Deliverables D3.2.1 and D3.2.2:

Implementation of the weak constraint 4D-Var in NEMOVAR

Bénédicte Lemieux-Dudon, Arthur Vidard
LJK

VODA: ANR-08-COSI-016

Contents

1 Introduction **4**

2 Theory **5**

2.1 Model error and notations 5

2.1.1 The dynamical model error 5

2.1.2 Applied notations and configuration 6

2.2 Weak constraint 4D-Var 7

2.2.1 The $\mathbf{4D-Var}_{\eta}$ method: augmented initial state vector with $N-1$ model error vectors η_k at each time step 7

2.2.2 The $\mathbf{4D-Var}_{\mathbf{x}}$ method: augmented initial state vector with $N-1$ state vectors at each time step 9

2.2.3 $\mathbf{4D-Var}_{\mathbf{x_k}}$: augmented initial state vector with picked states along the trajectory 10

2.3 Weak constraint incremental 4D-Var 12

2.3.1 Incremental 4D-var: recall and notations 12

2.3.2 Weak constraint incremental 4D-Var with augmented state vector of type $\mathbf{4D-Var}_{\mathbf{x_k}}$ 15

2.3.3 Weak constraint incremental 4D-Var of type $\mathbf{4D-Var}_{\eta_\mathbf{k}}$ 19

2.3.4 Comments on Weak constraint incremental 4D-Var of types $\mathbf{4D-Var}_{\mathbf{x_k}}$ and $\mathbf{4D-Var}_{\eta_\mathbf{k}}$ 25

2.3.5 Weak constraint incremental 4D-Var with augmented state vector of type $\mathbf{4D-Var}_{\mathbf{x_k}}$ $B^{1/2}$ condtioning 27

2.3.6 Weak constraint incremental 4D-Var of type $\mathbf{4D-Var}_{\eta_\mathbf{k}}$ $B^{1/2}$ conditioning 30

3 Implementation of the weak constraint 4D-Var with NEMO/NEMOVAR **32**

3.1 NEMO and NEMOVAR: general comments 32

3.1.1 Inner loop structure and minimizer drivers 33

3.1.2 Model and observation space vectors 34

3.1.3 Key difference between the congrad and cgmod inner drivers 35

3.1.4 Routines `simvar` to calculate the cost function gradient 36

3.1.5 Model and observation space vectors and the `ctlvec` structured type 36

3.1.6 Model and observation space vectors and file storage 38

3.2 Overall strategy to implement the weak constraint 4D-Var . . . 41
3.2.1 Nemo outer loop: a single run over the assimilation window . . . 42
3.2.2 No split strategy and consequences . . . 42
3.3 NEMO outer loop: analysis and implementation . . . 45
3.3.1 The increment modules: description . . . 45
3.3.2 Increment module: weak constraint 4D-Var implementation . . . 48
3.4 NEMOVAR inner loop: analysis and implementation . . . 51
3.4.1 Quantities and code naming . . . 51
3.4.2 Routine nemovar_init . . . 52
3.4.3 CONGRAD inner loop driver . . . 63
3.4.4 CGMOD inner loop driver . . . 67
3.4.5 What to change in CONGRAD and CGMOD inner drivers . . . 73
3.4.6 The simulation routines `simvar_x2x` and `simvar_y2x` . . . 73

Appendices **86**

A First Appendix: Weak 4D-Var implementation issues **86**
A.1 Augmented size of vectors of the model space . . . 86
A.1.1 Arrays of oceanographic field: dimension issue . . . 86
A.1.2 Off-line communication files: the record dimension issue . . . 89
A.1.3 Weak 4D-Var implementation: off-line communication files and time record issue . . . 91
A.1.4 Issues connected to J and ∇J calculation and strategy . . . 95

B Second Appendix: details on some features of NEMO/NEMOVAR **97**
B.1 Observation space vectors and corresponding variables in the code . . . 97
B.1.1 Key observation space vectors . . . 97
B.1.2 Storage strategy . . . 97
B.1.3 Illustration with profile observations . . . 98
B.2 The linearization of the balance operator at each inner loop iteration . . . 99

C Third Appendix **101**
C.1 Cost function approximation around the optimum . . . 101
C.1.1 Minimizer and pre-conditionning techniques . . . 101
C.1.2 NEMOVAR congrad minimizer . . . 102

D Fourth Appendix **103**
D.1 Topic to investigate . . . 103
D.1.1 Hints to plan the work . . . 103

Chapter 1

Introduction

In operational variational data assimilation applied to ocean modeling, background and observation errors are considered as dominant compared to other errors of the system. In particular, the dynamical model error is neglected. Such approximation is referered to **strong constraint** variational data assimilation. In the framework of 4D-Var schemes, strong constraint approaches require to restrict the length of the assimilation window, so that model errors may still be neglected at the end of the window. In particular, the strong constraint incremental 4D-Var approaches, requires short assimilation windows because one reduce the departure of the model and its linear tangent approximation.

In academic research, weak constraint data assimilation have been investigated since the seventies (Sasaki, 1970). Weak constraint 4D-Var consists in taking into account the dynamical model error. It means that ocean model equations are required to be satified only weakly.

Or goal is to investigate how to introduce weak constraint approach in the NEMO, NEMOVAR data assimilation sytem. NEMOVAR enables currently to run a strong constraint incremental 4D-Var, where the control applies on increments to 5 oceanographic fields initial condition. These fields are the temperature and salinity t, s, the horizontal velocities u, v, and the sea surface height η.

There exist several formulation of the weak constraint incremental 4D-Var formulations, which differ on the choice of augmented control vector. The aim of this document is to make a review of the different weak constraint formulations, and to inquire how to implement them in the NEMO/NEMOVAR system.

The first chapter of this document proposes a short review of the weak constraint 4D-Var formulations. The second chapter, is a step by step analysis of the NEMO and NEMOVAR 4D-Var sequence of instructions, to inquire what implementations have to be performed to enable a weak constraint assimilation system.

Chapter 2

Theory

2.1 Model error and notations

2.1.1 The dynamical model error

The true state of a system is described with $x^t(r,t)$ a vector field which depends on the space and time coordinates r and t. A dynamical model denoted $\mathcal{T}$ enables to transform the true vector state from time t to $t+dt$. We suppose that the dynamical model is imperfect, and we introduce the dynamical model error $\eta(r,t)$ as follows:

$$\begin{array}{ll} x^t(r,t+dt) & = \mathcal{T}\left(x(r,t),t\right)+\epsilon(r,t) \\ x^t(r,0) & = x_0^t \end{array} \tag{2.1}$$

Let us suppose that we have only acces to a space and time discrete representation of the true vector field. We denote x_k the discrete state vector of the system at time t_k, and set to n its number of spatial components $x(r_i,t_k)$ with $i=1,n$.

There exists a representation error between the true vector field $x^t(t_k)$ and its discrete vector representation x_k:

$$\epsilon_k^r = x_k - x^t(t_k)$$

In the following we omit the representation error.

We hereafter assume that the true state is the true discretized state vector denoted x_k^t. The discrete version of the dynamic model $\mathcal{T}$ is denoted $\mathcal{M}$. It enables to calculate the state vector x_{k+1} from x_k. There exists a dynamical model error because:

- the analytical equations from which is derived the model $\mathcal{M}$, do not capture the real system behavior,
- the dicretization of these analytical equations introduces errors,
- some model errors also come from an incomplete knowledge of the model parameters, and the forcing field evolution.

In the transition equation from states x_{k-1}^t and x_k^t, the model error is represented with vector η_k:

$$x_k^t = \mathcal{M}_{k-1,k}\left(x_{k-1}^t\right) + \eta_k \tag{2.2}$$

The model error indentically applies in the transition equation between states x_{k-1} and x_k:

$$\left\{ \begin{array}{ll} x_k & = \mathcal{M}_{k-1,k}\left(x_{k-1}\right) + \eta_k \\ x_{k=0} & = x_0 \end{array} \right. \tag{2.3}$$

For short, we may sometimes denote the above equation as follows:

$$\left\{ \begin{array}{ll} x_k & = \mathcal{M}_k\left(x_{k-1}\right)) + \eta_k \\ x_{k=0} & x_0 \end{array} \right. \tag{2.4}$$

2.1.2 Applied notations and configuration

In the next sections, we review the weak 4D-Var formalism of the litterature in the framework of the model described by equation (2.3). Let us define the space and time configuration:

- the assimilation window covers the time interval $[t_0, t_N = t_0 + N\,dt]$ with $N+1$ discrete time steps, and N time intervals
- the multivariate state vector is made of 4 fields defined on a spatial grid of size n,
- the number of components of the initial state vector x_0 is M with $M = 4 \times n$.

The model equation (2.3) is defined for k running from 0 to N. The last state x_N is not controled in the assimilation system. We have:

$$\begin{array}{lll} x_0 & = \mathbf{1}\, x0 & y_0^o \\ x_1 & = \mathcal{M}_{0,1}\left(x_0\right) + \eta_1 & y_1^o \\ x_2 & = \mathcal{M}_{1,2}\left(x_1\right) + \eta_2 & y_2^o \\ \vdots & \vdots & \vdots \\ x_k & = \mathcal{M}_{k-1,k}\left(x_{k-1}\right) + \eta_k & y_k^o \\ \vdots & & y_{N-1}^o \\ x_{N-1} & = \mathcal{M}_{N-2,N-1}\left(x_{N-2}\right) + \eta_{N-1} & \\ x_N & = \mathcal{M}_{N-1,N}\left(x_{N-1}\right) & \end{array} \tag{2.5}$$

We moreover denote the integration of the model from 0 to k as follows:

$$x_k = \mathcal{M}_{0,k}\left(x_0, \eta_0, \ldots, \eta_{k-1}\right) + \eta_k \tag{2.6}$$

which can be written :

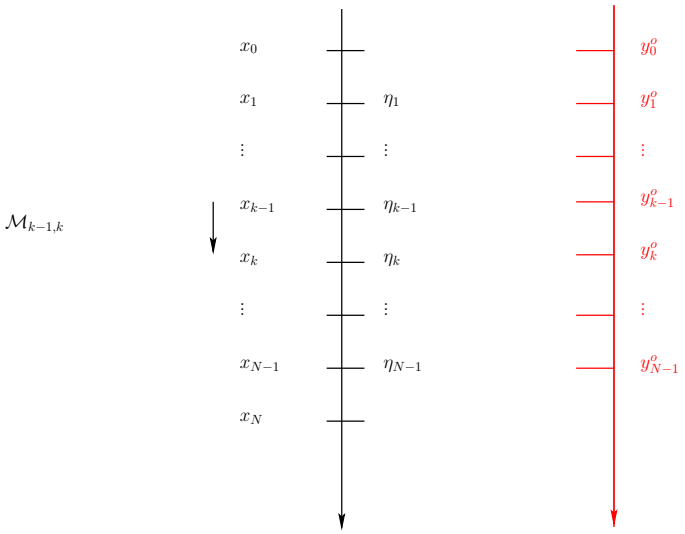


Figure 2.1: Time steps, state, error and observation vectors.

- for $k = N - 1$:

$$x_{N-1} = \mathcal{M}_{0,N-1}\left(x_0, \eta_0, \ldots, \eta_{N-2}\right) + \eta_{N-1} \tag{2.7}$$

- and for $k = N$:

$$x_N = \mathcal{M}_{0,N}\left(x_0, \eta_0, \ldots, \eta_{N-1}\right) \tag{2.8}$$

2.2 Weak constraint 4D-Var

In Trémolet (2006) and Trémolet (2007), several formalisms of the weak constraint 4D-var are shown. In this document, we adopt the naming conventions of Trémolet (2006). In sections 2.2.1, 2.2.2, we present the $\mathbf{4D-Var}_{\eta}$, $\mathbf{4D-Var}_{\mathbf{x}}$ methods. These two methods involve an augmented state vector with a problem size which is N times larger compared to the strong constraint 4D-var problem. In section 2.2.3, we slightly modify the previous approach in order to reduce the size of the assimilation system: we will call this method $\mathbf{4D-Var}_{\mathbf{x_k}}$.

2.2.1 The $\mathbf{4D-Var}_{\eta}$ method: augmented initial state vector with $N-1$ model error vectors η_k at each time step

Instead of only controling the initial state vector x_0, the $\mathbf{4D-Var}_{\eta}$ method applies on a state vector augmented with the $N-1$ error vectors η_k for time step index k from 1 to $N-1$:

$$X = \begin{pmatrix} x_0 \\ \eta_1 \\ \eta_2 \\ \vdots \\ \eta_k \\ \vdots \\ \eta_{N-1} \end{pmatrix} \tag{2.9}$$

To construct the cost function, we assume the classical hypothesis of non correlated background, observation and model errors. The size of each η vectors is M [1], the problem size is therefore $N \times M$.

The augmented size of the problem requires supplementary regularization terms. These terms are chosen such that the solution verifies the minimal norm constraint for the whole η vector defined as follows:

$$\eta = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_k \\ \vdots \\ \eta_{N-1} \end{pmatrix} \tag{2.10}$$

We further suppose that the η_k vectors are uncorrelated trough time. In that case, the dynamical model error covariance matrix $Q = E[\eta\,\eta^T]$, of size $((N-1)\times M)\times((N-1)\times M)$, is a diagonal block matrix. And the k^{th} diagonal Q block related to the time step k is:

$$Q_k = E[\eta_k\,\eta_k^T] \tag{2.11}$$

On these assumptions, the 4D-var cost function may then be split in 3 terms:

$$\begin{aligned} J\left(x_0, \eta_1, \ldots, \eta_k, \ldots, \eta_{N-1}\right) &= \|x_0 - x_0^b\|^2_{B^{-1}} \\ &+\tfrac{1}{2}\sum_{k=0}^{N-1} \|\mathcal{H}_k\left(\mathcal{M}_k\left(x_0, \ldots, \eta_{k-1}\right) + \eta_k\right) - y_k^o\|^2_{{R_k}^{-1}} \\ &+\tfrac{1}{2}\sum_{k=1}^{N-1} \|\eta_k\|^2_{{Q_k}^{-1}} \end{aligned} \tag{2.12}$$

Comment: one can note that the $\mathbf{4D-Var}_\eta$ approach is not suited for parallelization of the gradient computation. Let us call the q^{th} observation term of J:

$$J^{o,q} = \left(\mathcal{H}_k\left(\mathcal{M}_q\left(x_0, \eta_1, \ldots, \eta_{q-1}\right) + \eta_q\right) - y_k^o\right)^T\ {R_k}^{-1}\ \left(\mathcal{H}_k\left(\mathcal{M}_q\left(x_0, \eta_1, \ldots, \eta_{q-1}\right) + \eta_q\right) - y_k^o\right) \tag{2.13}$$

[1] Each state vector as M components which is the grid size multiplied by the number of multivariate variables.

It will contribute to the $q+1$ folowing gradient sections:

$$\nabla J^{o,q} = \begin{pmatrix} \frac{\partial J^{o,q}}{\partial x_0} \\ \frac{\partial J^{o,q}}{\partial \eta_1} \\ \frac{\partial J^{o,q}}{\partial \eta_2} \\ \vdots \\ \frac{\partial J^{o,q}}{\partial \eta_q} \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{2.14}$$

The q+1$^{\text{th}}$ will also contribute to these sections as well as depends on $\eta_1, \eta_2, \ldots, \eta_{k-1}$ and η_k.

2.2.2 The $4D-Var_x$ method: augmented initial state vector with $N-1$ state vectors at each time step

Similarly to the $\mathbf{4D-Var}_\eta$ formalism, the augmented state vector size is $N \times M$, but it is now made of the initial state vector x_0 plus the $N-1$ state vectors at time $t_1, \ldots, t_{N-1}$ of the trajectory:

$$X = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_{N-1} \end{pmatrix} \tag{2.15}$$

This weak constraint 4D-var formalism is called $\mathbf{4D-Var_x}$.
The relationship 2.3 involves:

$$\begin{array}{ll} \eta_1 & = x_1 - \mathcal{M}_1(x_0) \\ \ldots & \ldots \\ \eta_k & = x_k - \mathcal{M}_k(x_{k-1}) \\ \ldots & \ldots \\ \eta_{k+1} & = x_{k+1} - \mathcal{M}_{k+1}(x_k) \\ \ldots & \ldots \\ \eta_{N-1} & = x_{N-1} - \mathcal{M}_{N-1}(x_{N-2}) \end{array}$$

Comment:

- if we make the assumptions of section 2.2.2, an if we apply the same regularization approach, both methods $\mathbf{4D-Var_x}$ and $\mathbf{4D-Var}_\eta$ are equivalent: instead of controling the $N-1$ error vectors η_k, $N-1$ state vectors x_k are controlled. The application of the minimal norm regularization on the whole η vector can be equivalently applied on the right hand side of equations (2.16) with the same Q_k norm, which gives the following cost function is therefore:

$$\begin{array}{ll} J\left(x_0, x_1, \ldots, x_k, \ldots, x_{N-1}\right) & = \|x_0 - x_0^b\|^2_{B^{-1}} \\ & +\frac{1}{2}\sum_{k=1}^{N-1} \|x_k - \mathcal{M}_k\left(x_{k-1}\right)\|^2_{Q_k} \\ & +\frac{1}{2}\sum_{k=0}^{N-1} \|\mathcal{H}_k\left(x_k\right) - y_k^o\|^2_{R_k{}^{-1}} \end{array} \tag{2.16}$$

- This approach is more suited for parallelization.

2.2.3 $\mathbf{4D-Var_{x_k}}$: augmented initial state vector with picked states along the trajectory

The methods applied in the previous sections 2.2.1 and 2.2.2 involve large problem size. Instead of controling $N-1$ supplementary vectors at each time steps, one can control a smaller sample of state vectors along the trajectory.

We developpe hereafter the equations of such method when the initial state vector x_0 is augmented with only two supplementary state vectors picked at time steps t_{n_β} and t_{n_γ}. It leads to three assimilation subwindows with trajectories that may show discontinuity at subwindow junctions. The state vector trajectories are denoted x_α, x_β and x_γ with initial state written $x_{\alpha,0}$, $x_{\beta,0}$ and $x_{\gamma,0}$ respectively.

The x^α trajectory has a standard background denoted x^b. On the contrary, x^β and x_γ has non standard backgrounds: their background is defined as the last state predicted by the model at the end of the previous subtrajectory: $\mathcal{M}_{0,n_\alpha}\left(x_{\alpha,0}\right)$ and $\mathcal{M}_{n_\alpha,n_\beta}x_{\beta,0}$.

Table 2.1 and figure 2.2 shows a summary of the quantities involved.

trajectory	initial state	background
x_α	x_0^α	x_0^b
x_β	x_0^β	$\mathcal{M}_{0,n_\alpha}(x_0^\alpha)$
x_γ	x_0^γ	$\mathcal{M}_{n_\alpha,n_\beta}(x_0^\beta)$

Table 2.1: Quantities involved in the $\mathbf{4D-Var_{x_k}}$ example with 2 supplementary controled state vectors.

In such representation, we have three distinct model equations:

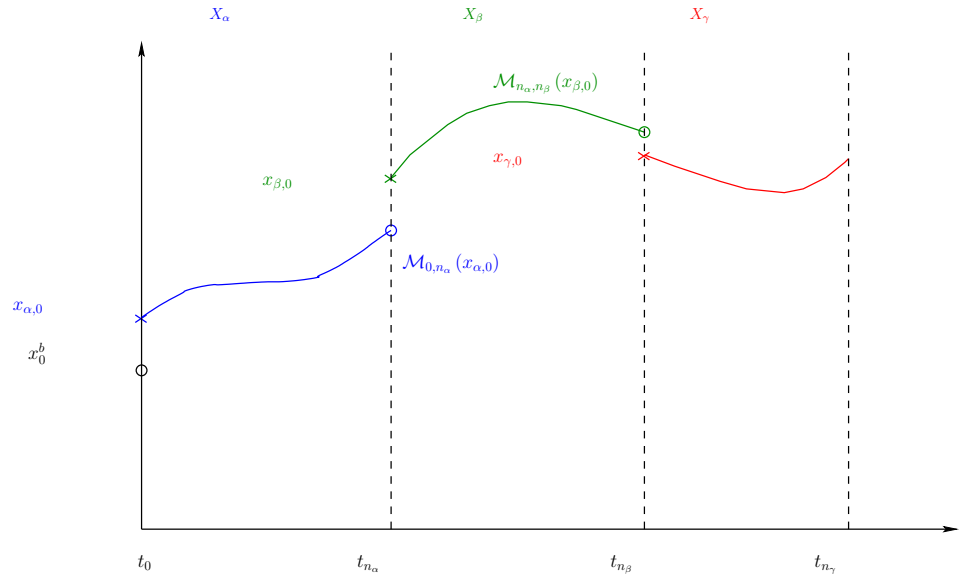


Figure 2.2: Trajectories of the $\mathbf{4D-Var_{x_k}}$ example with 2 supplementary controled state vectors.

$$\begin{array}{lll} x_{\alpha,k} & = \mathcal{M}_{0,k}\left(x_{\alpha,0}\right) & \text{for} \quad k=0 \quad \text{to} \quad n_\alpha \\ x_{\beta,k} & = \mathcal{M}_{n_\alpha,k}\left(x_{\beta,0}\right) & \text{for} \quad k=n_\alpha \quad \text{to} \quad n_\beta \\ x_{\gamma,k} & = \mathcal{M}_{n_\beta,k}\left(x_{\gamma,0}\right) & \text{for} \quad k=n_\beta \quad \text{to} \quad n_\gamma \end{array}$$

If we assume that the three background states show no correlation of their errors, the corresponding cost function can be written:

$$\begin{array}{ll} J\left(x_{\alpha,0},x_{\beta,0},x_{\gamma,0}\right) & = \frac{1}{2}\left(x_{\alpha,0}-x^b\right)^T {Q_\alpha}^{-1}\left(x_{\alpha,0}-x^b\right) \\ & +\frac{1}{2}\left(x_{\beta,0}-M_{0,n_\alpha}(x_{\alpha,0})\right)^T {Q_\beta}^{-1}\left(x_{\beta,0}-M_{0,n_\alpha}(x_{\alpha,0})\right) \\ & +\frac{1}{2}\left(x_{\gamma,0}-M_{n_\alpha,n_\beta}(x_{\beta,0})\right)^T {Q_\gamma}^{-1}\left(x_{\gamma,0}-M_{n_\alpha,n_\beta}(x_{\beta,0})\right) \\ & +\frac{1}{2}\sum_{k=0}^{n_\alpha-1}\left\|y_k^o-H_k\left(M_{0,k}(x_{\alpha,0})\right)\right\|^2_{{R_k}^{-1}} \\ & +\frac{1}{2}\sum_{k=n_\alpha}^{n_\beta-1}\left\|y_k^o-H_k\left(M_{n_\alpha,k}(x_{\beta,0})\right)\right\|^2_{{R_k}^{-1}} \\ & +\frac{1}{2}\sum_{k=n_\beta}^{n_\gamma-1}\left\|y_k^o-H_k\left(M_{n_\beta,k}(x_{\gamma,0})\right)\right\|^2_{{R_k}^{-1}} \end{array} \tag{2.17}$$

The weak constraint is introduced by adding penalty terms measuring the discontinuities at the junctions of the pieces of trajectories. We note that x_α, x_β and x_γ are not dymically dependent. They are correlated trough the two junction terms.

The penalty terms can also be viewed has a regularization term of minimal norm type, which is applied to the model error vectors η_{n_β} and η_{n_γ} :

$$\begin{array}{ll} \eta_{n_\beta} & = x_{\beta,0}-\mathcal{M}_{0,n_\alpha}x_{\alpha,0} \\ \eta_{n_\gamma} & = x_{\gamma,0}-\mathcal{M}_{n_\alpha,n_\beta}x_{\beta,0} \end{array}$$

Gradient The size of the control vector and the cost function gradient, is multiplied by the number of subwindows defined. In the three subwindow case, we have the following expression for $\nabla J = (\nabla_{x^\alpha} J, \nabla_{x^\beta} J)^T$:

$$\begin{array}{ll} \nabla_{x^\alpha} J & = {Q_\alpha}^{-1} \left(x^\alpha - x^b\right) + \sum_{k=0}^{n_\alpha - 1} {M_{0,k}}^T {H_k}^T {R_k}^{-1} \left(H_k \left(M_{0,k}(x^\alpha)\right) - y_k^o\right) \\ & + M_{0,k}^T {Q_\alpha}^{-1} \left(M_{0,n_\alpha}(x^\alpha) - x^\beta\right) \\ \nabla_{x^\beta} J & = {Q_\beta}^{-1} \left(x^\beta - M_{0,n_\alpha}(x^\alpha)\right) + \sum_{k=n_\alpha}^{n_\beta - 1} {M_{n_\alpha,k}}^T {H_k}^T {R_k}^{-1} \left(H_k \left(M_{n_\alpha,k}(x^\beta)\right) - y_k^o\right) \end{array} \tag{2.18}$$

Hessian It has been shown that the Hessian $\mathcal{H}$ of the assimilation system suffers from a bad conditionning number. Let us calculate the hessian blocks $\mathcal{H}_{\alpha,\alpha}$, $\mathcal{H}_{\beta,\beta}$, $\mathcal{H}_{\alpha,\beta}$ in our simple case:

$$\mathcal{H}_{\alpha,\alpha} = \frac{\partial^2 J}{\partial^2 x^\alpha} \quad \mathcal{H}_{\beta,\beta} = \frac{\partial^2 J}{\partial^2 x^\beta} \quad \mathcal{H}_{\alpha,\beta} = \frac{\partial^2 J}{\partial x^\alpha \partial x^\beta} \quad \mathcal{H}_{\beta,\alpha} = {\mathcal{H}_{\alpha,\beta}}^T \tag{2.19}$$

to construct the $\mathcal{H}$ auto-adjoint operator (see appendix):

$$\mathcal{H} = \begin{pmatrix} \mathcal{H}_{\alpha,\alpha} & \mathcal{H}_{\alpha,\beta} \\ \mathcal{H}_{\beta,\alpha} & \mathcal{H}_{\beta,\beta} \end{pmatrix} \tag{2.20}$$

$$\begin{array}{ll} \mathcal{H}_{\alpha,\alpha} & = {Q_\alpha}^{-1} + \sum_{k=0}^{n_\alpha - 1} {M_{0,k}}^T {H_k}^T {R_k}^{-1} M_{0,k} H_k \\ & + {M_{0,n_\alpha}}^T {Q_\beta}^{-1} M_{0,n_\alpha} \\ \mathcal{H}_{\beta,\beta} & = {Q_\beta}^{-1} + \sum_{k=n_\alpha}^{n_\beta - 1} {M_{0,k}}^T {H_k}^T {R_k}^{-1} M_{0,k} H_k \\ \mathcal{H}_{\beta,\alpha} & = -{M_{0,n_\alpha}}^T {Q_\beta}^{-1} \\ \mathcal{H}_{\alpha,\beta} & = -{Q_\beta}^{-1} M_{0,n_\alpha} \end{array} \tag{2.21}$$

This result is consistent with the auto-adjoint property of $\mathcal{H}$, in particular:

$$\begin{array}{ll} {\mathcal{H}_{\alpha,\beta}}^T & = -{Q_\beta}^{-1} M_{0,n_\alpha} \\ & = -{M_{0,n_\alpha}}^T {Q_\beta}^{-1} \\ & = \mathcal{H}_{\beta,\alpha} \end{array} \tag{2.22}$$

2.3 Weak constraint incremental 4D-Var

In this section, we develope the equations for the weak constraint incremental 4D-Var. Before that, let us make a short recall on the incremental 4D-Var.

2.3.1 Incremental 4D-var: recall and notations

Incremental 4D-var is applied when the model operators (observation and/or dynamic models) are non linear, which implies a non quadratic $\mathcal{J}$ cost function. The minimization of $\mathcal{J}$ is performed with an iterative process, through successive minimization of quadratic approximations of $\mathcal{J}$.

The order zero of the iterative process starts with the background state vector x_0^b and consists in searching $\delta x_0^{(1)}$ such that $x_0^{(1)} = x_0^b + \delta x_0^{(1)}$ is the minimum of the $J^{(1)}$, the first quadratic approximate of $\mathcal{J}$. Each quadratic approximation corresponds to one incremental 4D-Var outer loop.

The g^{th} outer loop starts with ${x_0}^{(g-1)}$, the $(g-1)^{\text{th}}$ estimate of the minimum of $\mathcal{J}$. The g^{th} quadratic approximation $J^{(g)}$ is built through a linearization of the model operators (dynamic and observation) around ${x_0}^{(g-1)}$ (which is usually called **reference state**), by applying a small perturbation $\delta {x_0}^{(g)}$ on ${x_0}^{(g-1)}$ in the $\mathcal{J}$ function:

$$\begin{aligned} {x_0}^{(g)} &= {x_0}^{(g-1)} + \delta {x_0}^{(g)} \\ &= {x_0}^{r} + \delta {x_0}^{(g)} \end{aligned} \tag{2.23}$$

The $J^{(g)}$ quadratic approximation minimization provides the $\delta {x_0}^{(g)}$ optimum value, the new current estimate of the minimum of $\mathcal{J}$ being ${x_0}^{(g)}$. The minimization of the $J^{(g)}$ functions is also an iterative process whose loops are the so-called incremental 4D-Var inner loops. The sequence outer loop + inner loops is repeated untill the current iterate ${x_0}^{(g)}$ meets a certain stop criterion (usually a maximum number of outer loops). When the criterion is met, let say after N outer loops, the final increment is the sum of partial increments:

$$\sum_{g=1}^{N} \delta {x_0}^{(g)} \tag{2.24}$$

and the analysed state vector (or control vector) x_0^a is given by:

$$x_0^a = x_0^b + \textstyle\sum_{g=1}^{N} \delta {x_0}^{(g)} \tag{2.25}$$

Let us now provide more details regarding the linearized model operators, and the related $\mathcal{J}$ quadratic approximates. At outer loop g, the linearization is applied around the current **reference state** $x_0^r = x_0^{(g-1)}$. The **Linear Tangent hypothesis** must be valid, which is to say:

$$\mathcal{M}_{0,k}\left({x_0}^{(g-1)} + \delta {x_0}^{(g)}\right) = \mathcal{M}_{0,k}\left({x_0}^{(g-1)}\right) + \left.\frac{\partial \mathcal{M}_{0,k}}{\partial x_0}\right|_{{x_0}^{(g-1)}} \delta {x_0}^{(g)} + \circ\left(\|\delta {x_0}^{(g)}\|\right) \tag{2.26}$$

and:

$$\begin{aligned} \mathcal{H}_k \circ \mathcal{M}_{0,k}\left({x_0}^{(g-1)} + \delta {x_0}^{(g)}\right) &= \mathcal{H}_k \circ \mathcal{M}_{0,k}\left({x_0}^{(g-1)}\right) \\ &+ \left.\frac{\partial \mathcal{H}_k}{\partial x_k}\right|_{{x_k}^{(g-1)}} \left.\frac{\partial \mathcal{M}_{0,k}}{\partial x_0}\right|_{{x_0}^{(g-1)}} \delta {x_0}^{(g)} + \circ\left(\|\delta {x_0}^{(g)}\|\right) \end{aligned} \tag{2.27}$$

with:

$${x_k}^{(g-1)} = \mathcal{M}_{0,k}\left({x_0}^{(g-1)}\right) \tag{2.28}$$

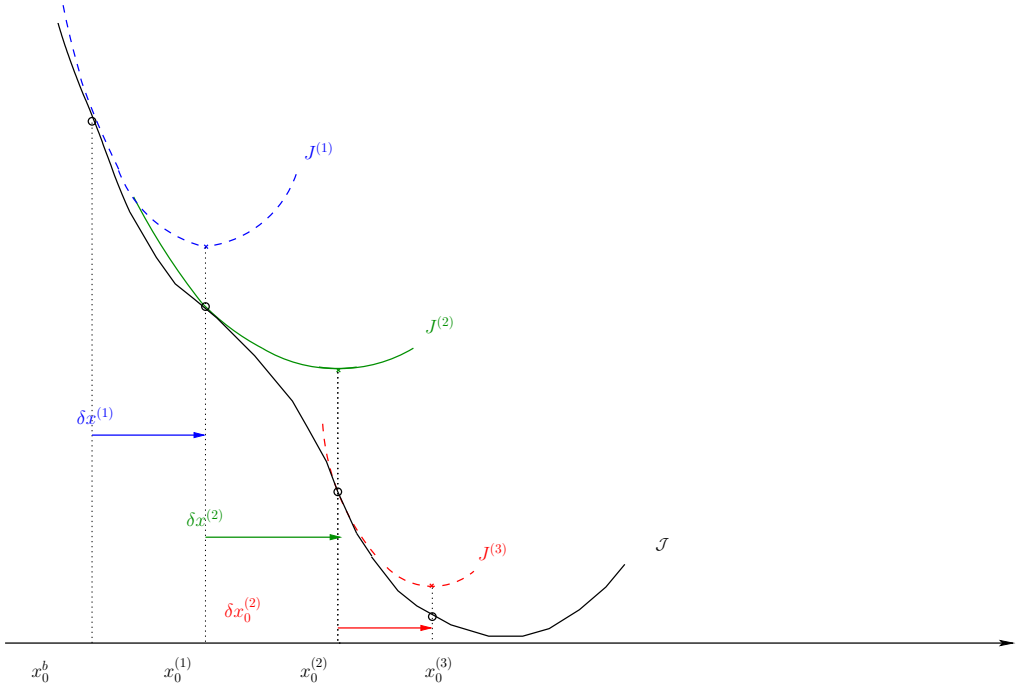


Figure 2.3: Three outer loops of the incremental 4D-Var.

Simplified notations

- We sometimes denote b the current reference state:

$$b = {x_0}^r = {x_0}^{(g-1)}$$

which is equivalent to say that:

$$b = {x_0}^b + \sum_{p=1}^{g-1} {\delta x_0}^{(p)} \tag{2.29}$$

- while the searched increment of outer loop g is simply denoted:

$${\delta x_0}^{(g)} = \delta x_0 \tag{2.30}$$

The quadratic cost function

$$\begin{aligned} J^{(g)} \quad &= \tfrac{1}{2} \left(x_0^{\alpha,r} + \delta x_0^{\alpha} - x_0^{\alpha,b} \right)^T {Q_\alpha}^{-1} \left(x_0^{\alpha,r} + \delta x_0^{\alpha} - x_0^{\alpha,b} \right) \\ &+ \tfrac{1}{2} \sum_{k=0}^{n_\alpha - 1} \left(y_k^o - \mathcal{M}_{k,0}(x_0^{\alpha,r}) - \mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0^{\alpha} \right)^T {R_k}^{-1} \left(y_k^o - \mathcal{M}_{k,0}(x_0^{\alpha,r}) - \mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0^{\alpha} \right) \end{aligned} \tag{2.31}$$

2.3.2 Weak constraint incremental 4D-Var with augmented state vector of type $4D-Var_{x_k}$

Let us come back to the the weak constraint 4D-Var formulation of type $\mathbf{4D-Var_{x_k}}$ shown in section 2.2.1 (augmented initial state vector with picked states along the trajectory), but this time in the incremental case.

Equations: no conditioning

We split the assimilation window in two subwindows only, to simplify equations. The two subtrajectories x^α and x^β can be discontinuous. The background vector has two components $x_0^{\alpha,b}$ and $x_0^{\beta,b}$:

$$x_0^b = \begin{pmatrix} x_0^{\alpha,b} \\ x_0^{\beta,b} \end{pmatrix} \tag{2.32}$$

The quadratic approximations of cost function $\mathcal{J}$ is $J^{(g)}$, and we have:

- 1st outer loop ($g=1$): $J^{(1)}$ is linearized around x_0^b with

$$\delta x_0^1 = \arg\min J^{(1)}$$

- g^{th} outer loop: $J^{(g)}$ is linearized around $x_0^{(g-1)}$ with

$$x_0^{(g-1)} = x_0^b + \sum_{k=1}^{g-1} \delta x_0^k \tag{2.33}$$

and

$$\delta x_0^g = \arg\min J^{(g)}$$

With augmented vector notations, for the g^{th} outer loop, we have:

- the reference state vector is:

$$x_0^{(g-1)} = \begin{pmatrix} x_0^{\alpha,(g-1)} \\ x_0^{\beta,(g-1)} \end{pmatrix} \tag{2.34}$$

- and the searched increment vector is:

$$\delta x_0^{(g)} = \begin{pmatrix} \delta x_0^{\alpha,(g)} \\ \delta x_0^{\beta,(g)} \end{pmatrix} \tag{2.35}$$

After inner loop minimization:

- the increment is:

$$\delta x_0^{(g)}$$

- the cumulated increment is:

$$\sum_{k=1}^{g} \delta x_0^k$$

- the new reference state is:

$$x_0^{(g)} = x_0^b + \sum_{k=1}^{g} \delta x_0^k$$

With augmented vector notations, for the g^{th} outer loop, we have:

$$x_0^{(g)} = \begin{pmatrix} x_0^{\alpha,(g-1)} + \delta x_0^{\alpha,(g)} \\ x_0^{\beta,(g-1)} + \delta x_0^{\beta,(g)} \end{pmatrix} \tag{2.36}$$

To simplify the notation, the reference state vector $x_0^{(g-1)}$ is now denoted by x_0^r, and the current searched increment is denoted by δx_0:

$$x_0{}^{(g)} = x_0^r + \delta x_0 = \begin{pmatrix} x_0^{\alpha,r} + \delta x_0^{\alpha} \\ x_0^{\beta,r} + \delta x_0^{\beta} \end{pmatrix} \tag{2.37}$$

Let us derive $J^{(g)}$ from $\mathcal{J}(x^{\alpha,r} + \delta x^{\alpha}, x^{\beta,r} + \delta x^{\beta})$:

$$\begin{aligned} \mathcal{J} \;&= \tfrac{1}{2} \left(x_0^{\alpha,r} + \delta x_0^{\alpha} - x_0^{\alpha,b} \right)^T Q_\alpha{}^{-1} \left(x_0^{\alpha,r} + \delta x_0^{\alpha} - x_0^{\alpha,b} \right) \\ &+ \tfrac{1}{2} \left(x_0^{\beta,r} + \delta x_0^{\beta} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r} + \delta x_0^{\alpha}) \right)^T Q_\beta{}^{-1} \left(x_0^{\beta,r} + \delta x_0^{\beta} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r} + \delta x_0^{\alpha}) \right) \\ &+ \tfrac{1}{2} \textstyle\sum_{k=0}^{n_\alpha - 1} \left(y_k^o - \mathcal{H}_k \left(\mathcal{M}_{0,k}(x_0^{\alpha,r} + \delta x_0^{\alpha}) \right) \right)^T R_k{}^{-1} \left(y_k^o - \mathcal{H}_k \left(\mathcal{M}_{0,k}(x_0^{\alpha,r} + \delta x_0^{\alpha}) \right) \right) \\ &+ \tfrac{1}{2} \textstyle\sum_{k=n_\alpha}^{n_\beta - 1} \left(y_k^o - \mathcal{H}_k \left(\mathcal{M}_{n_\alpha,k}(x_0^{\beta,r} + \delta x_0^{\beta}) \right) \right)^T R_k{}^{-1} \left(y_k^o - H_k \left(\mathcal{M}_{n_\alpha,k}(x_0^{\beta,r} + \delta x_0^{\beta}) \right) \right) \end{aligned} \tag{2.38}$$

This step consists in the linearization of the model operators $\mathcal{M}$ and $\mathcal{H}$ around the reference state [2] :

$$\begin{aligned} J^{(g)} \;&= \tfrac{1}{2} \left(x_0^{\alpha,r} + \delta x_0^{\alpha} - x_0^{\alpha,b} \right)^T Q_\alpha{}^{-1} \left(x_0^{\alpha,r} + \delta x_0^{\alpha} - x_0^{\alpha,b} \right) \\ &+ \tfrac{1}{2} \left(x_0^{\beta,r} + \delta x_0^{\beta} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) - \mathbf{M}_{n_\alpha,0}\, \delta x_0^{\alpha} \right)^T Q_\beta{}^{-1} \left(x_0^{\beta,r} + \delta x_0^{\beta} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) - \mathbf{M}_{n_\alpha,0}\, \delta x_0^{\alpha} \right) \\ &+ \tfrac{1}{2} \textstyle\sum_{k=0}^{n_\alpha - 1} \left(y_k^o - \mathcal{H}_k \mathcal{M}_{0,k}(x_0^{\alpha,r}) - \mathbf{H}_k\, \mathbf{M}_{0,k}\, \delta x_0^{\alpha} \right)^T R_k{}^{-1} \left(y_k^o - \mathcal{H}_k \mathcal{M}_{0,k}(x_0^{\alpha,r}) - \mathbf{H}_k\, \mathbf{M}_{0,k}\, \delta x_0^{\alpha} \right) \\ &+ \tfrac{1}{2} \textstyle\sum_{k=n_\alpha}^{n_\beta - 1} \left(y_k^o - \mathcal{H}_k \mathcal{M}_{n_\alpha,k}(x_0^{\beta,r}) - \mathbf{H}_k\, \mathbf{M}_{n_\alpha,k}\, \delta x_0^{\beta} \right)^T R_k{}^{-1} \left(y_k^o - \mathcal{H}_k \mathcal{M}_{n_\alpha,k}(x_0^{\beta,r}) - \mathbf{H}_k\, \mathbf{M}_{n_\alpha,k}\, \delta x_0^{\beta} \right) \end{aligned} \tag{2.39}$$

Let us further introduce the innovation vector $d_k^{\alpha,(g-1)}$ and $d_k{}^{\beta,(g-1)}$ (taken at reference state):

[2] We further omit the outer loop superscript exponent and simply denote the g^{th} searched increment δx_0.

$$d_k^{\alpha,(g-1)} = \begin{pmatrix} {d_k}^{\alpha,(g-1)} \\ {d_k}^{\beta,(g-1)} \end{pmatrix} = \begin{pmatrix} y_k^o - \mathcal{M}_{0,k}\left(x_0^{\alpha,r}\right) \\ y_k^o - \mathcal{M}_{0,k}\left(x_0^{\beta,r}\right) \end{pmatrix} \tag{2.40}$$

This gives in equation 2.39:

$$\begin{aligned} J^g \;&= \tfrac{1}{2}\left(x_0^{\alpha,r} + \delta x_0^\alpha - x_0^{\alpha,b}\right)^T {Q_\alpha}^{-1}\left(x_0^{\alpha,r} + \delta x_0^\alpha - x_0^{\alpha,b}\right) \\ &+\tfrac{1}{2}\left(x_0^{\beta,r} + \delta x_0^\beta - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) - \mathbf{M}_{n_\alpha,0}\,\delta x_0^\alpha)\right)^T {Q_\beta}^{-1}\left(x_0^{\beta,r} + \delta x_0^\beta - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) - \mathbf{M}_{n_\alpha,0}\,\delta x_0^\alpha)\right) \\ &+\tfrac{1}{2}\textstyle\sum_{k=0}^{n_\alpha-1}\left(\mathbf{H}_k\,\mathbf{M}_{0,k}\,\delta x_0^\alpha - {d_k}^\alpha\right)^T {R_k}^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{0,k}\,\delta x_0^\alpha - {d_k}^\alpha\right) \\ &+\tfrac{1}{2}\textstyle\sum_{k=n_\alpha}^{n_\beta-1}\left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,\delta x_0^\beta - {d_k}^\beta\right)^T {R_k}^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,\delta x_0^\beta - {d_k}^\beta\right) \end{aligned} \tag{2.41}$$

From equation 2.41 can be derived the linear gradient of J^g with respect to the two sections of the increment vector, δx_0^α and δx_0^β:

$$\begin{aligned} \nabla_{x^\alpha} J^g \;&= {Q_\alpha}^{-1}\left(x_0^{\alpha,r} + \delta x_0^\alpha - x_0^{\alpha,b}\right) \\ &-{\mathbf{M}_{0,n_\alpha}}^T {Q_\beta}^{-1}\left(x_0^{\beta,r} + \delta x_0^\beta - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) - \mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha\right) \\ &+\textstyle\sum_{k=0}^{n_\alpha-1}{\mathbf{M}_{0,k}}^T {\mathbf{H}_k}^T {R_k}^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{0,k}\,\delta x_0^\alpha - {d_k}^\alpha\right) \\ \nabla_{x^\beta} J^g \;&= {Q_\beta}^{-1}\left(x_0^{\beta,r} + \delta x_0^\beta - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) - \mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha\right) \\ &+\textstyle\sum_{k=n_\alpha}^{n_\beta-1}{\mathbf{M}_{n_\alpha,k}}^T {\mathbf{H}_k}^T {R_k}^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,\delta x_0^\beta - {d_k}^\beta\right) \end{aligned} \tag{2.42}$$

For algorithmic matters, it is a common practice to make appear the Hessian of the cost function as well as the initial gradient (i.e., before entering inner loops when $\delta x = 0$), in gradient equation (2.42). By doing this, we hilight the *Supplementary terms* introduced by the weak incremental 4D-Var $\left(\delta x_0^\alpha, \delta x_0^\beta\right)$ formulation:

$$\begin{aligned} \nabla_{x^\alpha} J^g \;&= \overbrace{{Q_\alpha}^{-1}\left(x_0^{\alpha,r} - x_0^{\alpha,b}\right) - \sum_{k=0}^{n_\alpha-1}{\mathbf{M}_{0,k}}^T {\mathbf{H}_k}^T {R_k}^{-1}\,{d_k}^\alpha}^{\text{Initial gradient}} \\ &\underbrace{+{Q_\alpha}^{-1}\left(\mathbf{I} + \sum_{k=0}^{n_\alpha-1} Q_\alpha\,{\mathbf{M}_{0,k}}^T {\mathbf{H}_k}^T {R_k}^{-1}\mathbf{H}_k\,\mathbf{M}_{0,k}\right)\delta x_0^\alpha}_{\text{Hessian}} \\ &\underbrace{-{\mathbf{M}_{0,n_\alpha}}^T {Q_\beta}^{-1}\left(x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) + \delta x_0^\beta - \mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha\right)}_{\textit{Supplementary term}} \end{aligned} \tag{2.43}$$

and:

$$\begin{aligned}\nabla_{x^\beta} J^g &= \overbrace{Q_\beta{}^{-1}\left(x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})\right) - \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1}\, d_k{}^\beta}^{\text{Initial gradient}} \\ &\underbrace{+ Q_\beta{}^{-1}\left(\mathbf{I} + \sum_{k=n_\alpha}^{n_\beta-1} Q_\beta\, \mathbf{M}_{n_\alpha,k}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1} \mathbf{H}_k\, \mathbf{M}_{n_\alpha,k}\right) \delta x_0^\beta}_{\text{Hessian}} \\ &\underbrace{- Q_\beta{}^{-1}\, \mathbf{M}_{0,n_\alpha}\, \delta x_0^\alpha}_{\textit{Supplementary term}}\end{aligned} \tag{2.44}$$

Always for algorithmic matters, let us try to get rid of matrix inversion, by multiplying gradient equation (2.43) by $\mathbf{Q}_\alpha^{-1}$, and equation (2.44) by $\mathbf{Q}_\beta^{-1}$:

$$\begin{aligned}Q_\alpha\, \nabla_{x^\alpha} J^g &= \left(x_0^{\alpha,r} - x_0^{\alpha,b}\right) - \textstyle\sum_{k=0}^{n_\alpha-1} Q_\alpha\, \mathbf{M}_{0,k}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1}\, d_k{}^\alpha \\ &+ \delta x_0^\alpha + \textstyle\sum_{k=0}^{n_\alpha-1} Q_\alpha\, \mathbf{M}_{0,k}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1} \mathbf{H}_k\, \mathbf{M}_{0,k}\, \delta x_0^\alpha \\ &- Q_\alpha\, \mathbf{M}_{0,n_\alpha}{}^T\, Q_\beta{}^{-1} \left(\underbrace{x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) + \delta x_0^\beta - \mathbf{M}_{0,n_\alpha}\, \delta x_0^\alpha}_{A}\right)\end{aligned} \tag{2.45}$$

and:

$$\begin{aligned}Q_\beta\, \nabla_{x^\beta} J^g &= \left(x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})\right) - \textstyle\sum_{k=n_\alpha}^{n_\beta-1} Q_\beta\, \mathbf{M}_{n_\alpha,k}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1}\, d_k{}^\beta \\ &+ \delta x_0^\beta + \textstyle\sum_{k=n_\alpha}^{n_\beta-1} Q_\beta\, \mathbf{M}_{n_\alpha,k}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1} \mathbf{H}_k\, \mathbf{M}_{n_\alpha,k}\, \delta x_0^\beta \\ &- \mathbf{M}_{0,n_\alpha}\, \delta x_0^\alpha\end{aligned} \tag{2.46}$$

In $\nabla_{x^\alpha} J^g$ gradient equation just above, we highlight one quantity (braces) designated by the letter A. In $\nabla_{x^\alpha} J^g$ equation, one can gather terms to retrieve quantity A:

$$\begin{aligned}Q_\beta\, \nabla_{x^\beta} J^g &= \left(\overbrace{x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) + \delta x_0^\beta - \mathbf{M}_{0,n_\alpha}\, \delta x_0^\alpha}^{A}\right) - \textstyle\sum_{k=n_\alpha}^{n_\beta-1} Q_\beta\, \mathbf{M}_{n_\alpha,k}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1}\, d_k{}^\beta \\ &+ \textstyle\sum_{k=n_\alpha}^{n_\beta-1} Q_\beta\, \mathbf{M}_{n_\alpha,k}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1} \mathbf{H}_k\, \mathbf{M}_{n_\alpha,k}\, \delta x_0^\beta\end{aligned} \tag{2.47}$$

This provides an algorithm, which avoid inverting the $\mathbf{Q}$ matrices:

1. calculate A:

$$\begin{aligned}A &= x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) + \delta x_0^\beta - \mathbf{M}_{0,n_\alpha}\, \delta x_0^\alpha \\ &= \textstyle\sum_{p=1}^{n-1} \delta x_0^{\beta,(p)} - \mathbf{M}_{0,n_\alpha}\, \sum_{p=1}^{n-1} \delta x_0^{\alpha,(p)} + \delta x_0^\beta - \mathbf{M}_{0,n_\alpha}\, \delta x_0^\alpha\end{aligned} \tag{2.48}$$

2. calculate $Q_\beta \nabla_{x^\beta} J^g$ (last subtrajectory first), where terms have been gathered to make appear the adjoint state $\delta x_0^{\beta *}$:

$$Q_\beta \nabla_{x^\beta} J^g = \left(\overbrace{x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) + \delta x_0^\beta - \mathbf{M}_{0,n_\alpha} \delta x_0^\alpha}^{A} \right) + Q_\beta \delta x_0^{\beta *} \tag{2.49}$$

3. calculate $Q_\alpha \nabla_{x^\alpha} J^g$, where one can find the adjoint state $\delta x_0^{\alpha *}$, and where $\mathbf{Q}_\beta^{-1} A$ is replaced by $\nabla_{x^\beta} J^g - \delta x_0^{\beta *}$ (see equation 2.49):

$$\begin{aligned} Q_\alpha \nabla_{x^\alpha} J^g &= \left(x_0^{\alpha,r} - x_0^{\alpha,b} + \delta x_0^\alpha \right) + Q_\alpha \delta x_0^{\alpha *} \\ &\quad - Q_\alpha \, \mathbf{M}_{0,n_\alpha}{}^T \left(\nabla_{x^\beta} J^g - \delta x_0^{\beta *} \right) \end{aligned} \tag{2.50}$$

Note that `cgmod` algorithm in NEMOVAR, calculate both $\nabla_x J^g$ and $\mathbf{Q} \nabla_x J^g$.

Comment about the 4D – Var$_{\mathbf{x_k}}$ weak incremental 4D-Var formulation

- The cost function observation term is split in independant parts, each of them being connected to one subtrajectory, and only depending on the local subtrajectory initial state. Numerically, it means that one can perform seperatly the minimization of each subwindow observation term (inner loop).

- One numerical problem concerns the required inversion of the square root error covariance matrices Q_α and Q_β.

2.3.3 Weak constraint incremental 4D-Var of type 4D – Var$_{\eta_\mathbf{k}}$

Cost function and gradient equations developped in section 2.3.2, can be rewritten with respect to the state vector augmented with model error $\left(x_0^\alpha, \eta^\beta\right)$. The link between the 2 formulations is given by equations:

$$\boxed{x_0^{\beta,(g)} = \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(g)}\right) + \eta^{\beta,(g)}} \tag{2.51}$$

where g stands for the g^{th} outer loop.

But we see further on that equation 2.51 is in fact approximated at the step of linearization of the cost function.

The background of the η state vector, is a **null** vector since no discontinuity exists at subtrajectories junction when starting the 4D-Var process:

$$\begin{aligned}\eta^{\beta,b} &= x_0^{\beta,(0)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(0)}\right)\\ &= x_0^{\beta,b} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,b}\right)\\ &= 0\end{aligned} \tag{2.52}$$

Let us now look at outer loops 1, 2 to generalize the results at any outer loops:

$$\eta^{\beta,(1)} = x_0^{\beta,(1)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(1)}\right) \tag{2.53}$$

$$\eta^{\beta,(2)} = x_0^{\beta,(2)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(2)}\right) \tag{2.54}$$

These equations can be linearized around the reference state, which gives for the former:

$$\begin{aligned}\eta^{\beta,(1)} &= x_0^{\beta,(1)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(1)}\right)\\ &\simeq x_0^{\beta,(0)} + \delta x_0^{\beta,(1)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(0)}\right) - \mathbf{M}_{0,n_\alpha}\delta x_0^{\alpha,(1)}\end{aligned} \tag{2.55}$$

and for the later:

$$\begin{aligned}\eta^{\beta,(2)} &= x_0^{\beta,(2)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(2)}\right)\\ &\simeq x_0^{\beta,(1)} + \delta x_0^{\beta,(2)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(1)}\right) - \mathbf{M}_{0,n_\alpha}\delta x_0^{\alpha,(2)}\end{aligned} \tag{2.56}$$

One recognizes $\eta^{\beta,(1)}$ vector in 2.56:

$$\begin{aligned}\eta^{\beta,(2)} &= x_0^{\beta,(2)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(2)}\right)\\ &\simeq x_0^{\beta,(1)} + \delta x_0^{\beta,(2)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(1)}\right) - \mathbf{M}_{0,n_\alpha}\delta x_0^{\alpha,(2)}\\ &\simeq \underbrace{x_0^{\beta,(1)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(1)}\right)}_{\eta^{\beta,(1)}} + \delta x_0^{\beta,(2)} - \mathbf{M}_{0,n_\alpha}\delta x_0^{\alpha,(2)}\end{aligned} \tag{2.57}$$

The $\eta^{\beta,(2)}$ increment expression of equation 2.54 can therefore be rewritten as follows:

$$\begin{aligned}\eta^{\beta,(2)} &= x_0^{\beta,(2)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(2)}\right)\\ &\simeq \eta^{\beta,(1)} + \underbrace{\delta x_0^{\beta,(2)} - \mathbf{M}_{0,n_\alpha}\delta x_0^{\alpha,(2)}}_{\delta\eta^{\beta,(2)}}\end{aligned} \tag{2.58}$$

One can generalize the above equation at any outer loop order.

Let us do it for the g^{th} outer loop, which shows how equation (2.51) is approximated:

$$\boxed{\begin{aligned}\eta^{\beta,(g)} &= x_0^{\beta,(g)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(g)}\right)\\ &\simeq \eta^{\beta,(g-1)} + \underbrace{\delta x_0^{\beta,(g)} - \mathbf{M}_{0,n_\alpha}\delta x_0^{\alpha,(g)}}_{\delta\eta^{\beta,(g)}}\end{aligned}} \tag{2.59}$$

and let us now repeat the process for any outer loop from $(g-1)$ to 1:

$$\begin{aligned}\eta^{\beta,(g-1)} &= x_0^{\beta,(g-1)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(g-1)}\right)\\ &\simeq \underbrace{x_0^{\beta,(0)} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,(0)}\right)}_{\eta^{\beta,(0)}=\eta^{\beta,b}=0} + \underbrace{\sum_{p=1}^{g-1}\delta x_0^{\beta,(p)} - \mathbf{M}_{0,n_\alpha}\sum_{p=1}^{g-1}\delta x_0^{\alpha,(p)}}_{\sum_{p=1}^{g-1}\delta\eta^{\beta,(p)}}\\ &\simeq \eta^{\beta,1} + \textstyle\sum_{p=2}^{g-1}\delta\eta^{\beta,(p)}\end{aligned} \tag{2.60}$$

As a result, two important equations provinding the link between formulations $\mathbf{4D-Var_{x_k}}$ and $\mathbf{4D-Var}_{\eta_\mathbf{k}}$, in the framework of the linearization of incremental 4D-Var:

$$\boxed{\delta\eta^{\beta,(g)} \simeq \delta x_0^{\beta,(g)} - \mathbf{M}_{0,n_\alpha}\delta x_0^{\alpha,(g)}} \tag{2.61}$$

$$\boxed{\sum_{p=1}^{g-1}\delta\eta^{\beta,(p)} = \sum_{p=1}^{g-1}\delta x_0^{\beta,(p)} - \mathbf{M}_{0,n_\alpha}\sum_{p=1}^{g-1}\delta x_0^{\alpha,(p)}} \tag{2.62}$$

Moreover, it highlights the already seen cumulative nature of the increment:

$$\boxed{\eta^{\beta,(g)} \simeq \eta^{\beta,b} + \sum_{p=1}^{g-1}\delta\eta^{\beta,(p)} + \delta\eta^{\beta,(g)}} \tag{2.63}$$

Cost function

Background terms of the $J^{(g)}$ function are therefore (omitting the g exponents on increments):

- unchanged regarding α subtrajectory:

$$J^{b,\alpha,(g)} = \|x_0^{\alpha,r} + \delta x_0^{\alpha} - x_0^{\alpha,b}\|^2_{\mathbf{Q}_\alpha}$$

- transformed as follows regarding the β subtrajectory:

$$\begin{aligned}J^{b,\beta,(g)} &= \|x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}\left(x_0^{\alpha,r}\right) + \delta x_0^{\beta} - \mathbf{M}_{0,n_\alpha}\delta x_0^{\alpha}\|^2_{\mathbf{Q}_\beta}\\ &= \|\eta^{\beta,r} + \delta\eta^{\beta}\|^2_{\mathbf{Q}_\beta}\end{aligned} \tag{2.64}$$

Regarding observation terms:

- the formulation is unchanged for terms related to observations dispatched along the α subtrajectory:

$$J^{o,\alpha,(g)} = \tfrac{1}{2}\textstyle\sum_{k=0}^{n_\alpha-1}\|y_k^o - \mathcal{H}_k\mathcal{M}_{0,k}(x_0^{\alpha,r}) - \mathbf{H}_k\,\mathbf{M}_{0,k}\,\delta x_0^{\alpha}\|^2_{R_k} \tag{2.65}$$

- the formulation is transformed for terms related to observations dispatched along the β subtrajectory, with first the non quadratic cost function:

$$\mathcal{J}^{o,\beta} = \frac{1}{2}\sum_{k=n_\alpha}^{n_\beta-1} \|\mathcal{H}_k\left(\mathcal{M}_{n_\alpha,k}(x_0^{\beta,r}+\delta x_0^\beta)\right) - y_k^o\|^2 \tag{2.66}$$

and second, the quadratic approximate:

$$J^{o,\beta,(g)} = \frac{1}{2}\sum_{k=n_\alpha}^{n_\beta-1} \|\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,\delta x_0^\beta - \left(y_k^o - \mathcal{H}_k\left(\mathcal{M}_{n_\alpha,k}(x_0^{\beta,r})\right)\right)\|^2 \tag{2.67}$$

Let us now introduce equation (2.61) in equation (2.67):

$$\begin{aligned} J^{o,\beta,(g)} &= \sum_{k=n_\alpha}^{n_\beta-1} \|\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,\left(\mathbf{M}_{0,n_\alpha}\delta x_0^\alpha + \delta\eta^\beta\right) - \left(y_k^o - \mathcal{H}_k\left(\mathcal{M}_{n_\alpha,k}(x_0^{\beta,r})\right)\right)\|^2 \\ &= \sum_{k=n_\alpha}^{n_\beta-1} \|\mathbf{H}_k\,\left(\mathbf{M}_{n_\alpha,k}\mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha + \mathbf{M}_{n_\alpha,k}\,\delta\eta^\beta\right) - d_k\|^2 \\ &= \sum_{k=n_\alpha}^{n_\beta-1} \|\mathbf{H}_k\,\left(\mathbf{M}_{0,k}\delta x_0^\alpha + \mathbf{M}_{n_\alpha,k}\,\delta\eta^\beta\right) - d_k\|^2 \end{aligned} \tag{2.68}$$

Finally, let us gather background and observation terms:

$$\boxed{\begin{aligned} J^{(g)} &= \frac{1}{2}\|x_0^{\alpha,r} + \delta x_0^\alpha - x_0^{\alpha,b}\|^2_{\mathbf{Q}_\alpha} + \frac{1}{2}\|\eta^{\beta,r} + \delta\eta^\beta\|^2_{\mathbf{Q}_\beta} \\ &+\frac{1}{2}\sum_{k=0}^{n_\alpha-1} \|y_k^o - \mathcal{H}_k\mathcal{M}_{0,k}(x_0^{\alpha,r}) - \mathbf{H}_k\,\mathbf{M}_{0,k}\,\delta x_0^\alpha\|^2_{R_k} \\ &+\frac{1}{2}\sum_{k=n_\alpha}^{n_\beta-1} \|\mathbf{H}_k\,\left(\mathbf{M}_{n_\alpha,k}\mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha + \mathbf{M}_{n_\alpha,k}\,\delta\eta^\beta\right) - d_k\|^2 \end{aligned}} \tag{2.69}$$

Note that equation (2.68) can be developped in four terms:

$$\begin{aligned} J^{o,\beta,(g)} &= \sum_{k=n_\alpha}^{n_\beta-1} \left(d_k - \mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha\right)^T \mathbf{R}_k^{-1}\left(d_k - \mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha\right) \\ &+ \sum_{k=n_\alpha}^{n_\beta-1} \left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,\delta\eta^\beta\right)^T \mathbf{R}_k^{-1}\left(-d_k + \mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha\right) \\ &+ \sum_{k=n_\alpha}^{n_\beta-1} \left(-d_k + \mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha\right)^T \mathbf{R}_k^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,\delta\eta^\beta\right) \\ &+ \sum_{k=n_\alpha}^{n_\beta-1} \left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,\delta\eta^\beta\right)^T \mathbf{R}_k^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,\delta\eta^\beta\right) \end{aligned} \tag{2.70}$$

This result will be applied to developpe the gradient.

Gradient

Let us derive:

- partial derivatives of term $J^{o,\alpha,(g)}$ from equation (2.65). It only depends on x_0^α:

$$\begin{aligned} \frac{\partial J^{o,\alpha,(g)}}{\partial x_0^\alpha} &= \sum_{k=0}^{n_\alpha-1} \mathbf{M}_{0,k}^T\mathbf{H}_k^T\,\mathbf{R}_k^{-1}\mathbf{H}_k\,\mathbf{M}_{0,k}\,\delta x_0^\alpha - \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,k}^T\mathbf{H}_k^T\,\mathbf{R}_k^{-1}\,d_k \\ &= \delta x_0^{\alpha *} \end{aligned} \tag{2.71}$$

- partial derivatives of term $J^{o,\beta,(g)}$ from equation (2.70). Because it depends both on x_0^α and η, we have:

  with respect to x_0^α:

$$\begin{array}{ll} \frac{\partial J^{o,\beta,(g)}}{\partial x_0^\alpha} & = \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,n_\alpha}^T \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \mathbf{M}_{0,n_\alpha} \delta x_0^\alpha \\ & - \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,n_\alpha}^T \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} d_k \\ & + \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,n_\alpha}^T \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \delta\eta^\beta \end{array} \tag{2.72}$$

  and with respect to η^β:

$$\begin{array}{ll} \frac{\partial J^{o,\beta,(g)}}{\partial \eta^\beta} & = \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \delta\eta^\beta \\ & + \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \mathbf{M}_{0,n_\alpha} \delta x_0^\alpha - \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} d_k \end{array} \tag{2.73}$$

Let us now deduce the gradient of both terms of $J^{o,(g)} = J^{o,\alpha,(g)} + J^{o,\beta,(g)}$:

- $J^{o,(g)}$ partial derivative with respect to η^β only derives from equation (2.73). The grouping of terms enable to make appear variable δx_0^β:

$$\frac{\partial J^{o,\beta,(g)}}{\partial \eta^\beta} = \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \left(\underbrace{\mathbf{M}_{0,n_\alpha} \delta x_0^\alpha + \delta\eta^\beta}_{\delta x_0^\beta} \right) - \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} d_k \tag{2.74}$$

  In the above equation, one can recognize the adjoint vector:

$$\begin{array}{ll} \frac{\partial J^{o,\beta,(g)}}{\partial \eta^\beta} & = \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \delta x_0^\beta - \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} d_k \\ & = \delta x_0^{\beta *} \end{array} \tag{2.75}$$

  This result is not surprising since chain rules for derivatives applied on $J\left(\delta x_0^\alpha, \delta x_0^\beta\right)$ and : $J\left(\delta x_0^\alpha, \eta^\beta\right)$ shows:

$$\boxed{\frac{\partial J^{o,\beta,(g)}}{\partial \eta^\beta} = \frac{\partial J^{o,\beta,(g)}}{\partial x_0^\beta} = \delta x_0^{\beta *}} \tag{2.76}$$

- $J^{o,(g)}$ partial derivative with respect x_0^α, derive from (2.71) and (2.72):

$$\begin{aligned}\frac{\partial J^{o(g)}}{\partial x_0^\alpha} &= \overbrace{\sum_{k=0}^{n_\alpha-1} \mathbf{M}_{0,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{0,k} \, \delta x_0^\alpha - \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \, d_k}^{\delta x_0^{\alpha *}} \\ &+ \textstyle\sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,n_\alpha}^T \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \mathbf{M}_{0,n_\alpha} \, \delta x_0^\alpha - \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,n_\alpha}^T \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \, d_k \\ &+ \textstyle\sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,n_\alpha}^T \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \, \mathbf{H}_k \, \mathbf{M}_{n_\alpha,k} \, \delta \eta^\beta\end{aligned} \tag{2.77}$$

and second by grouping terms to make appear δx_0^β, we have:

$$\begin{aligned}\frac{\partial J^{o,(g)}}{\partial x_0^\alpha} &= \delta x_0^{\alpha *} + \textstyle\sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,n_\alpha}^T \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \overbrace{\left(\mathbf{M}_{0,n_\alpha} \, \delta x_0^\alpha + \delta \eta^\beta\right)}^{\delta x_0^\beta} \\ &- \textstyle\sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{0,n_\alpha}^T \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \, d_k\end{aligned} \tag{2.78}$$

which is equivalent to:

$$\begin{aligned}\frac{\partial J^{o,(g)}}{\partial x_0^\alpha} &= \delta x_0^{\alpha *} \\ &+ \mathbf{M}_{0,n_\alpha}^T [\textstyle\sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \, \delta x_0^\beta - \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \, d_k]\end{aligned} \tag{2.79}$$

where one can recognize the adjoint variable $\delta x_0^{\beta *}$ within the brackets:

$$\frac{\partial J^{o,(g)}}{\partial x_0^\alpha} = \delta x_0^{\alpha *} + \mathbf{M}_{0,n_\alpha}^T \, \delta x_0^{\beta *} \tag{2.80}$$

Finally, we have for the full gradient:

$$\boxed{\frac{\partial J^{(g)}}{\partial x_0^\alpha} = \mathbf{Q}_\alpha^{-1} \left(x_0^{\alpha,r} + \delta x_0^\alpha - x_0^{\alpha,b}\right) + \delta x_0^{\alpha *} + \mathbf{M}_{0,n_\alpha}^T \, \delta x_0^{\beta *}} \tag{2.81}$$

and

$$\boxed{\begin{aligned}\frac{\partial J^{(g)}}{\partial \eta^\beta} &= \mathbf{Q}_\beta^{-1} \, \delta \eta - \textstyle\sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \, d_k + \sum_{k=n_\alpha}^{n_\beta-1} \mathbf{M}_{n_\alpha,k}^T \mathbf{H}_k^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k} \, \delta x_0^\beta \\ &= \mathbf{Q}_\beta^{-1} \, \delta \eta + \delta x_0^{\beta *}\end{aligned}} \tag{2.82}$$

2.3.4 Comments on Weak constraint incremental 4D-Var of types $4D-Var_{\mathbf{x_k}}$ and $4D-Var_{\eta_\mathbf{k}}$

For the seek of a suited algorithm the cost function and gradient equations (2.41) and (2.42) can be rewritten to make appear:

- the cumulated α component increment $\sum_{p=1}^{(g-1)} \delta x_0^{\alpha,(p)}$,
- the η component increment cumulated and current $\sum_{p=1}^{(g-1)} \delta\eta^{(p)} + \delta\eta^{(g)}$:

Cost function In equation (2.41) it gives:

$$\begin{aligned} J^g &= \tfrac{1}{2} \left(\overbrace{x_0^{\alpha,r} - x_0^{\alpha,b}}^{\sum_{p=1}^{(g-1)} \delta x_0^{\alpha,(g)}} + \delta x_0^\alpha \right)^T {Q_\alpha}^{-1} \left(x_0^{\alpha,r} - x_0^{\alpha,b} + \delta x_0^\alpha \right) \\ &+ \tfrac{1}{2} \left(\overbrace{x_0^{\beta,r} + -\mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})}^{\sum_{p=1}^{(g-1)} \delta\eta^{(p)}} + \overbrace{\delta x_0^\beta - \mathbf{M}_{n_\alpha,0}\, \delta x_0^\alpha}^{\delta\eta^{(g)}} \right)^T {Q_\beta}^{-1} \left(x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) + \delta x_0^\beta - \mathbf{M}_{n_\alpha,0}\, \delta x_0^\alpha \right) \\ &+ \tfrac{1}{2} \sum_{k=0}^{n_\alpha - 1} \left(\mathbf{H}_k\, \mathbf{M}_{0,k}\, \delta x_0^\alpha - {d_k}^\alpha \right)^T {R_k}^{-1} \left(\mathbf{H}_k\, \mathbf{M}_{0,k}\, \delta x_0^\alpha - {d_k}^\alpha \right) \\ &+ \tfrac{1}{2} \sum_{k=n_\alpha}^{n_\beta - 1} \left(\mathbf{H}_k\, \mathbf{M}_{n_\alpha,k} \underbrace{\delta x_0^\beta}_{\mathbf{M}_{n_\alpha,0}\, \delta x_0^\alpha + \delta\eta^{(g)}} - {d_k}^\beta \right)^T {R_k}^{-1} \left(\mathbf{H}_k\, \mathbf{M}_{n_\alpha,k}\, \delta x_0^\beta - {d_k}^\beta \right) \end{aligned} \tag{2.83}$$

which finally may resume to:

$$\begin{aligned} J^g &= \tfrac{1}{2} \left(\sum_{p=1}^{(g-1)} \delta x_0^{\alpha,(g)} + \delta x_0^\alpha \right)^T {Q_\alpha}^{-1} \left(\sum_{p=1}^{(g-1)} \delta x_0^{\alpha,(g)} + \delta x_0^\alpha \right) \\ &+ \tfrac{1}{2} \left(\sum_{p=1}^{(g-1)} \delta\eta^{(p)} + \delta\eta^{(g)} \right)^T {Q_\beta}^{-1} \left(\sum_{p=1}^{(g-1)} \delta\eta^{(p)} + \delta\eta^{(g)} \right) \\ &+ \tfrac{1}{2} \sum_{k=0}^{n_\alpha - 1} \left(\mathbf{H}_k\, \mathbf{M}_{0,k}\, \delta x_0^\alpha - {d_k}^\alpha \right)^T {R_k}^{-1} \left(\mathbf{H}_k\, \mathbf{M}_{0,k}\, \delta x_0^\alpha - {d_k}^\alpha \right) \\ &+ \tfrac{1}{2} \sum_{k=n_\alpha}^{n_\beta - 1} \left(\mathbf{H}_k\, \mathbf{M}_{n_\alpha,k} \left(\mathbf{M}_{n_\alpha,0}\, \delta x_0^\alpha + \delta\eta^{(g)} \right) - {d_k}^\beta \right)^T {R_k}^{-1} \left(\mathbf{H}_k\, \mathbf{M}_{n_\alpha,k} \left(\mathbf{M}_{n_\alpha,0}\, \delta x_0^\alpha + \delta\eta^{(g)} \right) - {d_k}^\beta \right) \end{aligned} \tag{2.84}$$

Gradient equations The equations (2.42) become:

$$\nabla_{x^\alpha} J^g = \overbrace{\underbrace{Q_\alpha{}^{-1} \overbrace{\left(x_0^{\alpha,r} - x_0^{\alpha,b}\right)}^{\sum_{p=1}^{(g-1)} \delta x_0^{\alpha,(g)} + \delta x_0^\alpha} - \sum_{k=0}^{n_\alpha - 1} \mathbf{M}_{0,k}{}^T \mathbf{H}_k{}^T R_k{}^{-1} d_k{}^\alpha}_{} + \underbrace{Q_\alpha{}^{-1} \left(\mathbf{I} + \sum_{k=0}^{n_\alpha - 1} Q_\alpha \mathbf{M}_{0,k}{}^T \mathbf{H}_k{}^T R_k{}^{-1} \mathbf{H}_k \mathbf{M}_{0,k}\right) \delta x_0^\alpha}_{\text{Hessian}}}^{\text{Initial gradient}} \underbrace{-\mathbf{M}_{0,n_\alpha}{}^T Q_\beta{}^{-1} \left(\underbrace{x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})}_{\sum_{p=1}^{(g-1)} \delta\eta^{(p)}} + \underbrace{\delta x_0^\beta - \mathbf{M}_{0,n_\alpha} \delta x_0^\alpha)}_{\delta\eta^{(g)}} \right)}_{\textit{Supplementary term}} \tag{2.85}$$

and:

$$\nabla_{x^\beta} J^g = \overbrace{\underbrace{Q_\beta{}^{-1} \overbrace{\left(x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})\right)}^{\sum_{p=1}^{(g-1)} \delta\eta^{(p)}} - \sum_{k=n_\alpha}^{n_\beta - 1} \mathbf{M}_{n_\alpha,k}{}^T \mathbf{H}_k{}^T R_k{}^{-1} d_k{}^\beta}_{} + \underbrace{Q_\beta{}^{-1} \left(\mathbf{I} + \sum_{k=n_\alpha}^{n_\beta - 1} Q_\beta \mathbf{M}_{n_\alpha,k}{}^T \mathbf{H}_k{}^T R_k{}^{-1} \mathbf{H}_k \mathbf{M}_{n_\alpha,k}\right) \delta x_0^\beta}_{\text{Hessian}}}^{\text{Initial gradient}} \underbrace{-Q_\beta{}^{-1} \mathbf{M}_{0,n_\alpha} \delta x_0^\alpha}_{\textit{Supplementary term}} \tag{2.86}$$

It finally leads to:

$$\nabla_{x^\alpha} J^g = \overbrace{\underbrace{Q_\alpha{}^{-1} \left(\sum_{p=1}^{(g-1)} \delta x_0^{\alpha,(g)} + \delta x_0^\alpha\right) - \sum_{k=0}^{n_\alpha - 1} \mathbf{M}_{0,k}{}^T \mathbf{H}_k{}^T R_k{}^{-1} d_k{}^\alpha}_{} + \underbrace{Q_\alpha{}^{-1} \left(\mathbf{I} + \sum_{k=0}^{n_\alpha - 1} Q_\alpha \mathbf{M}_{0,k}{}^T \mathbf{H}_k{}^T R_k{}^{-1} \mathbf{H}_k \mathbf{M}_{0,k}\right) \delta x_0^\alpha}_{\text{Hessian}}}^{\text{Initial gradient}} \underbrace{-\mathbf{M}_{0,n_\alpha}{}^T Q_\beta{}^{-1} \left(\sum_{p=1}^{(g-1)} \delta\eta^{(p)} + \delta\eta^{(g)}\right)}_{\textit{Supplementary term}} \tag{2.87}$$

and:

$$\begin{aligned}\nabla_{x^\beta} J^g &= \overbrace{Q_\beta{}^{-1}\left(\sum_{p=1}^{(g-1)}\delta\eta^{(p)}\right) - \sum_{k=n_\alpha}^{n_\beta-1}\mathbf{M}_{n_\alpha,k}{}^T\,\mathbf{H}_k{}^T\,R_k{}^{-1}\,d_k{}^\beta}^{\text{Initial gradient}} \\ &\underbrace{+\,Q_\beta{}^{-1}\left(\mathbf{I}+\sum_{k=n_\alpha}^{n_\beta-1}Q_\beta\,\mathbf{M}_{n_\alpha,k}{}^T\,\mathbf{H}_k{}^T\,R_k{}^{-1}\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\right)\delta x_0^\beta}_{\text{Hessian}} \\ &\underbrace{-Q_\beta{}^{-1}\,\mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha}_{\textit{Supplementary term}}\end{aligned} \tag{2.88}$$

2.3.5 Weak constraint incremental 4D-Var with augmented state vector of type $4D-Var_{\mathbf{x_k}}$ $B^{1/2}$ condtioning

Equations: $B^{1/2}$ condtioning

For conditioning matters, the change of variable $\delta u = B^{-1/2}x$ is usually introduced in order to accelerate convergence. Let us develope the corresponding equations in the weak constraint 4D-Var scheme.

Variables are defined as follows (still setting the current outer loop as the g^{th}):

- The control vector is written u_0

$$u_0 = \begin{pmatrix} u_0^{\alpha,b} \\ u_0^{\beta,b}\end{pmatrix} = \begin{pmatrix} Q_\alpha{}^{-1/2}x_0^{\alpha,b} \\ Q_\beta{}^{-1/2}x_0^{\beta,b}\end{pmatrix} \tag{2.89}$$

- The reference control vector is:

$$u_0^r = \begin{pmatrix} u_0^{\alpha,r} \\ u_0^{\beta,r}\end{pmatrix} = \begin{pmatrix} Q_\alpha{}^{-1/2}x_0^{\alpha,r} \\ Q_\beta{}^{-1/2}x_0^{\beta,r}\end{pmatrix} \tag{2.90}$$

- The searched increment is:

$$\delta u_0 = \begin{pmatrix} \delta u_0^{\alpha} \\ \delta u_0^{\beta}\end{pmatrix} = \begin{pmatrix} Q_\alpha{}^{-1/2}\delta x_0^{\alpha} \\ Q_\beta{}^{-1/2}\delta x_0^{\beta}\end{pmatrix} \tag{2.91}$$

Equation (2.41) giving $J^{(g)}$ becomes:

$$\begin{aligned} J^g &= \tfrac{1}{2}||u_0^{\alpha,r}+\delta u_0^\alpha - u_0^{\alpha,b}||^2 \\ &+\tfrac{1}{2}||u_0^{\beta,r}+\delta u_0^\beta - Q_\beta{}^{-1/2}\mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) - Q_\beta{}^{-1/2}\mathbf{M}_{0,n_\alpha}\,Q_\alpha{}^{1/2}\delta u_0^\alpha)||^2 \\ &+\tfrac{1}{2}\textstyle\sum_{k=0}^{n_\alpha-1}\left(\mathbf{H}_k\,\mathbf{M}_{0,k}\,Q_\alpha{}^{1/2}\delta u_0^\alpha - d_k{}^\alpha\right)^T R_k{}^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{0,k}\,Q_\alpha{}^{1/2}\delta u_0^\alpha - d_k{}^\alpha\right) \\ &+\tfrac{1}{2}\textstyle\sum_{k=n_\alpha}^{n_\beta-1}\left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,Q_\beta{}^{1/2}\delta u_0^\beta - d_k{}^\beta\right)^T R_k{}^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\,Q_\beta{}^{1/2}\delta u_0^\beta - d_k{}^\beta\right)\end{aligned} \tag{2.92}$$

where vector d_k is:

$$\begin{pmatrix} {d_k}^{\alpha} \\ {d_k}^{\beta} \end{pmatrix} = \begin{pmatrix} y_k^o - \mathcal{M}_{0,k}\left(x_0^{\alpha,r}\right) \\ y_k^o - \mathcal{M}_{n_\alpha}\left(x_0^{\beta,r}\right) \end{pmatrix} \tag{2.93}$$

Equation 2.42 giving $\nabla J^{(g)}$

$$\begin{aligned} \nabla_{u^\alpha} J^g &= \left(u_0^{\alpha,r} + \delta u_0^\alpha - u_0^{\alpha,b}\right) \\ &\quad - {Q_\alpha}^{T/2} \mathbf{M}_{0,n_\alpha}{}^T \, {Q_\beta}^{-T/2} \left(u_0^{\beta,r} + \delta u_0^\beta - {Q_\beta}^{-1/2} \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) - {Q_\beta}^{-1/2} \mathbf{M}_{0,n_\alpha} {Q_\alpha}^{1/2} \, \delta u_0^\alpha\right) \\ &\quad + \textstyle\sum_{k=0}^{n_\alpha - 1} {Q_\alpha}^{T/2} \mathbf{M}_{k,0}{}^T \, \mathbf{H}_k{}^T \, {R_k}^{-1} \left(\mathbf{H}_k \, \mathbf{M}_{0,k} {Q_\alpha}^{1/2} \, \delta u_0^\alpha - {d_k}^{\alpha}\right) \\ \nabla_{u^\beta} J^g &= \left(u_0^{\beta,r} + \delta u_0^\beta - {Q_\beta}^{-1/2} \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r}) - {Q_\beta}^{-1/2} \mathbf{M}_{0,n_\alpha} {Q_\alpha}^{1/2} \, \delta u_0^\alpha\right) \\ &\quad + \textstyle\sum_{k=n_\alpha}^{n_\beta - 1} {Q_\beta}^{T/2} \mathbf{M}_{k,n_\alpha}{}^T \, \mathbf{H}_k{}^T \, {R_k}^{-1} \left(\mathbf{H}_k \, \mathbf{M}_{k,n_\alpha} {Q_\beta}^{1/2} \, \delta u_0^\beta - {d_k}^{\beta}\right) \end{aligned} \tag{2.94}$$

For algorithmic matters, it is a common practice to make appear the Hessian of the cost function as well as the initial gradient (i.e., before entering inner loops when $\delta x = 0$), which gives in equation (2.94):

$$\begin{aligned} \nabla_{u^\alpha} J^g &= \overbrace{\left(u_0^{\alpha,r} - u_0^{\alpha,b}\right) - \sum_{k=0}^{n_\alpha - 1} {Q_\alpha}^{T/2} \mathbf{M}_{k,0}{}^T \, \mathbf{H}_k{}^T \, {R_k}^{-1} \, {d_k}^{\alpha}}^{Initial \quad gradient} \\ &\quad + \underbrace{\left(\mathbf{I} + \sum_{k=0}^{n_\alpha - 1} {Q_\alpha}^{T/2} \mathbf{M}_{k,0}{}^T \, \mathbf{H}_k{}^T \, {R_k}^{-1} \, \mathbf{H}_k \, \mathbf{M}_{0,k} {Q_\alpha}^{1/2}\right)}_{Hessian} \delta u_0^\alpha \\ &\quad \underbrace{- {Q_\alpha}^{T/2} \mathbf{M}_{0,n_\alpha}{}^T \, \mathbf{Q}_\beta^{-T/2} \left(\overbrace{u_0^{\beta,r} - {Q_\beta}^{-1/2} \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})}^{{Q_\beta}^{-1/2}\left(x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})\right)} + \overbrace{\delta u_0^\beta - {Q_\beta}^{-1/2} \mathbf{M}_{0,n_\alpha} {Q_\alpha}^{1/2} \, \delta u_0^\alpha}^{{Q_\beta}^{-1/2}\left(\delta x_0^\beta - \mathbf{M}_{0,n_\alpha} \, \delta x_0^\alpha\right)} \right)}_{Supplementary \quad term} \end{aligned} \tag{2.95}$$

and for β component:

$$\nabla_{u^\beta} J^g = \overbrace{\underbrace{\left(u_0^{\beta,r} - {Q_\beta}^{-1/2}\mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})\right)}_{{Q_\beta}^{-1/2}\left(x_0^{\beta,r}-\mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})\right)} - \sum_{k=n_\alpha}^{n_\beta-1} {Q_\beta}^{T/2}\mathbf{M}_{k,n_\alpha}{}^T\,\mathbf{H}_k{}^T\,{R_k}^{-1}\,{d_k}^\beta}^{\text{Initial gradient}}$$
$$+ \overbrace{\left(\mathbf{I} + \sum_{k=n_\alpha}^{n_\beta-1} {Q_\beta}^{T/2}\,\mathbf{M}_{k,n_\alpha}{}^T\,\mathbf{H}_k{}^T\,{R_k}^{-1}\,\mathbf{H}_k\,\mathbf{M}_{k,n_\alpha}{Q_\beta}^{1/2}\right)}^{\text{Hessian}}\,\delta u_0^\beta$$
$$\underbrace{-\left(\mathbf{Q}_\beta{}^{-\mathbf{1/2}}\,\mathbf{M}_{0,n_\alpha}{Q_\alpha}^{1/2}\,\delta u_0^\alpha\right)}_{\text{Supplementary term}} \tag{2.96}$$

Now remember that:

$$Q_\beta{}^{-1/2}\left(\delta x_0^\beta - \mathbf{M}_{0,n_\alpha}\,\delta x_0^\alpha\right) = Q_\beta{}^{-1/2}\,\delta\eta^\beta \tag{2.97}$$

$$Q_\beta{}^{-1/2}\left(x_0^{\beta,r} - \mathcal{M}_{0,n_\alpha}(x_0^{\alpha,r})\right) = Q_\beta{}^{-1/2}\sum_{p=1}^{(g-1)}\delta\eta^{\beta,(p)} \tag{2.98}$$

One can introduce η^β increments with ${Q_\beta}^{1/2}$ conditionning:

$$\boxed{\delta\tilde{\eta}^\beta = Q_\beta{}^{-1/2}\,\delta\eta^\beta} \tag{2.99}$$

If one introduce equation 2.99 in equations (2.97) and (2.98), and further put the latter in gradient components of equations (2.95) and (2.96), one finally have:

$$\nabla_{u^\alpha} J^g = \overbrace{\left(u_0^{\alpha,r} - u_0^{\alpha,b}\right) - \sum_{k=0}^{n_\alpha-1} {Q_\alpha}^{T/2}\mathbf{M}_{k,0}{}^T\,\mathbf{H}_k{}^T\,{R_k}^{-1}\,{d_k}^\alpha}^{\textit{Initial gradient}}$$
$$+ \underbrace{\left(\mathbf{I} + \sum_{k=0}^{n_\alpha-1} {Q_\alpha}^{T/2}\mathbf{M}_{k,0}{}^T\,\mathbf{H}_k{}^T\,{R_k}^{-1}\,\mathbf{H}_k\,\mathbf{M}_{0,k}{Q_\alpha}^{1/2}\right)}_{\textit{Hessian}}\,\delta u_0^\alpha$$
$$\underbrace{-{Q_\alpha}^{T/2}\mathbf{M}_{0,n_\alpha}{}^T\,\mathbf{Q}_\beta^{-T/2}\left(\sum_{p=1}^{(g-1)}\delta\tilde{\eta}^{\beta,(p)} + \delta\tilde{\eta}^\beta\right)}_{\textit{Supplementary term}} \tag{2.100}$$

and for β component:

$$\begin{aligned}\nabla_{u^\beta} J^g &= \overbrace{\left(\sum_{p=1}^{(g-1)} \delta\tilde{\eta}^{\beta,(p)}\right) - \sum_{k=n_\alpha}^{n_\beta-1} {Q_\beta}^{T/2}{\mathbf{M}_{k,n_\alpha}}^T {\mathbf{H}_k}^T {R_k}^{-1} {d_k}^\beta}^{\text{Initial gradient}} \\ &+ \overbrace{\left(\mathbf{I} + \sum_{k=n_\alpha}^{n_\beta-1} {Q_\beta}^{T/2}\,{\mathbf{M}_{k,n_\alpha}}^T {\mathbf{H}_k}^T {R_k}^{-1}\,\mathbf{H}_k\,\mathbf{M}_{k,n_\alpha}{Q_\beta}^{1/2}\right)\delta u_0^\beta}^{\text{Hessian}} \\ &\underbrace{-\left(\mathbf{Q_\beta}^{-1/2}\,\mathbf{M}_{0,n_\alpha}{Q_\alpha}^{1/2}\,\delta u_0^\alpha\right)}_{\text{Supplementary term}}\end{aligned} \tag{2.101}$$

Finally the lattest equation can be simplified further as follows:

$$\begin{aligned}\nabla_{u^\beta} J^g &= \left(\textstyle\sum_{p=1}^{(g-1)} \delta\tilde{\eta}^{\beta,(p)}\right) - \sum_{k=n_\alpha}^{n_\beta-1} {Q_\beta}^{T/2}{\mathbf{M}_{k,n_\alpha}}^T {\mathbf{H}_k}^T {R_k}^{-1} {d_k}^\beta \\ &+ \left(\textstyle\sum_{k=n_\alpha}^{n_\beta-1} {Q_\beta}^{T/2}\,{\mathbf{M}_{k,n_\alpha}}^T {\mathbf{H}_k}^T {R_k}^{-1}\,\mathbf{H}_k\,\mathbf{M}_{k,n_\alpha}{Q_\beta}^{1/2}\right)\delta u_0^\beta \\ &\underbrace{-\mathbf{Q_\beta}^{-1/2}\left({Q_\beta}^{1/2}\,\delta u_0^\beta - \mathbf{M}_{0,n_\alpha}{Q_\alpha}^{1/2}\,\delta u_0^\alpha\right)}_{\delta\tilde{\eta}^\beta}\end{aligned} \tag{2.102}$$

and:

$$\begin{aligned}\nabla_{u^\beta} J^g &= \left(\textstyle\sum_{p=1}^{(g-1)} \delta\tilde{\eta}^{\beta,(p)}\right) - \sum_{k=n_\alpha}^{n_\beta-1} {Q_\beta}^{T/2}{\mathbf{M}_{k,n_\alpha}}^T {\mathbf{H}_k}^T {R_k}^{-1} {d_k}^\beta \\ &+ \left(\textstyle\sum_{k=n_\alpha}^{n_\beta-1} {Q_\beta}^{T/2}\,{\mathbf{M}_{k,n_\alpha}}^T {\mathbf{H}_k}^T {R_k}^{-1}\,\mathbf{H}_k\,\mathbf{M}_{k,n_\alpha}{Q_\beta}^{1/2}\right)\left(\mathbf{Q_\beta}^{-1/2}\,\mathbf{M}_{0,n_\alpha}{Q_\alpha}^{1/2}\delta u_0^\alpha + \delta\tilde{\eta}^\beta\right) \\ &-\delta\tilde{\eta}^\beta\end{aligned} \tag{2.103}$$

2.3.6 Weak constraint incremental 4D-Var of type $4D-Var_{\eta_\mathbf{k}}$ $B^{1/2}$ condtioning

The quadratic cost function $J^{(g)}$ is:

$$\begin{aligned}J^g &= \tfrac{1}{2}||u_0^{\alpha,r} + \delta u_0^\alpha - u_0^{\alpha,b}||^2 + \tfrac{1}{2}||\tilde{\eta}^{\beta,r} + \delta\tilde{\eta}^\beta||^2 \\ &+\tfrac{1}{2}\textstyle\sum_{k=0}^{n_\alpha-1}\left(\mathbf{H}_k\,\mathbf{M}_{0,k}\,{Q_\alpha}^{1/2}\delta u_0^\alpha - {d_k}^\alpha\right)^T {R_k}^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{0,k}\,{Q_\alpha}^{1/2}\delta u_0^\alpha - {d_k}^\alpha\right) \\ &+\tfrac{1}{2}\textstyle\sum_{k=n_\alpha}^{n_\beta-1}\left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\left(\mathbf{M}_{0,n_\alpha}\,{Q_\alpha}^{1/2}\delta u_0^\alpha + {Q_\beta}^{1/2}\,\delta\tilde{\eta}^\beta\right) - {d_k}^\beta\right)^T \\ &{R_k}^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{n_\alpha,k}\left(\mathbf{M}_{0,n_\alpha}\,{Q_\alpha}^{1/2}\delta u_0^\alpha + {Q_\beta}^{1/2}\,\delta\tilde{\eta}^\beta\right) - {d_k}^\beta\right)\end{aligned} \tag{2.104}$$

The partial derivatives of $J^{(g)}$ can be found easily by noting that:

$$\frac{\partial J^{(g)}}{\partial u_0^\alpha} = \mathbf{Q}_\alpha^{\mathbf{1/2}} \frac{\partial J^{(g)}}{\partial x_0^\alpha} \tag{2.105}$$

$$\frac{\partial J^{(g)}}{\partial \tilde{\eta}_0^\beta} = \mathbf{Q}_\beta^{\mathbf{1/2}} \frac{\partial J^{(g)}}{\partial \eta_0^\beta} \tag{2.106}$$

Based on the above equations, and on the results of equations (2.81) and (2.82), (see section related to $\left(\delta x_0^\alpha, \delta \eta^\beta\right)$ formulation), we have:

$$\boxed{\frac{\partial J^{(g)}}{\partial u_0^\alpha} = \left(\sum_{p=1}^{g-1} \delta u_0^{\alpha,(p)} + \delta u_0^\alpha\right) + \mathbf{Q}_\alpha^{\mathbf{1/2}} \left(\delta x_0^{\alpha *} + \mathbf{M}_{0,n_\alpha}^T \delta x_0^{\beta *}\right)} \tag{2.107}$$

$$\boxed{\frac{\partial J^{(g)}}{\partial \tilde{\eta}^\beta} = \left(\sum_{p=1}^{g-1} \delta \tilde{\eta}^{\beta,(p)} + \delta \tilde{\eta}^\beta\right) + \mathbf{Q}_\beta^{\mathbf{1/2}} \underbrace{\delta x_0^{\beta *}}} \tag{2.108}$$

Chapter 3

Implementation of the weak constraint 4D-Var with NEMO/NEMOVAR

The NEMOVAR constraint incremental 4D-Var controls increments of 5 oceanographic fields defined at initial time step of the assimilation window: the temperature and salinity t, s, the horizontal velocities u, v, and the sea surface height η. NEMOVAR controls increments of these oceanographic fields defined at initial time step of the assimilation window.

Among the possible weak constraint incremental 4D-var formulations, one consists in controling increments defined not only at the initial time step but at several chosen time steps of the assimilation window. The assimilation window is split into subwindows, and trajectory discontinuities at the junctions are allowed. The discontinuities are penalized in the cost function by terms measuring the distance between the last and initial states of two succesive subwindows. In comparison to the strong constraint approach, the size of the state vector is augmented by the number of subwindows.

The aim of this document is to propose implementations to set up a weak constraint 4D-Var algorithm for the NEMO and NEMOVAR system. For that purpose, a step by step code analysis is necessary. Part of it will be detailed in appendices, while others will be presented in the main text.

In the next section, we first provide an overview of the NEMO and NEMOVAR strong constraint incremental 4D-Var. Second, we highlight the main issues we shall face when implementing the weak constraint 4D-Var. Then, we propose implementation in the code, keeping in mind that we wish to remain less invasive as possible.

3.1 NEMO and NEMOVAR: general comments

When NEMO and NEMOVAR simulations are part of a 4D-Var assimilation experiment, we refer to them as the *outer loop* and *inner loop* executables. We shall also designate them by nemo.exe and nemovar.exe, which are the names of the executables.

In the following, we describe the organization of the NEMO and NEMOVAR outer and inner loops.

3.1.1 Inner loop structure and minimizer drivers

The nemovar executable is seperated in three sequential calls to subroutines:

1. `nemovar_init`,
2. `nemovar_main`,
3. `nemovar_final`.

The `nemovar_main` subroutine organizes the call to one of the three **inner loop drivers** provided in NEMOVAR. Each **inner loop driver** applies its own minimizer algorithm on the quadratic cost function $J^{(g)}$. The user can choose one algorithm or the other by changing the `ncgtype` parameter in the namelist block `namopt` (see namelist_nemovar). It involves a call to:

- either to `nemovar_inner_congrad`, which relies on the `congrad_main` routine,
  $\rightarrow$ *CONGRAD* minimizer
- or to `nemovar_inner_cgmod`, which relies on the `cgmod_main` routine,
  $\rightarrow$ *CGMOD* minimizer
- or to `nemovar_inner_rpcg`, which calls `rcpg_main`,
  $\rightarrow$ *RPCG* minimizer

The three minimizers are based on Conjugate Gradient descent algorithms (hereafter CG). The driver routines can be found in source nemovar_inner_drivers.F90. They all organize the inner loop iterations in a similar sequence of operations:

1. They prepare the first inner loop iteration:
   (a) by loading the cumulated increment vectors δx_0,
   (b) and computating the initial J and ∇J values at the reference state x_r^r:
   $$x_0^r = x_0^b + \delta x_0$$
2. They launch the loop iterations in a reverse mode communication with the minimizer routines (either congrad_main, or cgmod_main, or rcpg_main).
   At each iteration, the minimizer argument are the current gradient vector as input, and the new increment vector as output.
3. They break the loop on a stop criterion.

4. They update the cumulated increment variable with the final current increment.
5. They write or append the following off-line communication files:
   - `increment#.nc` which is required between nemovar.exe and nemo.exe,
   - `restart_ctlvec_#` which is required between two succesive executions of nemovar.exe.

In this document, we omit the description of the `rpcg` driver, because a working version is not yet available.

3.1.2 Model and observation space vectors

The table 3.1.2 shows the important vectors defined in the model space, while tables **??** and **??** show the corespondance between weighted and unweighted vectors (i.e., application of the background error covariance matrix) for congrad and cgmod respectively.

Table 3.1: Important model and observation space vectors of the data assimilation system.

Quantity	**Notation**
State vector	x_0
State vector trajectory	$x_{k+1} = \mathcal{M}_{k,k+1}(x_k)$
Increment	δx_0
Background vector and trajectory	x_0^b and x_k^b
Reference state vector when entering a new inner loop	$x_0^r = x_0^b + \delta x_0$
Model counter part of the reference state vector trajectory	$\mathcal{H}_k(x_k^r)$
Innovation	$d_k = y_k^o - \mathcal{H}_k(x_k^r)$
Increment linear trajectory	$\delta x_{k+1} = \mathbf{M}_{k,k+1}\,\delta x_k$
Model counterpart of the increment linear trajectory	$\mathbf{H}_k \delta x_k$
Initial state of the adjoint	$x_N^* = \mathbf{H}_N^T {R_N}^{-1}(\mathbf{H}_N \delta x_N - d_N)$
Adjoint state trajectory	$x_{k-1}^* = \mathbf{M}_{k-1,k}^T x_k^* + \mathbf{H}_{k-1}^T {R_{k-1}}^{-1}(\mathbf{H}_{k-1}\delta x_{k-1} - d_{k-1})$

Table 3.2: Weighted and unweighted model space vectors handled by the **congrad** conjugate gradient.

Quantity	**Not weighted**	**Weighted**
Background vector	x_0^b	$b = B^{-1/2}\, x_0^b$
Current increment End of inner loop (outer loop g)	$\delta x_0^{(g)}$	$\delta u_0^{(g)} = B^{-1/2}\, \delta x_0^{(g)}$
Cummulated increment Entering inner loop (outer loop g)	$\sum_{p=1}^{g-1} \delta x_0^p$	$B^{-1/2} \sum_{p=1}^{g-1} \delta x_0^p$
Reference state vector when entering a new inner loop	$x_0^r = x_0^b + \sum_{p=1}^{g-1} \delta x_0^p$	$s = B^{-1/2}\, x_0^r$

Table 3.3: Weighted and unweighted model space vectors handled by the **cgmod** congugate gradient.

Quantity	**Not weighted**	**Weighted**
Background vector	x_0^b	$b = B^{-1}\, x_0^b$
Current increment End of inner loop (outer loop g)	$\delta x_0^{(g)}$	$\delta u_0^{(g)} = B^{-1}\, \delta x_0^{(g)}$
Cummulated increment Entering inner loop (outer loop g)	$\sum_{p=1}^{g-1} \delta x_0^p$	$B^{-1} \sum_{p=1}^{g-1} \delta x_0^p$
Reference state vector when entering a new inner loop	$x_0^r = x_0^b + \sum_{p=1}^{g-1} \delta x_0^p$	$s = B^{-1}\, x_0^r$

3.1.3 Key difference between the congrad and cgmod inner drivers

A key difference between `congrad` and `cgmod` inner drivers relies on the control vector they handle.

- in `congrad`, the control variable is the increment weighted by $\mathbf{B^{1/2}}$:

$$\begin{aligned} J^g &= \tfrac{1}{2} || \sum_{q=1}^{g-1} \delta u_0^{(q)} + \delta u_0^g ||^2 \\ &+ \tfrac{1}{2} \sum_{k=0}^{n-1} \left(\mathbf{H}_k\, \mathbf{M}_{k,0}\, B^{1/2} \delta u_0 - d_k \right)^T R_k{}^{-1} \left(\mathbf{H}_k\, \mathbf{M}_{k,0}\, B^{1/2} \delta u_0 - d_k \right) \end{aligned} \tag{3.1}$$

$$\nabla J^g = \left(\sum_{q=1}^{g-1} \delta u_0^{(q)} + \delta u_0^g \right) + \sum_{k=0}^{n-1} B^{T/2} \mathbf{M}_{k,0}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1} \left(\mathbf{H}_k\, \mathbf{M}_{k,0} B^{1/2}\, \delta u_0^g - d_k \right) \tag{3.2}$$

- in `cgmod`, there are two types of control variables handled, the unweighted increment δx_0^g, and the increment weighted by $\mathbf{B^{-1}}$:

$$\begin{aligned} J^g &= \tfrac{1}{2}\left(\delta x_0^g + \textstyle\sum_{q=1}^{g-1} \delta x_0^{(q)}\right)^T B^{-1} \left(\delta x_0^g + \textstyle\sum_{q=1}^{g-1} \delta x_0^{(q)}\right) \\ &\quad + \tfrac{1}{2} \textstyle\sum_{k=0}^{n-1} \left(\mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0^g - d_k\right)^T R_k{}^{-1} \left(\mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0^g - d_k\right) \end{aligned} \tag{3.3}$$

$$\nabla J^g = B^{-1} \left(\delta x_0^g + \textstyle\sum_{q=1}^{g-1} \delta x_0^q\right) + \textstyle\sum_{k=0}^{n-1} \mathbf{M}_{k,0}{}^T \, \mathbf{H}_k{}^T \, R_k{}^{-1} \left(\mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0^g - d_k\right) \tag{3.4}$$

More details are provided section 3.4.4.

3.1.4 Routines `simvar` to calculate the cost function gradient

The J and ∇J calculation is performed with calls to the `simvar_` simulation routines defined in the sim_var.F90 source. The `simvar_` routines performs elementary matrix-vector operations that are classified in four groups according to the spaces they map. The congrad and cgmod inner drivers only apply calls to:

- simvar_x2x that performs a mapping from control space to itself. The header of the routine details the mapping options:

```
!!                  1) B
!!               or 2) B^1/2
!!               or 3) H^T R^-1 H
!!               or 4) B^T/2 H^T R^-1 H B^1/2
```

- simvar_y2x that performs a mapping from observation space to control space, with mapping options detailed in the header:

```
!!                  1) H^T R^-1
!!               or 2) B^T/2 H^T R^-1
!!               or 3) B H^T
```

The `rpcg` driver calls simvar_x2y, for a mapping from control space to observation space, and simvar_y2y for a mapping from observation space to itself. We will not describe these mapping routines.

3.1.5 Model and observation space vectors and the `ctlvec` structured type

The `simvar_` input and output arguments are variables of a key stuctured type called `ctlvec`. This type is specific to the nemovar.exe inner loop. It is defined in source control_vectors.F90:

```
TYPE ctlvec
   LOGICAL :: &
      & lalloc = .FALSE.                ! Allocation status
   INTEGER :: &
      & n3d    = 0                      ! Number of 3D fields
   INTEGER :: &
      & n2d    = 0                      ! Number of 2D fields
   INTEGER :: &
      & nex    = 0                      ! Number of extra data
   INTEGER :: &
      & nsize3d                         ! Size of each 3D field
   INTEGER :: &
      & nsize2d                         ! Size of each 2D field
   INTEGER :: &
      & nsize                           ! Total size of control vector
   REAL(wp), POINTER, DIMENSION(:) :: &
      & pdata                           ! Data
END TYPE ctlvec
```

The main field of the `ctlvec` type is `%padata` which is a 1D pointer array. It is dedicated to store 2D and 3D oceanic fields as well as observation data. Key vectors declared as `ctlvec` are:

- control and state vectors (i.e., s, x),
- control and state vector increments (i.e., δu, δx),
- observation or innovation vectors (i.e., y and d),
- non linear or linear model counterpart to obsevation vectors (i.e., $\mathcal{H}(x)$ or $H\,\delta x$),
- gradients of the cost function.

Several methods are defined to handle the `ctlvec` type:

- alloc_ctlvec, init_ctlvec, dealloc_ctlvec for allocation, initializing, deallocationg a `ctlvec` variable,
- setval_ctlvec, getval_ctlvec to load data from or into field %pdata,
- read_ctlvec, write_ctlvec which calls NEMO IOM routines to get or put the `ctlvec` field data in NEMO IOM files,
- and finaly, routines to perform operations on the %pdata fields (dot product, minimum or maximum value, norms, etc).

Model space vectors and `ctlvec` variable The number of 2D and 3D fields to be included in the `ctlvec` 1D pointer `%pdata` is controled by the integer variables `n2d` and `n3d`. Moreover, there exist index pointers to retrieve any given 2D or 3D oceanographic field stored in the 1D pointer . Index pointers are variables `n3d_t`, `n3d_ubs`, `n3d_ubu`, `n3d_ubv` and `n2d_ubssh`. They target the temperature, the unbalanced salinity, horizontal velocities,

and sea surface height, respectively.

The `nubsal`, `nubvel`, `nubssh` integer variables set the fields which are controled in the data assimilation system. These settings are controlled trough the namelist block `nambal`. The consistency with field index pointers is set as follows in the code:

```
! Setup model-space control vector size
n3d = 1 + nubsal + 2 * nubvel
n2d = nubssh
nex = 0

! Pointers for the control vector
n3d_t     = 1
n3d_ubs   = ( n3d_t + nubsal              ) * nubsal
n3d_ubu   = ( n3d_t + nubsal +     nubvel ) * nubvel
n3d_ubv   = ( n3d_t + nubsal + 2 * nubvel ) * nubvel
n2d_ubssh = nubssh
```

In nemovar.exe, some vectors of the assimilation system are directly loaded from file into `ctlvec` variables. Others are loaded in 3D or 2D arrays.

3.1.6 Model and observation space vectors and file storage

Many operations in the inner and outer loop consist in reading/writing vectors from/to files. The storage and related actions differ according to the data assimilation vector type. Let us detail this latter point.

Observation vector Observations can be either SLA, SST, temperature and/or salinity profiles, and lagrangian data. The I/O operations involving observations are the following:

- On the side of NEMO:
  - nemo stores the observation vector y^o, and the model counterpart $\mathcal{H}(x)$ in *feedback* files (the nemo.exe observation operator performs the model counter part calculation).
    The *feedback* files are of netcdf type with a well-defined structure. They enable the off-line communication between nemo.exe and nemovar.exe regarding the observation quantities (ref). There is one feedback file per assimilation cycle, and per data type.
- On the side of NEMOVAR:
  - nemovar loads the *feedback* file content in the `var` field of a structured type called `*datqc`, `*` being a prefix related to the observation type (e.g., `sla`, `sst`, `prof` or `lag`). The structured type also contains the `vext` field where is stored (after calculation by the nemovar inner loop):
    - the non-incremental innovation $y^o - \mathcal{H}(x^r)$,
    - the incremental linearized model counter part $\mathbf{H}\,\delta x_0$,

  - and finally the $\mathbf{R^{-1}}$ weighted model/observation misfit.

  More information on the structured type `*datqc` is provided in appendix B.1.1.

- nemovar inner drivers operates the transfer of the `datqc` innovation vectors into a single `ctlvec` variable called `z_y`, which is further applied to calculate the observation term of J and ∇J:

```
! Allocate control vectors in observation space
CALL alloc_ctlvec( z_y, 0, 0, nsize_obs )
(...)
!-----------------------------------------------------------------------
! B.2 Compute initial gradient and cost
!-----------------------------------------------------------------------
! Set y = -d (minus the innovation vector)
IF ( ln_t3d ) THEN
   ivar = 1
   DO jset = 1, nprofsets
      CALL setval_ctlvec( prodatqc(jset)%nvprot(ivar), ntob_off(jset), &
         &                -1.0_wp * prodatqc(jset)%var(ivar)%vext(:,mp3dinn), z_y )
   END DO
ENDIF
```

Background vectors and cumulated increments The I/O operations involving background or increment are the following:

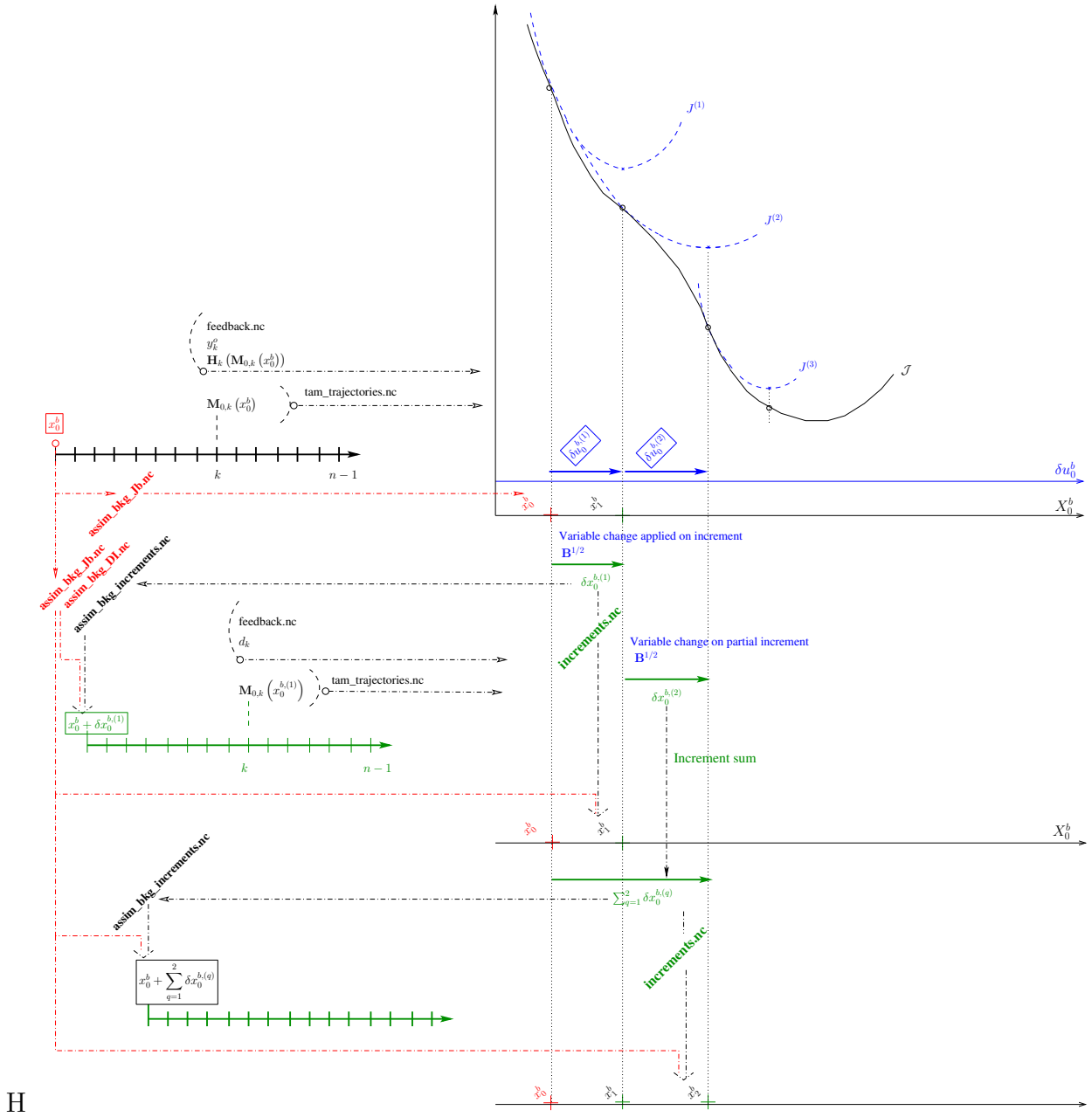
- NEMO stores the background state:
  - in the `assim_background_Jb` file, for off-line communication with NEMOVAR,
  - and in the `assim_background_DI`, or `assim_background_IAU` files, for off-line communication with itself.
- NEMOVAR stores the cumulated increment:
  - in the `increment` files, for off-line communication with NEMO [1]
  - in the `restart_ctlvec` files, for off-line communication with itself.

In particular, within nemovar.exe, the **background** and **cumulated increment** fields (temperature, salinity, horizontal velocities and sea surface height) can either be:

- loaded in 3D or 2D arrays in the nemovar_init step by the mean of NEMO IO routines,
- or directly loaded into `ctlvec` variables within the inner driver routines (congrad, cgmod).

At the end of the inner loop, the cummulated increment arrays are updated with the current increment `ctlvec` variable and further written to file.

[1] the piano python scripts targets the latter increment file with the symbolic link `assim_background_increment` for further reading by NEMO



H

Figure 3.1: Sketch showing an outer loop followed by inner loops.

NEMO/NEMOVAR off-line communication files

The sketch shown in figure 3.1 sumarizes the intrication between nemo.exe and nemovar.exe through the off-line communication files. Let us give a summary of the main files at hand in an assimilation experiment with nemo and nemovar:

First outer loop

1. NEMO reads the `restart` file.
2. NEMO writes the background files:
   - `assim_background_Jb` for communication with nemovar.exe,
   - `assim_background_DI` for communication with itself.
3. NEMO writes the `feedback` observation file (with observation and model counterpart).
4. NEMO writes the `tam_trajectory` files to record the trajectory for futher operator linearization around the trajectory states.

Subsequent outer loops

1. NEMO reads the `restart` file to reload all the background information.
2. NEMO reads the `assim_background_DI` file to reload the background fields controled through the state vector,
3. NEMO reads `assim_background_increment` to load the cumulated increments, and further update the *now* fields as a combination of the background and increment fields, e.g.:
$$tn = t_{\text{bkg}} + t_{\text{inc}}$$
4. NEMO writes the `feedback` observation file.
5. NEMO writes the `tam_trajectory` files to record the trajectory for futher operator linearization around the trajectory states.

Slight changes exist in the case of `IAU` option (see section 3.3.1).

In the next section, we propose a development strategy and the issues to be solved.

3.2 Overall strategy to implement the weak constraint 4D-Var

This section presents a quick overview of the strategy and problems for implementing the weak incremental 4D-Var.

3.2.1 Nemo outer loop: a single run over the assimilation window

We plan to run the NEMO outer loop in a single simulation as if the assimilation window was not split into subwindows. Our guiding lines for the implementation are:

- handle the discontinuous weak 4D-Var trajectories with the IAU capabilities for applying the background and increment,
- regarding these later options, avoid to split the netcdf outputs that are needed for the off-line communication between nemo and nemovar.

This strategy enables to keep unchanged the piano python script because the same off-line communication files will therefore be handled.

3.2.2 No split strategy and consequences

The no split strategy has consequences that we outline hereafter. Let us suppose in the following that we work with a number of `nsw` subwindows.

How to handle discontinuous trajectories in nemo outer loops?

To simulate a discontinuous trajectory in a single NEMO run, we must combine and apply the `nsw` background and increments states at the right time of the assimilation window (e.g., at the initial time step of each of the `nsw` assimilation subwindows if DI option is chosen). This approach involve the following requirements:

- In the first *nemo.exe outer loop*, we must record the `nsw` subwindow initial states of the background trajectory in a single background file. This is true for background files `assim_background_Jb` and `assim_background_DI` (slight changes exist for `_IAU` files).
- In the *nemovar.exe inner loop*, we must be able to calculate an increment for each of the `nsw` assimilation subwindow initial state (e.g., background for the first outer loop, and reference state for subsequent outer loops), and to record them in a single increment file. The files concerned are `increment` and `restart_ctlvec` or `restart_precvec`.
- For subsequent *nemo.exe outer loops*, we must succeed in applying at each of the `nsw` subwindow initial time step, an initial state that is the combination of the corresponding background and increment states.

Investigate the increment nemo assimilation module This issue will be detailed in section 3.3.1.

Augmented size for vectors of the model space: how to increase the dimension of arrays or ctlvec variables in nemovar.exe?

The weak 4D-var approach involves a number of 3D, 2D fields or a total number of components, augmented by a factor equal to the number of assimilation subwindows `nsw`. In nemovar.exe, the state or control vectors, the reference or background vectors, as well as the increment vectors are loaded from file:

- either in **2D or 3D arrays** which have shape and content that conforme to the ocean fields s, t, u, v or η.
- or in the `%pdata` **1D pointer** of a `ctlvec` variable.

We propose the following strategy.

One supplementary dimension for 2D and 3D arrays The main 3D and 2D arrays are the increment `inc_` and background `bkg_` arrays, which are loaded in the `nemovar_init` routine. One dimension must be added to both type of arrays, with an index ranging from 1 to the `nsw`.

Handle two `ctlvec` variables: a reduced and a full size variable The *init_ctlvec* subroutine of source control_vectors.F90, calculates the ad hock numbers of 3D and 2D fields to be included in the `ctlvec` variable. In many elementary routines called in nemovar.exe, it is assumed that the `ctlvec` variables store a `%pdata` 1D pointer only composed of the regular number of oceanographic fields t, s, u, v, and η. This is an issue since we have to deal with `nsw` more fields.

To avoid as much as possible to modify the nemovar routines, we propose the following strategy. Source control_vectors.F90 includes several routines to handle the `ctlvec` variables. It is in particular possible to define the number of 3D and 2D fields that compose the control vector with the *init_ctlvec* subroutine (see section 3.1.5). We propose to lean on this latter point, and handle two `ctlvec` variables instead of a single one:

- the regular `ctlvec` variable which store all the model fields defined over the *nsw* assimilation subwindows,
- a smaller `ctlvec` variable which only store the model fields of the current assimilation subwindow.

The *reduced* `ctlvec` variable should be locally declared, filled with the corresponding data, and handled as input argument of routines that are planned to receive a `ctlvec` variable with the regular number of oceanographic fields. Locally, a loop on the *nsw* subwindows will terefore be necessary to apply these routines on the `nsw ctlvec` variables.

For that purpose, we implement two new subroutines in source `control_vectors.F90` in order to operate the extraction from the full to reduced `ctlvec` variables:

- SUBROUTINE ctlvec_2_swctlvec:
  which extracts the current subwindow `ctlvec` variable from the full `ctlvec` variable,

- SUBROUTINE swctlvec_2_ctlvec:
  which updates the full `ctlvec` variable with the reduced `ctlvec` variable.

What about the size of the observation space vectors?

On the contrary to model space vectors, the size of observation space vectors remain unchanged since the assimilation window remains itself unchanged. Observation and model counter part are stored in feedback files with the corresponding time information. They are selectively retrieved by nemovar.exe on time condition when the observation term of J and ∇J are calculated.

Communication files: how to deal with a time record dimension issue?

For NEMO and NEMOVAR, the I/O modules that enable to read or write 2D or 3D field variables must be modified to handle a time record dimension. The I/O modules are in the NEMO/OPA_SRC/IOM directory.

Background and increment files The state vector size is augmented by a factor equal to `nsw`. It requires to store more than one time record in some of the netcdf files listed as off-line communication files in section 3.2.2. It is especially true for background and increment for which it is not a current option in the code. Slight modifications in the nemo I/O `iom_nf90_rstput` subroutine are needed.

Feedback observation files The NEMO outer loop essentially calculates the model counterpart to observation and stores it into the feedback files. No change is required, since the obsevation module records the time in feedback files.

I/O time record issue We examine the I/O issue in appendix A.1.2, and propose the changes to perform. Here is the sumary (see details in the appendix section A.1.3):

- For the reading interfaces `iom_get` and `iom_nf90_get`:
  No changes have to be done inside the interface subroutine themselves. However, the `ktime` argument, which stores the time index record of interest will have to be set in the NEMO and NEMOVAR routines, which calls the `iom_get` and `iom_nf90_get` interface.

- For the writting interface `iom_rstput` and `iom_nf90_rstput`:
  Changes must be done in the interface subroutines (detailed in appendix), and the NEMO and NEMOVAR calling subroutine must. The changes we propose are detailed in appendix A.1.2 section A.1.2. On the side of the NEMO and NEMOVAR routines

which calls the `iom_rstput` interface, the only change to make consists in paying attention to the two first arguments `kt` and `kwrite` that are oftenly both set to 1. They must be replaced by the write time step value.

A list of NEMO and NEMOVAR calling subroutines which are impacted with this issue are shown in table A.1.3 of appendix A.1.2. Changes in the calling syntax is also listed in the table.

3.3 NEMO outer loop: analysis and implementation

In this section we underline the main change to apply on the NEMO code.

3.3.1 The increment modules: description

There is mainly two important assimilation modules that have been added to NEMO in order to perform data assimilation experiments: the **observation module** and the **increment module**. The former enables to read many formats of observation files (including feedback), and further to process the data before assimialtion, in particular by calculating the model counterpart to observation. We will not detail this module because no change is required to set up a weak constraint incremental 4D-Var. We shall however focus on the increment related modules that can be found in the NEMO/OPA_SRC/ASM directory, which handle the reading, writting, combination and application of the background and increment vectors.

Module `asmpar`

The `asmpar` module stores parameters necessary to handle and combine increment and background:

- the names of the nemo and nemovar.exe off-line communication files:
  - `assim_background_DI.nc` or `assim_background_IAU.nc`, `assim_background_Jb.nc`,
  - and `assim_background_increment.nc`;
- time steps referenced to the initial assimilation window time step `nit000`:
  - `nitbkg` where to write the background in files,
  - `nitdin` where to combine the increment to the background when the Direct Initialization (hereafter DI) option is requested,
  - `nitiaustr` and `nitiaufin`, which is the time interval limits over which the increment is progessively combined to the current state. This kind of increment application is applied when the IAU option is requested.

The IAU option enables to apply the increment in a smooth manner because the total increment is progressively combined to the current state by means of a weight window.
See paragraph 3.3.1 for a description of the important NEMO namelist parameters that enable to control the increment application process.

Module asmbkg

The asmbkg module enables to write the background vector into background files with routine asm_bkg_wri. The argument of the routine is kt, the current time step (first time step being nit000-1). When kt is either equal to nitbkg or to nitdin, the current state is written in the background files.

Module asminc

The asminc module enables to apply the increment either with DI or IAU options. It stores four subroutines:

1. **Subroutine** asm_inc_init

   - **If IAU option is activated**, it allocates an array to store weights, and calculated these weights, in order to progressively apply the increment on the window:

     ```
     IF ( ln_asmiau ) THEN
       ALLOCATE( wgtiau( icycper ) )
     ```

   - It allocate increment arrays *_bkginc, and load increment fields from the increment file assim_background_increment:

     ```
     ALLOCATE( t_bkginc(jpi,jpj,jpk) )
     ALLOCATE( s_bkginc(jpi,jpj,jpk) )
     ALLOCATE( u_bkginc(jpi,jpj,jpk) )
     ALLOCATE( v_bkginc(jpi,jpj,jpk) )
     ALLOCATE( ssh_bkginc(jpi,jpj)   )
     ...
       IF ( ( ln_trainc ).OR.( ln_dyninc ).OR.( ln_sshinc ) ) THEN
          ...
          CALL iom_open( c_asminc, inum )
     ```

   - **If DI option is activated**, it allocates the background arrays *_bkg, and load background fields from file assim_background_DI:

     ```
     IF ( ln_asmdin ) THEN
       ...
       ALLOCATE( t_bkg(jpi,jpj,jpk) )
       ALLOCATE( s_bkg(jpi,jpj,jpk) )
       ALLOCATE( u_bkg(jpi,jpj,jpk) )
       ALLOCATE( v_bkg(jpi,jpj,jpk) )
       ALLOCATE( ssh_bkg(jpi,jpj)   )
       ...
             CALL iom_open( c_asmdin, inum )
     ```

2. **Subroutines `tra_asm_inc`, `dyn_asm_inc`, and `ssh_asm_inc`**
   These three routines combine the increment arrays `*_bkginc` either to the background fields (DI option) or progressively to states of the defined IAU time interval. Lets describe the algorithm with the tracer fields:

   - **If IAU option is activated**: the increment is combined to the *after* tracer fields (i.e., `ta` and `sa`), by the mean of the weight array, only if the current time step is consistent with the IAU time step interval:

```
IF ( ln_asmiau ) THEN
...
 IF ( ( kt >= nitiaustr_r ).AND.( kt <= nitiaufin_r ) ) THEN
 ...
  zincwgt = wgtiau(it) / rdt   ! IAU weight for the current time step
  ta(:,:,jk) = ta(:,:,jk) + t_bkginc(:,:,jk) * zincwgt
  sa(:,:,jk) = sa(:,:,jk) + s_bkginc(:,:,jk) * zincwgt
```

   - **if DI option is activated**: the increment is combine to the *now* tracer fields (i.e., `tn` and `sn`), only if the current time step is consistent with the `nitbkgr` time step:

```
IF ( kt == nitdin_r ) THEN
...
 tn(:,:,:) = t_bkg(:,:,:) + t_bkginc(:,:,:)
 sn(:,:,:) = s_bkg(:,:,:) + s_bkginc(:,:,:)
```

Namelist parameters to control background/increment options

The namelist block `nam_asminc` (see `namelist_nemo`) store parameters to control the increment process. The `nam_asminc` options are effective if the NEMO code is precompiled with the cpp key `key_asminc` (the `lk_asminc` logical parameter is than set to true in the code).

- either `ln_asmdin` or `ln_asmiau` must be set to TRUE:
  It triggers either the **DI** or **IAU** option,
- `ln_trainc`, `ln_dyninc`, and `ln_sshinc`:
  They seperately control the application of the tracer increment on tracer fields, the dynamic increment on velocity fields, and the η increment on sea surface height field,
- `ln_bkgwri` triggers the writing of the background files when set to TRUE.

Be aware that in a NEMO/NEMOVAR an incremental 4D-Var simulation:

- the trigger of the writing of the background files is only required at the first outer loop.
- it must be deactivated for subsequent outer loop (`ln_bkgwri` set to FALSE), while the trigger of increment application must be activated with:
  - either `ln_asmdin` or `ln_asmiau` set to TRUE,
  - and at least one of the following parameters set to TRUE:
    `ln_trainc`, `ln_dyninc`, and `ln_sshinc`

3.3.2 Increment module: weak constraint 4D-Var implementation

The `asmpar` module

We add to the `asmpar` module some new parameters to handle the assimilation subwindows:

- `ln_asmsw`:
  It is the logical variable that must be set to `.true.` if assimilation subwindows are needed (which is the case for the weak constraint 4D-Var).

- `nsw`:
  It is the integer that stores the number of assimilation subwindows.

- `isw`:
  It is the integer that will help to track the current assimilation subwindow,

- `nitsw`:
  It is an array of integers that stores the time step boundaries of each subwindow,

- `nit000_sav`, `nitend_sav`:
  It is integers to save the full assimilation window time boundaries (it will prevent to apply changes on low level routines that will have to run on subwindows instead of the full assimilation window; `nit000` and `nitend` will be replaced by the current subwindow time step boundaries, while the native time step boundaries will be saved for further use).

We subsequently introduce a new module called `asmsubwin`, which contains two subroutines to handle the `asmpar` module parameters:

- `asm_spltwin`:
  It enables to cut the full assimilation window into `nsw` subwindows, by initializing the `nitsw` array.

- `swap_nit` and `swapbck_nit`:
  It enables to set the global variables `nit000` and `nitend` to the current assimilation subwindow time limits and vice versa.

Increment subroutines: sequence of calls in `opa` and `step`

`opa` At the time step `nit000-1`:

- It calls subroutine `opa_init`, which itself calls `asm_inc_init`:

```
IF( lk_asminc ) CALL asm_inc_init     ! Initialize assimilation increments
```

  `nit000` is added to `nitbkg`, `nitdi` and `nitiau`, which are no longer referenced to `nit000`.

- If `ln_bkgwri` is set to true (first outer loop), the background routine `asm_bkg_wri` is called and the background file for time step `nit000-1` is written.
- If `ln_asmdi` set to true (subsequent outer loops, Direct Initialization option), it calls the increment and background application, with `*_asm_inc`, `*` being `tra`, `dyn` and/or `ssh`.

```
istp = nit000 - 1
IF( lk_asminc ) THEN
   IF( ln_bkgwri ) THEN
       CALL asm_bkg_wri( istp )    ! Output background fields
   ENDIF
   IF( ln_asmdin ) THEN                        ! Direct initialization
      IF( ln_trainc ) CALL tra_asm_inc( istp )    ! Tracers
      IF( ln_dyninc ) CALL dyn_asm_inc( istp )    ! Dynamics
      IF( ln_sshinc ) CALL ssh_asm_inc( istp )    ! SSH
```

Then `opa` launch a loop ranging from `nit000` to `nitend` within which it calls the `stp` routine of source `step.F90`.

`step` In the case of IAU option, for the tracer and dynamic fields, this is `step` routine that applies the increment to the current state by calling the `*_asm_inc` subroutines:

```
 IF(  ln_asmiau .AND. &
    & ln_trainc     ) &
    & THEN
    CALL tra_asm_inc( kstp )       ! apply tracer assimilation increment
 ENDIF
 (...)
 IF(  ln_asmiau .AND. &
    & ln_dyninc     ) &
    &  THEN
    CALL dyn_asm_inc( kstp )       ! apply dynamics assimilation increment
 ENDIF

IF( ln_trjwri ) CALL tam_trj_wri( kstp )          ! Output trajectory fields
IF( lk_asminc ) THEN
   IF( ln_bkgwri ) CALL asm_bkg_wri( kstp )       ! Output background fields
   IF( ln_asmdin ) THEN                           ! Direct initialization
      IF( ln_trainc ) CALL tra_asm_inc( kstp )        ! Tracers
      IF( ln_dyninc ) CALL dyn_asm_inc( kstp )        ! Dynamics
      IF( ln_sshinc ) CALL ssh_asm_inc( kstp )        ! SSH
   ENDIF
ENDIF
```

The `ssh_asm_inc` routine however is called

```
CALL dyn_spg( kstp, indic )    ! surface pressure gradient
```

Modifications in `step` and `opa` Background writting as increment application are both performed accoding to time requirements. For that purpose, the current time step is tested against the time parameters `nitbkg`, `nitdin`, `nitiau_str` or `nitiau_fin` (see 3.3.1). For instance, in the case of IAU option, the test is the following:

```
IF ( ( kt >= nitiaustr ).AND.( kt <= nitiaufin ) THEN
```

In the case of several assimilation subwindows, the time parameters change at any new assimilation subwindow. We propose to update their value, after any background writting or increment application.

- `nitbkg` is directly updated at the end of subroutine `asm_bkg_wri` in source `bkgwri.F90`:

```
   IF ( ln_asmsw ) THEN
      isw = isw + 1
      IF ( kt == nitbkg_r )  nitbkg_r  = nitbkg_r + nitsw(isw)    ! Background time referenced to nit000
      IF ( kt == nitdin_r )  nitdin_r  = nitdin_r + nitsw(isw)    ! Background time for DI referenced to nit000
   ENDIF

END SUBROUTINE asm_bkg_wri
```

- `nitdin`, `nitiau_str` or `nitiau_fin` are updated with subroutine `asm_inc_nitptr` through calls either in the `opa` or `step` sources:

  - In `opa`, right after applying the DI increment

```
      IF( ln_asmdin ) THEN                        ! Direct initialization
         IF( ln_trainc ) CALL tra_asm_inc( istp )    ! Tracers
         IF( ln_dyninc ) CALL dyn_asm_inc( istp )    ! Dynamics
         IF( ln_sshinc ) CALL ssh_asm_inc( istp )    ! SSH
#if defined key_asmsw
            CALL asm_inc_nitptr( istp )
#endif
      ENDIF
```

  - In `step`, after calls to the `*_asm_inc` routines:
    Pay attention that the `ssh` increment is applied in the `dyn_spg` subroutine/

```
      CALL dyn_spg( kstp, indic )    ! surface pressure gradient
(...)
#if defined key_asmsw
  IF(  ln_asmiau .AND.    &
     &( ln_trainc .OR.    &
     &  ln_dyninc .OR.    &
     &  ln_sshinc     ) ) &
     & THEN
    CALL asm_inc_nitptr( kstp )
  ENDIF
#endif
```

Subroutine `asm_inc_nitptr` involve the following instructions:

```
IF ( ln_asmiau ) THEN

   IF ( ( kt >= nitiaustr_r ).AND.( kt <= nitiaufin_r ) ) THEN
      ! If several subwindows, increment time window pointers for IAU
      isw = isw + 1
      nitiaustr_r = nitiaustr_r + nitsw(isw)    ! Start of IAU interval referenced to nit000
      nitiaufin_r = nitiaufin_r + nitsw(isw)    ! End of IAU interval referenced to nit000
   ENDIF

ELSEIF ( ln_asmdin ) THEN

   IF ( kt == nitdin_r ) THEN
```

```
         ! If several subwindows, increment time window pointers for DI
         isw = isw + 1
         nitdin_r    = nitdin_r + nitsw(isw)       ! Background time for DI referenced to nit000
       ENDIF

ENDIF
```

3.4 NEMOVAR inner loop: analysis and implementation

In this section we provide an analysis of the NEMOVAR code. We describe the algorithm of the following key routines:

- `nemovar_init`,
- `congrad_inner_driver`,
- `cgmod_inner_driver`,
- `simvar_x2x`,
- and `simvar_y2x`.

For each key routine, we propose some implementations to build a weak contraint formulation of the incremental 4D-Var. Let us first give a list of the important quantities with their corresponding name in the code.

3.4.1 Quantities and code naming

Table 3.4 gives the correspondance between model space quantities and their names within the code.

Table 3.4: Correspondance between model space vectors and code variables of array type.

Quantity	**Code variable**
Background x_0^b	`t_bkg`, `s_bkg`, `u_bkg`,...etc.
Cumulated increment $\sum_{q=1}^{g-1} \delta x_0$	`t_inc`, `s_inc`, `u_inc`,...etc.
Reference state x_0^r	`t_bkg` + `t_inc`,...etc.
State trajectory $x_{k+1} = \mathcal{M}_{k,k+1}(x_k)$	`tn`, `sn`, `un`,...etc.
Increment δx_0	`tn_tl`, `sn_tl`,...etc.
Increment trajectory $\delta x_{k+1} = \mathbf{M}_{k,k+1}\, \delta x_k$	`tn_tl`, `sn_tl`,...etc.
Initial adjoint state $x_n^* = \mathbf{H}_n^T\, {R_n}^{-1} (\mathbf{H}_n \delta x_n - d_n)$	`tn_ad`,`sn_ad`, `un_ad`,...etc.
Adjoint states $x_{n-1}^* = \mathbf{M}_{n-1,n}^T x_n^* + \mathbf{H}_{n-1}^T\, {R_{n-1}}^{-1} (\mathbf{H}_{n-1} \delta x_{n-1} - d_{n-1})$	`tn_ad`,`sn_ad`, `un_ad`,...etc.
Adjoint state trajectory with respect to unbalanced increment $\delta x_{u,n-1}$ $x_{n-1}^* = \mathbf{M}_{n-1,n}^T x_n^* + \mathbf{K}^{T/2}\, \mathbf{H}_{n-1}^T\, {R_{n-1}}^{-1} (\mathbf{H}_{n-1} \mathbf{K}^{1/2} \delta x_{u,n-1} - d_{n-1})$	XXX
Adjoint state trajectory with respect to normalized unbalanced increment δu_{n-1} $x_{n-1}^* = \mathbf{M}_{n-1,n}^T x_n^* + {\mathbf{B_u}}^{1/2} \mathbf{K}^{T/2}\, \mathbf{H}_{n-1}^T\, {R_{n-1}}^{-1} \left(\mathbf{H}_{n-1} \mathbf{K}^{1/2} {\mathbf{B_u}}^{1/2} \delta u_{n-1} - d_{n-1}\right)$	XXX

3.4.2 Routine nemovar_init

Let us describe the sequence of important instructions of the `nemovar_init` code.

Initialization of the nemovar namelist block parameters

One example may be the reading of the `namalg` block, which is operated this way:

```
NAMELIST/namalg/ ln_diamat, nvarex, noutmax, ninnmax,  noutit, &
                 ln_ref, lnorrep
...
! Set namalg namelists defaults
ln_diamat = .FALSE.
ln_ref    = .FALSE.
REWIND( numnam )
READ  ( numnam, namalg )
```

4D-Var implementation The three following parameters are added to the `namalg` block of the `nanmelist_nemovar`:

- `ln_asmsw` which has to be set to TRUE to activate the control of several subwindow initial states,

- `nsw` which is an integer that must be equal to the number of assimilation subwindows,
- `isw` which is the current subwindow.

We change the nemovar_init declaration section accordingly:

```
NAMELIST/namalg/ ln_diamat, nvarex, noutmax, ninnmax,  noutit, nsw, &
                 ln_asmsw, ln_ref, lnorrep
```

Tangent and adjoint variables reset to zero

```
CALL oce_tam_init(0)
```

It allocates and initializes to zero the tangent linear and adjoint fields for the inner loop (case of argument set to 0). The impacted variables are arrays whose names in the code have the `*_tl` or `*_ad` suffixes, where `*` can be `tn`, `sn`, etc, for instance.

4D-Var implementation No change required.

Initialization of number and pointer indices of fields stored in `ctlvec` variable

```
n3d_t     = 1
n3d_ubs   = ( n3d_t + nubsal              ) * nubsal
n3d_ubu   = ( n3d_t + nubsal +     nubvel ) * nubvel
n3d_ubv   = ( n3d_t + nubsal + 2 * nubvel ) * nubvel
```

The controled 3D and 2D fields are organized in a 1D pointer array in the NEMOVAR code. Some pointer variables are declared to retrieve the range of index corresponding to a particular field:

- `n3d_t` for temperature,
- `n3d_ubs` for (unbalanced) salinity,
- `n3d_ubu` for (unbalanced) velocity,
- `n3d_ubv` for (unbalanced) zonal velocity,
- `n3d_ubssh` for unbalanced sea surface height.

The pointer integer indices are set at runtime depending on the `nubsal`, `nubvel` values set by the user in the `namelist_nemovar`.

4D-Var implementation Field pointers must now be arrays of integer instead of integers, since we deal with `nsw` fields of each type (temperature, salinity, velocities and sea surface height) per subwindow.

For that purpose, the modified `asmpar` module (see section 3.3.1) is applied to define the number of assimilation subwindows `nsw`:

```
USE asmpar, ONLY : &
  & nsw,
  & ...
```

To handle both a full and a reduced `ctlvec` variable, we must duplicate the variables related to field number and field index, which enable to retrieve an oceanographic field in a `ctlvec` variable (either a full or a reduced `ctlvec` variable). Field number and field index are variables declared in the `varctl` module:

- `n3d`, `n2d`, `nex`:
  the number of 2D and 3D fields (plus extra variable) stored in the 1D full control vector.
- `n3d_t`, `n3d_ubs`, `n3d_ubu`, `n3d_ubv`, `n2_ubssh`:
  the pointer indices to retrieve a given field in the 1D full control vector.

We chose to define a twin module called `swvarctl` to include:

- variables to store the number of 3D and 2D fields of the reduced `ctlvec`:
  `n3d_sw` and `n2d_sw`
- array variables that store the pointer indices targetting each field of a given subwindow:
  e.g.: for temperature fields, we define 1D array `n3d_t_sw` of size `nsw`, with `n3d_t_sw(isw)` enabling to target the temperature fields of subwindow `isw`.

The `swvarctl_init` subroutine initializes the former variables.

Initialization of the background arrays `*_bkg`

This is done with a call to `bkg_init` subroutine of the `bkg` module:

```
CALL bkg_init
```

This routine allocates the `_bkg` arrays to store the background fields, and calls:

- `bkg_rea`, which reads and loads the background fields from the `assim_background_state_Jb.nc` file, and further stores the previous fields in the `*_bkg` arrays.
- or `bkg_ref`, which constructs a reference background, and stores the fields in the `_bkg` arrays.

It is important to note that the `_bkg` arrays are only loaded to construct the balance operator, since the background fields are not required for the computation of the quadratic cost function $J^{(g)}$ and gradient. This issue is detailed in appendix A.1.1 with especially a comparison between the treatment of the `_bkg` arrays, and the increment fields stored in the `_inc` arrays.

4D-Var implementation o take into account the augmented size of the background with the number of subwindows, a dimension must be added to the `*_bkg` arrays:

```
#if defined key_asmsw
      ALLOCATE( t_bkg(1:nsw,jpi,jpj,jpk) )
      ALLOCATE( s_bkg(1:nsw,jpi,jpj,jpk) )
      ALLOCATE( u_bkg(1:nsw,jpi,jpj,jpk) )
      ALLOCATE( v_bkg(1:nsw,jpi,jpj,jpk) )
      ALLOCATE( ssh_bkg(1:nsw,jpi,jpj)   )
      ALLOCATE( gcx_bkg(1:nsw,jpi,jpj)   )
```

This supplementary dimension involves issues in the reading process of the background file. The augmented dimension involves I/O subroutine modifications to take into account the existence of several time records in the background files (but also in increment files). This issue is extensively treated in appendi A.1.2.

The `bkg_rea` as the `bkg_ref` subroutines must be modified. On the side of `bkg_rea`, we must enable the reading of all the time records stored in the background file:

```
#if defined key_asmsw
      DO i = 1, nsw
         ! i index corresponds to the ktime optional argument that stores the time record index
         ! the kstart, kcount time dimension are set accordingly
         CALL iom_get( inum, jpdom_autoglo, 'un', u_bkg(i,:,:,:), i )
         CALL iom_get( inum, jpdom_autoglo, 'vn', v_bkg(i,:,:,:), i )
         CALL iom_get( inum, jpdom_autoglo, 'tn', t_bkg(i,:,:,:), i )
         CALL iom_get( inum, jpdom_autoglo, 'sn', s_bkg(i,:,:,:), i )
         CALL iom_get( inum, jpdom_autoglo, 'sshn', ssh_bkg(i,:,:), i )
         CALL iom_get( inum, jpdom_autoglo, 'gcx', gcx_bkg(i,:,:), i )
      END DO
```

Call to `day_init` and `day(nit000)`

Routine `day` with argument `nit000` calculates the integer `ndastp`, which is the time step in yyyymmdd format.

ndastp in opa.F90 This opa.F90 that organizes the loop on time step `istp` from `nit000` to `nitend`, and calls to the `stp` routine of step.F90:

```
     istp = nit000
     DO WHILE ( istp <= nitend .AND. nstop == 0 )
        CALL stp( istp )
        istp = istp + 1
```

Before this loop, opa.F90 calls `opa_init`, which itself calls subroutine `istate_init`, whithin which the `day_init` routine is at last called. The `day_init` routine further calls `day_rst`, which calculates the `ndastp` date on the basis of the restart time step and date `nit000`, and `ndat000`:

```
SUBROUTINE day_init
  (...)
  CALL day_rst( nit000, 'READ' )
  ! set the calandar from ndastp (read in restart file and namelist)
  nyear   =   ndastp / 10000
```

Unless ndrstp is set to 2, we have:

```
ndastp = ndate0 - 1
```

Further (but before the loop on istp), opa.F90 calls asm_bkg_wri, istp being set to nit000−1. Subroutine asm_bkg_wri handles this latter special case:

```
IF ( kt == nitbkg_r ) THEN
    (...)
    ! Treat special case when nitbkg = 0
    IF ( nitbkg_r == nit000 - 1 ) THEN
       zdate = REAL( ndate0 )
    (...)
    ELSE
       CALL calc_date( nit000, nitbkg_r, ndate0, idate )
       zdate = REAL( idate )
    ENDIF

    ! Write the information
    CALL iom_rstput( kt, nitbkg_r, inum, 'rdastp' , zdate   )
    CALL iom_rstput( kt, nitbkg_r, inum, 'un'     , un      )
    (...)
IF ( kt == nitdin_r ) THEN
    (...)
    ! Treat special case when nitbkg = 0
    IF ( nitdin_r == nit000 - 1) THEN
       zdate = REAL( ndastp )
    ELSE
       zdate = REAL( ndastp )
    ENDIF

    ! Write the information
    CALL iom_rstput( kt, nitdin_r, inum, 'rdastp' , zdate   )
```

We can see that the initial background states are written in background files at nitbkg and at nitdin, if there is a consistency in namelist data. Caution: At opa step, to enter the asm_bkg_wri condition, nitbkg_r or nitdin_r must be set to nit000−1. Moreover:

- For background_increment_Jb, the written date zdate is ndate0, or calulated according to nitbkg_r.
- For background_increment_DI, the written date is always ndastp=ndate0-1:
  Problem?!: we will not have the right date written for DI combination with the current state.
  Unless if we exclude to apply DI, for several nsw assimilation subwindows;

In step.F90, the day routine is called at each time step to update the calendar date to the current time step kstp, which especially impacts ndastp. One question is to understand how to write several nsw initial states in the background file, and if the different ndastp date will be written in the file: updating nitbkg_r is sufficient? What to do with nitdin_r writting? See the issue in bkg_init, while reading the background and the ndastp variable. Proposal: call calc_date of the asminc module?

opa_init calls istate before the first call to asm_bkg_wri. The istate routine calls day_rst where ndast is initialized to ndate0 − 1 (Why?). In asm_bkg_wri the background record is written at any nitbkg_r on one side, and at any nitdin_r on the other side. The iom_rstput routine enables in particular to write the corresponding date in yyyymmdd format (see field 'rdastp'). Two cases are handled: (i) if nitbkg_r = nit000 − 1; (ii) and else.

Initialization of the increment arrays *_inc

This is done with a call to inc_init subroutine of the inc module:

```
CALL inc_init( ndastp )
```

The increment initialization consists in reading the cumulated increment from the increment file recorded at the end of the last cycle of inner loops. It stores the increment fields in the *_inc arrays.

4D-Var implementation dimension must be added to the *_inc arrays, to take into account the augmented size of the background with the number of subwindows:

```
#if defined key_asmsw
      ALLOCATE( t_inc(1:nsw,jpi,jpj,jpk) )
      ALLOCATE( s_inc(1:nsw,jpi,jpj,jpk) )
      ALLOCATE( u_inc(1:nsw,jpi,jpj,jpk) )
      ALLOCATE( v_inc(1:nsw,jpi,jpj,jpk) )
      ALLOCATE( ssh_inc(1:nsw,jpi,jpj)   )
      ALLOCATE( gcx_inc(1:nsw,jpi,jpj)   )
```

Argument ndastp must be checked (equal to kinc in the code below):

```
   SUBROUTINE inc_init( kinc )

         CALL iom_get( inum, 'time' ,       zinc )
         IF ( NINT(zinc) /= kinc) THEN
            WRITE(numout,*)'W A R N I N G'
            WRITE(numout,*)'Increment valid time       = ', nint(zinc)
            WRITE(numout,*)'is different from expected = ', kinc
            WRITE(numout,*)
         ENDIF

         CALL iom_get( inum, 'z_inc_dateb', z_inc_dateb )
         CALL iom_get( inum, 'z_inc_datef', z_inc_datef )
#if defined key_wk4dv
         DO i = 1, nsw
            ! i is the time record index
            CALL iom_get( inum, jpdom_autoglo, 'bckint' ,   t_inc(i,:,:,:), i )
            CALL iom_get( inum, jpdom_autoglo, 'bckins' ,   s_inc(i,:,:,:), i )
            CALL iom_get( inum, jpdom_autoglo, 'bckinu' ,   u_inc(i,:,:,:), i )
            CALL iom_get( inum, jpdom_autoglo, 'bckinv' ,   v_inc(i,:,:,:), i )
            CALL iom_get( inum, jpdom_autoglo, 'bckineta' , ssh_inc(i,:,:), i )
         END DO
#else
         CALL iom_get( inum, jpdom_autoglo, 'bckint' ,   t_inc(:,:,:), 1 )
         CALL iom_get( inum, jpdom_autoglo, 'bckins' ,   s_inc(:,:,:), 1 )
         CALL iom_get( inum, jpdom_autoglo, 'bckinu' ,   u_inc(:,:,:), 1 )
         CALL iom_get( inum, jpdom_autoglo, 'bckinv' ,   v_inc(:,:,:), 1 )
         CALL iom_get( inum, jpdom_autoglo, 'bckineta' , ssh_inc(:,:), 1 )
#endif
```

Set up of the background correlation errors

This is done trough the call to the bge_init subroutine of module bge_setup (BGE/bge_setup.F90). This routine sets parameters of the background correlation errors. The namelist block nambge controls how the background correlation errors are calculated.

```
!-----------------------------------------------------------------------
!       nambge    Background error correlation parameters
!-----------------------------------------------------------------------
!
!  nbgecor    Type of correlation
!             0 = No correlations
!             1 = Diffusion: Explicit 2D horizontal + Explicit 1D vertical
!             2 = Diffusion: Implicit 3 x 1D
(...)
!  nbgecof    Method to compute the diffusion coefficients
!             0 = Constant
!             1 = Around equator param. + vertical param.
!             2 = Rossby radius param. + vertical param.
!             3 = Read in a file
!             4 = Around equator param. + mixed layer param.
!  nbgenor    Method to compute the normalization
!             0 = exact method
!             1 = randomization method
!             2 = randomization method with spatial averaging
!             3 = approximation of the theoritical factor with smoothed
!                 diffusion coefficients
!  nbgenes    Size of current ensemble for computing the diffusion
!           normalization factors with the randomization method
```

The `bge_difcof` is called and according to the value of the `nbgecof` parameter in the namelist block `nambge`, different subroutines included in COR/difcof_setup.F90 are further called:

```
      & nbgecof,      & ! Method to compute the diffusion coefficients
                        !   0 = Constant
                        !   1 = Around equator param. + vertical param.
                        !   2 = Rossby radius param. + vertical param.
                        !   3 = Read in a file
                        !   4 = Around equator param. + mixed layer param.
```

In particular, if `nbgecof` is set to 4, the `compute_mxl` subroutine is applied. This latter routine requires the background arrays `_bkg`.

4D-Var implementation The shape of the background arrays must be set conditionnal to `key_asmwin` in subroutine `compute_mxl`.

Balance operator initialization

This is done through the call to `bkg_bal_init` subroutine (BAL/bkg_bal.F90). It initializes the balance operator that makes part of the B error covariance matrix (see appendix B.2). The balance operator is linearized around the reference state.

The bkg_bal_init calls:

```
! Salinity balance initialization

CALL bkg_bal_sal

! SSH balance initialization

CALL bkg_bal_ssh
```

```
! Velocity balance initialization

CALL bkg_bal_vel
```

It requires the background and increment arrays because it requires the reference state:

```
tn(:,:,:) = t_bkg(:,:,:) + t_inc(:,:,:)
sn(:,:,:) = s_bkg(:,:,:) + s_inc(:,:,:)

(...)

! Initialize for avt (background) computation
ub(:,:,:) = u_bkg(:,:,:)
un(:,:,:) = u_bkg(:,:,:)
vb(:,:,:) = v_bkg(:,:,:)
vn(:,:,:) = v_bkg(:,:,:)
```

4D-Var implementation *Important: inc and bkg arrays must have a fourth dimension corresponding to the assimilation subwindows.*
Important: other arrays concerned by the size problem u_b, u_n ?

It also computes the "active points in the ocean" by adding unity each time the mask array is different from zero, and the result is stored in:

```
! Control variable array sizes (unmasked points)

i3d_t_size      = i3d_tpts
i3d_ubs_size    = i3d_tpts * nubsal
i3d_ubu_size    = i3d_upts * nubvel
i3d_ubv_size    = i3d_vpts * nubvel
i2d_ubssh_size  = i2d_tpts * nubssh

itot_size  = i3d_t_size   + i3d_ubs_size   + i3d_ubu_size &
   &       + i3d_ubv_size + i2d_ubssh_size
```

Potential problems here?

Background standard deviation construction

The bkg_err_init subroutine (BGE/bkg_err.F90) enables to construct the background standard deviation, which are stored in variables:

```
& sd_uana_bkg,   & !: Standard deviation of u_bkg error
& sd_vana_bkg,   & !: Standard deviation of v_bkg error
& sd_tana_bkg,   & !: Standard deviation of t_bkg error
& sd_sana_bkg,   & !: Standard deviation of s_bkg error
& sd_sshana_bkg    !: Standard deviation of ssh_bkg error
```

The std variables are:

- either estimated through `bkg_err_def_std`,
- or read in files with `bkg_err_rea_std` (not yet developed in nemovar version X).

```
SUBROUTINE bkg_err_init
(...)
   IF ( lrbksd ) THEN
      ! Read from file
      CALL bkg_err_rea_std
   ELSE
      ! Parameterized options
      CALL bkg_err_def_std
   ENDIF
   IF ( ( nsd3dbkgp == 1 ) .OR. ( nsd3dbkgp == 2 ) ) THEN
      ! Read from file: available on a regular grid
      CALL bkg_err_read_reg
   ENDIF

   CALL bkg_err_diag
  (...)
END SUBROUTINE bkg_err_init
```

In order to construct the std variables, the `bkg_err_def_std` loads the background arrays:

```
SUBROUTINE bkg_err_def_std
(...)
   ! Compute the background dS/dT
   tn(:,:,:) = t_bkg(:,:,:)
   sn(:,:,:) = s_bkg(:,:,:)
```

4D-Var implementation One goal is to be able to define specific background standard deviations for each assimilation subwindow and initial backround state. For that purpose, a supplementary dimension must be added to the std variable arrays, following the approach applied to the background field variables:

```
#if defined key_asmsw
   REAL(wp), DIMENSION(:,:,:,:), ALLOCATABLE :: &
      & sd_uana_bkg, &  ! Standard deviation of u_bkg error
      & sd_vana_bkg, &  ! Standard deviation of v_bkg error
      & sd_tana_bkg, &  ! Standard deviation of t_bkg error
      & sd_sana_bkg     ! Standard deviation of s_bkg error

   REAL(wp), DIMENSION(:,:,:), ALLOCATABLE :: &
      & sd_sshana_bkg   ! Standard deviation of ssh_bkg error
#else
   REAL(wp), DIMENSION(:,:,:), ALLOCATABLE, TARGET :: &
      & sd_uana_bkg, &  ! Standard deviation of u_bkg error
      & sd_vana_bkg, &  ! Standard deviation of v_bkg error
      & sd_tana_bkg, &  ! Standard deviation of t_bkg error
      & sd_sana_bkg     ! Standard deviation of s_bkg error
```

```
   REAL(wp), DIMENSION(:,:), ALLOCATABLE , TARGET :: &
      & sd_sshana_bkg   ! Standard deviation of ssh_bkg error
#endif
```

And a loop on assimilation subwindows must be introduced. It is however not possible to put it directly in `nemovar_init` subroutine. The `bkg_err_diag` routine indeed applies the tangent model operator to the standard deviation (dynamical and observation tangent model operators), with instructions that are not compatible with the subwindow loop:

```
      tn_tl(:,:,:) = sd_tana_bkg(:,:,:) * tmask(:,:,:)
      sn_tl(:,:,:) = sd_sana_bkg(:,:,:) * tmask(:,:,:)
      un_tl(:,:,:) = sd_uana_bkg(:,:,:) * umask(:,:,:)
      vn_tl(:,:,:) = sd_vana_bkg(:,:,:) * vmask(:,:,:)
      sshn_tl(:,:) = sd_sshana_bkg(:,:) * tmask(:,:,1)

      CALL dia_obs_tan_init
      CALL dia_obs_tan( nit000 - 1 )

      DO jstp = nit000, nitend

         ! Call the observation operators
         CALL dia_obs_tan( jstp )

      END DO

      (...)

      IF ( ln_t3d ) THEN
         ivar = 1
         DO jset = 1, nprofsets
            ! Update section bounds
            jini  = jend + 1
            jend  = jend + prodatqc(jset)%nvprot(ivar)
            prodatqc(jset)%var(ivar)%vext(:,mp3dbesdana) = prodatqc(jset)%var(ivar)%vext(:,n
         END DO
      ENDIF
```

The instructions involving the `prodatqc` variable must remain outside of the subwindow loop. For that reason, we selectively introduce the loop inside the subroutines of the `bkg_err` module. We apply the routines `swap_nit` and `swapbck_nit` where it is required (one can note that `nit000` is applied at some places):

```
#if defined key_asmsw
```

```
      DO isw = 1, nsw
#endif
      CALL swap_nit
      (...)
      CALL swapbck_nit
#if defined key_asmsw
      ENDDO
#endif
```

The call to `bkg_err_diag` enables to construct the variables `%var(ivar)%vext(:,mp3dbesdana)` that are (to check) the interpolated background standard deviation, which are defined on the whole assimilation window. Here is the corresponding instruction for profile data:

```
  IF ( ln_t3d ) THEN
     ivar = 1
     DO jset = 1, nprofsets
        ! Update section bounds
        jini   = jend + 1
        jend   = jend + prodatqc(jset)%nvprot(ivar)
        prodatqc(jset)%var(ivar)%vext(:,mp3dbesdana) = prodatqc(jset)%var(ivar)%vext(:,mp3dt
     END DO
```

bkg_sta_bal For the weak 4D-var, a loop is neccesary, exactly as for bkg_bal_init, since the `_bkg` arrays are required.

tam_trj_init This routine is in `NEMO/OPA_SRC/TAM/tamtrj.F90`. It only reads the `namtam` block of the nemotam namelist.

dia_mat_bge This routine makes part of the `diamat` module. The routines of the `diamat` module enable to calculate the normalization coefficients, which must be applied after the calculation of the univariate correlations with the implicit or explicit diffusion operator.

The `dia_mat_bge` reads the correlation coefficient into the `background.normalization.nc` file:

```
       cl_nor = 'background.normalization.nc'
       CALL nor_rea_bge( cl_nor, istr, zlamcort, zlamcors, zlamcoru, &
          &              zlamcorv, zlamcore )
```

Then it runs the `ran_dia_bge` subroutine that is part of the `ran_diamat` module. This step enables to aplly a randomize method to calculate the normalization This module calls the standard deviation variables of the `bkg_err` module: `sd_tana_bkg`, `sd_sana_bkg`, etc.

3.4.3 CONGRAD inner loop driver

Quadratic cost function and its gradient

With congrad driver, the incremental 4D-Var equations are given with respect to the increment in the control vector space δu_0:

$$\begin{aligned} J^g &= \tfrac{1}{2}||s + \delta u_0 - u_0^b||^2 \\ &+ \tfrac{1}{2}\sum_{k=0}^{n-1} \left(\mathbf{H}_k\, \mathbf{M}_{k,0}\, B^{1/2}\delta u_0 - d_k\right)^T R_k{}^{-1} \left(\mathbf{H}_k\, \mathbf{M}_{k,0}\, B^{1/2}\delta u_0 - d_k\right) \end{aligned} \tag{3.5}$$

where:

- s is the reference state x_0^r weighted by $B^{-1/2}$, i.e., the sum of the background state and the current cumulated increment:

$$s = B^{-1/2}\left(x_0^b + \sum_{q=1}^{g-1} \delta x_0^q\right)$$

- δu_0 is the searched increment weighted by $B^{-1/2}$, i.e., the argument that will minimizes J^g,
- u_0^b is the background state vector weighted by $B^{-1/2}$.

$$\nabla J^g = \left(s + \delta u_0 - u_0^b\right) + \sum_{k=0}^{n-1} B^{T/2}\mathbf{M}_{k,0}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1} \left(\mathbf{H}_k\, \mathbf{M}_{k,0} B^{1/2}\, \delta u_0 - d_k\right) \tag{3.6}$$

The congrad inner driver reorganizes the quadratic gradient equation (3.6) with matrix $\mathbf{A}$ and vector $\mathbf{b}$, which are indpendant of the increment δu_0, and constant quantities on a given outer loop:

$$\begin{aligned} \nabla J^g &= \overbrace{\left(s - u_0^b\right) - \sum_{k=0}^{n-1} B^{T/2}\mathbf{M}_{k,0}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1} d_k}^{-\mathbf{b}} \\ &+ \underbrace{\left(\mathbf{I} + \sum_{k=0}^{n-1} B^{T/2}\mathbf{M}_{k,0}{}^T\, \mathbf{H}_k{}^T\, R_k{}^{-1}\mathbf{H}_k\, \mathbf{M}_{k,0} B^{1/2}\right)}_{\mathbf{A}=\textbf{Hessian matrix}} \delta u_0 \\ &= \mathbf{A}\,\delta u_0 - \mathbf{b} \end{aligned} \tag{3.7}$$

The CONGRAD inner driver also takes into account that before entering any inner loops, whatever is g the outer loop iteration, the current increment δu_0 is null.

This implies that:

- the following value for the cost function observation term:

$$J^{o,(g),\text{bef. inner loops}}\left(\delta u_0 = 0\right) = \frac{1}{2}\sum_{k=0}^{n-1} d_k^T \, {R_k}^{-1} \, d_k$$

- and an initial gradient which is equal to the $\mathbf{b}$ vector:

$$\begin{array}{ll} \nabla J^{(g)\text{bef. inner loops}} & = \left(s - u_0^b\right) - \sum_{k=0}^{n-1} B^{T/2} {\mathbf{M}_{k,0}}^T \, {\mathbf{H}_k}^T \, {R_k}^{-1} d_k \\ & = -\mathbf{b} \end{array} \tag{3.8}$$

Taking into account initial gradient formulae (3.8), the linear equation to solve in the least square sens [2] can be rewritten:

$$\mathbf{A}\,\delta u_0 = \nabla J^g - \nabla J^{g\text{before inner loops}} \tag{3.9}$$

Moreover, it can be noted that before any inner loop iteration (e.g., $\delta u_0 = 0$):

- for the first outer loop iteration $g = 1$, we have a null cumulated increment $s - u_0^b$, so that

$$J^{b,g=1}\left(\delta u_0 = 0\right) = 0$$

- while for subsequent outer loops, we simply have

$$J^{b,g>1}\left(\delta u_0 = 0\right) = ||s - u_0^b||^2$$

CONGRAD inner driver steps

1. It allocates the `ctlvec` variables with especially:

   - `z_vtcl%pdata`:
     to store the current outer loop searched increment $\delta u_0^{(g)}$ (i.e., the control vector).
   - `z_btcl%pdata`:
     to store the opposite of the cummulated increment (null if $g = 1$):

$$-\left(s - u_0^b\right) = -u_0^b - \sum_{q=1}^{g-1} \delta u_0^{(g-1)} + u_0^b$$

     which is the current increment resulting from the previous cycle of analysis (one outer loop + inner loops minimization process).

[2] The squared Hessian matrix is not full rank.

- `z_gtcli`:
  to store the initial gradient.
- `z_gtcl`:
  to store the Hessian increment product when $g > 1$, or the "relative gradient".
- `z_gctlf`:
  to store the final gradient
- It allocates the `congrad_wrkspc` variable which a structured type with `ctlvect` variables.

2. It loads `congrad_wrkspc` variable if $g > 1$.
   (called subroutine is `diaprc_restart`).

3. It loads the cumulated increment in `z_btcl%pdata` in the case it is not null ($g > 1$).
   (called subroutine is `diactl_restart`, see **??**).

4. It computes the initial gradient and cost function terms:

   - It calculates $J^{b,\text{bef. inner loops}}$:
     - which is either the norm of `z_btcl%pdata` if $g > 1$,
     - or the norm of `z_vtcl%pdata` if $g = 1$ (unuseful instruction: null value!).
   - It loads the innovation from the `datqc` to the `z_y ctlvec` variable.
   - It calculates the initial gradient with equation (3.8) through a call to the simulator routine `simvar_y2x`.
     `z_gtcli` stores the result. If it is not the first outer loop, the result is completed further with the cumulated increment.

5. It deallocates `z_y`.

6. It resets to zero the variables that stores the model counter part of the linear trajectory of the increment $\mathbf{H}_k\mathbf{M}_k\delta x_0$.
   which is to say the `vext(:,mp3dtan)` field of the `datqc(jset)%var(ivar)` (see appendix B.1.1).

7. It calculates $J^{o,\text{bef. inner loops}}$ through a call to `obscost`.
   This routine calculates the observation forcing: the difference between `vext(:,mp3dtan)` and `vext(:,mp3dinn)`, and loads the result into `vext(:,mp3dwdt)`
   CAUTION: in `obs_cost`, local `ctlvec` variables `z_y` and `z_ry` are allocated to store the observation forcing weighted or not by R^{-1}. No call to the `dot_product` subroutine of the `control_vector` module is applied between the local `ctlvec` variables. The dot product is performed by applying `mpp_sum_inter` with variable `vext(:,mp3dwdt)` as argument.

8. The initial gradient is updated with the cumulated increment if $g > 1$,
   and the initial background term of the cost function is calculated (again, why?).

9. The total initial cost function is stored in `z_cost`.

10. Diagnostics are applied through the call to `diaopt`.

11. It saves the initial gradient and its norm, and performs the gradient test, if `ln_tst_grad` is true (call to `tstgrd`).

12. The initial gradient is copied into the congrad variable `z_gtcl%pdata`, which will further store the Hessian-increment product, while `z_gtcli%pdata` will keep storing the initial gradient, see equation (3.9).

13. The Lanczos loop starts:

    - it launches `congrad_main` with especially arguments:

`z_vctl`	Increment
`z_gctl`	Initial gradient, Hessian-increment product
`z_cost`	Cost function

      (3.10)

    - at each iteration, the gradient is calculated again by the mean of the call to `simvar_x2x` (see equation (3.6)):

      $$\left(\sum_{k=0}^{n-1} B^{T/2} {\mathbf{M}_{k,0}}^T \, {\mathbf{H}_k}^T \, {R_k}^{-1} \mathbf{H}_k \, \mathbf{M}_{k,0} B^{1/2} \right) \delta u_0 \tag{3.11}$$

      The corresponding instruction is:

      ```
      CALL simvar_x2x( congrad_wrkspc%p_w, z_gctl, 'B^T/2 H^T R^-1 H B^1/2' )
      ```

      where `congrad_wrkspc%p_w` stores the current increment.
      Check that the `obscost` routine is called in the simulator.

14. In case the end of the congrad minimizer signifies the end of the Lanczos loop through the flag `i_flag_rc`, then:

    - The final gradient calculation is prepared through a last call to the simulator `simvar_x2x` with input argument `z_vctl`, and output `z_gctlf`.
    - $J^{o,\text{final}}$ is calculated through a call to `obscost`.
    - The final ∇J^{f} is computed as the sum of:

∇J^i	`z_gctlf`
δu_0	`z_vctl`
∇J^f	`z_gctli`

      (3.12)

    - $J^{b,\text{final}}$ is calculated with still two cases depending on $g > 1$,

- J^{final} is calculated also.
- Optimization diagnostics are performed, as well as gradient tests.
- The cumulated increment is updated if $g > 1$ (remember that `z_btcl` stores the opposite of the cummulated increment):

```
z_bctl%pdata(:) = z_bctl%pdata(:) - z_vctl%pdata(:)
```

- The background cost function is updated (still two cases according to g).
- $\delta x_0^{(g)}$, the current increment in the state vector space is computed with a call to `simvat_x2x` that only applies $B^{1/2}$ on $\delta u_0^{(g)}$.
  The result is stored in `z_dxmod`.
  See instruction:

```
CALL simvar_x2x( z_vctl, z_dxmod, 'B^1/2' )
```

- The increment arrays `tn_inc`, `sn_inc`,... are updated with the current increment through a call to the `updinc` subroutine.

15. Outside of the loop:
    - statistics on observation are made with `obs_wri_ana`,
    - the increment file is updated through the call to `rstinc` (see inc.F90 source),
    - the `diactl_restart` routine updates the cumulated increments, and `diaprc_restart` updates the congrad workspace (see **??**),
    - variables of type control vector are deallocated.

3.4.4 CGMOD inner loop driver

Quadratic cost function and its gradient

With the cgmod driver, the incremental 4D-Var equations are written with respect to δx_0, the increment in the space of state vectors:

$$\begin{aligned} J^g \quad &= \tfrac{1}{2} \left(\delta x_0 + \textstyle\sum_{q=1}^{g-1} \delta x_0^{(q)} \right)^T B^{-1} \left(\delta x_0 + \textstyle\sum_{q=1}^{g-1} \delta x_0^{(q)} \right) \\ &+ \tfrac{1}{2} \textstyle\sum_{k=0}^{n-1} \left(\mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0 - d_k \right)^T {R_k}^{-1} \left(\mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0 - d_k \right) \end{aligned} \tag{3.13}$$

where:

- δx_0 is the searched increment for the g^{th} outer loop made of the unbalanced increment of the oceanographic fields,
- $\sum_{q=1}^{g-1} \delta x_0^q$ is the cumulated increment calculated during the prvious outer loops.

The cost function gradient can be written as follows:

$$\nabla J^g = B^{-1}\left(\delta x_0 + \sum_{q=1}^{g-1}\delta x_0^q\right) + \sum_{k=0}^{n-1}\mathbf{M}_{k,0}{}^T\,\mathbf{H}_k{}^T\,R_k{}^{-1}\left(\mathbf{H}_k\,\mathbf{M}_{k,0}\,\delta x_0 - d_k\right) \tag{3.14}$$

The cgmod inner driver reorganizes the gradient equation (3.14) with the matrix $\mathbf{A}$ and vector $\mathbf{b}$, which are indpendant of the increment δx_0, and therefore constant quantities on a given outer loop:

$$\begin{aligned}\nabla J^g &= \overbrace{B^{-1}\sum_{q=1}^{g-1}\delta x_0^q - \sum_{k=0}^{n-1}\mathbf{M}_{k,0}{}^T\,\mathbf{H}_k{}^T\,R_k{}^{-1}d_k}^{-\mathbf{b}} \\ &+ \underbrace{\left(B^{-1} + \sum_{k=0}^{n-1}\mathbf{M}_{k,0}{}^T\,\mathbf{H}_k{}^T\,R_k{}^{-1}\mathbf{H}_k\,\mathbf{M}_{k,0}\right)}_{\mathbf{A}=\textbf{Hessian matrix}}\delta x_0\end{aligned} \tag{3.15}$$

where $\mathbf{A}$ is the Hessian matrix.

At first outer loop:

- the cummulated increment is a null quantity:

$$\sum_{q=1}^{g-1}\delta x_0^q = 0$$

- the initial value of the gradient before entering the inner loop $\nabla J^{g=0,\text{bef. inner loops}}$ verifies:

$$\nabla J^{g=0,\text{bef. inner loops}} = -\sum_{k=0}^{n-1}\mathbf{M}_{k,0}{}^T\,\mathbf{H}_k{}^T\,R_k{}^{-1}d_k \tag{3.16}$$

For subsequent outer loops, the initial value of the gradient before entering the inner loop $\nabla J^{g,\text{bef. inner loops}}$ verifies:

$$\nabla J^{g,\text{bef. inner loops}} = B^{-1}\sum_{q=1}^{g-1}\delta x_0^q - \sum_{k=0}^{n-1}\mathbf{M}_{k,0}{}^T\,\mathbf{H}_k{}^T\,R_k{}^{-1}d_k = b \tag{3.17}$$

Equation 3.15 can therefore be written:

$$\begin{aligned}\nabla J^g &= \mathbf{A}\,\delta x_0 - \mathbf{b} \\ \nabla J^g - \nabla J^{g,\text{bef. inner loops}} &= \mathbf{A}\,\delta x_0\end{aligned} \tag{3.18}$$

The minimum gradient is the solution of the linear system:

$$\mathbf{A}\,\delta x_0 = \mathbf{b} \tag{3.19}$$

One can note that equation (3.15) may be transformed with a $\mathbf{B}^{-1}$ factorization, which leads to the $\mathbf{A}$ and $\mathbf{b}$ expressions:

$$\begin{array}{ll} \mathbf{b} & = \mathbf{B}^{-1}\left(\sum_{q=1}^{g-1}\delta x_0^q - \sum_{k=0}^{n-1}\mathbf{B}\,\mathbf{M}_{k,0}{}^T\,\mathbf{H}_k{}^T\,R_k{}^{-1}d_k\right) & = \nabla J^{g,\text{bef. inner loops}} \\ \mathbf{A} & = \mathbf{B}^{-1}\left(\mathbf{I} + \sum_{k=0}^{n-1}\mathbf{B}\mathbf{M}_{k,0}{}^T\,\mathbf{H}_k{}^T\,R_k{}^{-1}\mathbf{H}_k\,\mathbf{M}_{k,0}\right)\,\delta x_0 & \end{array} \tag{3.20}$$

The minimum gradient is also the solution of the linear system:

$$\mathbf{B}\,\mathbf{A}\,\delta x_0 = \mathbf{B}\,\mathbf{b} \tag{3.21}$$

This enables to rewrite the gradient equation (3.15) for any outer loop such as the Hessian increment product equal to the following gradient difference:

$$\mathbf{B}\,\mathbf{A}\,\delta x_0 = \mathbf{B}\,\nabla J^g - \mathbf{B}\,\nabla J^{g,\text{bef. inner loops}} \tag{3.22}$$

CGMOD important `ctlvec` variable

Quantity	`ctlvec` variable	**notation**
Cumulated increment	`z_btcl`	$\sum_{q=1}^{g-1}\delta x_0^q$
Weighted cumulated increment	`z_wbtcl`	$B^{-1}\sum_{q=1}^{g-1}\delta x_0^q$
Increment	`z_vtcl`	δx_0
Weighted increment	`z_wvtcl`	$B^{-1}\,\delta x_0$

It is important to note that B^{-1} is never required. Second comment, δx_0 is the unbalanced increment. When refering to B, we mean here B_u, the diagonal block matrix filled with univariate backround error covariance matrices related to the controled oceanographic fields.

CGMOD inner driver steps

1. It allocates the `ctlvec` variables with especially:

   - `z_vtcl%pdata`:
     to store the current outer loop searched increment δx_0 (null when entering the driver).
   - `z_btcl%pdata`:
     to store the opposite of the cummulated increment (null quantity for first outer loop $g = 1$).
   - `z_gtcli` and `z_wgtcli`:
     to store the initial gradient (weighted or not) before any inner loops.

- `z_gtcl`:
  to store the Hessian increment product when $g > 1$, or the "relative gradient".
- `z_gctlf`:
  to store the final gradient
- It allocates the cgmod workspace variables.

2. It loads the cumulated and weighted cumulated increment in `z_btcl` and `z_wbtcl` if not zero ($g > 1$).
   (called subroutine is `diactl_restart`, see **??**).

3. It computes the initial gradient and cost function terms:

   - It calculates $J^{b,\text{bef. inner loops}}$ which is:
     - either the dot product between `z_btcl%pdata` and `z_wbtcl%pdata` if $g > 1$,
     - or the dot product between `z_vtcl%pdata` and `z_wvtcl%pdata` if $g = 1$ (unuseful instruction: null value!).
   - It loads the innovation d_k from the `datqc` to the `z_y ctlvec` variable.
   - It calculates one term of the initial gradient by applying equation (3.17) through a call to the simulator routine `simvar_y2x`:

$$-\sum_{q=1}^{n-1} \mathbf{M}_k^T \, \mathbf{H}_k^T \, d_k \tag{3.23}$$

     The result is stored in `z_gtcli`. If it is the first outer loop, there result is not modified further with the cumulated increment because it is a null quantity.

4. `z_y` is deallocated.

5. Variable `vext(:,mp3dtan)` meant to store the model counter part of the linear trajectory of the increment $\mathbf{H}_k \mathbf{M}_k \delta x_0$ are reset to zero.
   See appendix B.1.1 for more information on field `vext(:,mp3dtan)` of the `datqc(jset)%var(ivar)` variable.

6. It calculates $J^{o,\text{bef. inner loops}}$ through a call to `obscost`.
   This routine calculates the observation forcing: the difference between `vext(:,mp3dtan)` and `vext(:,mp3dinn)`, and loads the result into `vext(:,mp3dwdt)`
   CAUTION: in `obs_cost`, local `ctlvec` variables `z_y` and `z_ry` are allocated to store the observation forcing weighted or not by R^{-1}. No call to the `dot_product` subroutine of the `control_vector` module is applied between the local `ctlvec` variables. The dot product is performed by applying `mpp_sum_inter` with variable `vext(:,mp3dwdt)` as argument.

7. A call to `simvar_x2x` enables to calculate `z_wgctli` from `z_gctli` by applying the $\mathbf{B}$ covariance matrix.

8. Both the weigthed and not weighted initial gradient variables `z_gctli` and `z_wgctli` are updated by adding the cumulated increment `z_bctl` and `z_wbctl` if $g > 1$.

```
z_gctli%pdata(:)  = z_gctli%pdata(:)  - wbkgsc * z_wbctl%pdata(:)
z_wgctli%pdata(:) = z_wgctli%pdata(:) - wbkgsc * z_bctl%pdata(:)
```

   Pay attention to the application of the weighted (B^{-1}) cumulated increment in line 1, which is added to the unweighted initial gradient and provides the $\mathbf{b}$ vector. The second line, provides $\mathbf{B}\,\mathbf{b}$.

9. The initial background term of the cost function is calculated again (see `z_costb`). Why?

10. The total initial cost function is stored in `z_cost`.

11. Diagnostics are applied through the call to `diaopt`.

12. The initial gradient and its norm are saved, and the gradient test is performed, if `ln_tst_grad` is true (call to `tstgrd`).

13. The CG loop starts:

- it launches `cgmod_main` with especially arguments:

```
z_gctli
z_wgctli
z_vctl
z_cgwork
```
(3.24)

  where `z_cgwork` is a target for the following `ctlvec`:

```
z_gctl  => z_cgwork%z_ctvecs(1)   ! grad_x J
z_wgctl => z_cgwork%z_ctvecs(2)   ! B grad_x J
z_wrk1  => z_cgwork%z_ctvecs(3)   ! Workspace
z_wrk2  => z_cgwork%z_ctvecs(4)   ! Workspace
z_gctlf => z_cgwork%z_ctvecs(5)   ! Workspace
z_wvctl => z_cgwork%z_ctvecs(6)   ! B^-1 dx
```

- at each iteration:
  - the gradient can be calculated through the call to `simvar_x2x`:
    ```
    CALL simvar_x2x( z_cgarg, z_cgres, 'H^T R^-1 H' )
    ```
  - or if northo = 2, $\mathbf{B}$ is applied to ∇J:
    ```
    CALL simvar_x2x( z_cgarg, z_cgres, 'B' )
    ```

14. When the cgmod minimizer launch the signal to end the inner loop then:

- the final gradient calculation is prepared through a call to the simulator `simvar_x2x` with input argument `z_vctl`, and output `z_gctlf`. It calculates the following term of equation (3.20):

$$\texttt{z_gctlf} = \sum_{k=0}^{n-1} \mathbf{M}_{k,0}^T \, \mathbf{H}_k^T \, R_k^{-1} \mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0 \tag{3.25}$$

- $J^{o,\text{final}}$ is calculated through a call to `obscost`, and stored in `fcosto`.
- It calculates `z_wgctlf` through a call to:

```
CALL simvar_x2x( z_gctlf, z_wgctlf, 'B' )
```

- The final gradient weighted or not is computed:
  - The weighted gradient `z_wctlf` is the sum of:

$$\begin{array}{ll} \left(\mathbf{B} \sum_{k=0}^{n-1} \mathbf{M}_{k,0}^T \, \mathbf{H}_k^T \, R_k^{-1} \, \mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0\right) & \texttt{z_wgctlf} \\ \mathbf{B}^{-1} \, \delta x_0 & \texttt{z_wvctl} \\ \mathbf{B} \, \nabla J^i & \texttt{z_wgctli} \end{array} \tag{3.26}$$

  - The unweighted gradient `z_ctlf` is the sum of:

$$\begin{array}{ll} \sum_{k=0}^{n-1} \mathbf{M}_{k,0}^T \, \mathbf{H}_k^T \, R_k^{-1} \mathbf{H}_k \, \mathbf{M}_{k,0} \, \delta x_0 & \texttt{z_wgctlf} \\ \delta x_0 & \texttt{z_vctl} \\ \nabla J^i & \texttt{z_wgctli} \end{array} \tag{3.27}$$

- $J^{b,\text{final}}$ is calculated (two cases depending on $g > 1$), and stored in `fcostb`.
- J^{final} is calculated afterward, and stored in `z_costf`.
- Optimization diagnostics are performed, as well as the gradient test.
- `z_dxmod` is allocated (it is a `ctlvec` variable to store the current increment).
- The cumulated increment (weigthed or not) is updated if $g > 1$ (remember that `z_btcl` stores the opposite of the cummulated increment):

```
z_bctl%pdata(:) = z_bctl%pdata(:) - z_vctl%pdata(:)
z_wbctl%pdata(:) = z_wbctl%pdata(:) - z_wvctl%pdata(:)
```

  Why?: to REcalculate the backgound cost function? Else, at the stage, `z_bctl` will not be used any more. The storage of the new cumulated increment will be done through the call to `t_inc` arrays to which will be added the current increment after balance (see `updinc` subroutine)
- The background cost function is REcalculated (still two cases according to g). Why?. Caution to condition with `noutit` and `noutmax`.

- `z_dxmod%pdata` loads the $\delta x_0^{(g)}$ values.
- A call to the `updinc` subroutine with `z_dxmod` as argument, enables to balance the increment fileds (i.e., temperature, salinity,...), and to further update the cumulated increment fields `tn_inc`, `sn_inc`,...etc.

```
CALL updinc( z_dxmod )
```

15. Outside of the loop:

- statistics on observation are made with `obs_wri_ana`,
- the increment file is updated through the call to `rstinc` (see inc.F90 source),
- through the `diactl_restart` routine the cumulated increment files `increments#` are updated,
- variables of `ctlvec` type are deallocated.

3.4.5 What to change in CONGRAD and CGMOD inner drivers

3.4.6 The simulation routines `simvar_x2x` and `simvar_y2x`

Inner driver calls to `simvar_x2x`

In the congrad inner driver, at each inner loop iteration, the simvar_x2x routine is called:

```
CALL simvar_x2x( congrad_wrkspc%p_w, z_gctl, 'B^T/2 H^T R^-1 H B^1/2' )
```

The complete routine enables to apply the Hessian operator $(\mathbf{I} + \mathbf{B}^{T/2}\,\mathbf{H}^T\,\mathbf{R}^{-1}\,\mathbf{H}\,\mathbf{B}^{1/2})$ to the control vector increment δu. This operation provides in particular, the difference between the current and initial iteration gradient (see appendix 3.4.3).

We propose hereafter a step by step analysis of the `simvar_x2x` subroutine in order to identify where and how applying a loop on the assimilation subwindows index `isw`, and where using the reduced `ctlvec` variable.

Apply $\mathbf{B_u}^{1/2}$ the unbalanced error covariance matrix to the increment

```
CALL bge_covsqr( p_x2 )
```

The call to subroutine `bge_covsqr` enables to apply the univariate correlation operator to δu:

$$\delta x_0 = B_u^{1/2}\,\delta u_0$$

The `ctlvec` type variable `px2` is the input and output argument. It stores the current control vector increment δu_0.

To deal with weak 4D-var specificities, we propose here to:

- loop on the subwindow index `isw`,
- extract a reduced control vector (attached to a single subtrajectory) from the whole control vector
  let us say `px2_sw` extracted from `px2`,
- call `bge_covsqr` with the reduced control vector as argument `px2_sw`,
- update the corresponding section of the whole control vector `px2` with the modified reduced control vector `px2_sw`.

Apply the linear tangent model operator H M to δx

The tangent linear model propagates the increment vector across each observation time steps. Then the tangent linear observation operator (the operator itself if it is linear) transforms δx_k into a model counterpart increment:

$$\mathbf{H}_k \, \delta x_k = \mathbf{H}_k \, \mathbf{M}_{k-1,k} \, \delta x_{k-1}$$

The model operators are linearized around the successive trajectory states (e.g., x_{k-1}). To propagate the increment time step by time step:

- the closest trajectory states must therefore be loaded from the `tam_trajectories.nc` files [3].
- x_{k-1} state being loaded, a call to the numeric tangent model enables to perform steps of integration:
  $$\mathbf{M}_{k-1,k}(x_{k-1})$$
- and finally, a call to the linear tangent observation operator $\mathbf{H}_k(x_k)$ enables to calculate the model counterpart to observation.

This process, will enable to finally construct at each observation time, the model counterpart of the linear trajectory increment:

$$\mathbf{H}_k \mathbf{M}_{k-1,k} \, \delta x_{k-1} - d_k$$

Let us describe the instructions of the code to get the linear trajectory of the increment.

Variable to store the tangent linear model counterpart: reset to zero

[3] The trajectory is recorded at frequency `nittfrqtrj` during the nemo direct run. Time interpolation may be necessary to estimate the model counterpart at the right observation time.

```
CALL obs_tan_init( ln_t3d, ln_s3d, ln_sla, ln_sst, ln_lag )
```

`obs_tan_init` operates a reset to zero of the field `%var()%vext(:,mp3dtan)` of the `*datqc` variable.
This field will enable to store the model counter part of the linear tangent trajectory of the increment, on the whole assimilation window, at each observation point:

$$\mathbf{H}_k \mathbf{M}_k \, \delta x_{k-1}$$

Variables dedicated to the storage of data belonging to observation space are described in appendix (B.1.1).

For the weak 4D-Var approach, no nsw loop is expected here. The call to obs_tan_init should be moved before the call to `bge_covsqr` outside of the begining of the `isw` loop.

Variables storing the linear tangent fields corresponding to the increment: allocation and reset

```
CALL oce_tam_init(1)
```

The subroutine (/NEMOTAM/OPATAM_SRC/oce_tam.F90) is called with argument set to 1. In this case, it allocates and resets to zero the field variables dedicated to the storage of the successive states of the linear trajectory of the increment δx_k. It is the 3D and 2D "fields":

- tracer fields `tn_tl`, `sn_tl`,
- velocity fields `un_tl`, `vn_tl`,
- and sea surface height `sshn_tl`

For the weak 4D-Var, this reset operation must be included in the main nsw loop of the simvar_x2x routine.

Apply $\mathbf{K}^{1/2}$ the balance operator on the state vector increment

The subroutine `balopttan` operates a multi variate balance on the increment fields (see appendix B.2). The `balopttan` input argument is a `ctlvec` variable that stores the current unbalanced control vector increment δx. Output arguments are the state vector field variables `tn_tl`, `sn_tl`, `un_tl`, `vn_tl` and `sshn_tl`. They are either 3D or 2D arrays.

For the weak 4D-Var, the balance instruction must be included in the main `isw` loop to calculate balanced field increments successively for each increment section.

We propose that the input argument remains a `ctlvec` variable but of reduced size and defined on a single assimilation subwindow. Then, the process would be:

- extract a single subwindow `ctlvec` variable,
- apply `balopttan` to it,
- update the wright section of the whole `ctlvec` state vector with the balanced `ctlvec` variable.

Projection of the tangent state vector fields `tn_tl`, `sn_tl`, `un_tl`, etc.
Projection is operated with the call to `proinc`, which must be in the loop.

Reading the restart of the trajectory

```
CALL trj_rea(nit000 -1,1)
```

The dynamic and tracer fields `tn`, `sn`, `un`, `vn`, and `sshn` are needed to compute the linear tangent and adjoint operator, since the linearization of the model is made around the restart state vector at first. In the strong constraint 4D-Var scheme, we have a single piece of trajectory on the full assimilation window, a single initial state to control. A unique restart defined at time step `nit000-1` is read with the call to `trj_rea`.

In the weak constraint 4D-Var scheme, the trajectory is split in `isw` pieces, each of them having its own initial state that must be controled. We have a therefore `nsw` restart to read.
The call to `trj_rea` must be included in the main `isw` subwindow loop [4].

Initialization of the observation counter for the tangent linear model equivalent vector

```
CALL dia_obs_tan_init
```

Ths instruction performs the initialization of the observation counter for the computation of the tangent linear model counterpart. Appendix B.1.1 provides more information on the structured types applied to store data of the observation space.

Let us give an example, with profile data type. This call to `dia_obs_tan_init` leads to reset to zero, the profile observation counter, which is stored in parameter `prodatqc(jprofset)%nprofup` (for all the `nprofsets` profile sets). The reset is effective for the whole assimilation window.

[4] `'tam_trajectory'_it` with `it = kstp - nit000 + 1`.

This instruction does not need to be in the loop, and should be moved before the begining of the loop.

Linear tangent version of the observation operator* H*: first call

```
CALL dia_obs_tan(nit000-1)
```

Taking profile data as example, subroutine `dia_obs_tan` calls the subroutine `obs_pro_opt_tan` (source `obs_oper_tam.F90`). Input arguments of the `obs_pro_opt_tan` routine are the increment fields `tn_tl`, etc. The output argument is the `*datqc` structured type storing observation data and meta data.

This routine performs the interpolation from the model grid to the observation point of the increment fields:

$$\mathbf{H}\,\delta x$$

The argument of `dia_obs_tan` is here `nit000-1`:
Is-it an initialization of the **H** coefficient at the restart time step `nit000-1`?
The result is stored in vector `prodatqc%var()%vext(...,mp3tan)`.

In the weak 4D-Var, this initialization have to be applied at the `nsw` subtrajectory restart states. The restart time to apply being `nitsw-1`.

This instruction must be in the subwindow loop, paying attention to the right time step to apply

Initial state settings for the tangent model

```
CALL istate_init_tan
```

The subroutine `istate_init_tan` (module `istate_tam`) involves the use of the variables storing the increment fields (i.e., `tn_tl`, `sn_tl`, `un_tl`, etc.). It includes hidden reference to time step `nit000`.

Weak 4D-Var approache
The above comment shows the necessity to set the variables `nit000` and `nitend` to the `isw` subwindow time step limits, at the begining of every `isw` loop, and to set them back to their native values at the end of the loop. We propose the following isntructions to do so:

- save temporary variables the `nit000` and `nitend` time steps before assimilation subwindow loops:

```
nit000_save = nit000
nitend_save = nitend
```

- Then, put the following instruction after the begining of the loop :

```
nit000 = nitsw(isw)
nitend = nitsw(isw+1)
```

  where `nitsw(isw)` and `nitsw(isw+1)` are the subwindow time step boundaries.

- and at the end of subwindow loop, `nit000` and `nitend` time steps must be set back to their saved values.

Loop on time steps to apply the linear tangent models $\mathbf{H}_k\mathbf{M}_{k-1,k}$ ***on*** δx_{k-1}

```
1   DO jstp= nit000,nitend
2       CALL step_tan(jstp)
3       CALL dia_obs_tan(jstp)
4   ENDDO !jstp loop
```

The linear tangent dynamic model is first applied (line 2) on increment fields `tn_tl`, etc..., which corresponds to equation:

$$\delta x_k = \mathbf{M}_{k-1,k}\delta x_{k-1}$$

Then the linear tangent version of the observation operator is applied (line 3):

$$\mathbf{H}_k\delta x_k$$

The resulting model counterpart of the linear trajectory of the increment is stored in the global variable `prodatqc%var()%vext(...,mp3tan)`. At the end of the loop, the model observation discrepancy at any observation time steps is available all across the assimilation window with the two following vectors:

- $\mathbf{H}\delta x_k$ stored in vector `prodatqc%var()%vext(...,mp3tan)`,
- and d_k stored in vector `prodatqc%var()%vext(...,mp3dinn)`.

For the weak 4D-Var implementation, the `jstp` loop must be inside the main `isw` subwindow loop. Caution: the `nit000` and `nitend` must be changed for the right time limits corresponding to the current assimilation subwindow.

Observation error covariance matrix applied

```
   IF ( ln_t3d ) THEN
      ivar = 1
      DO jset = 1, nprofsets
         CALL setval_ctlvec( prodatqc(jset)%nvprot(ivar), ntob_off(jset), &
            &                prodatqc(jset)%var(ivar)%vext(:,mp3dtan), z_y )
      END DO
   ENDIF
  (...)
  CALL oer_rpy(z_y, z_rpy,...)
```

The call to `oer_rpy` enables to calculate the weighted. The above instruction is prepared with the transfer of the whole `prodatqc%var()%vext(...,mp3tan)` vector in the `ctlvec` variable `z_y`. This is done through the call to subroutine `setval_ctl`. The weighing with the inverse of the observation error covariance matrix is then applied on `z_y`: The output argument is the `ctlvec` variable `z_rpy`. Both `z_y` and `z_rpy` must be of augmented size. The latter variable is further stored in the `prodatqc%var()%vext(...,mp3dwt)` with the call to subroutine `getval_ctl`.

4D-Var implementation
This whole step must be outside the assimilation subwindow loop, because the variable `prodatqc%var()%vext` are meant to store observation data on the whole assimilation window. The variable `z_rpy` stores therefore the whole weighted observation forcing.

Variables storing the adjoint variable: allocation and reset

```
CALL oce_tam_init(1)
```

If called with argument set to 1, this instruction allocates and resets to zero the field variables dedicated to the storage of the successive adjoint variable states. It is the 3D and 2D "fields":

- tracer fields `tn_ad`, `sn_ad`,
- velocity fields `un_ad`, `vn_ad`,
- and sea surface height `sshn_ad`

For the weak 4D-Var, this reset operation must be included in the main nsw loop of the simvar_x2x routine.

Allocation and initialization of the adjoint state vector field variables

```
CALL oce_tam_init(2)
```

The adjoint variable is allocated and set to zero with the call to `oce_tam_init` (argument set to 2). The adjoint variables are fields `tn_ad`, `sn_ad`, `un_ad`, `vn_ad` and `sshn_ad`.

4D-Var implementation
The reset operation must also be included in the `nsw` loop.

Initialization of the observation counter

```
CALL dia_obs_adj_init
```

Note: because of the adjoint equations are retrotrajectories, the data counter parameter `prodatqc(jprofset)%nprofup` applied for profile data, is here set to the total number of profile `prodatqc(jprofset)%nprof`. Then the count is applied back to zero.
For the weak 4D-Var, this reset operation must not be in the main nsw loop of the simvar_x2x routine.

Initialization of the retro trajectory reading

```
CALL trj_rea(nitend,-1)
```

It is used for linearization of the dynamical model. Allocation + reset? In the loop: yes.

Loop on time steps to perform a retro propagation of the vector storing the model observation discrepancy

The following routine calls the adjoint version of the observation operator (i.e., $\mathbf{H}^T$) at each time step:

```
DO jstp= nitend,nit000
    CALL dia_obs_adj(jstp)
```

Then it applies the adjoint of the dynamical model with the following instruction:

```
    CALL step_adj(jstp)
ENDDO !jstp loop
```

4D-Var implementation
Both calls must be in the assimilation subwindow loop, taking care (as already mentionned), to substitute to `nitend` and `nit000`, the local limits.
Note: In NEMOTAM/OPATAM_SRC/step_tam.F90 subroutine `step_adj` at `jstp` calls `trj_rea` with argument, the end of the previous step.

```
CALL trj_rea( ( kstp - 1 ), -1 )
```

Check and better understand the instruction

```
CALL istate_init adj_
```

Adjoint version of the observation operator applied at initial step

```
CALL dia_obs_adj(nit000-1)
```

4D-Var implementation
Must be in the subwindow loop.

`simvar_y2x`

The three `nemovar_inner_drivers` apply the `simvar_y2x` subroutine. For this preliminary study, we only focus on congrad and cgmod. For both drivers the call to `simvar_y2x` enables to calculate a term of the initial gradient $\nabla J^{g,\text{bef. inner loops}}$ before inner loop iterations start:

- congrad applies it to map the innovation vector into the control vector space (i.e., the δu_0 space):

```
CALL simvar_y2x( z_y, z_gctli, 'B^T/2 H^T R^-1' )
```

- cgmod applies it to map the innovation vector into the "state vector" space [5] (i.e., the δx_0 space):

```
CALL simvar_y2x( z_y, z_gctli, 'H^T R^-1' )
```

Let us analyse the instructions step by step in `simvar_y2x`.

Observation error covariance matrix applied on the simvar input argument `p_y`

```
CALL oer_rpy(p_y, z_y,-1)
```

This call enables to weight the `ctlvec` variable `p_y` with the inverse of the observation error covariance. The output `z_y ctlvec` is transferred further in the `%vext(:,mp3dwdt)` field (see variable `datqc`).

[5] The state vector might be designated as x_0 rather than δx_0.

```
IF ( ln_t3d ) THEN
   ivar = 1
   DO jset = 1, nprofsets
      CALL getval_ctlvec( prodatqc(jset)%nvprot(ivar), ntob_off(jset), &
         &                prodatqc(jset)%var(ivar)%vext(:,mp3dwdt), z_y )
   END DO
ENDIF
```

Weak 4D-Var:
These set of instructions must remain outside of the assimilation subwindow loop. The `z_y ctlvec` must be of augmented size.

Reset of the adjoint fields

```
CALL oce_tam_init( 2 )
```

The adjoint fields `tn_ad`, `sn_ad`, `un_ad`, `vn_ad` and `sshn_ad` are reset to zero.
4D-Var implementation

Initialization of the observation counter for the adjoint integration steps

```
CALL dia_obs_adj_init
```

Let us give an example of initialization of the observation counter for profile data. The data counter parameter `prodatqc(jprofset)%nprofup` is set to the total number of profile `prodatqc(jprofset)%nprof`. The count will be applied back to zero because of the adjoint follows retro-trajectories.
This instruction must not be in the subwindow loop because observation counters are global parameters for the whole assimilation window.

Initialization of the retro trajectory reading

```
CALL trj_rea(nitend,-1)
```

It is used for linearization of the dynamical model. Allocation + reset? In the loop: yes.

Loop on time steps to perform a retro propagation of the vector storing the model observation discrepancy

The following routine calls the adjoint version of the observation operator (i.e., $\mathbf{H}^T$) at each time step:

```
DO jstp= nitend,nit000
    CALL dia_obs_adj(jstp)
```

Then it applies the adjoint of the dynamical model with the following instruction:

```
    CALL step_adj(jstp)
ENDDO !jstp loop
```

4D-Var implementation
Both calls must be in the assimilation subwindow loop, taking care to substitute to `nitend` and `nit000` the local subwindow limits.

Check and better understand the instruction

```
CALL istate_init_adj
```

Adjoint version of the observation operator applied at initial step

```
    CALL dia_obs_adj(nit000-1)
```

4D-Var implementation

Projection of initial adjoint fields

```
    CALL proincadj( tn_ad, sn_ad, un_ad, vn_ad, sshn_ad )
```

The input and output arguments are the adjoint fields `tn_ad`, etc. The projection is described in subroutine `proincadj` as:

```
Project the gradient onto the dual of the subspace of
      !!              allowed control variables using the adjoint of the
      !!              sequence of projection operators.
```

4D-Var implementation
The projection must be in the subwindow loop, i.e., there is one initial adjoint set of fields per subwindow which provides a subwindow section of the gradient.

Balance operator applied to the adjoint fields

```
CALL baloptadj( p_x, tn_ad, sn_ad, un_ad, vn_ad, sshn_ad )
```

Congrad works with the normalized control vector δu, while cgmod works with δx_u which is the unbalanced increment. When entering the `simvar` subroutine, the balanced increment is unbalanced (input argument). When leaving the `simvar` routine the the gradient (or equivalently the adjoint),

Appendices

Appendix A

First Appendix: Weak 4D-Var implementation issues

A.1 Augmented size of vectors of the model space

A.1.1 Arrays of oceanographic field: dimension issue

The weak 4D-var approach involves an augmented state vector with a number of 3D, 2D fields multiplied by the number of assimilation subwindows `nsw`. The nemovar inner loop algorithm:

- starts with the reading of the cumulated increment and the background vectors, which are respectively stored in files:
  - `increments_`,
  - `assim_background_Jb`.
- ends with the update of the cumulated increment with the current increment, before writing it into the `increments_` file.

Currently, the arrays dedicated to store the fields of the control or state vectors are not built to support the supplementary dimension corresponding to subwindows (this dimension can be viewed as a pseudo time dimension). This is especially the case of the arrays dedicated to store increment and background oceanographic fields. They can be found in the code as:

- `_bkg`,
- `_inc`.

Before proposing a treatment, let us investigate how these arrays are applied in the code.

Background arrays

The _bkg arrays are t_bkg, s_bkg, u_bkg, v_bkg and ssh_bkg. In the inner loop, they enable to store the background vector components written in the background file assim_background_state_Jb.nc They are required during the minimization at each inner loop iteration in order to make the variable change from the control vector back to the state vector, through the call to the balopttan subroutine (bal_opt.F90) in the _simvar.

Increment arrays

Nemovar initialization Allocatable arrays of type t_inc, s_inc, u_inc, v_inc and ssh_inc are allocated in subroutine inc_init and enable to load the increment vector component of the previous outer loop. The inc_init routine is called in subroutine nemovar_init in (nemovar.F90). In the nemovar_init, the subroutine bkg_bal_init (bkg_bal.F90), which itself calls the bkg_sal_bal where the current now tracers are updated with the background and increment arrays.

None of the *_inc arrays are used again before the end of the inner loop minimization.

Nemovar process at the end of the minimization The subroutine updinc (upd_inc.F90) is called at the end of each three minimizer of the nemovar_inner_drivers. Its argument is the ctlvec varaible z_pdxmod which stores the analysed increment components in its field %pdata. This latter routine:

- makes the extraction of the %pdata vector components and fills the 3D and 2D arrays zt, zs, zu, zv and zssh through the call to balopttan, which operates the balance of the fields.
- updates the arrays t_inc, s_inc, u_inc, v_inc and ssh_inc with the cummulated increments components with instructions:

```
t_inc = t_inc + zt
s_inc = s_inc + zs
...
```

The _bkg type arrays for the background storage

The table A.1.1 sumarizes where the background arrays _bkg are called in the inner loop. They are required after each minimization iteration in order to perform the variable change from the control vector back to the state vector. This operation involves the application of the balance operator in the balopttan subroutine, The balance operator is a linearized operator around the reference state (see appendix A.1.1 that provides further informations).

Table A.1: The the _bkg arrays in the inner loop.

Arrays	subroutine	calling routine (source)	Purpose
all	bkg_init	nemovar_init (nemovar.F90)	arrays loaded (bkg_rea or bkg_ref)
tracer only	bkg_bal_sal	bkg_bal_init called in nemovar_init (nemovar.F90)	initialize balance operator $t_n = t_{bkg} + t_{inc}$ $s_n = s_{bkg} + s_{inc}$
all fields	balopttan (bal_opt.F90)	called in simvar_ (sim_var.F90)	

The _inc type arrays for the cumulated increment storage

Table A.1.1 sumarizes where the _inc arrays are called in the inner loop. Appendix A.1.1 provide further informations.

Table A.2: The the _inc arrays in the inner loop.

Fields	subroutine	calling routine (source)	Purpose
all	inc_init	nemovar_init (nemovar.F90)	arrays loaded from "increment_" file
tracer only	bkg_bal_sal	bkg_bal_init called in nemovar_init (nemovar.F90)	initialization of balance operator $t_n = t_{bkg} + t_{inc}$ $s_n = s_{bkg} + s_{inc}$
all	updinc	called in the 3 minimizers (nemovar_inner_drivers.F90)	balance and update of arrays
all	rstinc	called in the 3 minimizers (nemovar_inner_drivers.F90)	update of "increment_" file

Arrays of oceanographic fields in the inner loop: summary

One can note that the _inc arrays are loaded and updated at the end of the inner loop process, and finally written back in the _increment file, On the other hand the _bkg arrays are simply applied to update inner loop variables connected to the balance operator. Further, none of these arrays are applied to update the ctlvec variables dedicated to the background and cumulated increment.

Proposed strategy

To overcome the problem of the inappropriate size of arrays dedicated to store 1D, 2D and 3D fields in a weak 4D-Var implementation, one option is to add one dimension to take into

account the nsw initial states that must be controled.

The _bkg array case Weak 4D-Var implementation:

Because the _bkg arrays are required at each inner loop iteration for the linearization of the balance operator, it is mendatory to store them from the begining to the end of the minimization iterations, with the supplementary dimension.

The _inc array case Weak 4D-Var implementation:

The _inc arrays are only necessary at the begining and end of the nemovar inner loop. One can afford to allocate and load the _inc arrays with a supplementary dimension in the nemovar_init subroutine, then to deallocate them during the minimization loops, and to reload them at the end of the process.

A.1.2 Off-line communication files: the record dimension issue

The NEMO/IOM module holds the I/O subroutines in the NEMO/OPA_SRC/IOM directory. These I/O subroutines are organized into two generic interfaces in sources iom.F90 and iom_nf90.F90. The augmented state vector includes nsw sections corresponding to the nsw successive initial conditions to control. Elementary routines that read or write state or control vector however assume that no handling of time records has to be applied. We propose below an analysis of routines dedicated to read or write fields or parameters, and for which the time record issue can be encountered. We have only covered simulation case that applies the netcdf format (see option jpnf90).

A list of the I/O file and calling routines impacted with this issue are shown in table A.1.3 of this appendix.

Off-line communication file: reading interface

The two generic interfaces at hand are:

1. iom_get (iom.F90),
2. and iom_nf90_get (iom_nf90.F90).

The iom_get interface includes the following routines: iom_g0d, iom_g1d, iom_g2d, iom_g3d. These interface subroutines operate calls to:

- either iom_get_g0d (iom.F90)
- or iom_get_g123d (iom.F90)

All the `iom_get` interface routines (e.g., iom_get_g0d and iom_get_g123d) calls the second generic interface `iom_nf90_get`.

The second generic interface `iom_nf90_get` includes the routines:

- iom_nf90_g0d,
- iom_nf90_g123d.

Both `iom_get` and `iom_nf90_get` interfaces can recover several time records in I/O files

When the netcdf format is applied (`jpnf90` option), `iom_get` and `iom_nf90_get` interfaces can recover several time records in I/O files. This can be done with the optional argument `ktime`, as well as arguments `kstart` and `kcount`:

- `ktime` is simply the **time record index**,
- `kstart` and `kcount` are arrays of integers meant to store the start and range of indices in the space and time dimensions. These indices enable to target the right section of data to be recovered in a netcdf file. In the case of a 3D field, the number of dimension is 4 in order to include the time dimension. In the case of a scalar, `kstart` and `kcount` as only one dimension which is the time record index.

Some details of the process

The argument `ktime` can always be set when calling subroutines of interface `iom_get`:

- the argument `ktime` can be set when calling subroutine `iom_g0d` of interface `iom_get`:

```
SUBROUTINE iom_g0d( kiomid, cdvar, pvar, ktime)
(...)
   CASE (jpnf90);   CALL iom_nf90_get( kiomid, idvar, pvar, ktime=itime)
```

- the arguments `ktime`, `kstart`, and `kcount` can be set when calling subroutine `iom_123d` of interface `iom_get`:

```
SUBROUTINE iom_get_123d( kiomid, kdom  , cdvar ,   &
      &                  pv_r1d, pv_r2d, pv_r3d,   &
      &                  ktime , kstart, kcount,   &
      &                  knolink  )
   (...)
   INTEGER                 , INTENT(in), OPTIONAL :: ktime  ! record number
   INTEGER , DIMENSION(:) , INTENT(in), OPTIONAL :: kstart ! start position of the reading in each axis
   INTEGER , DIMENSION(:) , INTENT(in), OPTIONAL :: kcount ! number of points to be read in each axis
   (...)
 istart(idmspc+1) = itime
   (...)
      CASE (jpnf90) ; CALL iom_nf90_get( kiomid, idvar, inbdim, istart, icnt, &
         &                               ix1, ix2, iy1, iy2,                  &
         &                               pv_r1d, pv_r2d, pv_r3d )
```

As can be seen above, both subroutines calls interface iom_nf90_get and transmit the kstart, and kcount argument. The ktime, kstart and kcount arguments are recovered by the iom_nf90_get generic interface subroutines, with especially:

- iom_nf90_g123d where the kstart array has dimspc dimension plus one, with its last dimension set to ktime,
- and iom_nf90_g0d that launches NF90_GET_VAR with start argument equals to ktime,

To get selectively a given time record, the code analysis shows that the only important argument to pass is ktime. It would be different if we want to selectively get fields on reduced space domain.

Off-line communication file: writting interface

The two generic interface at hand are:

1. iom_rstput (iom.F90),
2. and iom_nf90_rstput (iom_nf90.F90).

The iom_rstput interface includes the following routtines: iom_rp1d, iom_rp2d, iom_rp3d, iom_rp0d. The latter subroutines all calls a single routine of the iom_nf90_rstput interface: iom_nf90_rp0123d.

The key writtingsubroutine iom_nf90_rp0123d is not written to write several time records. We explain our proposal for modifying the code.

A.1.3 Weak 4D-Var implementation: off-line communication files and time record issue

Reading: the iom_get interface

Weak 4D-Var implementation:

The code analysis shows that recovering the time dimension can be handled with the only time record index ktime.

Any time the iom_get interface is called in the inner loop, one must check if time records has to be recovered, and set the ktime argument accordingly.

Writing: the iom_rstput interface

Weak 4D-Var implementation:
The only change required in the inner loop routines that calls the iom_rstput interface, consists in paying attention to the two first arguments kt and kwrite that are oftenly set both to 1, and to replace them by the write time step value.

Writing: proposed implementation in the iom_rstput interface

The proposed implementation consists in adding the following instructions to handle time record in iom_nf90_rstput.

The iom_nf90_rstput interface handles the iom_file variable of type file_descriptor:

```
TYPE, PUBLIC ::   file_descriptor
  CHARACTER(LEN=240)                          ::   name     !: name of the file
  INTEGER                                     ::   nfid     !: identifier of the file (0 if closed)
  INTEGER                                     ::   iolib    !: library used to read the file (jpioipsl, jpnf90 or jprstdimg)
  INTEGER                                     ::   nvars    !: number of identified varibles in the file
  INTEGER                                     ::   iduld    !: id of the unlimited dimension
  INTEGER                                     ::   irec     !: writing record position
  CHARACTER(LEN=32)                           ::   uldname  !: name of the unlimited dimension
  CHARACTER(LEN=32), DIMENSION(jpmax_vars)    ::   cn_var   !: names of the variables
  INTEGER, DIMENSION(jpmax_vars)              ::   nvid     !: id of the variables
  INTEGER, DIMENSION(jpmax_vars)              ::   ndims    !: number of dimensions of the variables
  LOGICAL, DIMENSION(jpmax_vars)              ::   luld     !: variable using the unlimited dimension
  INTEGER, DIMENSION(jpmax_dims,jpmax_vars)   ::   dimsz    !: size of variables dimensions
  REAL(kind=wp), DIMENSION(jpmax_vars)        ::   scf      !: scale_factor of the variables
  REAL(kind=wp), DIMENSION(jpmax_vars)        ::   ofs      !: add_offset of the variables
END TYPE file_descriptor
```

One can note that the field iom_file(kiomid)%luld(idvar) when set to true, indicates that the variable with id idavar does have an unlimited dimension.

Before writting any variables, the file is created at the first call of the iom_nf90_open routine, and dimensions are defined with t being declared as unlimited dimension.

The algorithm of the iom_nf90_rstput subroutine is the following:

If kt and kwrite input arguments are equal:

- Define the variable dimensions if not already done:
  nav_lon, nav_lat, nav_lev, time_counter
  To test if the instruction is carried out:

```
IF( iom_file(kiomid)%nvars == 0 ) THEN
```

  The time_counter dimension variable id is 4.
  (Note that the %irec field is applied as a boolean to see if file is in define or write mode).

- Define the other variables if not already done:
  Four types of fields can be written in file: pv_rxd with x equal to 0,1,2 and 3, for 0,1,2 and 3D fields respectively.
  Except px_r0d, all fields depend on the time_counter id:

```
IF(     PRESENT(pv_r0d) ) THEN ; idims = 0
ELSEIF( PRESENT(pv_r1d) ) THEN ; idims = 2 ; idimid(1:idims) = (/    3,4/)
ELSEIF( PRESENT(pv_r2d) ) THEN ; idims = 3 ; idimid(1:idims) = (/1,2  ,4/)
ELSEIF( PRESENT(pv_r3d) ) THEN ; idims = 4 ; idimid(1:idims) = (/1,2,3,4/)
ENDIF
(...)
IF( .NOT. PRESENT(pv_r0d) ) THEN ; iom_file(kiomid)%luld(idvar) = .TRUE.
ELSE                             ; iom_file(kiomid)%luld(idvar) = .FALSE.
```

If `kt` and `kwrite` input arguments are equal:

- If variable dimension are defined **but not written**
  (applies the `iom_file(kiomid)%dimsz(1, 1)` as a boolean flag to check)
- For all variable, time variable dimension included,
  NO `start` or `count` arguments are applied in the netcdf subroutines:

```
CALL iom_nf90_check(NF90_INQ_VARID( if90id, 'time_counter', idmy ), clinfo)
CALL iom_nf90_check(NF90_PUT_VAR( if90id, idmy, kt               ), clinfo)
```

- Update of the length of the dimensions `x`, `y`, `z` and `t`, on which the dimensions variables depend (length stored in the `iom_file%dimsz(dims,vars)`):

```
iom_file(kiomid)%dimsz(1:2, 2) = iom_file(kiomid)%dimsz(1:2, 1)
CALL iom_nf90_check(NF90_INQUIRE_DIMENSION( if90id, 3, len = iom_file(kiomid)%dimsz(1,3) ), clinfo)
iom_file(kiomid)%dimsz(1  , 4) = 1   ! unlimited dimension
```

- At last, write the field data:

```
IF(     PRESENT(pv_r0d) ) THEN
  CALL iom_nf90_check(NF90_PUT_VAR( if90id, idvar, pv_r0d                ),clinfo)
ELSEIF( PRESENT(pv_r1d) ) THEN
  CALL iom_nf90_check(NF90_PUT_VAR( if90id, idvar, pv_r1d(              :)),clinfo)
ELSEIF( PRESENT(pv_r2d) ) THEN
  CALL iom_nf90_check(NF90_PUT_VAR( if90id, idvar, pv_r2d(ix1:ix2, iy1:iy2)),clinfo)
(...)
```

We propose to modify the `iom_nf90_rstput` the following way:

- introduce a new local variable to read the current time variable in the opened file.

```
INTEGER              :: kt_read             ! previous recorded ocean time-step
```

- `kt` will be further compared to kt_read, and applied to trigger the increment of the unlimited dimension index.

Further, add an alternate case to the first writing of the dimension variables:

- If variable have been already defined and written
  Read the last written `time_counter` value and store it in `kt_read`:

```
IF( iom_file(kiomid)%dimsz(1, 1) == 0 ) THEN
(...)
   CALL iom_nf90_check(NF90_INQUIRE_DIMENSION( if90id, 4, len = iom_file(kiomid)%dimsz(1,4) )       , clinfo)
   CALL iom_nf90_check(NF90_INQ_VARID( if90id, 'time_counter', idmy )                               , clinfo)
   CALL iom_nf90_check(NF90_GET_VAR( if90id, idmy, kt_read, start=(/iom_file(kiomid)%dimsz(1,4)/)  ), clinfo)
(...)
```

- Compare the current time step and `kt_read`, and if it is larger:

```
IF (kt > kt_read) THEN
   ! the length of the time record must be increased and updated
   iom_file(kiomid)%dimsz(1,4) = iom_file(kiomid)%dimsz(1,4) + 1
   CALL iom_nf90_check(NF90_PUT_VAR( if90id, idmy, kt, start=(/iom_file(kiomid)%dimsz(1,4)/)  ), clinfo)
```

which is to say:

- update the length of the `t` dimension on which `time_counter` dimension variable depends,
- write the current time in the `time_counter` variable of the file, using the netcdf `start` [1] argument.

At last, write the field data, using the netcdf `start` argument with the time record index updated with the current record number `iom_file(.)%dimsz(1,4)`:

```
IF(     PRESENT(pv_r0d) ) THEN
           CALL iom_nf90_check(NF90_PUT_VAR( if90id, idvar, pv_r0d                    ), clinfo)
        ELSEIF( PRESENT(pv_r1d) ) THEN
           ! weak 4D-Var: enable multiple records in the same file
           !CALL iom_nf90_check(NF90_PUT_VAR( if90id, idvar, pv_r1d(                 :) ), clinfo)
           CALL iom_nf90_check( NF90_PUT_VAR( if90id, idvar, pv_r1d(:)                  , start=(/1, iom_file(kiomid)%dimsz
        ELSEIF( PRESENT(pv_r2d) ) THEN
           ! weak 4D-Var: enable multiple records in the same file
           !CALL iom_nf90_check(NF90_PUT_VAR( if90id, idvar, pv_r2d(ix1:ix2, iy1:iy2   ) ), clinfo)
           CALL iom_nf90_check(NF90_PUT_VAR( if90id, idvar, pv_r2d(ix1:ix2, iy1:iy2)    , start=(/1, 1, iom_file(kiomid)%di
        ELSEIF( PRESENT(pv_r3d) ) THEN
           ! weak 4D-Var: enable multiple records in the same file
           !CALL iom_nf90_check(NF90_PUT_VAR( if90id, idvar, pv_r3d(ix1:ix2, iy1:iy2, :) ), clinfo)
           CALL iom_nf90_check(NF90_PUT_VAR( if90id, idvar, pv_r3d(ix1:ix2, iy1:iy2, :) , start=(/1, 1, 1, iom_file(kiomid)
        ENDIF
(...)
```

I/O routines: time arguments to modify for weak 4D-Var implementation

Table A.1.3 shows the list of inner loop routines that calls NEMO I/O interfaces `iom_get` and `iom_rstput`, and whose time arguments must be changed.

[1] The netcdf `start` array stores the starting indices in all the space and time dimensions (to set where to start reading the data in the file), which had been updated with the current record number `iom_file(.)%dimsz(1,4)`.

File	calling subroutine	Read/write status	IOM interface	Argument to set
background	**bkg_init** nemovar.F90 (nemovar_init)	**read**	iom_get (iom.F90)	`ktime` time record index
increment	**inc_init** nemovar.F90 (nemovar_init)	**read**	iom_get (iom.F90)	`ktime` (time record index)
	rstinc inc.F90 nemovar_inner_	**write**	iom_rstput (iom.F90)	`kt` and `kwrite` time step
inner loop restart	`diactl_restart` dia_res.F90 nemovar_inner_ jp_read	**read**	iom_get (iom.F90)	`ktime` time record index
	`diaprec_restart` dia_res.F90 nemovar_inner_ jp_write	**write**	iom_rstput (iom.F90)	`kt` and `kwrite` time step

Caution: changes to be made in the `write_ctlvec` and `read_ctlvec` subroutines

A.1.4 Issues connected to J and ∇J calculation and strategy

The computation of J and ∇J is required before each new inner loop iteration launched by the `nemovar_inner_drivers` routines. The computation of ∇J is performed through calls to the simulation routines `simvar`. Appendix section 3.4.3 provides information on the operations performed by the `nemovar_inner_drivers` minimizer CONGRAD to compute J and ∇J to prepare each new inner loop iteration.

Key points to note, in the frame of a weak 4D-var formulation:

- the size of ∇J is augmented: the number of components is multiplied by `isw`.
- there are `isw` background and observation terms in the J equation:
  - each observation term only depends on one subtrajectory initial state (important point to implement parallel computations),
  - each background term (except the first) depends on two successive subtrajectory initial states.

We face the issue connected to the vector size, and the number of 3D and 2D fields. It is assume in the `simvar` routines that the state vector only includes four 3D fields and a single 2D fields. It is not suited for the weak 4D-var formulation.

Issues connected to error covariance matrices

The construction of the background error covariance matrices includes three main parts:

1. the B_u matrix, which is a block diagonal matrix made of univariate error covariance matrices targetting t, s, u, v and η. Each block is constructed in two steps:
   - the calculation of the error correlation coefficients (implicit or explicit diffusion operator).
   - the calculation of the background error variances.
2. the balance operator $\mathcal{K}^{1/2}(x)$, which sets the physical equilibrium.

At full-term, we will apply different error covariance matrices for each section of the background vector x_0^α, x_0^β, x_0^γ. One question to answer by that time is do we wish to have different correlation and variance coefficients, or do we simply want to set the proper balance operator.

Issues connected to cost function

To calculate J, the I/O operations that are performed in the `nemovar_inner_drivers` have to be modified in particular:

- when loading the cumulated increment of the "restart_ctlvec" file inbto the `z_bctl ctlvec` before entering the inner loop,

  ```
  CALL diactl_restart( jp_read, nitrt, noutit, z_bctl )
  ```

  The `diactl_restart` routine (source MIN/dia_res.F90) calls subroutine `read_ctlvec` (source MIN/control_vectors.F90) which loads fields into a `ctlvec` variable.

- when updating the "increment_" file at the end of the minimization:

  ```
  CALL diactl_restart( jp_write, nitrt, noutit, z_bctl )
  ```

  The `diactl_restart` routine calls subroutine `read_ctlvec` which writes the `ctlvec` variable into the files.

In both instruction, the augmented increment must be written, and changes to take into account the "time" dimension must be operated in the `iom_get` and `iom_put` interfaces.

Issues connected to gradient

In order to calculate all the components of the ∇J vector we modify the `simvar_` routine:

- we introduce a loop on the `isw` subwindow,
- we apply the reduced size `ctlvec` variable when necessary.

Appendix B

Second Appendix: details on some features of NEMO/NEMOVAR

B.1 Observation space vectors and corresponding variables in the code

B.1.1 Key observation space vectors

In the inner loops, the three key observation space vectors are:

- the innovation:
$$d_k = -y_k^o + \mathcal{M}(x_0^{b,(g-1)})$$
- the model counterpart of the increment linear trajectory at observation points:
$$\mathbf{H}_k \mathbf{M}_k \, \delta x_0$$
- the weighted model observation discrepency, which results from the combination of the two vectors mentionned above:
$$\mathbf{R}^{-1}(\mathbf{H}_k \mathbf{M}_k \, \delta x_0 - d_k)$$

B.1.2 Storage strategy

The data of the observation space are stored into files called feedback, which ensure the off-line communication between NEMO and NEMOVAR. During the code execution (nemo or nemovar), these data are read and stored into structured type variables.

Within the inner loops in particular, the module `obsstore` (see `NEMOASSIM/VAR_SRC/OBSTAM/obssto` defines the variable that will store different types of observation. Table B.1 provide a sumary of these variables.

Data	**Native**	**Quality controlled**	nb of sets	number of var
Profile	`profdata`	`prodatqc`	`nprofsets`	`nprofvars`
SLA	`sladata`	`sladatqc`	`nslasets`	`nslavars`
SST	`sstdata`	`sstdatqc`	`nsstsets`	`nsstvars`
Lagrangian data	`lagdata`		`nlag`	

Table B.1: Structured type to store observations.

Variables of type `datqc` include different fields shown in table B.2. The `vext` field is a key variable to construct the observation terms in the cost function and its gradient.

Fields	**Profile data**	**Depth**	**Model counterpart**	**Extra data**
Variable	`vobs`	`vdep`	`vmod`	`vext`
Dimension	1D	1D	1D	2D

Table B.2: Structured type to store observation data: profile.

We illustrate how it works in the next section, taking variable `prodatqc` dedicated to profile observation as an example.

B.1.3 Illustration with profile observations

For profile data, two structured types are defined (see the `NEMO/OPA_SRC/OBS/obs_profiles_def.F90` source):

- `obs_prof_var` to store the **profile data**,
- and `obs_prof` to store the **profile information**.

Variable `prodatqc` is of type `obs_prof`, and holds the `vext` fields where the innovation, the model counterpart of the linear trajectory increment, and the weighted model observation discrepancy are stored. The table B.3 shows the correspondance between those vectors and the code variables.

Table B.4 show other important fields of the `obs_prof_var` type, where are stored the data vectors and the meta data respectively (see `OBSTAM/obspoint.F90`).

Observation vectors	**Field** of `*datqc(jset)%var(ivar)`
Tangent linear model equivalent	`vext(:,mp3dtan)`
Innovation	`vext(:,mp3dinn)`
Resulting weighted forcing	`vext(:,mp3dwdt)`
Interpolated background standard deviation	`vext(:,mp3dbesdana)`

Table B.3: Vectors in the observation space stored in the `%vext` fields.

Information	Total nb of profile	Observation counter	Time step nearest to profile
Variable	`nprof`	`nprofup`	`mstp`

Table B.4: Structured type to store observation data information: profile

Note:

The `dia_obs_oper_tam.F90` uses the module `obs_profiles_def` by the mean of the module declaration:

```
USE obs_oper_tam
```

The latter module includes the subroutine `obs_pro_opt_tan` which computes the tangent linear model counterpart of the data (for profile observation).

B.2 The linearization of the balance operator at each inner loop iteration

The linearization of the balance operator $\mathcal{K}$ around the reference state x_0^r at each inner loop iteration is required, because δu_0, the increment of the control vector must be transformed into δx_0, the increment of the state vector. This step is performed by applying the square root of the multivariate background error matrix $B^{1/2}$:

$$\delta x_0 = \mathrm{B}^{1/2} \delta u_0 \tag{B.1}$$

The matrix B is made of two parts:

$$\mathrm{B}^{1/2} = \mathcal{K}^{1/2}\, \mathrm{B}_u^{1/2} \tag{B.2}$$

where:

- B_u is a block matrix of the univariate background covariance error matrices,
- $\mathcal{K}^{1/2}$ is the balance operator (which restores physical equilibrium between fileds).

The operator $\mathcal{K}^{1/2}$ appears in the backgound cost function. Its depends on x_0. The incremental approach involves the linearization of the operators of the cost function in order to construct a quadratic cost function.

To derive the right expression for the linearized balance operator, let us come back to the background term of the non quadratic cost function at outer loop g:

$$\begin{aligned} J^b\left(x_0 - x_0^b\right) &= \left(x_0 - x_0^b\right)^T \mathcal{K}\left(x_0\right)^{-T/2} \mathbf{B_u}^{-1} \mathcal{K}\left(x_0\right)^{-1/2} \left(x_0 - x_0^b\right) \\ &= \underbrace{[\mathcal{K}\left(x_0\right)^{-1/2} \left(x_0 - x_0^b\right)]}_{\text{Term to be linearized}}{}^T \mathbf{B_u}^{-1} [\mathcal{K}\left(x_0\right)^{-1/2} \left(x_0 - x_0^b\right)] \end{aligned} \tag{B.3}$$

The linearization must be performed around x_0^{g-1}, the reference state vector that we denote x_0^r:

$$x_0 = x_0^r + \delta x_0$$

Let us simplify the notation with $\mathcal{A} = \mathcal{K}^{1/2}$ and by dropping the initial time index, i.e., $x_0 = x$:

$$\begin{array}{ll} [\mathcal{A}(x)]_{i,j}.\left(x_j - x_j^b\right) & = [\mathcal{A}(x^r)]_{i,j}.\left(x_j^r - x_j^b\right) \\ & +\frac{\partial[\mathcal{A}(x^r)]_{j,j}}{\partial x_k}.\delta x_k\,.\left(x_j^r - x_j^b\right) + [\mathcal{A}(x^r)]_{i,k}\,.\delta x_k + \mathcal{O}\left(||\delta x||^3\right) \\ = & [\mathcal{A}(x^r)]_{i,j}.\left(x_j^r - x_j^b\right) \\ & +[\frac{\partial[\mathcal{A}(x^r)]_{i,j}}{\partial x_k}.\left(x_j^r - x_j^b\right) + [\mathcal{A}(x^r)]_{i,k}]\,.\delta x_k + \mathcal{O}\left(||\delta x||^3\right) \\ \approx & [\mathcal{A}(x^r)]_{i,j}.\left(x_j^r - x_j^b\right) \\ & +[\frac{\partial[\mathcal{A}(x^r)]_{i,j}}{\partial x_k}.\left(x_j^r - x_j^b\right) + [\mathcal{A}(x^r)]_{i,k}]\,.\delta x_k \end{array} \tag{B.4}$$

Connexion with personnal communication from Arthur The balance operator is linearized around the the reference state vector x_0^r:

$$B = \frac{\partial K}{\partial x_0^r}^{T/2} B_{\mathrm{u}} \frac{\partial K}{\partial x_0^r}^{1/2} \tag{B.5}$$

Appendix C

Third Appendix

C.1 Cost function approximation around the optimum

$$\begin{aligned} J(x^{\alpha a} + h\, e_\alpha, x^{\beta a} + h'\, e_\beta) &\simeq J(x^{\alpha a}, x^{\alpha a}) + \begin{pmatrix} \partial_{x^\alpha} J & \partial_{x^\alpha} J \end{pmatrix} \begin{pmatrix} h\, e_\alpha \\ h'\, e_\beta \end{pmatrix} \\ &\quad + \begin{pmatrix} h\, e_\alpha & h'\, e_\beta \end{pmatrix} \begin{pmatrix} \partial_{x^\alpha, x^\alpha} J & \partial_{x^\alpha, x^\beta} J \\ \partial_{x^\beta, x^\alpha} J & \partial_{x^\beta, x^\beta} J \end{pmatrix} \begin{pmatrix} h\, e_\alpha \\ h'\, e_\beta \end{pmatrix} \end{aligned} \tag{C.1}$$

C.1.1 Minimizer and pre-conditionning techniques

TO DO IF TIME.

The Hessian conditionning number

A well-known result about preconditionner is that $\mathcal{H}$, the hessian of the assimilation system is the inverse of the analysis error covariance matrix P^a:

$$\begin{aligned} \mathcal{H} &= P^{a\,-1} \\ &= B^{-1} + \mathbf{H}\, R^{-1}\, \mathbf{H}^{-1} \end{aligned} \tag{C.2}$$

During iterations of basic CG minimization, neither $\mathcal{H}$ nor P^a are calculated or approximated. The simple pre-conditionning approach is based on the crude approximation:

$$\mathcal{H} \simeq B^{-1} \tag{C.3}$$

Which leads to the variable change (note the simultaneous adimentionalisation of the problem):

$$v = B^{-1/2} x \tag{C.4}$$

C.1.2 NEMOVAR congrad minimizer

CONGRAD is a Lanczos type PCG minimizer (Lanczos to handle Matrix-vector product input and PCG for Precontionning Conjugate Gradient). Two options for pre-conditioning is proposed:

1. Ritz preconditioner
2. LBFGS preconditioner

LBFGS preconditioner: Limited memory preconditioning

A set of vectors of size n is retained from past PCG computations, and used together with B to derive an approximation of the inverse of $B^{-1} + \mathbf{H}R^{-1}\mathbf{H}^T$,and this approximation is then used to precondition the PCG iterations on the next outer iteration.

Appendix D

Fourth Appendix

D.1 Topic to investigate

D.1.1 Hints to plan the work

We plan to implement in NEMOVAR the third weak constraint 4D-var option. We have to check:

- Build a very simple case to verify the conditionning problem before starting developments on NEMOVAR
  What causes the bad conditionning number?

- Other hint: apply the last step full state vector as observation, the problem does not require a regularization, which permits the application of diagonal matrices Q_α and Q_β. Perform tests and development in such simple framework (wait for `NEMOVAR/VODA` merge, simple way of handling implicit background covariance, matrices and nupvel can now be applied to apply increment and not only balanced information).
- Tools to estimate the conditionning number of the Hessian without knowing the components of the matrix:
  - Some minimizers estimate the conditionning number,
  - What about the nemovar minimizer? congrad (new version), cgmod,
- Implementation possibilies in NEMOVAR:
  - Vector augmentation (see Elisabeth Remy'work),
  - Increment application,
  - Routine too encapsulate the congrad minimizer (i.e., fragmented actions with matrix vector multiplication as input)?
  - if problem: other minimizer?

Bibliography

Trémolet, Y. (2006), 'Accounting for an imperfect model in 4d-var', *Q. J. R. Meteorol. Soc.* **132**(621), 2483–2504.

Trémolet, Y. (2007), 'Model-error estimation in 4d-var', *Q. J. R. Meteorol. Soc.* **133**(626), 1267–1280.
URL: *http://dx.doi.org/10.1002/qj.94*