Deliverable D1.3:

NEMO Tangent & Adjoint Models (NemoTam)


Arthur Vidard, Franck Vigilant
INRIA/LJK, Grenoble

Charles Deltel, Rachid Benshila
LOCEAN-IPSL, Paris

VODA: ANR-08-COSI-016
## Contents

1 NEMOTAM Reference Manual

1.1 Introduction ................................................................................. 5
  1.1.1 Abstract .............................................................................. 5
  1.1.2 Introduction .......................................................................... 5

1.2 Tangent and adjoint models (TAM) basics ...................................... 7
  1.2.1 TAM, but what for? ............................................................. 7
  1.2.2 Tangent and Adjoint coding principle ................................. 7
  1.2.3 Potential issues .................................................................... 8

1.3 Direct model changes .................................................................... 8
  1.3.1 Storing the non-linear trajectory .......................................... 8
  1.3.2 Modified routines in the direct code ..................................... 10
  1.3.3 Modified declarations in the direct code .............................. 10

1.4 Tangent linear model approximations ........................................... 10
  1.4.1 Active variables degraded to passive .................................... 10
  1.4.2 Simplifications ..................................................................... 11

1.5 Hand coding status ................................................................. 11

1.6 Available options in NEMOTAM ................................................ 11

1.7 The generic validation interface .................................................. 11
  1.7.1 Introduction ........................................................................ 11
  1.7.2 Adjoint test ......................................................................... 16
  1.7.3 Tangent-Linear validation .................................................. 17
  1.7.4 Tangent-Linear Hypothesis (HLT) validity ............................ 19

1.8 MPI parallelization ...................................................................... 20
  1.8.1 General features .................................................................. 20
  1.8.2 Adjoint of MPI_ALLGATHER ........................................ 21

2 NEMOTAM User’s Guide

2.1 Source code structuration ......................................................... 23

2.2 Getting started tutorial ............................................................. 24
  2.2.1 Installing the code using the NEMOVAR GIT repository .......... 24
  2.2.2 Installing the code using the NEMOTAM SVN repository ........ 25
  2.2.3 Which executable for which test? ....................................... 26

2.3 Demonstrators ........................................................................... 26
2.3.1 The ORCA2 global configuration .............................. 26
2.3.2 The GYRE regional configuration .............................. 32
2.3.3 The POMME configuration (toy model for handling open boundaries) ... 33
2.4 How-to update NEMOTAM with your contribution? ............... 34
2.4.1 Coding rules for TAM ........................................ 34
2.4.2 Workflow for introducing additional developments ............. 35
2.5 Frequently asked questions ..................................... 36
2.5.1 I can’t use GDB after compiling with FCM .................... 36
2.5.2 I have many commas in the namelist generated by piano ...... 36
2.5.3 I have negative values in my cost function terms ............ 36
2.5.4 My run stops with ’PROBLEM WITH THE MATRIX OR WITH THE PRECONDITIONER’ on stderr ................. 36
Chapter 1

NEMOTAM Reference Manual

1.1 Introduction

1.1.1 Abstract

The development of the tangent linear and adjoint models (TAM in the following) of the dynamical core of the NEMO ocean engine (NEMOTAM) is a key objective of the VODA project. TAM are widely used for variational assimilation applications, but they are also powerful tools for the analysis of physical processes, since they can be used for sensitivity analysis, parameter identification and for the computation of characteristic vectors (singular vectors, Liapunov vectors, etc.).

In the framework of VODA, a work package has been set-up in order to develop a comprehensive NEMOTAM package, and to define an effective long-term development strategy for ensuring synchronisation of NEMOTAM with future NEMO releases. This is a heavy task, but it is worth the effort since NEMOTAM will benefit all NEMO users for the wide range of applications described above.

Ideally, this strategy should be defined to allow NEMOTAM to adapt to future NEMO developments as quickly and as efficiently as possible, so that new releases of NEMOTAM can be made soon after new releases of NEMO. This will require careful coordination between the main development teams of NEMO, NEMOTAM and possibly NEMOVAR (INRIA, NEMO Team, CERFACS, ECMWF).

1.1.2 Introduction

The NEMO ocean engine (Madec 2008) was previously known as the OPA model (Madec et al. 1998). It used to have a TAM (called OPATAM), fully hand-coded and maintained mainly by A. Weaver. OPATAM was initially developed for a Pacific ocean configuration, and targeted at variational data assimilation applications in the framework of OPAVAR (Weaver et al. 2003, 2005). OPATAM/OPAVAR were extended to other regional basins (Mediterranean sea (Rémy 1999), North Atlantic 1/3°(Forget et al. 2008), South Atlantic 1°), to the global ocean (ORCA 2° (Daget et al. 2009)), and were used for methodological studies such as control of the 3D model error (Vidard 2001), control of the surface forcing and open boundary conditions (Deltel 2002,
Vossepoel et al. 2003). OPATAM was also used for sensitivity studies (Sévellec et al. 2008), singular vectors (Moore et al. 2003, Sévellec et al. 2009), etc.

For several reasons, mainly because of lack of workforce, OPATAM, OPAVAR and related developments were not included in the standard release of OPA. As a consequence, synchronisation of OPATAM with OPA’s releases could not be achieved on a regular basis, and all developments were on individual branches, without feedback to the OPATAM/OPAVAR system. The pool of potential users was reduced consequently. It is important not to repeat this error in the future, so as to ensure that NEMOTAM become a widely used community tool.

A NEMOTAM working group was initiated in the framework of a CNRS/INSU/LEFE ASSIM-2006 project, to investigate the feasibility of using TAPENADE (Hascoët & Pascual 2004), an AD tool, to speed up the writing of TAM for OPA9, the dynamical ocean component of NEMO. The goal of this working group was twofold. The first goal was to identify the strengths and weaknesses of TAPENADE by applying it directly to the NEMO source code, with as little human intervention as possible, in order to build a NEMOTAM prototype. The second goal was to define, based on the experience deriving the prototype, a strategy for developing a general-purpose NEMOTAM that both respects the NEMO code style/structure and gives acceptable computer performance (in terms of both CPU and memory requirements) for realistic ocean configurations. Providing feedback to the TAPENADE developers was an important aspect of this work to help improve future versions of the TAPENADE software.

The results from this feasibility study demonstrated that TAPENADE was able to produce tangent-linear and adjoint models for NEMO, albeit for a fixed and somewhat simplified configuration (ORCA $2^{\circ}$ with all non-differentiable options switched off (Tber et al. 2007)). Even for this simplified configuration, however, substantial human intervention and additional work was required to obtain a useable product from the raw TAPENADE-generated code. Three main drawbacks with TAPENADE were identified for this application. First, the memory management and CPU performance of the raw code were rather poor. Second, the current version of TAPENADE generates single-processor code only and cannot handle directives from the C-PreProcessor (CPP keys), which are widespread in NEMO. Third, the technique of binomial checkpointing that is used in TAPENADE to handle nonlinearities (see (Tber et al. 2007)) is not compatible, at least in its present implementation, with the incremental algorithm of NEMOVAR, which employs separate executables and (possibly) different resolutions for the outer and inner loops. Improved memory management and extensions to support MPP and CPP keys are planned in future versions of TAPENADE so the first two deficiencies are not fundamental. The third deficiency, however, is more problematic and it is likely that the trajectory management for nonlinearities in NEMOTAM will be done differently from TAPENADE, possibly along the lines of the simpler strategy implemented in OPAVAR. The modifications required to make TAPENADE or whatever other AD tool, compatible with the multi-incremental approach are really substantial and cannot be done in a short or medium term. Moreover the numerical performances of the TAPENADE generated TAM do not allow yet their use for ‘big’ configurations and for operational applications.

From that experience it has been decided to go toward the hand-coding approach, the use of AD tool being left aside for the time being. We may reconsider this in a medium or long term though.

1.2 Tangent and adjoint models (TAM) basics

1.2.1 TAM, but what for?

The development of tangent and adjoint models is an important step in addressing sensitivity analysis and variational data assimilation problems in Oceanography. Sensitivity analysis is the study of how model output varies with changes in model inputs. The sensitivity information given by the adjoint model is used directly to gain an understanding of the physical processes. In data assimilation, one considers a cost function which is a measure of the model-data misfit. The adjoint sensitivities are used to build the gradient for descent algorithms. Similarly the tangent model is used in the context of the incremental algorithms to linearize the cost function around a background control.

1.2.2 Tangent and Adjoint coding principle

The original program $P$, whatever its size and run time, computes a function

$$F : X \in \mathbb{R}^m \rightarrow Y \in \mathbb{R}^n$$

which is the composition of the elementary functions computed by each run-time instruction. In other words if $P$ executes a sequence of elementary statements $I_k$, $k \in [1, .., p]$, then $P$ actually evaluates

$$F = f_p \circ f_{p-1} \circ ... \circ f_1$$

where each $f_k$ is the function implemented by $I_k$. Therefore one can apply the chain rule of derivative calculus to get the Jacobian matrix $F'$, i.e. the partial derivatives of each component of $Y$ with respect to each component of $X$. Calling $X_0 = X$ and $X_k = f_k(X_{k-1})$ the successive values of all intermediate variables, i.e. the successive states of the memory throughout execution of $P$, we get

$$F'(X) = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \times ... \times f'_1(X_0)$$  \hspace{1cm} (1.1)

The derivatives $f'_k$ of each elementary instruction are easily built, and must be inserted in the differentiated program so that each of them has the values $X_{k-1}$ directly available for use. This process yields analytic derivatives, that are exact up to numerical accuracy. In practice, two sorts of derivatives are of particular importance in scientific computing: the tangent (or directional) derivatives, and the adjoint (or reverse) derivatives. The tangent derivative is the product

$$dY = F'(X) \times dX$$

of the full Jacobian times a direction $dX$ in the input space. From equation (1.1), we find

$$dY = F'(X) \times dX = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \times ... \times f'_1(X_0) \times dX$$  \hspace{1cm} (1.2)

which is most cheaply executed from right to left because matrix $\times$ vector products are much cheaper than matrix $\times$ matrix products. This is also the most convenient execution order because it uses the intermediate values $X_k$ in the same order as the program $P$ builds them. On the other
hand the adjoint derivative is the product $X_{ad} = F'^*(X) \times Y_{ad}$ of the transposed Jacobian times a weight vector $Y_{ad}$ in the output space. The resulting $X_{ad}$ is the gradient of the dot product $(Y \cdot Y_{ad})$. From equation (1), we find

$$X_{ad} = F'^*(X) \times Y_{ad} = f'^*(X_0) \times \ldots \times f'^*(X_{p-2}) \times f'^*(X_{p-1}) \times Y_{ad} \quad (1.3)$$

which is also most cheaply executed from right to left. However, this uses the intermediate values $X_k$ in the inverse of their building order in $P$.

For specific details about practical adjoint coding refer to (Giering & Kaminski 1998).

### 1.2.3 Potential issues

- the major approximations: non differentiability issues, simplification of the direct model before linearization, validity of the tangent linear hypothesis over some time window, etc.
- the validation interface: classical tests for checking the correctness of tangent and adjoint models
- numerical problems
- numerical cost issues

### 1.3 Direct model changes

Some modifications and additions are compulsory to integrate the Tangent and Adjoint Module to NEMO. They mainly consist in:

- changing subroutines or parameters declaration from `private` to `public` for:
  - initialization
  - re-use
- adding a subroutine to save the non-linear trajectory
- adding parameters
- updating namelist

#### 1.3.1 Storing the non-linear trajectory

To evaluate the tangent and/or the adjoint at a time $t$, some actual parameters of the direct model are needed at the corresponding time $t$. To fulfill this purpose a subroutine, `tamtrj_wri` is added to write the trajectory. It is controlled with one logical parameter: `In_trjwri`. The trajectory is saved upon a given frequency “nitrjfrq” only if the logical parameter is set to TRUE.

**New subroutine tam_trj_wri**

A new subdirectory TAM is created into OPA_SRC named TAM. The file is `tamtrj.F90`. The purpose of this routine is to write on disk the model state trajectory for use with 4DVar. It is called in the `opa_model` routine for the initialization, and in the `stp` routine in order to perform the actual saving of the trajectory. The currently stored fields are listed in Table 1.1.
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Associated ccp keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp</td>
<td></td>
</tr>
<tr>
<td>emps</td>
<td></td>
</tr>
<tr>
<td>un</td>
<td></td>
</tr>
<tr>
<td>vn</td>
<td></td>
</tr>
<tr>
<td>tn</td>
<td></td>
</tr>
<tr>
<td>sn</td>
<td></td>
</tr>
<tr>
<td>ta</td>
<td></td>
</tr>
<tr>
<td>sa</td>
<td></td>
</tr>
<tr>
<td>tb</td>
<td></td>
</tr>
<tr>
<td>sb</td>
<td></td>
</tr>
<tr>
<td>avmu</td>
<td></td>
</tr>
<tr>
<td>avmv</td>
<td></td>
</tr>
<tr>
<td>aeiu</td>
<td>lk_traldf_eiv</td>
</tr>
<tr>
<td>aeiv</td>
<td>lk_traldf_eiv</td>
</tr>
<tr>
<td>aeiw</td>
<td>lk_traldf_eiv</td>
</tr>
<tr>
<td>uslp</td>
<td>key_ldfslp</td>
</tr>
<tr>
<td>vslp</td>
<td>key_ldfslp</td>
</tr>
<tr>
<td>wslpi</td>
<td>key_ldfslp</td>
</tr>
<tr>
<td>wslpj</td>
<td>key_ldfslp</td>
</tr>
<tr>
<td>avs</td>
<td>key_zdfddm</td>
</tr>
<tr>
<td>strdmp</td>
<td>key_tradmp</td>
</tr>
<tr>
<td>hmp</td>
<td>key_tradmp</td>
</tr>
</tbody>
</table>

Table 1.1: Fields stored in the reference trajectory
Specific management of the initial state

For the initial state of the model, we call the tam_trj_wri in the 'step' routine just before the tracer block. We, then, have access to some parameters (tke and sbc for instance) as they are not yet computed in opa (outside 'step').

Linear interpolation

For efficiency matters, the whole trajectory is of course not saved. Only a predefined subset of the time steps is saved. The frequency on which the trajectory is saved is controlled by namelist parameter nittrjfrq. The intermediate steps will be estimated in TAM by the mean of a linear interpolation.

Namelist

Parameters related to the model state saving is controlled by the namelist namtam with:

- ln_trjwri: Logical switch for writing out state trajectory
- nittrjfrq: saving frequency of the trajectory

1.3.2 Modified routines in the direct code

- opa.F90:
  - include initialilization of the module that handles the trajectory saving (USE tamtrj)
- step.F90:
  - include calls for saving the trajectory (CALL tam_trj_wri)
  - add temporary variables (zta, zsa) to save ta/sa (this is a temporary fix because these variables are currently used as workspace in dynamic block)

1.3.3 Modified declarations in the direct code

The following table describes the change from PRIVATE to PUBLIC of some routines and parameters.

1.4 Tangent linear model approximations

1.4.1 Active variables degraded to passive

Some active variables are treated as passive (mainly due to non-differentiability characteristics).

- avmu, avmv: the vertical eddy viscosity. The turbulent closure scheme zdftke induces some non-differentiabilities for there variables. They are used in routine dynzdf_imp_tam. F90.
<table>
<thead>
<tr>
<th>Modules</th>
<th>Routines</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynadv</td>
<td>dyn_adv_ctl</td>
<td></td>
</tr>
<tr>
<td>daymod</td>
<td>day_mth</td>
<td></td>
</tr>
<tr>
<td>lib_mpp</td>
<td>mpi_comm_opa</td>
<td></td>
</tr>
<tr>
<td>par_oce</td>
<td>jpni, jpnj, jpnij</td>
<td></td>
</tr>
<tr>
<td>zdftke</td>
<td>tke_rst</td>
<td></td>
</tr>
<tr>
<td>eosbn2</td>
<td>neos_init</td>
<td></td>
</tr>
<tr>
<td>tradmp</td>
<td>cofdis, dtacof, dtacof_zoom</td>
<td>strdmp, ttrdmp, resto</td>
</tr>
<tr>
<td>traqsr</td>
<td>nksr, gdsr</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.2: List of modified declarations in the direct code NEMO

- uslp, vslp: the slope of neutral surface (slope of isopycnal surfaces referenced locally) are treated as passive in order to not recomputed them at each step. They are used in routine traldf_iso_tam.F90.
- wslpi/j: vertical mixing coefficient due to lateral mixing. They are used in routine trazdf_imp_tam.F90.

1.4.2 Simplifications

- removal of the upstream part of the second order centered scheme for traadv_cen2_tam.F90. It introduces some non-differentiabilities. Some complementary test must be held to fully implement the second order scheme
- removal of zonal mean lateral diffusive heat and salt transport (pht_ldf and pst_ldf)

1.5 Hand coding status

The current status for the hand coding part of NEMOTAM / NEMOVAR is gathered in Tables 1.3 and 1.4.

1.6 Available options in NEMOTAM

This sections contains information on current available configuration for NEMOTAM. To have an overview on available routines, please see the TAM development progress table. We use the following method: The physical choices are set through cpp keys and namelist parameters. We use below a similar table used for NEMO. See tables 1.5 and 1.6.
<table>
<thead>
<tr>
<th>Modules called by step: tam</th>
<th>Module(s) called by previous one</th>
<th>Options</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>cla_div_tam.F90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cla_tam.F90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>divcur_tam.F90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dynhpg_tam.F90</td>
<td></td>
<td>hpg_zco, hpg_zps, hpg_sco, hpg_hel, hpg_wdj, hpg_djc, hpg_rot</td>
<td></td>
</tr>
<tr>
<td>dynnxt_tam.F90</td>
<td></td>
<td>lk_yvl, key_obc, key_bdy, key_agrif</td>
<td>non-linear, not in reference, not in reference</td>
</tr>
<tr>
<td>dynvor_tam.F90</td>
<td></td>
<td>dyn_vor_en, dyn_vor_een, dyn_vor_ene, dyn_vor_mix</td>
<td>not in reference, not in reference</td>
</tr>
<tr>
<td>dynzdf_tam.F90</td>
<td>dynzdf_exp_tam.F90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dynzdf_imp_tam.F90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dynmem_tam.F90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wzvmod_tam.F90</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1.3: Hand coding status of NEMOTAM, Dynamics part
<table>
<thead>
<tr>
<th>Modules called by step</th>
<th>Module(s) called by previous one</th>
<th>Options</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ocean tracers (TRA)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>traadvTam.F90</td>
<td>traadv_cen2Tam.F90</td>
<td></td>
<td>no upstream scheme (equivalent to zcofi=0)</td>
</tr>
<tr>
<td></td>
<td>traadv_tvdTam.F90</td>
<td></td>
<td>not differentiable</td>
</tr>
<tr>
<td></td>
<td>traadv_musclTam.F90</td>
<td></td>
<td>not differentiable</td>
</tr>
<tr>
<td></td>
<td>traadv_muscl2Tam.F90</td>
<td></td>
<td>not differentiable</td>
</tr>
<tr>
<td></td>
<td>traadv_ubsTam.F90</td>
<td></td>
<td>not differentiable</td>
</tr>
<tr>
<td></td>
<td>traadv_qckTam.F90</td>
<td></td>
<td>not differentiable</td>
</tr>
<tr>
<td></td>
<td>traadv_eivTam.F90</td>
<td></td>
<td>not differentiable</td>
</tr>
<tr>
<td>trabcctam.F90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tradmpTam.F90</td>
<td></td>
<td></td>
<td>hlmp is saved in direct model to handle uslp, vslp</td>
</tr>
<tr>
<td>traldf Tam.F90</td>
<td>traldf_bilapTam.F90</td>
<td>key_ldf_ano</td>
<td></td>
</tr>
<tr>
<td></td>
<td>traldf_bilapgTam.F90</td>
<td></td>
<td>uslp, vslp treated as passive</td>
</tr>
<tr>
<td></td>
<td>traldf_lapTam.F90</td>
<td></td>
<td>not in reference</td>
</tr>
<tr>
<td>tranxtTam.F90</td>
<td>key_obc, key_bdy</td>
<td></td>
<td>not in reference</td>
</tr>
<tr>
<td></td>
<td>key_agrif</td>
<td></td>
<td>not in reference</td>
</tr>
<tr>
<td>traqsrTam.F90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trasbcTam.F90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trazdf Tam.F90</td>
<td>trazdf_impTam.F90</td>
<td>lk_vvl</td>
<td>non-linear</td>
</tr>
<tr>
<td></td>
<td>trazdf_expTam.F90</td>
<td>lk_vvl</td>
<td>wslpi, wslpj treated as passive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lk_vvl</td>
<td></td>
</tr>
<tr>
<td><strong>Surface boundary conditions (SBC)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sbcmodTam.F90</td>
<td>sbc_gyreTam.F90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sbc_fbxTam.F90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sbc_ssrTam.F90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sbc_fwbTam.F90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sbc_clotamTam.F90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sbc_anatamTam.F90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1.4: Hand coding status of NEMOTAM, Tracers and Surface Boundary Condition part
<table>
<thead>
<tr>
<th>Used for</th>
<th>Functionalities</th>
<th>CPP keys</th>
<th>Associated namelist logical</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Horizontal Diffusion/Viscosity</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lateral diffusion along isopycinal slopes</td>
<td>key_ldfslp</td>
<td></td>
</tr>
<tr>
<td><strong>Horizontal Tracer diffusion</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tracer lateral diffusion eddy induced velocity parametrization</td>
<td>key_traldf_eiv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tracer lateral laplacian</td>
<td></td>
<td>ln_traldf_lap</td>
</tr>
<tr>
<td></td>
<td>Iso-neutral application direction for operator</td>
<td></td>
<td>ln_traldf_iso</td>
</tr>
<tr>
<td></td>
<td>Tracer lateral diffusion 1d=f(depth), 2d=f(lat,ion) or 3d=f(lat,ion, depth)</td>
<td>key_traldf_c2d</td>
<td></td>
</tr>
<tr>
<td><strong>Horizontal Dynamics Viscosity</strong></td>
<td></td>
<td>nam_dynldf</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dynamic Laplacian operator lateral viscosity</td>
<td></td>
<td>ln_dynldf_lap</td>
</tr>
<tr>
<td></td>
<td>Dynamic biLaplacian operator lateral viscosity</td>
<td></td>
<td>ln_dynldf_bilap</td>
</tr>
<tr>
<td></td>
<td>Geopotential application direction viscosity</td>
<td></td>
<td>ln_dynldf_lap</td>
</tr>
<tr>
<td></td>
<td>Dynamic lateral viscosity 1d, 2d or 3d</td>
<td>key_dynldf_3cd</td>
<td></td>
</tr>
<tr>
<td><strong>Vertical viscosity Diffusivity</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Vertical diffusion based on TKE closure scheme</td>
<td>key_zdftke</td>
<td>namtke</td>
</tr>
<tr>
<td></td>
<td>Vertical double diffusion mixing for salinity</td>
<td>key_zdfddm</td>
<td>namddm</td>
</tr>
<tr>
<td><strong>Vertical convective processes</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>enhanced vertical diffusion parametrization</td>
<td></td>
<td>ln_zdfevd</td>
</tr>
<tr>
<td><strong>Surface forcing fields</strong></td>
<td></td>
<td>namsbc,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>key_forced_daily, key_tau_daily</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Initial state and damping</strong></td>
<td></td>
<td>key_tradmp</td>
<td>namdtp</td>
</tr>
<tr>
<td></td>
<td>Use a 3D monthly salinity data (Levitus)</td>
<td>key_dtasal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use a monthly temperature data (Levitus)</td>
<td>key_dtatem</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use a monthly sst data (Levitus)</td>
<td>key_dtasst</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sea surface temperature damping</td>
<td>namsbc,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sea surface salinity damping</td>
<td>namsbc,</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.5: Available option in NEMOTAM, part 1
<table>
<thead>
<tr>
<th>Used for</th>
<th>Functionalities</th>
<th>CPP keys</th>
<th>Associated namelist logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-line diagnostics</td>
<td>Eddy induced velocity (Gent-Williams parametrization)</td>
<td>key_diaeciv</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Thermocline, 20°C-28°C isotherm depths</td>
<td>key_diahth</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enable observation operator</td>
<td>key_diaobs</td>
<td></td>
</tr>
<tr>
<td>High Performances</td>
<td>Enable MPP computing with MPI</td>
<td>key_mpp_mpi</td>
<td></td>
</tr>
<tr>
<td>computing</td>
<td>reproductibility</td>
<td>key_mpi_send</td>
<td></td>
</tr>
<tr>
<td>Vertical coordinates</td>
<td>Full steps - Z</td>
<td>ln_zco</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Partial steps - PS</td>
<td>ln_zps</td>
<td></td>
</tr>
<tr>
<td>Pressure gradient</td>
<td>Filtered free surface algorithm</td>
<td>key_dynspgflt</td>
<td></td>
</tr>
<tr>
<td>treatment</td>
<td>Vorticity scheme</td>
<td>nam_dynvor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>enstrophy conserving</td>
<td>ln_dynvor_ens</td>
<td></td>
</tr>
<tr>
<td></td>
<td>enstrophy conserving</td>
<td>ln_dynvor_enn</td>
<td></td>
</tr>
<tr>
<td>Advection schemes</td>
<td>2nd order centered</td>
<td>ln_tradvcen2</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>Idealized Double Gyre configuration with a flat bottom</td>
<td>key_gyre</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Global ORCA configuration at 2deg resolution</td>
<td>key_orca_r2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Open boundaries POMME configuration at 0.25° resolution</td>
<td>key_pomme_r025</td>
<td></td>
</tr>
<tr>
<td>Other CPP keys</td>
<td>Geothermal boundary condition flux</td>
<td>nambbbc</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Diffusive Bottom Boundary Layer</td>
<td>key_trabbl_dif</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.6: Available option in NEMOTAM, part2
1.7 The generic validation interface

1.7.1 Introduction

The validation interface aims at showing the consistency of the TAM with respect to the direct code. To fulfill this goal we demonstrate that:

1. the adjoint code is the adjoint of its tangent code
2. the tangent linear code is the tangent linear of the direct code

with defined criteria.

In general, TAM modules are subdivided into four parts:

1. a tangent-linear routine
2. an adjoint routine
3. an adjoint test routine
4. a tangent-linear routine

Item 3 and 4 are optional. The strategy is to perform adjoint and tangent-linear tests for all critical sub-modules and the main module step_tam for time-step integration.

1.7.2 Adjoint test

By definition, $A^*$ (or $^tA$ in a finite-dimension case) is the adjoint operator of a linear operator $A$ defined on a Hilbert space if it has the following property:

$$ (Ax, y) = (x, {^tA}y) \quad (1.4) $$

In practice, we apply the comparison to random perturbations $\delta x$ in the model-space and $\delta y^*$ in the adjoint model-space. We have then:

$$ (A\delta x, \delta y^*) = (\delta x, {^tA}\delta y^*) \quad (1.5) $$

Adjoint test requirements

- A saved trajectory of the direct model
- An additionnal namelist (namtst for adjoint test, namtst_tlm for tangent test)

Unitary test versus Global test

Unitary test involves one or a limited set of routines. The perturbation vectors $\delta X$ and $\delta Y$ are the input/output variables identified as actives variables. There is no time evolution. Global test involves time evolution (loop on step_tam) and $\delta X$ and $\delta Y$ are input/output variables identified as the control vector (or variables of main interest).
Test Pass Criteria

We note (if exists) the relative error $Err$. The pass test criteria is based on the comparison of $Err$ with the epsilon machine $\varepsilon_{machine} = eps$.

$$Err = \frac{(A\delta x, \delta y^*) - (\delta x, 'A\delta y^*)}{(A\delta x, \delta y^*)}$$  \hspace{1cm} (1.6)

The adjoint test results output is the file `adjoint_test.output_xxx` with the following label:

<table>
<thead>
<tr>
<th>Test status</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test status is OK</td>
<td>$Err \leq Eps$</td>
</tr>
<tr>
<td>Test status is WARNING</td>
<td>$Eps &lt; Err \leq 100 \times Eps$</td>
</tr>
<tr>
<td>Test status is FAILED</td>
<td>$Err &gt; 100 \times Eps$</td>
</tr>
</tbody>
</table>

Table 1.7: Adjoint test labels definition

Run the adjoint test

- set the directory where the direct trajectory is stored
- in the namelist `namtst`
  - set `ln_tst_nemotam` to `.true.` (switch for M adjoint and tangent tests)
  - set `ln_tst_cpd_tam` to `.true.` (switch for adjoint tests of the sub-components)
  - set `ln_tst_stp_tam` to `.true.` (switch for adjoint tests of main components `step_tam`)
  - set `ln_tst_tan_cpd` and `ln_tst_tan` to `.false.` (switch for tangent tests)

1.7.3 Tangent-Linear validation

The tangent validation test checks that the Tangent Linear model $L$ is a first-order approximation of the direct model $M$. Then, we have the following second-order Taylor expansion:

$$M(X + p\delta X) = M(X) + pL(\delta X) + \varepsilon$$  \hspace{1cm} (1.7)

When the factor $p$ tends to zero:

1. $N_p = M(X + p\delta X) - M(X)$ tends to $L_p = pL(\delta X)$, first-order validation
2. $\varepsilon$ behaves as $O(p^2)$, second-order validation

this means that for a given $p_0$, for $p$ smaller than $p_0$ the error of Tangent Linear with respect to the model is decreasing as $p^2$.

Note that the second test (b) is not relevant if $M$ is linear.
<table>
<thead>
<tr>
<th>Test</th>
<th>Expression</th>
<th>behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>$\frac{\text{Norm}(N_{p,.})}{\text{Norm}(L_{p,.})}$</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>$\frac{\text{Norm}(N_{p} - L_{p,.})}{p^2}$</td>
<td>$Cst$</td>
</tr>
</tbody>
</table>

Table 1.8: Tangent Linear tests. First-order check (a) and second-order check (b)

**Build the tangent test**

The tangent test is embedded into nemotam. To build it the user must add the compilation key `key_tst_tlm` in the CPP key set. Otherwise, a error message will raise during the run.

**Run the tangent test**

To run the tangent test the code must

- set the directory where the direct trajectory is stored
- in the namelist `namst`
  - set ln_tst to .true.
  - set ln_tst_nemotam to .true. (switch for M adjoint and tangent tests)
  - set ln_tst_tan cpd or ln_tst_tan to .true. (switch for tangent tests)
  - set ln_tst_tan cpd tan or ln_tst_tan stp tan to .false. (switch for adjoint tests)
- in the namelist `namst_tlm`
  - set tlm_bch to .true. (branching: '0' for $M(X)$, '1' for $M(X + p\delta X)$ and '2' for $L_{pdX}$)
  - set curr_loop to .true. (current iteration)
  - set h_ratio to .true. (value of the ratio factor $p$ applied to $\delta X$)
Tangent test outputs

The output is saved with the name tan_diag.output_0000. The header gives information on the computed parameters. For iteration \( n \), the output gives the following data:

- Routine name, \( N_n = \mathcal{M}(X + p_n \delta X) - \mathcal{M}(X) \),
- Error, \( E_n = N_n / L(p_n \delta X) \),
- Error ratio, \( Er_n = (N_n - L(p_n \delta X)) / L(p_n \delta X) \).

To have an overview of the behaviour of the tangent iteration can be made by the user changing the factor ratio \( p_n \). The factor takes usually a geometric progression as: \( p_n = p_0 r^{-n} \), with initial value \( p_0 \) and common ratio \( r \).

1.7.4 Tangent-Linear Hypothesis (HLT) validity

The innovation vector validity relies on the validity of the Tangent Linear. Before any 4DVar assimilation we must ensure that the tangent model is valid within our assimilation time-window. Otherwise the optimisation may fail as the cost function gradient will be degraded or even wrong.

We define a maximum assimilation time-window \( T_{\text{max}} \). This time-window will be different for each configuration. According to each application the user its own criteria to define to define \( T_{\text{max}} \) to fulfill its needs. The size of \( T_{\text{max}} \) may vary from 1 to 3 months (Rémy 1999, Weaver et al. 2002, Deltel 2002).

To estimate \( T_{\text{max}} \) we study the behaviour of \( y_1 = \mathcal{M}_{t_0 \rightarrow t}(x_0 + \delta x) \) with respect to \( y_2 = \mathcal{M}_{t_0 \rightarrow t}(x_0) + L_{t_0 \rightarrow t} (\delta x) \).

The Tangent Linear will be valid as long as the difference \( y_1 - y_2 \) is "small". As discussed in (Deltel 2002) definition criteria of \( T_{\text{max}} \) will rely on threshold on the RMS of \( y_1 - y_2 \) and the coefficient correlation \( \rho_{y_1y_2} \). A correlation coefficient \( \rho_{y_1y_2} \) above 90\% is considered as very good.

\[
RMS(y_1 - y_2) = \sqrt{\frac{\sum_{i=1}^{n} (y_1^i - y_2^i)^2}{n}}, \quad n: \text{number of ocean points} \tag{1.8}
\]

\[
\rho_{y_1y_2} = \frac{\sum_{i=1}^{n} (y_1^i - \bar{y}_1) \cdot (y_2^i - \bar{y}_2)}{\sqrt{\sum_{i=1}^{n} (y_1^i - \bar{y}_1)^2} \cdot \sqrt{\sum_{i=1}^{n} (y_2^i - \bar{y}_2)^2}}, \quad n: \text{number of ocean points} \tag{1.9}
\]

The initial condition as well as the increment \( \delta x \) are critical to define \( T_{\text{max}} \). The user should work with a typical increment \( \delta x \) of its configuration.

HLT test tool

NEMOTAM offers the possibility to analyze the Tangent-Linear Hypothesis through a dedicated executable \textit{model_hlt.exe}. The increments \( \delta x \) is either given by the user either computed from two restarts file \( x_0 \) and \( x_1 \) (then \( \delta x = x_0 - x_1 \)). To compute \( y_1 \) and \( y_2 \), \textit{model_hlt.exe} should be run three times: two for the direct model trajectories (\( \mathcal{M}_{t_0 \rightarrow t}(x_0 + \delta x) \) and \( \mathcal{M}_{t_0 \rightarrow t}(x_0) \)) and one for
the tangent trajectory ($L_{t_0 \rightarrow t}(\delta x)$). It is driven through the namelists *namhlt* and *tl_amtrj*.

```
!-----------------------------------------------------------------------
! namhlt  tangent-linear hypothesis test parameters
! nstg  flag for to compute model with or without perturbation
! 0: M(X)
! 1: M(X+dX)
! 2: L(dX)
! ln_hltt Logical switch for applying temperature increments
! ln_hltt Logical switch for applying salinity increments
! ln_hltuv Logical switch for applying horizontal velocity increments
! ln_hltssh Logical switch for applying sea surface height increments
! ln_hnorm Logical switch to compute dx (F) or reading from file (T)
! rhstdt Upper bound of normalization factor for temperature
! rhstsd Upper bound of normalization factor for salinity
! rhstduv Upper bound of normalization factor for velocity
! rhstdssh Upper bound of normalization factor for sea surface height
!-----------------------------------------------------------------------
&namhlt
  nstg = 2
  ln_hltt = .TRUE.
  ln_hltt = .TRUE.
  ln_hltuv = .TRUE.
  ln_hltssh = .FALSE.
  c_hltinc = ""
  ln_hnorm = .TRUE.
  rhstsd = 1.
  rhstsd = 0.1
  rhstduv = 0.01
  rhstdssh = 0.01
/
!-----------------------------------------------------------------------
! namtl_trj  tangent-linear trajectory frequency output
! ln_trjwri_tan Logical switch ractjectory output
! nittrjfrq_tan frequency output for linear tangent
!-----------------------------------------------------------------------
&namtl_trj
  ln_trjwri_tan = .TRUE.
  nittrjfrq_tan = 96
/
```

### 1.8 MPI parallelization

#### 1.8.1 General features

We remind some basics for coding the corresponding adjoint of the tangent linear part:
<table>
<thead>
<tr>
<th>Operation</th>
<th>Tangent-Linear</th>
<th>Adjoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>communication</td>
<td>send</td>
<td>receive</td>
</tr>
<tr>
<td>communication</td>
<td>receive</td>
<td>send</td>
</tr>
<tr>
<td>communication</td>
<td>gather</td>
<td>scatter</td>
</tr>
<tr>
<td>arithmetic</td>
<td>scatter</td>
<td>gather</td>
</tr>
<tr>
<td>inpout/Output</td>
<td>write</td>
<td>read</td>
</tr>
<tr>
<td>inpout/Output</td>
<td>read</td>
<td>write</td>
</tr>
</tbody>
</table>

Table 1.9: ORCA2 Results of Adjoint test fo sub-routines

There is no tangent of \( lbc_{lnk} \). The direct routine is used. For the adjoint part we use the coding rule described in table 1.9

### 1.8.2 Adjoint of MPI\_ALLGATHER

From revision 3.2 of NEMO a new treatment of north fold horizontal bondary condition is performed using \( lbcnf.d.F90 \) routine. This induces a MPI\_ALLGATHER usage in \( libmpp.F90 \) module (\( mpp\_lbc\_north\_3d \) and \( mpp\_lbc\_north\_2d \)).

MPI\_ALLGATHER gathers data from all tasks and distribute it to all.

![Figure 1.1: MPI\_ALLGATHER task description](http://rc.usf.edu/tutorials/classes/tutorial/mpi/chapter8.html)

If we call \( zloc\_P_i \) the data from task \( (i) \) and the \( zglo \) the result of MPI\_ALLGATHER and \( zloc\_adj\_P_i, zglo\_adj \) their adjoint equivalent.

The adjoint counter-part of

\[
\text{MPI\_ALLGATHER}(zglo, zloc)
\]

is

MPI_SUM(zglo_adj)

\[ \text{zloc}\_\text{P1} = \text{zglo}\_\text{adj}(:,i) \]

As short demonstration, we consider a two-task case \( P_1 \) and \( P_2 \).

\text{MPI\_ALLGATHER(zglo, zloc)} is equivalent to:

\begin{verbatim}
for \( P_1 \):

(1) send and receive local data:
send(zloc1\_P1, P1, P2) ---> zloc1\_P2 = zloc1\_P1
recv(zloc2\_P1, P2, P1) ---> zloc2\_P1 = zloc2\_P2

( zlocK\_Pj : local data K of j task)

B- zglo operation
zglo\_P1(:,1) = zloc1\_P1
zglo\_P1(:,2) = zloc2\_P1

we then write the adjoint as follows:
\end{verbatim}

\text{zloc2\_P1*} = zloc2\_P1* + zglo\_P1*(:,2)
\text{zglo\_P1*}(,2) = 0
\text{zloc1\_P1*} = zloc1\_P1* + zglo\_P1*(:,1)
zglo\_P1*(:,1) = 0

\text{zglo\_tot*} = zglo\_P1* + zglo\_P2* (i.e mpp\_sum\_nfd)

(! because zloc\_P1* is not null here, thus: zloc1\_P1* = zloc\_P1* + zloc1\_P2*)

\text{zloc1\_P1*} = zglo\_tot*(:,1)
\text{zloc2\_P1*} = zglo\_tot*(:,2)
Chapter 2

NEMOTAM User’s Guide

2.1 Source code structuration

The project will work with three repositories (see Fig. 2.1b,c,d):

- a NEMOBASE repository, with an official NEMO released version currently the tag nemo_v3, available at: https://forge.ipsl.jussieu.fr/nemo/browser/tags/nemo_v3
- a NEMOVAR repository, with all developments related to the nemovar project, including new and modified files for NEMO, this is the standard version of NEMOVAR (currently tag N3.0-A2.0)
- a VODA repository including new development for NEMOTAM and NEMOVAR within the VODA project

![ GIT tree ](a) NEMOBASE (b) NEMOVAR (c) VODA

Figure 2.1: Code organization
The building process will look for both repositories, if any redundancy is found, a VODA files will overwrite the corresponding NEMOVAR file, and a NEMOVAR file will overwrite the corresponding NEMOBASE file in the building folder.

The NEMOBASE repository (see Fig. 2.1b) is the standard NEMO folder. The NEMOVAR repository (see Fig. 2.1c) is subdivided into 6 components: NEMO, NEMOQC, NEMOTAM, NEMOASSIM, UTIL and TOOL. The NEMOTAM folder is a mimic of the NEMO folder (meaning we have an OPATAM_SRC folder subdivided into components as ASM, DOM, DYN, LDF, TRA, ...). The VODA folder (see Fig. 2.1d) mimic the NEMOVAR folder, with additional contributions. This organization gives a clear snapshot of what is being developped and lets some room for further development.

### 2.2 Getting started tutorial

#### 2.2.1 Installing the code using the NEMOVAR GIT repository

**Checking out the code from the GIT repository**

You need to ask the VODA project team to have developer access to the VODA project on the gforge.inria.fr server and access to the NEMO - IPSL. You can then follow the steps described below.

- Create first a new directory, where the source code will be installed. For instance,
  ```bash
cd $VODA_GIT (=HOME/VODA-PROJECT/VODA_GIT e.g.)
```
- Install NEMOBASE
  ```bash
go to https://forge.ipsl.jussieu.fr/nemo/browser/tags/nemo_v3 (authentication required)
  and click on ‘zip archive’ at the bottom of the page.
  put this archive in $VODA_GIT, unzip it and rename the obtained directory NEMOBASE (this name is mandatory)
```  
- Install NEMOVAR
  ```bash
  Download the following bundles, located on the gforge server under the "files" tab:
  NEMOVAR: N3.0-A2.0 => nemovar_N3.0-A2.0.bundle
  and put them in the $VODA_GIT/BUNDLES directory.
  ```

In $VODA_GIT, you now have:

```bash
tree BUNDLES/
BUNDLES/
|-- NEMOBASE.bundle.20090902
`-- nemovar_N3.0-A2.0.bundle
```

Clone these bundles:
```
  git clone BUNDLES/NEMOBASE.bundle.20090902 NEMOBASE/
  git clone BUNDLES/nemovar_N3.0-A2.0.bundle NEMOVAR/
```
Building the code (in default mode: ORCA2 validation interface)

NEMOTAM makes use of the fcm based NEMOVAR compiling environment, a simple call to VODA/UTIL/build/fcmvmake.ksh with the usual arguments and the nemoall keyword should do the trick.

```
> fcmvmake.ksh -c $COMPILE -t $WORKDIR -B nemoall
```

the above requires that you create a VODA/UTIL/build/fcmconfig/bld/$COMPILE/nemo.cfg describing your compilation environment (examples are given in the same directory).

How to run the code is described in section 2.3.

2.2.2 Installing the code using the NEMOTAM SVN repository

Checking out the code from the SVN repository

NEMOTAM can be extracted using svn like NEMO. People should first to register on NEMO website www.nemo-ocean.eu. Then NEMOTAM is available through modipsl framework, so modipsl has to be extracted first.
Extract modipsl:

```
svn co http://forge.ipsl.jussieu.fr/igcmg/svn/modipsl/trunk modipsl
```

Extract NEMOTAM:

```
cd modipsl/util
./model NEMOTAM
```

**Building the code (in default mode : ORCA2 validation interface)**

Choose the configuration to build. You can edit the script fait_config.tam to add a new one.

```
cd modipsl/modeles/UTIL
./fait_config_tam ORCA2_TAM
```

You can build the direct model in the same way:

```
cd modipsl/modeles/UTIL
./fait_config ORCA2_LIM
```

To create the Makefiles, edit first modipsl/util/AA_make.gdef to find or add your compiler, then:

```
cd modipsl/util
./ins_make
```

To compile:

```
cd modipsl/modeles/NEMOTAM/WORK
gmake
```

### 2.2.3 Which executable for which test?

To run a bunch of tests to verify your configuration you need to compile several executable according to your test target. The graph 2.2 give an overview of the relationship between the executable and the available tests.

**Running the code**

### 2.3 Demonstrators

#### 2.3.1 The ORCA2 global configuration

ORCA is the generic name given to global ocean configurations. Its specificity lies on the horizontal curvilinear mesh used to overcome the North Pole singularity found for geographical meshes. ORCA is based on a 2 degrees Mercator mesh, (i.e variation of meridian scale factor as cosine of the latitude). In the northern hemisphere the mesh has two poles so that the ratio of anisotropy is nearly one everywhere. The vertical domain spreads from the surface to a depth of 5000m. There are 31 levels, with 10 levels in the to 100m. The time step depends on the resolution. It is 1h36’ for ORCA2 so that there is 15 time steps in one day.
VODA: ANR-08-COSI-016

Figure 2.2: Relationships between executable files (first column) and available tests (second column). The third column focuses on the main namelist’s parameters (’namtst’ for adjoint, gradient, tangent tests, ’namobs’ for single obs.experiment and ’namhlt’ for linear tangent hypothesis)
How to compile the ORCA2 configuration?

You must first follow the steps described in 2.2.1. The compilation process described below then relies upon the FCM (Flexible Configuration Manager) toolbox, which is included in VODA tree, see e.g. the following directory:

```
ls $VODA_GIT/VODA/UTIL/fcm/bin
```

Then, you may proceed in the following manner:

- Get sample input data, needed for the model to run (http://ljk.imag.fr/membres/Arthur.Vidard/docs/nemov3_testdata.tar.gz):
- Customize the first configuration file `nemo.cfg`, which is located according to your platform architecture (mac_intel here):
  ```
  vi $VODA_GIT/VODA/UTIL/build/fcmconfig/bld/mac_intel/nemo.cfg
  ```
- Customize the second configuration file, with CPP keys: `cppkeys.ORCA2_Z31.cfg`. A sample file is ready to be used, which you can modify:
  ```
  vi $VODA_GIT/VODA/UTIL/build/fcmconfig/nemo/cppkeys.ORCA2_Z31.cfg
  ```
- You are now ready to compile the code. You need to create a "scratch" directory (for example `SCRATCH_ORCA2` below), where compilation and execution will be performed:
  ```
  cd $VODA_GIT/VODA/UTIL/build
  ./fcmvmake.ksh -t $SCRATCH_ORCA2 -c mac_intel -B NEMO -G ORCA2_Z31
  ./fcmvmake.ksh -t $SCRATCH_ORCA2 -c mac_intel -B NEMOTAM -G ORCA2_Z31
  ```

How to run the ORCA2 configuration?

A sample minimal script is given, just as an example: The **direct** model NEMO

```
$VODA_GIT/VODA/UTIL/scripts/runnemo.ksh -t $SCRATCH_ORCA2 -A mac_intel -v
NEMO -g ORCA2_Z31
```

The **adjoint** model NEMO tests

```
$VODA_GIT/VODA/UTIL/scripts/runnemo.ksh -t $SCRATCH_ORCA2 -A mac_intel -j $TRAJ
-v NEMOTAM -g ORCA2_Z31
```

where `$TRAJ` is the reference trajectory directory. This script absolutely needs to be customized with respect to your working environment. Please have a look at this script before proceeding further. You will first need to create a reference trajectory by running only the **direct** model NEMO, and you then may want to use NEMOTAM to
- launch the generic TAM test interface,
- investigate the time span over which the tangent linear hypothesis remains valid,
- etc.

Testing the tangent model and the tangent linear hypothesis

As discussed previously, the aims of the test is to illustrate the correctness of the tangent and how we have dealt with the non-linearity (if it exits) of the direct model.
Figure 2.3: Tangent test type: Four kinds of test results. Top left (a), linear routine with a residual following the error machine. Top right (b), non-linear routine with a residual decreasing as p-order of 2. Bottom left (c), linear routine with a residual decreasing as p-order of 1. Bottom right (d), non-linear routine with a residual decreasing as p-order of 1. (Legend: Solid line is $N_p = M(x + p\delta x) - M(x)$, circle symbol $L_p = L(p\delta x)$, triangle symbol $N_p - L_p$)
Discussion

We have divided the results into four categories (a), (b), (c) and (d) for analysis (see table 2.1). Category (a), illustrated by solsor\_tan in figure 2.3, gathers tangent-linear routines behaving as expected w.r.t to their direct routine (also linear) counter-part: the computed residual is close to the error machine \( \frac{L_p}{N_p - L_p} \) is around \( 10^{15} \). Category (b), illustrated by eos\_insitu\_tan in figure 2.3, gathers tangent-linear routines behaving as expected w.r.t to their direct routine (not linear) counter-part: the computed residual is equivalent to a \( O(p^2) \). Categories (c) and (d), illustrated by dynspg\_flt\_tan and tra\_adv\_cen2\_tan in figure 2.3, gather suspicious cases we need to validate (or invalidate) through the choice made for the tangent-linear. Case (c) is for direct linear routine and (d) direct non-linear routine.

<table>
<thead>
<tr>
<th>Routine (L)</th>
<th>Category class</th>
</tr>
</thead>
<tbody>
<tr>
<td>solsor_tan</td>
<td>a</td>
</tr>
<tr>
<td>dynadv_tan</td>
<td>b</td>
</tr>
<tr>
<td>dynhpg_tan</td>
<td>a</td>
</tr>
<tr>
<td>traldf_lap_tan</td>
<td>b</td>
</tr>
<tr>
<td>trazdf_imp_tan</td>
<td>b or c ?</td>
</tr>
<tr>
<td>tra_zpshde_tan</td>
<td>b</td>
</tr>
<tr>
<td>trasbc_tan</td>
<td>b</td>
</tr>
<tr>
<td>bn2_tan</td>
<td>b</td>
</tr>
<tr>
<td>eos_lpt_tan</td>
<td>b</td>
</tr>
<tr>
<td>eos_insitu2d_tan</td>
<td>b</td>
</tr>
<tr>
<td>eos_insp_tan</td>
<td>b</td>
</tr>
<tr>
<td>eos_insitu_tan</td>
<td>b</td>
</tr>
<tr>
<td>dynspg_flt_tan</td>
<td>c</td>
</tr>
<tr>
<td>traadv_cen2_flt_tan</td>
<td>d</td>
</tr>
</tbody>
</table>

Table 2.1: ORCA2 Results of Tangent tests for sub-routines

The case of traadv\_cen2\_flt\_tan is fully understood as an approximation was done on the numerical scheme (the tangent linear does not include the upstream component of the numerical scheme). Below a certain threshold the upstream part (which is a \( O(p) \) in \( N_p \)) is no longer small compared to the second order scheme and thus, \( L_p \).

Testing the adjoint model

The adjoint test for the ORCA2 configuration has proven satisfactory for the unitary routine tests (see table 2.2) and the global test (see tables 2.3 and 2.4. For the latter, the integration time tested varied from 1 to 15 days.
<table>
<thead>
<tr>
<th>Routine (L)</th>
<th>$(L\delta x)^\top \cdot Wdy$</th>
<th>$\delta dx^\top \cdot L^\top W\delta y$</th>
<th>Rel. Err.</th>
<th>Mach. eps.</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>sol_sor_adj</td>
<td>.525406844734452E+15</td>
<td>.525406844734452E+15</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 1</td>
<td>.711167836790924E+12</td>
<td>.711167836790924E+12</td>
<td>.7E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 2</td>
<td>.711158686312387E+12</td>
<td>.711158686312387E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 3</td>
<td>.721405227628168E+12</td>
<td>.721405227628168E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 1</td>
<td>.711167836790924E+12</td>
<td>.711167836790924E+12</td>
<td>.7E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 2</td>
<td>.711158686312387E+12</td>
<td>.711158686312387E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 3</td>
<td>.721405227628168E+12</td>
<td>.721405227628168E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 2</td>
<td>.711167836790924E+12</td>
<td>.711167836790924E+12</td>
<td>.7E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 3</td>
<td>.711158686312387E+12</td>
<td>.711158686312387E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 1</td>
<td>.711167836790924E+12</td>
<td>.711167836790924E+12</td>
<td>.7E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 2</td>
<td>.711158686312387E+12</td>
<td>.711158686312387E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 3</td>
<td>.721405227628168E+12</td>
<td>.721405227628168E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 1</td>
<td>.711167836790924E+12</td>
<td>.711167836790924E+12</td>
<td>.7E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 2</td>
<td>.711158686312387E+12</td>
<td>.711158686312387E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 3</td>
<td>.721405227628168E+12</td>
<td>.721405227628168E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 2</td>
<td>.711167836790924E+12</td>
<td>.711167836790924E+12</td>
<td>.7E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 3</td>
<td>.711158686312387E+12</td>
<td>.711158686312387E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 1</td>
<td>.711167836790924E+12</td>
<td>.711167836790924E+12</td>
<td>.7E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 2</td>
<td>.711158686312387E+12</td>
<td>.711158686312387E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>sbcfwb_adj 3</td>
<td>.721405227628168E+12</td>
<td>.721405227628168E+12</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
</tbody>
</table>

Table 2.2: ORCA2 Results of the Adjoint tests for the main sub-routines
Table 2.3: ORCA2 Results of the Adjoint tests for the main routine step_tam for a run-window of one day. From line one to four, each control parameter is perturbed individually. Column six, all control parameter are perturbed.

<table>
<thead>
<tr>
<th>Routine (L)</th>
<th>$(L\delta x)^\top \cdot W_{dy}$</th>
<th>$\delta x^\top \cdot L^\top W \delta y$</th>
<th>Rel. Err.</th>
<th>Mach. eps.</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>step_adj U</td>
<td>.161275281471349E+18</td>
<td>.161275281471349E+18</td>
<td>.6E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj V</td>
<td>.318648865958325E+18</td>
<td>.318648865958325E+18</td>
<td>.4E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj T</td>
<td>.132120109179127E+19</td>
<td>.132120109179127E+19</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj S</td>
<td>.28136283249564E+18</td>
<td>.28136283249564E+18</td>
<td>.0E+00</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj SSH</td>
<td>.509250348807475E+10</td>
<td>.509250348807474E+10</td>
<td>.2E-14</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj</td>
<td>.206971044341451E+19</td>
<td>.206971044341451E+19</td>
<td>.1E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
</tbody>
</table>

Table 2.4: ORCA2 Results of the Adjoint tests for the main routine step_tam for a run-window of five days. From line one to four, each control parameter is perturbed individually. Column six, all control parameter are perturbed.

<table>
<thead>
<tr>
<th>Routine (L)</th>
<th>$(L\delta x)^\top \cdot W_{dy}$</th>
<th>$\delta x^\top \cdot L^\top W \delta y$</th>
<th>Rel. Err.</th>
<th>Mach. eps.</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>step_adj U</td>
<td>.102643067009108E+18</td>
<td>.102643067009108E+18</td>
<td>.2E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj V</td>
<td>.16782730282384E+18</td>
<td>.16782730282384E+18</td>
<td>.4E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj T</td>
<td>.893406215846413E+18</td>
<td>.893406215846413E+18</td>
<td>.7E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj S</td>
<td>.220114453479612E+18</td>
<td>.220114453479612E+18</td>
<td>.4E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj SSH</td>
<td>.265003707314821E+10</td>
<td>.265003707314821E+10</td>
<td>.4E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
<tr>
<td>step_adj</td>
<td>.137645166426475E+19</td>
<td>.137645166426475E+19</td>
<td>.6E-15</td>
<td>.2E-14</td>
<td>ok</td>
</tr>
</tbody>
</table>

2.3.2 The GYRE regional configuration

GYRE is an idealized configuration representing double gyres in the Northern hemisphere. It is $\beta$-plane with a regular grid spacing at 1 degree horizontal resolution. There is 31 verticles levels. The configuration is forced with analytical heat, freswater and wind-stress fields.

How to compile the GYRE configuration?
Similar to ORCA2 configuration replacing ORCA2_Z31 by GYRE_Z31.

How to run the GYRE configuration?
Similar to ORCA2 configuration replacing ORCA2_Z31 by GYRE_Z31.

Tangent-Linear Hypothesis (HLT) validity

To illustrate the tangent linear hypothesis validity test We use an initial state $x_0$ ten years from spin-up and an increment $\delta x$ computed from an previous assimilation with $(|\delta x_T| \leq 1, |\delta x_S| \leq 0.1$, $|\delta x_U| \leq 0.0022$ and $|\delta x_V| \leq 0.0019$). see figure 2.4.

here: place more stats on $\delta x$ (mean, min, max, std)?
here: add comments for the graph
Figure 2.4: **Tangent linear hypothesis validity for GYRE.Z31**: plot in function of time-window in days over 1 year. Top graphs are RMS of T, S, U and V variables in S.I units. Bottom graphs are related coefficient correlation.

### 2.3.3 The POMME configuration (toy model for handling open boundaries)

#### Description of the configuration

NEMO is distributed with some reference configurations allowing the users to set up a first application, and the developers to validate their developments. POMME is a small square box configuration \((30 \times 40 \times 46)\) grid points), extracted from NEMO DRAKKAR / ORCA025. All boundaries are open ocean boundaries (including the corners). This configuration was set up to facilitate developments and tests for regional configurations (however, it is not yet an official NEMO reference configuration). It should be soon included in the NVTK validation toolkit.

#### How to compile the POMME configuration?

You must first follow the steps described in 2.2.1. The compilation process described below then relies upon the FCM (Flexible Configuration Manager) toolbox, which is included in VODA tree, see e.g. the following directory:

```
ls $VODA_GIT/VODA/UTIL/fcm/bin
```

Then, you may proceed in the following manner:

- Get sample input data, needed for the model to run:

  ```
  wget http://www.locean-ipsl.upmc.fr/~cdlod/data/data_POMME_Z46.tar.gz
  ```

Customize the first configuration file `nemo.cfg`, which is located according to your platform architecture (mac_intel here):

```bash
vi $VODA_GIT/VODA/UTIL/build/fcmconfig/bld/mac_intel/nemo.cfg
```

Customize the second configuration file, with CPP keys: `cppkeys.POMME_Z46.cfg`. A sample file is ready to be used, which you can modify:

```bash
vi $VODA_GIT/VODA/UTIL/build/fcmconfig/nemo/cppkeys.POMME_Z46.cfg
```

You are now ready to compile the code. You need to create a ”scratch” directory (for example `SCRATCH_POMME` below), where compilation and execution will be performed:

```bash
cd $VODA_GIT/VODA/UTIL/build
./fcmvmake.ksh -t $SCRATCH_POMME -c mac_intel -B NEMO -G POMME_Z46
./fcmvmake.ksh -t $SCRATCH_POMME -c mac_intel -B NEMOTAM -G POMME_Z46
```

**How to run the POMME configuration?**

A sample minimal script is given, just as an example:

```bash
vi $VODA_GIT/VODA/UTIL/scripts/run_pomme.ksh
```

This script absolutely needs to be customized with respect to your working environment. Please have a look at this script before proceeding further. You will first need to create a reference trajectory by running only the direct model NEMO, and you then may want to use NEMOTAM to

- launch the generic TAM test interface,
- investigate the time span over which the tangent linear hypothesis remains valid,
- etc.

**Testing the tangent model and the tangent linear hypothesis**

*coming soon*

**Testing the adjoint model**

*coming soon*

### 2.4 How-to update NEMOTAM with your contribution?

#### 2.4.1 Coding rules for TAM

NEMOTAM modules should follow the NEMO convention. However additional rules has to be followed:

- For a given direct module `mod.F90 (that contains the subroutine sub)`, a corresponding module `mod_tam.F90` has to be created and it should contain the tangent linear subroutine called `sub_tan`, the adjoint subroutine `sub_adj` and the corresponding testing routine `sub_adj_tst`. 

• For a given active direct variable var, the corresponding tangent linear and adjoint variable should be named var_tl and var_ad respectively. The underscore is omitted for local active variable. To summarized, active variables un, pun, zun will give un_tl, pun_tl and zun_tl in the tangent module.

• For historical reasons, some modules do not follow these rules, feel free to update them.

If the TAM module is a direct translation of the corresponding OPATAM routine, a good practice is to account for it in the first line of the history (i.i stating that the 8.2 version was done by e.g Huey, Dewey, and Louie).

2.4.2 Workflow for introducing additional developments

coming soon
2.5 Frequently asked questions

2.5.1 I can’t use GDB after compiling with FCM

When I run the code with GDB for debugging purposes, the library lib_fcm_nemotam.a is not found, and thus gdb does not have access to symbolic informations associated with the modules of the code: we can’t set breakpoints, we can’t look for the content of variables, etc. This library is a temporary one generated by FCM during the build process, but it is removed once the compilation is finished.

This problem only appears on Mac OSX’s computers. For unknown reasons, the linker looses track when going from fcm’s temporary libraries to the final libraries. The temporary workaround is to recreate these temporary libraries from the object files (but you will have to redo it each time you compile):

```bash
cd $SCRATCH_POMME/fcmbuild/POMME_Z46_mac_intel.1.1/build/lib/
ar -q lib_fcm_nemotam.a ../obj/*.o
```

2.5.2 I have many commas in the namelist generated by piano

This happens if your config namelist have lines ending with a comma. This feature is accepted by FORTRAN, but the PIANO python script parser do not handle this. Removing the commas at the end of each line will solve the problem.

2.5.3 I have negative values in my cost function terms

See congrad.F90:

```fortran
Compute cost for diagnostic purposes only
pf=zf0+0.5*DOT_PRODUCT(...)
```

It is this term in congrad which appears to be < 0 and is written to the cost file, whereas we would expect to write the pcost from simvar.F90. Under investigation

2.5.4 My run stops with 'PROBLEM WITH THE MATRIX OR WITH THE PRECONDITIONER' on stderr

This problem was reported in the POMME and GYRE configurations.

First, it is worth noting that this message disappears if you set

```fortran
ln_tst_grad = .TRUE.
```

which is absolutely not a normal behaviour. Needs further investigation.

The problem is solved by increasing the namelist nmin parameter. For example, change

```fortran
nmin = 300 ! minimum of iterations for the SOR solver
```

into:

```fortran
nmin = 500 ! minimum of iterations for the SOR solver
```
in namsol. This behaviour is not normal, it suggests a convergence problem in the tangent barotropic solver. *Needs further investigation.*
List of Figures

1.1 MPI_ALLGATHER task description (ref: http://rc.usf.edu/tutorials/classes/tutorial/mpi/chapter8.html) ........................................ 21

2.1 Code organization ........................................ 23

2.2 Relationships between executable files (first column) and available tests (second column). The third column focuses on the main namelist’s parameters (’namtst’ for adjoint, gradient, tangent tests, ’namobs’ for single obs.experiment and ’namhlt’ for linear tangent hypothesis) ........................................ 27

2.3 Tangent test type: Four kinds of test results. Top left (a), linear routine with a residual following the error machine. Top right (b), non-linear routine with a residual decreasing as p-order of 2. Bottom left (c), linear routine with a residual decreasing as p-order of 1. Bottom right (d), non-linear routine with a residual decreasing as p-order of 1. (Legend: Solid line is $N_p = M(x + p\delta x) - M(x)$, circle symbol $L_p = L(p\delta x)$, triangle symbol $N_p - L_p$) ........................................ 29

2.4 Tangent linear hypothesis validity for GYRE_Z31: plot in function of time-window in days over 1 year. Top graphs are RMS of T, S, U and V variables in S.I units. Bottom graphs are related coefficient correlation. ........................................ 33
List of Tables

1.1 Fields stored in the reference trajectory ................................................. 9
1.2 List of modified declarations in the direct code NEMO ............................. 10
1.3 Hand coding status of NEMOTAM, Dynamics part ................................. 12
1.4 Hand coding status of NEMOTAM, Tracers and Surface Boundary Condition part . 13
1.5 Available option in NEMOTAM, part1 .................................................. 14
1.6 Available option in NEMOTAM, part2 .................................................. 15
1.7 Adjoint test labels definition ............................................................... 17
1.8 Tangent Linear tests. First-order check (a) and second-order check (b) ......... 17
1.9 ORCA2 Results of Adjoint test fo sub-routines ...................................... 20

2.1 ORCA2 Results of Tangent tests for sub-routines .................................. 30
2.2 ORCA2 Results of the Adjoint tests for the main sub-routines .................. 31
2.3 ORCA2 Results of the Adjoint tests for the main routine step_tam for a run-window of one day. From line one to four, each control parameter is perturbed individually. Column six, all control parameter are perturbed. .............................. 32
2.4 ORCA2 Results of the Adjoint tests for the main routine step_tam for a run-window of five days. From line one to four, each control parameter is perturbed individually. Column six, all control parameter are perturbed. .............................. 32
appendix : acronyms

- AD : Automatic differentiation
- AGRIF : Adaptive Grid Refinement In Fortran. Outil de raffinement de maillage et de nesting développé au sein de MOISE et inclus dans NEMO.
- ESOPA : Equipe Système OPA, qui gérait les développements de OPA. A été remplacée par la NEMO-Team après la mise en place du consortium NEMO
- MITgcm : Modèle de circulation général pour l’océan et l’atmosphère développé au MIT
- NEMO : Nucleus for European Modelling of the Ocean.
- NEMOTAM : TAM pour NEMO
- NEMO-Team : Equipe gérant les développements de NEMO
- NEMOVAR : système d’assimilation variationnelle de données pour NEMO
- OPA : Ocean Parallélisé. Le modèle direct d’océan, composant de NEMO.
- OPATAM : TAM pour OPA 8.x
- OPAVAR : système d’assimilation variationnelle de données pour OPA 8.x
- ORCA : grille globale pour OPA
- PONGO : Pôle d’Océan Numérique GrenOblois, regroupant les équipes MOISE (LJK/INRIA) et MEOM (LEGI)
- ROMS : Regional Ocean Modelling System
- TAM : Modèles Tangent et Adjoint
- TAPENADE : Générateur automatique de code tangent et adjoint développé par le projet TROPICS de l’INRIA
Bibliography


Tber, M. H., Hascoet, L., Vidard, A. & Dauvergne, B. (2007), Building the Tangent and Adjoint
codes of the Ocean General Circulation Model OPA with the Automatic Differentiation tool
TAPENADE, Research Report RR-6372, INRIA.
URL: http://hal.inria.fr/inria-00192415/en/


132, 2070–2083.

131, 3605–3625.

variational assimilation with a general circulation model of the tropical pacific ocean, Technical
memorandum 365, ECMWF.

assimilation with a general circulation model of the tropical pacific ocean. part i: Formulation,
internal diagnostics, and consistency checks.’, Mon. Weather Rev. 131(7), 1360–1378.