(km) ;
    dep:units = "m" ;
float dcoast(km, jm, im) ;
    dcoast:coordinates = "lat lon" ;
    dcoast:_FillValue = 1.e+20f ;
    dcoast:missing_value = 1.e+20f ;
float dcoast2d(jm, im) ;
    dcoast2d:coordinates = "lat lon" ;
    dcoast2d:_FillValue = 1.e+20f ;
    dcoast2d:missing_value = 1.e+20f ;
double dx(jm, im) ;
    dx:coordinates = "lat lon" ;
double dy(jm, im) ;
    dy:coordinates = "lat lon" ;
float dz(km) ;
    dz:units = "m" ;
float tmsk(km, jm, im) ;
    tmsk:coordinates = "lat lon" ;
float topo(jm, im) ;
    topo:long_name = "Bathymetry" ;
    topo:units = "meters" ;
    topo:coordinates = "lat lon" ;
    topo:title = "Bathymetry" ;
```

All the variables are mandatory except for *dcoast* and *dcoast2d*. The fist one is mandatory if diffusion filter is active *filter*=2 and the last is mandatory if at least one of the logical flag related to the coastal rejection is active *coastrej_xxx*=1

8.1.2 EOF file

The EOF file is mandatory and its name can be chosen in the *&covlst* namelist with the parameter *eof_flname* Below it is shown an output of the ncdump command.

```
$ ncdump -h eofs.nc
    netcdf eofs {
    dimensions:
            neof = 5 ;
            nlev = 283 ;
            jmt = 95 ;
```

```
        imt = 327 ;
        nreg = 31065 ;
    variables:
        float eva(neof, jmt, imt) ;
        float evc(neof, nlev, jmt, imt) ;
        float regs(jm, im) ;
            regs:long_name = "Full-grid EOF
            ↪ region" ;
```

All the variables are mandatory. Another possible structure of the file if the EOFs are not different in each grid cell is:

```
$ ncdump -h eofs.nc
    netcdf eofs {
    dimensions:
        nreg = 2 ;
        neof = 5 ;
        nlev = 283 ;
        jmt = 95 ;
        imt = 327 ;
    variables:
        float eva(neof, nreg) ;
        float evc(neof, nlev, nreg) ;
        float regs(jm, im) ;
            regs:long_name = "Full-grid EOF
            ↪ region" ;
```

Note that the number of level *nlev* are equal to double plus 1 the number of model vertical level *km* in the grid file. The relationship $nlev = 2 * km + 1$ must be strictly maintained. This is because the EOFs are three-variate with the state vector defined as $X = [eta, tem, sal]$. In case of bi-variate (temperature and salinity) the user should instert an array of 0 for eta.

8.1.3 SLA representation error file

The file is not mandatory and it is necessary only if *sla_hor_dep_err*=T in *&errlst* namelist. The file name must correspond to the variable *obserr_sla_flname* defined in the namelist. Below it is shown an output of the ncdump command.

```
$ ncdump -h SLA_repres.nc
    netcdf SLA_repres {
```

```
dimensions:
        jm = 95 ;
        im = 327 ;
        time_counter = UNLIMITED ; // (1 currently)
variables:
        float lat(jm, im) ;
                lat:standard_name = "latitude" ;
                lat:long_name = "latitude" ;
                lat:units = "degrees_north" ;
                lat:_CoordinateAxisType = "Lat" ;
        float lon(jm, im) ;
                lon:standard_name = "longitude" ;
                lon:long_name = "longitude" ;
                lon:units = "degrees_east" ;
                lon:_CoordinateAxisType = "Lon" ;
        float stde(time_counter, jm, im) ;
                stde:long_name = "SSH representation
                 ↪  error" ;
                stde:units = "m" ;
                stde:coordinates = "lat lon" ;
                stde:_FillValue = -1.f ;
                stde:missing_value = -1.f ;
```

8.1.4 Insitu representation error file

The file is not mandatory and it is necessary only if *ts_ver_dep_err*=T in *&errlst* namelist. The file name must correspond to the variable *obserr_ts_flname* defined in the namelist. Below it is shown an output of the ncdump command.

```
$ ncdump -h insitu_errors.nc
    netcdf insitu_errors {
    dimensions:
            depth = 401 ;
            jm = 95 ;
            im = 327 ;
    variables:
            double depth(depth) ;
            float sal(depth) ;
            float tem(depth) ;
            float lat(jm, im) ;
```

```
                lat:standard_name = "latitude" ;
                lat:long_name = "latitude" ;
                lat:units = "degrees_north" ;
                lat:_CoordinateAxisType = "Lat" ;
        float lon(jm, im) ;
                lon:standard_name = "longitude" ;
                lon:long_name = "longitude" ;
                lon:units = "degrees_east" ;
                lon:_CoordinateAxisType = "Lon" ;
        float sal_fc(jm, im) ;
                sal_fc:coordinates = "lat lon" ;
                sal_fc:long_name = "Representativeness
                 ↪  factor" ;
        float tem_fc(jm, im) ;
                tem_fc:coordinates = "lat lon" ;
                tem_fc:long_name = "Representativeness
                 ↪  factor" ;
```

The dimension depth can not correspond to the one of the grid because the profile error is linearly interpolated inside the code.

8.1.5 Equation of State file

The file is not mandatory and it is necessary only if *nneos*=2 or 3. The file name must correspond to the variable *eos_flname* defined in the *&covlst* namelist. This file is used to compute the thermo/haline expansion/contraction coefficients. Below it is shown an output of the ncdump command.

```
$ ncdump -h background.nc
    netcdf background {
    dimensions:
            jm = 95 ;
            im = 327 ;
            time_counter = UNLIMITED ; // (1 currently)
            km = 141 ;
    variables:
            float lat(jm, im) ;
                    lat:standard_name = "latitude" ;
                    lat:long_name = "latitude" ;
                    lat:units = "degrees_north" ;
```

```
        lat:_CoordinateAxisType = "Lat" ;
float lon(jm, im) ;
        lon:standard_name = "longitude" ;
        lon:long_name = "longitude" ;
        lon:units = "degrees_east" ;
        lon:_CoordinateAxisType = "Lon" ;
double time_counter(time_counter) ;
        time_counter:standard_name = "time" ;
        time_counter:long_name = "Time axis" ;
        time_counter:bounds =
         ↪  "time_counter_bnds" ;
        time_counter:units = "seconds since
         ↪  1950-01-01 00:00:00" ;
        time_counter:calendar = "gregorian" ;
        time_counter:axis = "T" ;
float vosaline(time_counter, km, jm, im) ;
        vosaline:standard_name =
         ↪  "sea_water_practical_salinity" ;
        vosaline:long_name =
         ↪  "sea_water_salinity" ;
        vosaline:units = "1e-3" ;
        vosaline:coordinates = "lat lon" ;
        vosaline:_FillValue = 1.e+20f ;
        vosaline:missing_value = 1.e+20f ;
        vosaline:online_operation = "instant" ;
        vosaline:interval_operation = "1 d" ;
        vosaline:interval_write = "1 d" ;
        vosaline:cell_methods = "time: point" ;
float votemper(time_counter, km, jm, im) ;
        votemper:standard_name =
         ↪  "sea_water_potential_temperature" ;
        votemper:long_name =
         ↪  "sea_water_potential_temperature" ;
        votemper:units = "degC" ;
        votemper:coordinates = "lat lon" ;
        votemper:_FillValue = 1.e+20f ;
        votemper:missing_value = 1.e+20f ;
        votemper:online_operation = "instant" ;
        votemper:interval_operation = "1 d" ;
```

```
                votemper:interval_write = "1 d" ;
                votemper:cell_methods = "time: point" ;
```

8.1.6 Climatological quality control file

The file is not mandatory and it is necessary only if *qc_clm*=T or *qc_eofbgr*=T. The filename must correspond to the variable *qc_clm_flname* defined in the namelist. This file is used to compute the absolute difference between the climatology and observations and/or the density anomaly to compute the background check on SLA variable. If they are above the value defined by *qc_clm_limT*, *qc_clm_limS* in the namelist for temperature and salinity, respectively, the observation is rejected. Below it is shown an output of the ncdump command.

```
$ ncdump -h climatology.nc
    netcdf climatology {
    dimensions:
            time = UNLIMITED ; // (12 currently)
            im = 327 ;
            jm = 95 ;
            km = 141 ;
    variables:
            double time(time) ;
                    time:standard_name = "time" ;
                    time:units = "months since 1-1-1" ;
                    time:calendar = "standard" ;
                    time:axis = "T" ;
            float lon(jm, im) ;
                    lon:standard_name = "longitude" ;
                    lon:long_name = "longitude" ;
                    lon:units = "degrees_east" ;
                    lon:_CoordinateAxisType = "Lon" ;
            float lat(jm, im) ;
                    lat:standard_name = "latitude" ;
                    lat:long_name = "latitude" ;
                    lat:units = "degrees_north" ;
                    lat:_CoordinateAxisType = "Lat" ;
            float votemper(time, km, jm, im) ;
                    votemper:long_name = "vosaline" ;
                    votemper:units = "m" ;
```

```
                votemper:coordinates = "lat lon" ;
        float vosaline(time, km, jm, im) ;
                vosaline:long_name = "vosaline" ;
                vosaline:units = "m" ;
                vosaline:coordinates = "lat lon" ;
```

8.1.7 Huber Norm file

The file is not mandatory and it is necessary only if at least one of the logical variable if the *&hublst* namelist is True. The filename must correspond to the variable *huberqc_flname* defined in the namelist. Below it is shown an output of the ncdump command.

```
$ ncdump -h hubernorm_limits_3dvar.nc
    netcdf hubernorm_limits_3dvar {
    dimensions:
            Nside = 2 ;
            Nnorm = 2 ;
            Nparam = 2 ;
    variables:
            float limits_xbt(Nside, Nnorm) ;
                limits_xbt:long_name = "huber_limits" ;
            float limits_tesac(Nside, Nnorm, Nparam) ;
                limits_tesac:long_name = "huber_limits"
                 ↪  ;
            float limits_argo(Nside, Nnorm, Nparam) ;
                limits_argo:long_name = "huber_limits"
                 ↪  ;
            float limits_buoy(Nside, Nnorm, Nparam) ;
```

8.1.8 Horizontal correlation length file for diffusive filter

The file is not mandatory and it is necessary only if *rd_corr* variable in *&diflst* namelist is True. The filename must correspond to the variable *crl_flname* in the same namelist. The *rx* and *ry* variables correspond to the correlation length in the x and y directions, respectively, and they have to be express in meters. Below it is shown an output of the ncdump command.

```
$ ncdump -h hcorr.nc
    netcdf hcorr {
```

```
dimensions:
        im = 327 ;
        jm = 95 ;
        km = 141 ;
variables:
        float lon(jm, im) ;
                lon:standard_name = "longitude" ;
                lon:long_name = "longitude" ;
                lon:units = "degrees_east" ;
                lon:_CoordinateAxisType = "Lon" ;
        float lat(jm, im) ;
                lat:standard_name = "latitude" ;
                lat:long_name = "latitude" ;
                lat:units = "degrees_north" ;
                lat:_CoordinateAxisType = "Lat" ;
        double rx(km, jm, im) ;
                rx:units = "meters" ;
                rx:coordinates = "lat lon" ;
                rx:_FillValue = 0. ;
                rx:missing_value = 0. ;
                rx:longname = "longitude correlation
                 ↪  radius" ;
        double ry(km, jm, im) ;
                ry:coordinates = "lat lon" ;
                ry:_FillValue = 0. ;
                ry:missing_value = 0. ;
                ry:longname = "latitude correlation
                 ↪  radius" ;
```

8.1.9 Weight file for diffusive filter

The file is not mandatory and it is necessary only if *rd_wgh* variable in *&diflst* namelist is True. The filename must correspond to the variable *wgh_flname* in the same namelist. The *coef* is used to convert correlation lengths to corresptive diffusive coefficients Below it is shown an output of the ncdump command.

```
$ ncdump -h weights.nc
    netcdf weights {
dimensions:
        im = 327 ;
```

```
        jm = 95 ;
        km = 141 ;
variables:
        float lon(jm, im) ;
                lon:coordinate = "lon lat" ;
        float lat(jm, im) ;
                lat:coordinate = "lon lat" ;
        float dy(jm, im) ;
                dy:girspacing\ y = "lon lat" ;
        float dx(jm, im) ;
                dx:girspacing\ x = "lon lat" ;
        double coef(km, jm, im) ;
                coef:_FillValue = 2.12199579096527e-314 ;
```

8.2 Misfits files

Misfits files are the files that contain the information concerning the difference between model and observations. They simple binary files and variables are to be written in double precision. There is a file for type of instrument and their name is hard-coded in the code. Below the list of possible instruments implemented in the code:

- ***arg_mis.dat***: file containg the Argo misfits
- ***gld_mis.dat***: file containg the Glider misfits
- ***xbt_mis.dat***: file containg the XBT misfits
- ***sla_mis.dat***: file containg the SLA misfits
- ***sst_mis.dat***: file containg the SST misfits
- ***tra_mis.dat***: file containg the Argo trajectory misfits
- ***trd_mis.dat***: file containg the Drifters trajectory misfits
- ***vdr_mis.dat***: file containg the Drifter velocitiy misfits
- ***gvl_mis.dat***: file containg the Glider velocity misfits

The files are read using the similar structure, but depending of the type of instruments they can have different variables.

8.2.1 Argo, Glider and XBT misfit files

They have in common the same files and example of how they are read in the code is reported below:

```
OPEN (511,FILE=drv%inpdir//'/arg_mis.dat',
 ↪  FORM='unformatted',STATUS='old',ERR=1111)
READ (511) arg%no
READ (511)
 ↪  &
  arg%ino(1:arg%no), arg%flg(1:arg%no),
   ↪  arg%par(1:arg%no)  &
  ,arg%lon(1:arg%no), arg%lat(1:arg%no)
   ↪  &
  ,arg%dpt(1:arg%no), arg%tim(1:arg%no)
   ↪  &
  ,arg%val(1:arg%no), arg%bac(1:arg%no)
   ↪  &
  ,arg%err(1:arg%no), arg%res(1:arg%no)
   ↪  &
  ,arg%ib(1:arg%no), arg%jb(1:arg%no), arg%kb(1:arg%no)
   ↪  &
  ,arg%pb(1:arg%no), arg%qb(1:arg%no), arg%rb(1:arg%no)
CLOSE (511)
```

Here below there is a brief explanation of the variable read:

- *no* : number of observations present in the file (64-bit integer)
- *ino* : Profile number (64-bit integer)
- *flg* : Validity flag (1: good, 0: bad) (64-bit integer)
- *par* : Variable parameter (1: temperature, 2: salinity) (64-bit integer)
- *lon* : Longitude (64-bit real)
- *lat* : Latitude (64-bit real)
- *dpt* : Depth (64-bit real)
- *tim* : time (julian day with respect to 1-1-1950) (64-bit real)
- *val* : value of the observation (64-bit real)
- *bac* : value of the model (64-bit real)

- *err* : observational error (meaningful only if *err_from_file* is True in *err_lst* namelist) (64-bit real)
- *res* : misfit (64-bit real)
- *ib* : i index of the nearest west point (64-bit integer)
- *jb* : j index of the nearest south point (64-bit integer)
- *kb* : k index of the nearest point below (64-bit integer)
- *pb* : distance from the nearest west point (64-bit real)
- *qb* : distance from the nearest south point (64-bit real)
- *rb* : distance from the nearest point below (64-bit real)

Note: *ib, jb, kb, pq, qb,rb* can be dummy values since the interpolation is performed inside the code.

8.2.2 SLA

```
OPEN (511,FILE=drv%inpdir//'/sla_mis.dat',
 ↪  FORM='unformatted',STATUS='old',ERR=1111)
READ(511) sla%no
READ (511)
 ↪  &
   sla%ino(1:sla%no),
    ↪  sla%ksat(1:sla%no),sla%flg(1:sla%no)&
  ,sla%lon(1:sla%no), sla%lat(1:sla%no),
   ↪  sla%tim(1:sla%no)&
  ,sla%val(1:sla%no), sla%bac(1:sla%no)
   ↪  &
  ,sla%err(1:sla%no), sla%res(1:sla%no)
   ↪  &
  ,sla%ib(1:sla%no), sla%jb(1:sla%no)
   ↪  &
  ,sla%pb(1:sla%no), sla%qb(1:sla%no),
   ↪  sla%dtm(1:sla%no)
CLOSE (511)
```

Here below there is a brief explanation of the variable read, only in the case they are different from the one listed in 8.2.1 section:

- *ino* : Track number (64-bit integer)

- *ksat* : Satellite ID (64-bit integer)
- *dtm* : Mean of the dynamic topography (64-bit real)

Note: All the observations belonging to the same track must be consecutive. Satellite ID should be in the same order of the *sla_sat_na* in the *&errlst* namelist.

8.2.3 SST

```
OPEN (511,FILE=drv%inpdir//'/sst_mis.dat',
 ↪  FORM='unformatted',STATUS='old',ERR=1111)
READ (511) sst%no
READ (511)
 ↪  &
       sst%ino(1:sst%no), sst%flg(1:sst%no),
        ↪  sst%par(1:sst%no) &
      ,sst%lon(1:sst%no), sst%lat(1:sst%no)
       ↪  &
      ,sst%dpt(1:sst%no), sst%tim(1:sst%no)
       ↪  &
      ,sst%val(1:sst%no), sst%bac(1:sst%no)
       ↪  &
      ,sst%err(1:sst%no), sst%res(1:sst%no)
       ↪  &
      ,sst%ib(1:sst%no), sst%jb(1:sst%no),
       ↪  sst%kb(1:sst%no)    &
      ,sst%pb(1:sst%no), sst%qb(1:sst%no),
       ↪  sst%rb(1:sst%no)
CLOSE (511)
```

Here below there is a brief explanation of the variable read, only in the case they are different from the one listed in 8.2.1 section:

- *ino* : Satellite ID (64-bit integer)

8.2.4 Glider velocity

```
OPEN (511,FILE=drv%inpdir//'/gvl_mis.dat',
 ↪  FORM='unformatted',STATUS='old',ERR=1111)
READ(511) gvl%no
READ(511)
 ↪  &
```

```
     gvl%ino(1:gvl%no), gvl%flg(1:gvl%no),
      ↪  gvl%par(1:gvl%no) &
    ,gvl%lon(1:gvl%no), gvl%lat(1:gvl%no),
     ↪  gvl%dpt(1:gvl%no) &
    ,gvl%tim(1:gvl%no), gvl%tms(1:gvl%no),
     ↪  gvl%tme(1:gvl%no) &
    ,gvl%val(1:gvl%no), gvl%bac(1:gvl%no),
     ↪  gvl%res(1:gvl%no) &
    ,gvl%err(1:gvl%no), gvl%nav(1:gvl%no)
     ↪  &
    ,gvl%ib(1:gvl%no), gvl%jb(1:gvl%no)
     ↪  &
    ,gvl%pb(1:gvl%no), gvl%qb(1:gvl%no)
CLOSE (511)
```

Here below there is a brief explanation of the variable read, only in the case they are different from the one listed in 8.2.1 section:

- *tms* : Starting time for averaging (64-bit real)
- *tme* : Final time for averaging (64-bit real)
- *nav* : Number of time steps for averaging (64-bit integer)

8.2.5 Drifter velocity

```
OPEN (511,FILE=drv%inpdir//'/vdr_mis.dat',
 ↪  FORM='unformatted',STATUS='old',ERR=1111)
READ (511) vdr%no
READ(511)
 ↪  &
   vdr%ino(1:vdr%no), vdr%flg(1:vdr%no),
    ↪  vdr%par(1:vdr%no) &
  ,vdr%lon(1:vdr%no), vdr%lat(1:vdr%no),
   ↪  vdr%dpt(1:vdr%no) &
  ,vdr%tim(1:vdr%no), vdr%tms(1:vdr%no),
   ↪  vdr%tme(1:vdr%no) &
  ,vdr%val(1:vdr%no), vdr%bac(1:vdr%no)
   ↪  &
  ,vdr%err(1:vdr%no), vdr%nav(1:vdr%no)
   ↪  &
```

```
   ,vdr%ib(1:vdr%no), vdr%jb(1:vdr%no)
    ↪  &
   ,vdr%pb(1:vdr%no), vdr%qb(1:vdr%no)
CLOSE (511)
```

Note: the only difference with respect to glider velocity 8.2.4 is the absence of the misfit that is computed inside the code

8.2.6 Argo and Drifters trajectory file

```
OPEN (511,FILE=drv%inpdir//'/tra_obs.dat',
 ↪  FORM='unformatted',STATUS='old',ERR=1111)
READ (511) tra%no, tra%nc, tra%nt, tra%im, tra%jm,
 ↪  tra%km, tra%dpt
READ (511)  tra%ino, tra%flg, tra%tim, tra%dtm,      &
            tra%loi, tra%lai, tra%lof, tra%laf,      &
            tra%err, tra%lob, tra%lab,               &
            tra%xob, tra%xmn, tra%erx,               &
            tra%yob, tra%ymn, tra%ery
 CLOSE (511)
```

Here below there is a brief explanation of the variable read:

- *no* : number of all observations present in the file (64-bit integer)
- *nc* : number of good observations present in the file (64-bit integer)
- *nt* : number of time step (64-bit integer)
- *im* : dimension of the grid in x (64-bit integer)
- *jm* : dimension of the grid in y (64-bit integer)
- *km* : dimension of the grid in vertical (64-bit integer)
- *ino* : profile number (64-bit integer)
- *flg* : quality flag (1: good, 0: bad) (64-bit integer)
- *tim* : time of the duration of the trajectory (64-bit real)
- *dtm* : time spent under surface (64-bit real)
- *loi* : initial observed longitude (64-bit real)
- *lai* : initial observed latitude (64-bit real)
- *lof* : final observed longitude (64-bit real)

- *laf* : final observed latitude (64-bit real)
- *err* : observational error in meters (64-bit real)
- *lob* : longitude of simulated position (64-bit real)
- *lab* : latitude of simulated position (64-bit real)
- *xob* : observed value in x direction (64-bit real)
- *xmn* : x coordinate of mean trajectory position (64-bit real)
- *erx* : observational error in x direction (64-bit real)
- *yob* : observed value in y direction (64-bit real)
- *ymn* : y coordinate of mean trajectory position (64-bit real)
- *ery* : observational error in y direction (64-bit real)

8.3 Output files

There are two types of output files the one related to the increments and one related to the diagnostics. The former are defined on the model space and are NetCDF files, the latter on the observational space and are NetCDF, simple binary or even ASCII file.

8.3.1 Increments

Increment files are five: one for each variable and they are listed below:

- *corr_tem.nc* for temperature
- *corr_sal.nc* for salinity
- *corr_uvl.nc* for zonal velocity
- *corr_vvl.nc* for meridian velocity
- *corr_eta.nc* for SLA

The variable name corresponds to the second part of the filename.

8.3.2 Diagnostics

- *OceanVar.diagnostics_####* is the standard output diagnostic file. It is divided per tile and is form of ASCII file.

- *iterated.dat* file contains information concerning the convergence of the minimizer.
- *file concerning the statistics of the output*. They can be in ASCII, simple bianary or NetCDF files.

  simple binary statistics of all instruments are in unique file called *obs.dat*

  ASCII they are divided per type of instrument (i.e *sla_mis.dat*, *arg_mis.dat*)

  NetCDF they are divided per type of instrument but currently they are available only for SLA, Argo, Glider, XBT and SLA.

For a complete description the reader should refer to the reference manual in the *html* directory and in particular to the routine *write_dia.F90* and *obs_str.F90*

Part III

Reproducibility

9 Reproducibility vs MPI reproducibility tradeoff

The possibility to opt for bit-for-bit reproducibility across parallel MPI processes is available in OceanVar at compile-time at the expense of performance: Across a range of tested configurations (see Github), bitwise equal output is guaranteed when changing the number of MPI processes and the domain decomposition across these processes, henceforth called **MPI-reproducibility**.

For example, when using 4 processes, OceanVar gives bitwise equal output irrespective of the domain decomposition being in a 2x2 manner (square) or a 4x1 (rectangular).

Bitwise reproduciblity might be useful for debugging purposes or to increase credibility of the model. We warn, however, that turning reproducibility on does not increase the precision of the problematic calculations. Instead it "forces" reproducibility by doing calculations in a sequential manner even if multiple processes are used. Therefore, MPI reproducibility comes at the cost of performance.

9.1 (De)Activating MPI-reproducibility

MPI-reproducibility is activated by default (at compile time), and can be deactivated by setting the $\mathrm{REPRODUCIBILITY}$ variable to *false* in the $\mathrm{src/configuration.mk}$ script before compiling.

If your configuration (computer, OS, library version, ...) is not in the list of configurations for which MPI reproducibility is guaranteed, than it is highly likely that differences between runs are decreased, if not bitwise equal.

9.1.1 Configurations

Guarantees for BR are given across the following sets of configurations. A “ground truth” per computer is given by the configuration.

9.2 Testing reproducibility with the test suite

In the OceanVar github, there is a test suite which automates the running, calculation and plotting of differences between several different runs. These runs can differ in which compiled executable, which OceanVar input data, which namelist options and which MPI processes are used.

A complete user guide can be found in the $\mathrm{README.md}$ file in the test suite directory, or equivalently as shown on the GitHub, helping the user to test easily test for bitwise reproducibility.

References

[DP08] Srdjan Dobricic and Nadia Pinardi. "An oceanographic three-dimensional variational data assimilation scheme". In: *Ocean modelling* 22.3-4 (2008), pp. 89–105.

[KR71] Michael Kwizak and André J Robert. "A semi-implicit scheme for grid point atmospheric models of the primitive equations". In: *Monthly Weather Review* 99.1 (1971), pp. 32–36.

[OS08] Peter R. Oke and Pavel Sakov. "Representation Error of Oceanic Observations for Data Assimilation". In: *Journal of Atmospheric and Oceanic Technology* 25.6 (June 2008), pp. 1004–1017. ISSN: 0739-0572. DOI: 10.1175/2007JTECHO558.1. URL: https://journals.ametsoc.org/doi/full/10.1175/2007JTECHO558.1 (visited on 06/07/2019).

[Sto+10] Andrea Storto et al. "Assimilating Along-Track Altimetric Observations through Local Hydrostatic Adjustment in a Global Ocean Variational Assimilation System". In: *Monthly Weather Review* 139.3 (Dec. 2010), pp. 738–754. ISSN: 0027-0644. DOI: 10.1175/2010MWR3350.1. URL: https://journals.ametsoc.org/doi/10.1175/2010MWR3350.1 (visited on 06/07/2019).

[Sto16] Andrea Storto. "Variational quality control of hydrographic profile data with non-Gaussian errors for global ocean variational data assimilation systems". In: *Ocean Modelling* 104 (Aug. 2016), pp. 226–241. ISSN: 1463-5003. DOI: 10.1016/j.ocemod.2016.06.011. URL: https://www.sciencedirect.com/science/article/pii/S1463500316300580 (visited on 03/10/2025).

[WC01] Anthony Weaver and Philippe Courtier. “Correlation modelling on the sphere using a generalized diffusion equation”. In: *Quarterly Journal of the Royal Meteorological Society* 127.575 (2001), pp. 1815–1846.

[WVA03] AT Weaver, Jérôme Vialard, and DLT Anderson. “Three-and four-dimensional variational assimilation with a general circulation model of the tropical Pacific Ocean. Part I: Formulation, internal diagnostics, and consistency checks”. In: *Monthly Weather Review* 131.7 (2003), pp. 1360–1378.