The ARPEGE/ALADIN Tech'Book: Implications of LAM aspects on the global model code

CY32/AL32

originally written by Mark Žagar

ever since updated by Claude Fischer

with contributions by Gabor Radnóti, Martin Janoušek, Alena Trojakova, Yann Seity

Toulouse, December 2006

Contents

1 Caution

Dear reader, possibly phaser, before going further through this document, the Author would like to stress that this paper indeed aims at giving guidelines both for the practical understanding, and the detailed knowledge of the bi-Fourier limited area "patches" inside the glorious Arpge/IFS code. However, the paper is not for the time being a bible pretending to indicate all the possible issues one can encounter when phasing the Aladin software. Therefore, nobody should restrict his/her knowledge and analysis only to those points listed in this document. For example, the routines listed in Appendix D are NOT the only ones to be checked during phasing, nor are they the only ones present in the Aladin library. This note focuses strongly on the features that are common to Arpge and Aladin in terms of functionality, but different in terms of actual coding. Thus, any change there, if only performed in the Arpge version, can lead to a bug in the Aladin one. Obviously, as this note grows with time, specific Aladin features tend to appear, like non-hydrostatic or coupling. They are presented since it is interesting to know about their interfacing or intertwinement with Arpge.

Enjoy your reading.

2 Introduction, general principles

This document points out and explains, from the scientific and technical point of view, the less obvious particularities of the local version of the ARPEGE/IFS model, named ALADIN. It is intended to help the model code developers of the global version to avoid possible undesirable consequences of their developments and modifications to ALADIN. It can also serve as one of the basic documents for phasers, and even newcomers to the project are not excluded from the reading community.

The differences may be grouped into two groups, both due to the special geometrical demands of ALADIN when compared to ARPEGE/IFS.

2.1 Spectral space

We always have to bear in mind two things:

1. Choice of the basic functions of the spectral representation, which is *bi-Fourier* $(Re, Im) \cdot (Re, Im)$ in ALADIN and *spherical harmonics* $(Re, Im)_{\text{zonal}} + L$ in ARPEGE/IFS.
2. Choice of the truncation: a *triangular* one (fig. 1a) in ARPEGE/IFS with 2 real numbers defining a component and an *elliptical* one (fig. 1b) in ALADIN with 4 real numbers defining a component plus the mean wind (the 0-th component in a series), which does not exist on a sphere.

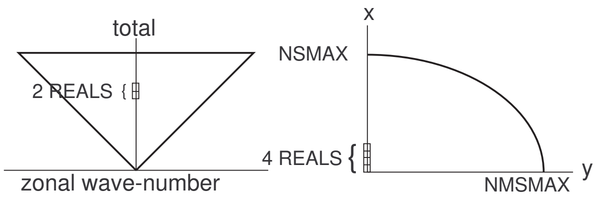

Figure 1: Schematic representation of a triangular (left) and elliptical truncation (right).

2.2 Grid-point space

Differences are:

1. Existence of lateral boundaries.
2. Necessity of the extension zone for biperiodizing the fields (see fig. 2).

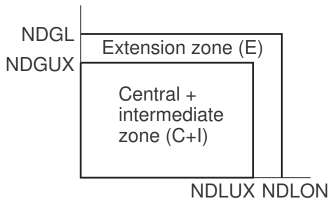

Figure 2: Schematic representation of the extension zone in ALADIN, and a few important definitions.

3. Projection onto a plane (Lambert, Mercator, rotated Mercator or polar-stereographic), causing in turn:

   - map factor $M \neq 1$, (Fig. 3)

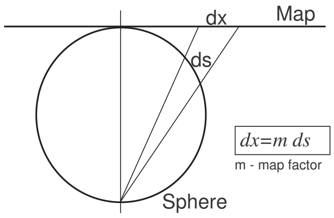

Figure 3: Polar-stereographic projection and definition of a map-factor.

   - y-axis is not really the geographical north for vector variables.

4. curvature terms $(\partial m/\partial x, \partial m/\partial y)$ `RATATX, RATATH`. This is common with stretched ARPEGE/IFS but with another definition: `RATATX` $\neq$ `0`.

2.3 Integration

Here we encounter two big differences, due to the fact that ALADIN is a limited area model:

1. Coupling with a large-scale model.
2. Consideration of boundaries for horizontal interpolations (in SL scheme).

Furthermore, we recall here briefly the configurations for the non-hydrostatic (NH) model, which now also exist in the global model. They are based on a corrective iteration for the NH semi-implicit scheme (in both Eulerian and Semi-Lagrangian cases). Two types of predictor/corrector schemes are presently implemented in the code:

1. the non-hydrostatic P/C scheme by J. Vivoda (keys LPC_FULL, LPC_NESC),
2. the non-hydrostatic iterating scheme by R. Bubnova (key LPC_OLD). The Bubnova solution only uses a small corrective step (keys LNHDYN and LPC_OLD and NSITER>0). For further details, we refer to Bubnova *etal.*, *Integration of the fully elastic equations cast in the hydrostatic pressure terrain-following coordinate in the framework of the ARPEGE-ALADIN NWP system*, MWR, volume 123, pp. 515-535, 1995.

Further down in this document, all the mentioned features (and more) will be presented in details.

3 Duplicated routines

There is no more any duplicated routine present in both ARPEGE/IFS and ALADIN model code, having the same name and the same function, but not the same source. Other duplicated routines existed in the ALADIN specific transform package `tal` in cycles AL15 and AL25T1-2. Since AL26T1, these duplicata have however been removed (`tfl` versions are used instead). Their principles and rules are described in the spectral transform section.

3.1 Technical point of view

The reasons for having duplicated routines in the past were:

- NO use of the `LELAM` key below the level of `SCAN2M` following an agreement with ECMWF. Thus, specific LAM calculations are performed under the `LRPLANE` key.

- a self-imposed splitting for the sake of security between model libraries (control level routines)

Note that until AL14, there were many duplicated routines such as `stepo(tl,ad).F90, scan2m(dm,tl,ad).F90`, that have been merged with ARPEGE in AL15. Later, in cycle CY28T1, five more control routines have been merged: `su0yom(a,b).F90, cnt3.F90, cnt4(tl,ad).F90`. Finally, `cnt4.F90` was merged for CY29. The merge however implies that at each phasing, a *careful inspection* of these routines has to be performed in the ARPEGE library, for ALADIN maintenance. Moreover, any change in these routines must be done in such a way not to harm neither ARPEGE/IFS, nor ALADIN. In particular, aladinists bear a bigger responsability for introducing ARPEGE-compliant modifications in `stepo` or `scan2mdm`.

The rule for control-level routines is that ARPEGE/IFS must be able to run with ALADIN libraries. To achieve this, the ARPEGE control routines contain all the original ARPEGE/IFS code, activated by `LELAM=.F.`, plus additional code for `LELAM=.T.`. During phasing, one has to ensure that new ARPEGE/IFS code is taken into account in the ALADIN section of code, without doing any harm to ALADIN, and that the ALADIN specificities remain untouched.

3.2 Scientific point of view

Scientifically, the reasons for the *past* duplicated routines may be grouped under the topics of:

- extra reading of coupling files: `cnt3.F90`
- coupling, bi-Fourier spectral representation: `cnt4(tl,ad).F90, cnt4.F90`
- call ALADIN transforms instead of ARPEGE ones: `stepo(tl,ad).F90, scan2m(dm,tl,ad).F90`
- no gridpoint scanning through the E zone: `scan2m(dm,tl,ad).F90`
- NH dynamics (which are fully coded only in ALADIN): for the direct model, as well as for the TL/AD models, the corrective step is treated in `cpg(tl,ad).F90` and `cpglag(tl,ad).F90`, and only minor tests are necessary at the level of the scanning.
- ALADIN canonical injections and spectral manipulations: `opk.F90, opm.F90`

4 Physical (grid-point) space

4.1 Extension zone and coupling

As mentioned before, in ALADIN, periodicity of the fields is assured by extending them into the meteorologically meaningless extension zone. This is done by applying a so-called biperiodization operator - a spline is created in the E-zone for minimizing a laplacian there. Performing any grid-point computations in this zone only increases the computational cost without any useful result. Luckily, Radnoti (95) introduced the changes which allow to partly or totally omit the E-zone from the grid-point computations. The key idea was to perform coupling at the end of the timestep, on the X^+ fields, instead of applying it at the beginning of the timestep, on the X^0 fields. Indeed, the semi-implicit timestepping, with L the SI operator (model linearized around a basic state) and R the non-linear residual, reads:

$$(I - \delta tL).X^+ = (I - \delta tL).X^0 + R(X^0, X^-)$$

If the Davies relaxation is applied at the beginning of the timestep, then one needs to carry $X^0 = \alpha X_i^0 + (1-\alpha)X_c^0$, and all the horizontal derivatives of the coupling data X_c^0 need to be known or computed in order to evaluate $R(X_c^0)$. If the derivatives are re-computed, then this implies gridpoint computations also in the E-zone, to assure biperiodicity. On the contrary, if coupling is applied at the end of the timestep, then one carries $X^+ = \alpha X_i^+ + (1-\alpha)X_c^+$, and only the term $(I - \delta tL).X_c^+$ needs to be evaluated. In gridpoint space, this calculation reduces to vertical derivatives, the horizontal parts are performed in spectral space after the transforms. For more details, we refer to Radnoti, *Comments on - A spectral limited-area formulation with time-dependent boundary conditions applied to the shallow-water equations* -, MWR, volume 123, pp. 3122-3124, 1995.

4.2 Dataflow of coupling

Since AL27, the dataflow has been adapted to Mats Hamrud's new model dataflow structure for gridpoint space. The former buffer GT3BUF has been replaced by a set of new buffers, GT3GMV, GT3GMVS and GT3GFL, that mimic the T0 model buffers GMV, GMVS, GFL (see Hamrud, 2003, or Tudor, 2003, for a detailed explanation of the new dataflow). In particular, the GT3* buffers are accessed with the T0 pointers from their T0 counterparts. Thus, their structure is a replicate of the GMV/GFL structure. An extra attribute has been created for the GFL variables, in order to control whether the associated GFL field has to be coupled or not (YGFL%NCOUPLING : 1 for a coupled GFL, 0 for a non coupled and -1 for coupling with constant value YGFL%REFVALC). YGMV/GMVS variables are supposed to be always coupled.

Coupling data are read by `elsirf.F90` at every new forecast segment, with swapping indexes to control on which half side of the coupling buffers the fields will be stored (thus overwriting some old, useless data). Time linear interpolation and Davies relaxation are performed at each timestep (`elsrw.F90`, `esrlxt1.F90` respectively), immediatly after the semi-implicit forward correction has been applied to the coupling fields (`eseimpls.F90`).

4.3 Extension zone and distributed memory

Furthermore, the E-zone computations also are avoided for the scanning in distributed memory mode. The difference is visible in `scan2mdm.F90` and `gp_model`, where `NGPTOT_CAP` replaces `NGPTOT` in the loop-calls to `CPG`, `CPGLAG` and `COBS`. Note however that GMV* and GFL* arrays absolutely must be set to zero from 1 to `NGPTOT`, and not only to `NGPTOT_CAP`, at the beginning of the adjoint gridpoint calculations (in `scan2mad.F90` and `gp_model_ad.F90`). Otherwise, irrelevant values can be present in the E-zone and lead to absurd spectral fields.

4.4 Semi-Lagrangian scheme

Due to the limited-area character of ALADIN, some modifications in the S-L computations had to be introduced.

4.4.1 Projected geometry

The interpolation grid has to be determined differently in ALADIN. So, if the cartesian geometry is required instead of a spherical one (key `LRPLANE`), then the routine `elascaw.F90` is called (`lascaw.F90` in ARPEGE/IFS). `elascaw.F90` computes the weights for the 3D SL interpolations. It is also called from the observation operator.

In addition, trajectories in ARPEGE/IFS are great circles while in ALADIN the trajectories are straight lines on the plane. Routine `elarmes.F90` is taking care for this part. `elarmes.F90` computes the position of the mid-point of the SL trajectory.

4.4.2 Lateral boundaries

The computation of the origin point of the trajectory in the vicinity of the boundaries causes problems. It happens that the origin point is outside the domain. In such an unfortunate, yet expected, case a special procedure is applied. Figure 4 explains it. This is coded in routine `elarmes.F90`.

Let us mention that, for interpolations and S-L trajectory computations, each chunk of data in a row is extended by one point to the left and two points to the right, by simply copying the data from the last meaningful point (routine `extx.F90`). In Arpge, the extra longitudes allow for cyclic East-West conditions, while in Aladin the neighbouring points are copied.

4.4.3 Treatment of the Coriolis parameter

Routine `elarche.F90` is called from `lapine(a,b).F90` under the `LRPLANE` key to implement the Rochas implicit treatment in the ALADIN geometry (key `LADVF`). In contrast to the treatment of boundaries in case when the true departure is outside of the C+I domain (previous subsection, values of the variables at the border are taken instead), the Coriolis effect is still computed from the position of the true departure.

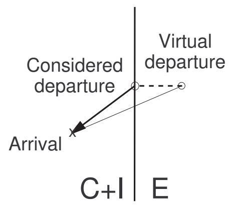

Figure 4: Drawing shows the principle of handling the departure point in case it would be outside of the C+I zone.

`elarche.F90` computes latitude, longitude, rotation matrix coefficients and the Coriolis vector term for a plane geometry. The key `LADVF` switches the Coriolis term from the RHS to the Lagrangian evolution in the momentum equation, so that one considers the SL evolution of:

$$\frac{d}{dt}(\vec{V} + 2\vec{\Omega} \wedge \vec{r})$$

4.5 Full-Pos

Similar problems as in the previous subsection appear. Interpolations near the edge of the domain can be problematic, so a little care must be taken to compute the halos for interpolation. Biperiodisation is treated in Full-Pos as an extrapolation rather than as an interpolation, which also helps having more useful data in the "E" zone since we cannot fully avoid using some data from it.

(a short extension by Ryad about Aladin specific tricks, hints and routines)

5 Spectral transforms

The most obvious consequence of going limited and plane is not using spherical harmonics any more. Instead, the bi-Fourier approach is the logical way to transform physical fields to spectral and back. Imagine that the plane from fig. 2 is first bent to join the left and right side, and then this tube is bent to join the upper and lower edge. A torus is obtained this way, having the property of being periodic in both directions. This is one way how to close the domain. Of course, in reality, a plane of the computational domain is not mapped on a torus, just the E-zone is there to connect the edges.

Before introducing the flow diagram, we have to stress that the transform routines are expanded over 3 different libraries (and clearcase projects): some intermediate routines to the transforms belong to the ALADIN library and they can be found in the `transform` directory. The core of the transforms, however, is external to the model, does have its own setup (especially also its own data distribution) and is located in a specific library, `tal (TA..)`. This transform package, coded by Gabor Radnoti under the general rules of coding defined for ARPEGE/IFS by Mats Hamrud, can be also used outside of the model, in any personal application. Finally, the underlying ARPEGE/IFS transforms are stored in the `tfl` project.

Let's insist on a few principles and properties of the transform package. The ALADIN one `tal` must be added to the ARPEGE/IFS version `tfl` for an ALADIN executable. Thus, the general idea is very simple: the `tfl` package is the ARPEGE/IFS code, the `tal` package is the self-consistent *incremental* ALADIN counterpart. A more detailed description of the organisation of the transform package can be found in Gabor Radnoti's outstanding documentation, available on the ALADIN web site. Note also that a set of ALADIN specific test programs does exist, in order to test the package separately from the model.

The reader should be aware of some general principles for maintenance and development:

- In the cycles AL15, AL25T1 and AL25T2, there existed in the package both twin identical fortran routines and ALADIN counterparts (modified with respect to ARPEGE/IFS). Twin identical routines had the same name in `tfl` and `tal`. These twins have been eliminated from AL26T1 onwards. Thus, the `tal` project does not contain duplicated routines anymore. Routines simply are named with the "e" in first position (`dir_trans` becomes `edir_trans`). For the setup routines, the "e" is added after the "su" prefix (`sump_trans_mod` becomes `suemp_trans_mod`).

- For calling an ALADIN transform routine from the model, care must be taken in the case where the callee is an ARPEGE/IFS routine, and where the transform routine does need a USE MODULE statement because it is embedded in a module procedure (CONTAINS). Indeed, the rule is that *no ALADIN transform module or interface can be declared in an ARPEGE/IFS routine*. Otherwise, one would need to load the `tal` package for ARPEGE compilation, which is voluntarily forbidden. In these cases, an interface routine has to be created in the ALADIN library, for calling the transform code. This (presumably short) routine is called by the ARPEGE code.

- Again, it is stressed that the `tfl` package is needed both for ARPEGE/IFS and ALADIN loading, while the `tal` package is reserved as an increment for the ALADIN loading. Any coding breaking this logic is certainly wrong.

- test programs can be run (simple direct and reverse transforms, vice versa transforms and test of the adjoint). These codes are maintained by GCO.

Since AL27, the input fields to the direct transforms are the GMVT1, GMVST1 and GFLT1 T1 gridpoint buffers (which basically contain all the model updated primitive fields). Output from the inverse transforms are the GMV, GMVS and GFL T0 gridpoint buffers, which also include the required horizontal derivatives. In older cycles, the GPP buffer was mainly used, along with GT1 or GT5 for specific calls.

In the west-east direction, the direct transforms are the same. Until AL25T2, the routine `etransdirh.F90` also linked (after some complicated calling sequence ...) to the T1 coupling so that the entering array, `GPP` was then biperiodic through coupling. Since AL26T1, the coupling code is called directly from `stepo.F90` just before `etransdirh.F90` (call to `ecoupl1.F90`). The flow diagram splashes over both model and transform package libraries, and therefore the project and directories are made explicit here:

```
arp/control/stepo.F90                →
ald/transform/etransdirh.F90         →
ald/transform/etransdir_mdl.F90      →
tal/external/edir_trans.F90          →
tal/module/edir_trans_ctl_mod.F90    →
tal/module/eftdir_ctl_mod.F90        →
(trgtol_mod.F90,ftdir_mod.F90)       →
xrd/support/fft992.F
```

It should be noted that the string of points overdimensioned by three points enters the direct Fourier transforms in the ALADIN case.

In the south-north direction, more special routines are written for ALADIN. The basic idea is still to respect as close as possible the ARPEGE/IFS set-up for Legendre transforms in `stepo.F90`, only of course performing a Fourier transform instead of a Legendre transform in N-S direction. Again, the flow diagram splashes over several projects:

```
arp/control/stepo.F90                →
ald/transform/etransdirh.F90         →
ald/transform/etransdir_mdl.F90      →
tal/external/edir_trans.F90          →
tal/module/edir_trans_ctl_mod.F90    →
tal/module/eltdir_ctl_mod.F90        →
(trltom_mod.F90,eltdir_mod.F90)      →
(.,tal/module/eledir_mod.F90,.       →
xrd/support/fft992.F
```

It is worth to be mentioned that the outcome from the spectral transforms follows a minimum logical order, prescribed by convention and hardcoded: for any primitive field ψ, valid at T0, the horizontal derivatives will be stored just behind it in memory. In addition, the meridional derivative ψ_M will be first, followed by the zonal derivative ψ_L. For the wind components, the logic is different and one has vorticity and divergence first, then U and V, then the zonal derivatives only. For T, q, $\widehat{P}$, $\widehat{d}$, the primitive, meridional, then zonal derivatives are stored in that order. See also the pointer structure TYPE_T0 in `gmv_subs.F90`.

As far as the adjoint spectral transforms are considered, we stress the difference in the metric term used for normalization after inverse `FFT992`. In ARPEGE Fourier transforms, the metric is basically NLOEN (the number of longitudes points on a row of latitude), while in ALADIN Fourier transforms, the metric is 2×NLOEN (eg, compare `tfl/module/ftdirad_mod.F90` with `tal/module/eftdirad_mod.F90`).

All in all, we see a philosophical challenge in the spectral transform organisation: the east-west transforms are Fourier in the global and in the limited case, so the differences are limited, while in the north-south direction the approach is very different. However, the coding is kept as parallel as possible, in order to have similar setups and calling sequences. Thus, for the N-S transforms for example, the setup for the Fourier transforms in ALADIN remains close to the one for ARPEGE Legendre transforms.

Since CY32, a new slicing of grid-point domain has been implemented by ECMWF for B-level parallelisation (under key LEQ_REGIONS in NAMPAR1). Default value is T for ARPEGE-IFS but is set to F in sump0.F90 under LELAM key (As the purpose of this modification was to have a more regular grid-point repartition of spherical global domains, it is not relevant for ALADIN).

6 Set-up

6.1 General comments

Special set-up or initialization routines are written for dealing with different geometrical properties in ALADIN. Some of them are closely related to their ARPEGE/IFS predecessors. In general, the letter "e" is inserted into the name of the routine to distinguish it.

- Grid-point space and geometry:
  - definition of the projection: specific calls to J.D. Gril's geometry package "MAKDO". As a consequence, from cycle AL25T1 onwards, only tangent Lambert, polar stereographic or secant/tangent Mercator projections are allowed. Since AL29T2, rotated Mercator geometry is accessible too. Furthermore, the domain is defined by its centre and half-lengths. The geometry package is stored in the Aladin library (`module/eggpack.F90`, `module/eggangles.F90`). Calls from ARPEGE are performed via interface routines (`utility/eggx_n.F90`, `utility/elalo2xy.F90`). `eggx_n.F90` is called from `echien.F90, sufpd.F90, sufpg2.F90`. `elalo2xy.F90` is called from `obatabs.F90, sufpg2.F90`.
  - set-up of the map-factor is different in ALADIN (routine `suegeo2.F90`), the geometrical specifications being close to those of stretched and tilted ARPEGE/IFS,
  - curvature terms (`suecuv.F90`).
- Spectral space: coefficients for the elliptic truncation have to be defined, and the spectral arrays are different, using 4 real numbers to define a particular wave (routine `suelap.F90`). Parameter RNLGINC, used to set up horizontal diffusion (it is the n_l variable in §8.1.2), only is initialized in Aladin since AL28T3. Its value is either 1 or 0, for a quadratic or linear grid respectively.
- Dynamics:
  - different horizontal diffusion parameters (`suehdf.F90, suedyn.F90`),
  - `suedyn.F90` also checks the interval of update of the boundary conditions.
- Coupling and biperiodization: we define the size of the coupling zone (`NBZONL, NBZONG`) and the size of the C+I zone (`NDLUNG..NDLUXG, NDGUNG..NDGUXG`), so that the E-zone length is obtained as the differences `NDLON-NDLUXG` and `NDGLG-NDGUXG`. `(NDLUXG, NDGUXG)` dimension the inner zone,

absent in ARPEGE/IFS, while the total number of points in ALADIN, including the E-zone, has a similar meaning than in ARPEGE/IFS (`NDLON, NDGLG`). Routine `suegeo1.F90` is called instead of `sugem1a.F90`. Routines for special ALADIN geometry, fitting of map factor, defining relaxation coefficients for coupling, are: `suegeo2.F90, suebig.F90, suebicu.F90`. Although parameters NDGUNG and NDLUNG still exist in the code, it is stressed that their actual value must be 1. The possibility to start the numbering of C+I points from other values (0 or > 1) has not been maintained in practice.

- 3DVAR: a variety of special routines are performing the set-up for variational configurations in ALADIN: `suejbbal.F90`, `suejbbalbeta.F90`, `suejbcor.F90`, `suejbcosu.F90`, `suejbcov.F90`, `suejbdat96.F90`, `suejbstd.F90`, `suejbtest.F90`. Mostly, differences arise from the spectral specificities (4 coefficients, bi-Fourier, ellipse). `suempvar.F90` is a specific routine to define data distribution from [0,NSMAX] in k^* wave space.
- Message passing: the routine `suemp.F90` substitutes `sump.F90`; differences:
  - extension zone, distribution of grid points to processors, splitting of grid in S-N direction (`suezone.F90`),
  - again the 4 versus 2 reals to describe one wave.

  B-level parallelization should work in ALADIN AL28T1, which means that either vertical level or longitude data distribution could be performed.
- Message passing in the transform package "tal" is initialised via `suemp*_mod.F90` routines.
- Control:
  - allocation of space particular to ALADIN is done in `su0yoma.F90` by calling `suello.F90`,
  - and the rest of the ALADIN particular set-up is performed through calls of the above mentioned set-up routines in `su0yomb.F90`.
- Full-Pos: in the ALADIN case, `sufpg.F90` calls `suefpg3.F90` to check if the required domain exists.

6.2 Calling sequence

- `SU0YOMA`:
  - `SUECT0`: initialize coupling parameters (NBICOU, NECRIPL, NECOTL, ...)
  - `SUDIM2`⇒`SUEDIM`: initialize dimensioning parameters: coupling zone (NBZONL, ...), coupling buffer (NG3D3, NGALEF, ...), digital filters
  - `SUELLO`: allocate FFT arrays, spectral bi-Fourier arrays, mean wind, horizontal diffusion arrays, coupling arrays
- `SU0YOMB`:
  - `SUEIOS`: packing parameters for I/O
  - `SUEGEO1`: domain parameters (center, half-lengths, grid mesh), DM-global gridpoint start addresses (NLOENG, ...), fail-save of YOMLEG parameters, MAKDO which issues DM-global gridpoint positions, compas, map factor
  - `SUETRANS`: calls TAL setup
  - `SUEMP`: message passing. This routine is a partial duplicate of the Arpge `SUMP`, so phasing is always required !
  - `SUELLR`: allocate SP7A1
  - `SUEGEO2`: set gridpoint start addresses (NSTAGP, NLOEN, ...), allocate gridpoint geometrical DM-local arrays (some of them will be biperiodicized and spectrally fitted later on)

- SUEVARGP: allocate and set gridpoint mask for σ_b in the C+I zone, specific parameters for bi-Fourier compactly supported structure functions
- SUELAP: set spectral space bi-Fourier addresses and arrays (NCPL4M, NESM0, NESM0G, RLEPINM, ...)
- SUELGES: allocate Aladin specific B-matrix components (SHUM, BFACT)
- SUEFFT: calls SET99 to setup FFT work arrays
- SUEBIG: biperiodisation and spectral fit of gridpoint positions (in radians), Coriolis parameter, map factor (truncated at NTRUN), RSTRET, geographical North vector in (x, y) (compas). DM-global values are then copied into DM-local arrays.
- SUVERT: sets vertical coordinate system: A_k, B_k, ΔB_k, etc ... (Arpge routine, recalled for completeness)
- SUECUV: curvature terms RATATX, RATATH
- SUEBICU: so-called α function used in the Davies relaxation for lateral boundary coupling
- SUEDYN: coupling frequency TEFRCL, inverse Helmholtz equation (semi-implicit vertical operators, but dependent also on the total wavenumber), horizontal diffusion, radiative upper boundary condition ("RUBC")
- SUEOPH: sets names for I/O files, including coupling files
- SUESC2: sets addresses for the coupling buffer and the coupling zone, allocates coupling buffers GMGT3 (map factor equivalent of GM), EALFAGT3 (EALFA), GMVSCPL (GMVS), GMVCPL (GMV), GFLCPL (GFL) and SP3UBC, allocates GPNHBUF for NHS model
- SUELCZ: Lanczos e601, includes a geographical consistency check with respect to C+I for the norm localization
- SUEJBBAL: for B matrix, reads `stabal96.bal` and initializes multivariate operators
- SUEJBCOV: for B matrix, reads `stabal96.cv` and initializes autocovariances for predictors
- SUESCAL: SCALP inner product for bi-Fourier (including NHS)

7 Modules, namelists, interfaces

Because of the features of *fortran90* compiler, all modules have to be fully integrated in ARPEGE/IFS libraries. They are stored under the `./arp/module` directory, and they are all named `yem*.F90` instead of `yom*.F90`. Namelists are under `./arp/ald_inc/namelist/nem*.h`. Routine interfaces which used to be under `./arp/ald_inc/interface/*.h` have been replaced in CY29 by generic interfaces and don't exist anymore.

8 Spectral space computations

Spectral differences impact on various configurations in the forecast part of the model as well as in the variational data assimilation and post-processing.

8.1 Forecasting model

Concerning the integrating part of the model, changes in geometry between ARPEGE/IFS and ALADIN are reflected in the computation of the semi-implicit horizontal operator and of the horizontal diffusion, which are computed in spectral space (ALADIN routine `espcm.F90` calls `espcsi.F90, espnhsi.F90, espchor.F90` instead of `spcm.F90` calling `spcsi.F90, spnhsi.F90, spchor.F90` in ARPEGE/IFS). A similar situation exists in Full-Pos spectral filtering (`espos.F90` for `spos.F90`). The fact that, in ARPEGE/IFS, two routines take care of spectral computations (for only one in ALADIN) is purely due to parallelisation reasons.

8.1.1 Semi-implicit horizontal operator

`espcsi.F90` is split from `espchor.F90` to optimize message passing and data distribution, both in ALADIN and ARPEGE. Namely, contrary to horizontal diffusion, solving the SI Helmholtz equation requires full vertical columns on one processor and thus one more transposition is done.

The treatment of the Coriolis parameter is different, the unstretched and untilted ARPEGE/IFS includes an option for SI treatment while ALADIN doesn't.

Furthermore, the storage of the SI matricial coefficients is different: in ARPEGE, each NFLEVG×NFLEVG matrix (one per total wavenumber) is read in `spcsi.F90`, since those matrices are computed and stored only once in the setup. In ALADIN, every NFLEVG×NFLEVG matrix is recomputed in `espcsi.F90` because it depends on every pair of waves (m, n) in the ellipse, and this system would be too huge to store in memory.

8.1.2 Horizontal diffusion

Characteristic *inverse* timescales for diffusion in ALADIN (RDIVORE, RDIDIVE, ..., computed in `suehdf.F90`) are different from ARPEGE/IFS (RDIVOR, RDIDIV, ..., computed in `suhdf.F90`) due to elliptic vs. triangular truncation.

Since AL28T1, horizontal diffusion is mostly controlled by the default computation in the setup. Thus, the former namelist variables HDIR* disappear. Instead, new control variables appear in NAMDYN: `RRDXTAU` and `RDAMP[X]` where `X` is in {VOR, DIV, T, Q etc...}. In the sequel, we focus on the situation since AL29T1 (as there were several bugfixes and logical changes in the way the horizontal diffusion strength should exactly be set up). For a precize overview, the reader is directed to Filip Vaňa's technical note "On the changing design of ARPEGE/ALADIN spectral horizontal diffusion setup" (December 15th, 2004). Here, we describe the case `LNEWHD=.TRUE.` and `LREPHD=.FALSE.`. From routine `suhdir.F90`, one can derive the default setup for the two models:

- Firstly, compute a reference grid length. Arpège:

$$l = \frac{2 * \pi * r_a}{NDLON}$$

- Aladin:

$$l = (0.5 * (\Delta_x^2 + \Delta_y^2))^{1/2}$$

`RRDXTAU` is understood as the ratio of the local grid mesh size l by the damping time for the shortest wave represented in the spectrum. `RRDXTAU` therefore has the physical dimension of a velocity. The `RDAMP[X]` parameters are field-selective factors in order to strengthen or lower the diffusion with respect to the reference (when $RDAMP[X] = 1$). The following sequence applies:

- $ZHDIR = l * (1 + 0.5 * n_l)^{2.5}/RRDXTAU$ and
  $ZRHDIR = RRDXTAU/(l * (1 + 0.5 * n_l)^{2.5})$
- if $RDAMP[X] > 0$ then $HDIR[X] = ZHDIR * RDAMP[X]$ and
  $HRDIR[X] = ZRHDIR/RDAMP[X]$
  else $HDIR[X] = 0$ and $HRDIR[X] = 0$

n_l is `RNLGINC` in the code.

If we go one step further, we can consider the computations of the inverse timescales themselves, which will be provided to `spchor(ad).F90` in Arpège and to `espc(ad).F90` in Aladin. These coefficients are obtained in `suhdf.F90` and `suehdf.F90`, respectively. We describe here the case LSLHD=.FALSE. and focus on divergence.

The actual strength of the diffusion will depend both on wavenumber and (pressure-)level. The "Laplacian eigenvalues" are set first:

- in Arpège:

$$D(n, REXPDH) = \left(\frac{\sqrt{\frac{n(n+1)}{NSMAX(NSMAX+1)}} - FRANDH}{1 - FRANDH}\right)^{REXPDH}$$

  where $FRANDH = 0$ and $REXPDH = 4$ by default.

- in Aladin:

$$D(m,n,REXPDH) = (\frac{\sqrt{\frac{m^2}{NMSMAX^2}+\frac{n^2}{NSMAX^2}} - FRANDH}{1-FRANDH})^{REXPDH}$$

with the same default values as in Arpège.

As next step, these spectral values are varied according to the vertical level under consideration: low, medium and high model layers are not diffused with the same intensity. In Arpège, the intensity is continuously shifted from the NSMAX reference (low) to a NSREFDH reference (high). If the default NSREFDH=NSMAX is retained, then the modification factor becomes p_1^0/p where $p_1^0 = SLEVDH * VP00$ (default is $1 * 101325$ Pa). Finally, the Arpège diffusion timescale reads:

- when the standard pressure for level $jlev$ verifies $p > p_1^0$:

$$RDIDIV(jlev, jn) = HRDIRDIV * D(n, REXPDH)$$

- otherwise, when $p > p_2^0 = SLEVDH2 * VP00$:

$$RDIDIV(jlev, jn) = HRDIRDIV * D(n, REXPDH) * \frac{p_1^0}{p}$$

In Aladin, the intensity varies according to two threshold pressures, but (strangely) the second one does not follow the same logic as in Arpège. Finally, one obtains:

- when the standard pressure for level $jlev$ verifies $p > p_1^0$:

$$RDIDIVE(jlev, jm, jn) = HRDIRDIV * D(m, n, REXPDH)$$

- when $p > p_1^{cr}$:

$$RDIDIVE(jlev, jm, jn) = HRDIRDIV * D(m, n, REXPDH) * \frac{p_1^0}{p}$$

- and above the level for p_1^{cr}:

$$RDIDIVE(jlev, jm, jn) = HRDIRDIV * D(m, n, REXPDH) * f(p, p_1^0, p_1^{cr})$$

(see `suehdf.F90`)

8.2 Variational analysis

All spectral computations are performed under the `LELAM` key, so that ALADIN specific coefficients are obtained (e.g. `sim4d.F90, chavar(ad).F90, chavarin(ad).F90`). The canonical injections differ for ALADIN, compared to ARPEGE/IFS, but the injections are merged inside the very same piece of code since cycle AL25T1 (see the xrd modules `spectral_fields.F90` and `control_vectors.F90`). The `LELAM` key again controls ALADIN spectral space specific treatment in `cain(in)(ad).F90`. Note also that the control vector in the ALADIN 3D-VAR is dimensioned with NSMAX, not NMSMAX. Here, NSMAX represents the total, elliptic wave truncation (and not the meridional one).

Weights in the J_b metric are all equal to 1. in ALADIN, while there are weights of 1. (for the zero imaginary parts) and 2. (for other spectral coefficients) in the ARPEGE/IFS.
See routine `arp/parallel/dot_product_ctlvec(pack_all_ms).F90`. This difference is linked to the non-filtering of those spectral coefficients that are always zero in ALADIN, while they are filtered out of the control vector in ARPEGE/IFS (therefore, NSPEC2=NSEFRE in ALADIN, but NSEFRE<NSPEC2 in ARPEGE). These zero terms correspond to the imaginary parts for $m = 0$ in the IFS, and to the spectral coefficients (3,4) for $m = 0$ and (2,4) for $n = 0$ in ALADIN.

Note also the existence of the NVARMAX dimension for the data distribution of cost functions (NSMAX in ARPEGE, NMSMAX in ALADIN).

9 Non-hydrostatic model

This part has been fully adapted for global model in CY31T2 and is no more specific to LAM. It has been deleted from this LAM documentation and we will now refer to Karim Yessad's documentation.

10 Observation operators

We can identify a number of localized consequences of the ALADIN specific geometry for the computation and the use of observation operators.

10.1 Compass (deviation from true N/E)

In the routines `obshor.F90` and `obshorad.F90`, the components of the observed wind are reoriented to correspond to the direction of the geographical axes. This is done either under the `NSTTYP>1` key/value, which is true if the pole of the collocation grid in ARPEGE/IFS is not at the real Earth pole, or under the `LELAM` key because the domain is a limited-area one. As an extra requirement, the compass values need to be communicated between processors in `mpobseq.F90` and `mpobseqad.F90`. The compass fields are treated as surface fields. In the adjoint code, the extra communication of the compass (arrays `YGOMEC%ECDL` and `YGOMEC%ECDM`) was introduced specially for Aladin (the `NSTTYP>1` case for ARPEGE/IFS was actually not run, and was surely bugged before this modification).

10.2 Data outside the domain

Figure 5 presents the problem. In a global model, data are of course always "in", but in ALADIN an observation, located outside the C+I area, is considered "missing" and therefore it is not used for any purpose. Pointer variables `MDBLAT` and `MDBLON` from the `ROBHDR/ODB` array contain this information (`MDBLAT` and `MDBLON` contain the latitude and longitude positions, respectively). For observations that fall outside the C+I(+rim) zone, the values of the projected position pointers `MDBTLA` and `MDBTLO` are set to `RMDI`. Otherwise, `MDBTLA` and `MDBTLO` contain the coordinates of the location of the observation on the projected grid. Inside the ARPEGE/ALADIN CANARI code, the retained, properly located data are encomprised in an overlapping area ± 15 degrees wider in each direction than the C+I domain. This overlapping allows for a proper interpolation of innovations in gridpoint space and a correct O.I. quality control with respect to surrounding observations, even in the case where the observation is in C+I but close to the border (the overlapping is defined in `arp/obs_preproc/obatabs.F90`). Note that this overlapping is defined in the projected ALADIN geometry, not in lat/lon geometry (call `EGGX`).

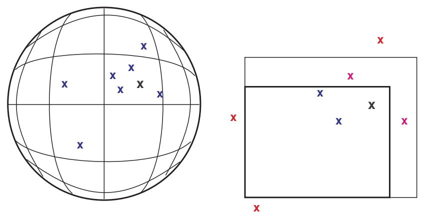

Figure 5: Drawing shows the observation operator problems occuring in LAM in case neighbouring observations lie outside the model domain.

Contrary to CANARI O.I., the first-guess screening process does not handle points outside the C+I zone. This means that the first-guess check might be biased for points close to the border, since some of the surrounding first-guess values will be missing. So far, this shortcoming has always been accepted. As indicated previously, the CANARI quality control explicitly computes guesses of the analyzed values by using the surrounding data, and compares with the observations (some kind of cross-validation). CANARI can handle points outside, but close, to the C+I zone. This is why the ± 15 degree rim zone is kept in `obatabs.F90`.

Since october 2001 and cycle AL15, the C+I data are selected at the level of ODB scripts, before entering the screening code (program `lamflag_odb.F90`). Thus, all the data are expected to verify the geometrical test in `obatabs.F90` in screening and 3D-VAR minimization.

10.3 Set of observations

The sets are defined in `ecset.F90`. For data that have `MDBTLA` and/or `MDBTLO` equal to `RMDI` - the test for observation points outside C+I performed in `obatabs.F90` - the observation is not added to a set, and some tables are reset to default. If all observations are inside C+I, then the total number of observations in the sets must be equal to the number of observations in the ODB base.

10.4 Distribution of observations

The distribution of observation locations among processors is set up in `mkglobstab.F90`. The positioning of observations in the LAM grid is specific, and uses the coordinates in the projected grid: for instance, `ZLATOBS() = ROBSU(:,MDBTLA) + _ONE_`. The positions are horizontal distances in projected geometry, normalized by the grid mesh size. The "+1" stands for starting the count of lat/lon at 1, while the ODB positions start at 0. The normalized distances are used afterwards to check for the processor that holds the gridpoint closest to the observation.

10.5 Weights for horizontal operators

The horizontal interpolations are performed via `slint.F90` and `slintad.F90`. For ALADIN, the routine `elascaw.F90` is called there to compute the weights in projected plane geometry. The on-demand SL communications are active as in ARPEGE/IFS. Note also that usually, the ALADIN interpolations are performed with the bi-linear weights (`NOBSHOR = 201`) while ARPEGE/IFS standard case is to use bi-cubic ones (`NOBSHOR = 203`).

10.6 Gridpoint scan and E-zone

In order to separate the scanning from the buffer I/O, one needs to specify two distinct end of scan/loop integer values in `cobs(tl,ad).F90`. IEND represents the scanning for gridpoint computations which is not done for the full E-zone latitudes, while IENDC stands for the buffer I/O end which is done for all gridpoints in the NPROMA packets, no matter which latitude they belong to. Thus, IEND is obtained from NGPTOT_CAP, while IENDC is obtained from NGPTOT. The computation is done in `scan2m(dm,tl,ad).F90` with the IGPCOMP integer.

11 Input/output (read/write of files)

This section was previously much dependent on whether shared or distributed memory was considered. Since cycle AL15/CY24T1 however, the shared memory code has disappeared and only the distributed memory version is accessible (even when run on only 1 processor).

11.1 Input

11.1.1 Spectral fields

The reading of spectral fields for the global and the limited area model is mostly unified and it is done in the routine `suspeca.F90`. The wind treatment depends on the nature of the model (global or LAM). In ARPEGE/IFS, the spectral variables for momentum are the velocity potential χ and the stream function ψ (recall: $u = \partial\chi/\partial x - \partial\psi/\partial y$, $v = \partial\chi/\partial y - \partial\psi/\partial x$) while, in ALADIN, they are the average cartesian components $\overline{u}$ and $\overline{v}$ plus vorticity ζ and divergence δ. Transformations from wind components (u, v) to vorticity/divergence (ζ, δ) and vice versa are done in the routine `euvgeovd.F90`, called at the end of `suspeca.F90`. In an ALADIN data file, the momentum is stored as u and v components of horizontal wind on the map (turned but not scaled by m) while, in an ARPEGE/IFS data file, it is χ and ψ.

11.1.2 Non-hydrostatic fields

While there are for the time being many possible choices for the NHS variables inside the model, it was decided to always write and read the very same fields from the ALADIN files. There is therefore a routine, `spnh_conv_nhvar.F90`, which performs the transformation between the scaled (model) fields to the unscaled (file) ones (valid both for ALADIN and ARPEGE). The transformations are the following:

$$\widehat{q} \rightarrow p - \pi$$

$$d_3, d_4 \rightarrow \frac{\partial w}{\partial z}$$

11.1.3 Grid-point fields

Reading of grid-point fields is also largely unified. The main routines for this purpose are `sugridf.F90` for surface fields and `sugridu.F90` for upper-air fields. The only non-unifiable part remaining is checking the consistency of read fields with the model parameters (`chien.F90` for ARPEGE/IFS and `cchien.F90, echien.F90` for ALADIN, under the `LELAM` key in `sugridvadm.F90` and `sugrxfu.F90`).

11.2 Output

The writing of files is unified as well (for full-postprocessed data: routines `wrsfp.F90, wrhfp.F90`, for the model: routine `wrmlppadm.F90`). The same order is respected for writing the data on model levels and pressure post-processed data. We will describe the model-level suite. In the ARPEGE/IFS case, there is first a short routine, `wrmlpp.F90`, an interface between the `iopack.F90`, called from `stepo.F90`, and `wrmlppadm.F90`, while in ALADIN, the routine `wrmlppadm.F90` is called directly from `iopack.F90`. The order of operations is as shown in table 1:

	ARPEGE/IFS	ALADIN
1.	Open file	
2.		Convert ζ, δ to u, v, unscale by m
3.	Send spectral fields to I/O processors	
4.	Send GP fields to I/O processors	
5.	Write spectral data; 2D, 3D	
6.	Write G-P data; surface, upper	
7.	Write XF data	
8.	Write CF data	
9.	Close file	

Table 1: Structure of tasks in `wrmlppadm.F90`

11.2.1 Full-Pos

Output from Full-Pos has been completely unified since CY21T2.

12 Data distribution and parallelization

12.1 General aspects

The principles of Arpège data distribution have been retained in general. The distribution depends on the space in which the computations are performed (grid-point, spectral, Fourier). Transpositions are used to communicate the data between processors before a change of space is called in the transform routines (`trmtol, trltom, trstom, trmtos, etrmtos, etrstom`). The transpositions are either called from the model routines (eg `trmtos, etrmtos`) or from the transform package (eg `trltom, trgtol`). Keep in mind that Fourier space distribution is totally setup in the transform package. Thus, just as for the spectral transforms, one has to examine 2 projects (model and transform package) to get a really complete picture of the data distribution.

3 crucial differences between ARPEGE/IFS and ALADIN have to be kept in mind:

- Until AL28T1, only the A-level parallelization was active in Aladin. Thus, the data were distributed along full bands of latitude in gridpoint space, along full rows of constant zonal wavenumber m in spectral space and neither the longitudinal splitting in gridpoint, nor the vertical level splitting in spectral could be activated. As a consequence, Aladin parallel jobs had to run with NPRTRV=NPRTRN=1. B-level has been validated for AL28T1 by G. Radnoti and the Hungarian team.

- one must be aware of the special do-loops in `scan2m*` and `gp_model*` for calling `cpg`, `cpglag` and `cobs`. While the scanning goes from 1 to NGPTOT in Arpège (where NGPTOT is the number of points in the packet of latitudes contained in the processor), the scanning goes from 1 to NGPTOT_CAP in Aladin. NGPTOT_CAP is the number of points in the packet of latitudes, not counting the points of those latitudes that lie entirely in the E-zone ! Thus, NGPTOT_CAP $\leq$ NGPTOT. However, reads and writes in the buffer arrays have still to be performed as in Arpège, over the full number of points (C+I+E), because the buffers are dimensioned with the same length in both models. This is the reason why two different integers are passed to `cpg, cpglag, cobs`: IEND (afterwards KNUMB) for the computational loops and IENDC (afterwards KNUMBE) for the buffer operations.

- the E-zone is treated in a special way in other parts of the code. Until AL25T2, data distribution could be balanced by tuning the distribution of the full E-zone latitudes over the processors (parameter TCDIS). Since AL26T1, TCDIS=0 is hardcoded in `suemplatb_mod.F90`. Otherwise, the latitudes are distributed in a fairly similar way to Arpège (routines `tfl/module/sumplat_mod.F90, tfl/module/sumplatb_mod.F90`), with extra calculations due to the E-zone in `tal/module/suemplatb_mod.F90` (using NDGUXG instead of NDGENG). Full-Pos also uses specific computations for extending surface fields.

In the setup, the routines `ald/setup/suemp.F90, tal/module/suemp_trans_mod.F90` do most of the computational distributed settings in Aladin. Part of the code is admittedly very similar to Arpège (`arp/setup/sump.F90, tfl/module/sump_trans_mod.F90`), but specific initialization is also active, such as the distribution of the zonal wavenumbers ("flip-invert" strategy) and of the latitudes (see above, routine `tal/module/suemplatb_mod.F90`).

12.2 Variational code

For 3D-VAR applications, an extra distribution was introduced from 0 to NSMAX (thus defined as if it were a meridional wavenumber distribution, see routine `ald/setup/suempvar.F90`). This allows for a more efficient parallel computation in the setup when the eigenvalue problems for the background error covariance matrix are solved, since the Aladin control vector (and B matrix as well) is dimensioned as in Arpège from 0 to NSMAX and not from 0 to NMSMAX as for the state vector. Note that this is an arbitrary choice (but technically the simplest one !) since this wavenumber is not the meridional one but the so-called total elliptic wavenumber. For similar reasons, a special dimensioning parameter NVARMAX is defined in `arp/setup/sudim.F90` for the proper dimensioning of the cost functions which has to be NSMAX in Arpège but NMSMAX in Aladin.

Yannick Tremolet has coded a specific spectral mapping for the distribution of the control vector in the IFS. Before CY26T1, the control vector was distributed by simple linear slices among all processors (so-called DISTRIBUTED_VECTOR type by M. Fisher). Now, the mapping defined through extra spectral space f90 data structures (SPECTRAL_FIELD type declaration), allows for a fully known distribution of the control vector which mimics the distribution of SPA3. The mapping parameters are defined and accessed by the modules `xrd/module/spectral_fields.F90` and `xrd/module/control_vectors.F90`. The mapped data distribution allows for an optimal message passing on MPP scalar platforms. The specific ALADIN code had to be introduced throughout the ARPEGE/IFS f90 modules. The main elements concerned are:

xrd/module/control_vectors.F90
xrd/module/control_vectors_data.F90
xrd/module/control_vectors_base.F90
xrd/module/control_vectors_oper.F90
xrd/module/control_vectors_para.F90
xrd/module/spectral_fields.F90
xrd/module/spectral_fields_data.F90
xrd/module/spectral_fields_oper.F90
xrd/module/spectral_fields_para.F90
arp/module/control_vectors_comm.F90
arp/module/spectral_columns.F90
arp/module/traj_main.F90
arp/parallel/dot_product_ctlvec.F90
arp/sinvect/lcztoald.F90
arp/utility/copy_spa2spec.F90
arp/utility/copy_spec2spa.F90
arp/utility/setimzero.F90
arp/utility/specimzero.F90
arp/utility/spec_trunc.F90
arp/utility/spec_part_copy.F90
arp/var/cain.F90
arp/var/cainad.F90
arp/var/cainin.F90
arp/var/caininad.F90
arp/var/cv2spa.F90
arp/var/spa2cv.F90

The coefficients local to one processor are stored in the following way: 2D (surface pressure), 3D (vorticity, divergence, temperature, specific humidity), TOVS brightness temperature, 1D (mean wind), 1D parameters (variational bias correction). For the 3D spectral fields, the NFLEVL coefficients over all vertical levels are stored contiguously (JLEV is the inner loop), for every spectral coefficient from 1 to NSPEC2. Some specific assignments are provided in `control_vectors.F90` to handle the mean wind stripe, which is not treated directly inside `spectral_fields.F90`, though the component spec%sp1d is allocated there. In the control vector type (CONTROL_VECTOR declaration in the code), the mean wind is stored in the component cv%lam1d. The logic is that the distributed control vector is stored in a CONTROL_VECTOR structure, which uses as an underground layer the SPECTRAL_FIELD structure, to perform the assignments and message passing controlled by the spectral mapping. These structures are also called in the code which computes the J_b term (it is defined inside the routine `control_vectors_para.F90(ctlvec_spnorm)`).

All the differences in Y. Tremolet's code, linked with ALADIN, come from the differences in the spectral spaces. They unfortunately splash throughout this code, but they are fully compatible with the usual spectral differences which are active elsewhere in the model code.

12.3 Extension of data at the "poles"

`eslextpol*` routines are counterparts of ARPEGE/IFS `slextpol*` ones. In CY27 there exist 4 versions of these routines: `(e)slextpol`; `(e)slextpol1`; `(e)slextpol1a` and `(e)slextpol2`. These routines are called for distributed memory semi-Lagrange model runs. The different versions serve the same goal, just for different semi-Lagrangian variants (whether halo on demand option LSLONDEM is on or off).

In semi-Lagrangian calculations in ALADIN, a special care has to be taken for the cases where the trajectory points are near or even outside the boundary of the C+I zone. The coordinates calculations of the trajectory points ensure that the trajectory points going outside stop at the C+I zone boundary (in `elarmes`). But for the horizontal interpolations, one value to the left and two values to the right, out of the C+I zone, are still needed. So simple copies of the boundary values are made (value(0)=value(1) and value(NDLUXG+1:NDLUXG+2)=value(NDLUXG)) for those purposes. And this is the job of **`eslextpol*`** routines. Note that no similar y-direction treatment is needed because the interpolation routines receive coordinates of the leftmost points at each of the four rows of the 4x4 interpolation region and hence if those points were above or below the C+I zone they are set back to the C+I zone boundaries.

In the global model, the routines make 2 operations, but only the second one is relevant for Aladin. These 2 operations are in connection with the semi-Lagrangian halo needed in distributed memory runs: when the geographical domain is splitted into subdomains, trajectory origin points may fall on a point that belongs to the core region of another processor. That's why the halo is introduced for ensuring overlapping of the processors' subdomains. In order that the halo operations after semi-Lagrangian communications (`slcomm*`) work well at the edges of the full domain, some extrapolations are additionally needed, and this is what `slextpol` routines are doing ("interpolating values copies" would be a better denomination than "extrapolations"):

1. Relevant only for global model: Pole treatments at North and South edges (not to be detailed here)
2. Relevant also for ALADIN but in a different way from ARP/IFS is the treatment at the East-West edges:
   - Aladin case:
     - A-level distribution (only North-South)→ full latitude is present on the processor: unlike in the global case, there is no real periodicity of the fields, therefore values of first and last (NDLUXG-th!!!) points are copied outwards to the halo region (the convention is that one extra point to the left and 2 to the right edge are introduced).
     - B-level distribution (East-West): one of the two above mentioned copies is performed if the given processor treats the first or the last (NDLUXG-th) point (this case is relevant only for aladin, in ARP/IFS b-level case the east-west halo together with the real periodicity of fields makes these extrapolations unnecessary).

Appendix A

Report on the parallelization level of Aladin at the time of March-October 2003

by Gabor Radnóti

Introduction

ALADIN has been able to run in parallel mode using MPI for many years, but there were strong limitations in its parallelization level with respect to ARPEGE/IFS:

- In ARPEGE/IFS optionally one can use the so called b-level parallelization. B-level parallelization means that the computational domain is distributed among processors in a two-dimensional way (in a-level parallelization all this is much simpler, distribution is done only in one dimension). The way of this two-dimensional distribution varies within the time-step and some transpositon routines take care of going from one kind of distribution to the other. For example in the gridpoint space fields are distributed in the two horizontal directions. In the spectral space fields are distributed according to groups of zonal wavenumbers and along the vertical. In some parts of spectral computations (e.g. semi-implicit part), however, one needs all the vertical levels simultaneously, therefore distribution is transposed such that all vertical levels are recollected and coefficients of the groups of zonal wavenumbers are further partitioned. The design of this b-level parallelization is suitable for ALADIN and in most parts of ALADIN code b-level requirements have been respected. Nevertheless this option has never worked in ALADIN. As time goes by, the situation would become worse and worse because developpers have no possibility to test their new developments in b-level environment therefore sometimes they may even fully neglect these constraints. That is why we have decided that it is high time to catch up with the parallelization level of the global model.
- Situation is very similar with the so called "lsplit" option, that intends to provide a perfect load balance for gridpoint computations both in a-level and in b-level parallelization by allowing to break the last "latitude-row" of a given a-set and to start the next a-set from this "breakpoint". An a-set in gridpoint space means a set of processors that treat the same group of latitude-rows (in a-level parallelization therefore an a-set is a single processor, in b-level all the processors working on the same group of latitudes, but over different longitudinal bends belong to the same a-set).

From the two paragraphs above one can see the purpose of b-level and lsplit: B-level parallelization is to reduce communication overhead (imagine a square domain with hundred horizontal bends and with a 10×10 sub-net: both represent a distribution to 100 processors, the former in a, the latter in b-level mode; the overhead is proportional to the total length of border lines, which is 5 times bigger in a-mode), Lsplit is to improve load-balance in grid-point space. Both aspects can be of high importance when having a MP system with hundreds of processors.

In 2001 the spectral transformations have been externalized and put into a separate package (tfl for ARPEGE/IFS and tal for ALADIN). Since during the transformations model fields "spend some time" in all possible "model spaces", basically all parallelization related setup has been moved to the package. In the external package one can run standalone tests without any model computations and it gives a possibility to test the parallelization design in a much simpler environment. Therefore we decided to make te first step toward b-level parallelization and lsplit inside the package. If it works, it proves that the design and setup of b-level parallelization is perfect for ALADIN and work can be continued on the model side.

Modifications and tests on tal package

b-level: After some basic tests it turned out that problems occur only with the momentum fields. This immediately suggested that the specificity of ALADIN, i.e. the mean wind, is most probably not treated in a b-level-conform way. Indeed, vertical distribution was not included in the code for mean wind after reading spectral wind fields and it was incorrectly (for b-level) distributed to all a-sets in direct transforms, where only the a-set holding wavenumber zero computes mean wind and it has to be distributed to all the other a-sets. After correcting these bugs b-level was perfectly working in the tal package.

Modified routines:

- `testtrans.F90`: Main program of standalone tal tests, vertical distribution of mean winds was included
- `euvtvd_mod.F90`: b-level conform distribution of mean wind within a b-set: all a-set processors within the same b-set (set of vertical levels) must receive mean winds from the processor holding wavenumber 0).
- `gath_grid_ctl_mod.F90`: a small tfl bug that was identified during tests (bug reported to IFS).

lsplit: The fact that lsplit option did not work in the package was in a way surprising because in this respect there seemed to be no principal difference between tfl and tal. However, it was easy to find that some differences exist and they are related to the aladin specific requirement to distinguish between C+I rows and E rows of the domain as far as computational demand is concerned. The tunable parameter called TCDIS is a pre-defined weight factor and in the aladin code it is taken into account when setting up gridpoint space a-sets. This was the only reason why tal and tfl setup differed conceptually. Nevertheless this concept contradicts in lsplit case with some assumptions of the code and these contradictory assumptions (all processors have as much as possible the same number of gridpoints to treat up to the level of division modulus, i.e. a group of processors have one more point than the rest according to the modulus) and these assumptions have never been correctly replaced in aladin. We decided to drop the TCDIS concept which does not only bring us immediatley to a correctly working lsplit option, but also makes the tal setup very close to the tfl one, which is a very convenient advantage.

ezone treatment: In the previous point i have mentioned the meaning of parameter TCDIS, that we have decided to drop. The solution that we have chosen instead is that we do almost the same distribution strategy as in arpege with the difference that for gridpoint computations, when we distribute latitudes among a-sets we consider NDGUX as total number of latitudes instead of NDGL and at the very end we attach all ezone rows to the last a-set (this means a very slight modification of `sumplat`, `sumplatb` and `sustaonl`==>`suemplat`, `suemplatb,suestaonl`). This is the case for gridpoint partitioning. In fourier space, before zonal direct transforms fields are transposed to full latitude bends. Therefore a repartitioning is performed where in principle the latitude set of each a-set may differ from that of gridpoint partitioning. This is controlled by `sumplatf`. (In lsplit case this partitioning necessarily differs from that of gp space, since in gp-space latitudes are broken, while in fourier space they must not be). Ezone transforms are not cheaper than C+I ones, so it is reasonable to use in fourier space the same partitioning as in arpege.i.e. not making difference between ezone rows and C+I rows in this respect.. It was tested in tal and it works perfectly. We have to see that with this new solution even when lsplit=false, fourier space distribution differs from gridpoint one, i.e. NDGLL=NDGENL will not be true anymore. Therefore in ALADIN code one has to be very careful with the correct usage of NDGLL versus NDGENL ! Modified routines:

- `suemplat_mod.F90`, `suemplatb_mod.F90`: new versions are much closer to tfl counterparts than before, TCDIS fully dropped.
- `suemp_trans_mod.F90`: calls `suestaonl` instead of `sustaonl`, and `sumplatf` instead of `suemplatf`

New routine: `suestaonl_mod.F90`: a slightly modified version of `sustaonl.F90`

Removed routine: `suemplatf_mod.F90`

After all these modifications the transform package works for all parallelization options. Performance tests are not meaningful with the package itself because no real gridpoint computations are involved.

Further fix on the package: Later at ALADIN tests a further small bug was found and corrected on tfl/tal package in routines `inv_trans.F90,einv_trans.F90`. Bug has been reported to IFS, so care is taken on its fix in higher cycles.

Modifications and tests on Aladin (configuration 001)

The fixes performed on the TAL package provided a firm basis for starting ALADIN tests and modifications. Before starting the testing, debugging, fixing actions I decided to re-design and recode the coupling data stream because it was known in advance that the original version is completely not suitable for b-level and lsplit requirements. Details are written down in a separate document.

Debugging of ALADIN in b-level with new coupling: After rewriting aladin coupling in the way described above I could start b-level and lsplit tests on aladin. Below I list the additional modifications that I had to introduce to make ALADIN configuration 001 work in b-level/lsplit mode:

- `espconvert.F90`: mis-use of NFLEVL<–>NFLEVG
- `espuv.F90`: initializing IVSETUV and introducing it to calling sequence of EINV_TRANS
- `sueorog.F90,suecuv.F90`: initializing IVSETSC and introducing it to calling sequence of EINV_TRANS,EDIR_TR
- `euvgeovd.F90`: mis-use of KLEV<–>KFLSUR + introducing kvsetuv
- `disgrid.F90`: Arpege bug, reported, care taken by GMAP
- `etrmtos.F90`: "mis-typing errors", trivial fixes.
- `sump0.F90`: remove protection of aladin lsplit
- `wrmlppadm.F90`: mis-use of NFLEVL<–>NFLEVG
- `ewrplppdm.F90`: introducing kvsetuv
- `suegeo2.F90`: wrong size and definition of NLOEN + forcing NBDYSZ=NGPTOT (see earlier remark)
- `suemp.F90`: 2 fixes:
  - care had to be taken that when spectral a-sets are further splitted for SI computations, the breaking of NSPEC2 arrays into several NSPEC2V arrays should not break inside a wavenumber pair, i.e. the 4 coefficients of a wavenumber pair should remain on the same processor.
  - wrong definition of NDGUXL
- `espnormave.F90`: removing mean wind from norm computation: if it is to be present, code for distribution of mean wind for diagnostic purpose has to be elaborated (In between it has been done, new routine `ecommspnorm`).
- `suspec.F90 suspeca.F90,suspeca.h`: AL15 code is bugged, corrected `suspeca` was taken from higher cycles. Furthermore, explicit array bounds had to be removed from dummies to provide norm identity after `suspeca` (a general little bug!!!!)

Performance tests After all the above described modification b-level and lsplit options worked in configuration 001. Norm identity was guaranteed up to 6 digits in a 6 hour integration. I made some performance tests with NPROC=16,NPRGPNS=NPRGPEW=NPRTRW=NPRTRV=4 on a domain with NFLEVG=31,NDLON=240,NDGL=216. The b-level option in this configuration brings a solid 10 percent performance improvement. LSPLIT does not make any measurable change in performance. The effectivity of LSPLIT highly depends on the NDLUXG/NPRGPNS ratio and modulus, so tests should be repeated with several such configurations.

Appendix B

2D vertical plane model

by Alena Trojakova

Specific to 2D vertical plane model ALADIN

A 2D vertical plane version of ALADIN model was introduced for the purpose of academic testing of existing model dynamics and its further development. The scope was not intended for real case studies (although with some dedicated coding it can be achieved, more details P.Benard or Y.Seity). The 2D model follows 3D version as much as possible, regardless of any efficiency. As a model domain (y, z)-plane was chosen, it corresponds to plane geometry (e.i. not spherical projected geometry, the map factor $m = 1$) and its position doesn't correspond to any place on the Earth. Physically 2D model represents a "true" 2D model, which means that we solve full 3D equations, including prognostic equation for u and all x-partial derivatives (although they are equal to zero thanks to (NMSMAX=0)). As 2D model is more subjected to reflection of waves at upper and lower boundaries it is usually necessary to apply some sort of artificial dumping called sponge.

Input/Output

An initial or coupling file is in standard "FA"- format, but they cannot be obtained by classical model configurations (923,E927,EE927,etc). Current way to get an input file is to run an off-line tool called "2D acadfa", which allows to create predefined idealized condition or you can modify its code for desired specification.

An output file is also standard "FA"- file. Post-processing of outputs via standard model configuration doesn't work (actually never become really needed). For the conversion from "FA" to ASCII format a special off-line tool called "flux" has to be used.

Both tools are available from J.Masek or P.Benard.

Code specific

The code of 2D model is characterized by NMSMAX=0 and also as an academic case by LMAP=F (common also for 3D academic case).

Setup

- SUEGEO1: setting of Coriolis parameter (possible via namelist NEMGEO)
  - setting of default value RCORI_ACAD=0
  - namelist value of RCORI_ACAD is read and truncated if out of range
- SUEBIG - ELLIPS special treatment for NMSMAX=0
- CNT3 - SUPONG : Initialize YOMPONG, module that controls sponge for vertical plane 2D model. Technical limitations of the sponge:
  - one processor (needed for correct sponge setup)
  - time constant LBC (background profiles do not evolve in time)

- NH model: 2D experiment with NPROMA = NDGL, use NPROMA = -NDGL in namelist &NAMDIM to aviod optimization of NPROMA in setup. (needed by spectral transforms called below level of CPG to compute grad(vd))

Miscellaneous

- CCHIEN: no subtrucation packing
- GPSPNG: gridpoint sponge
- tfl/module/FTINV_MOD.F90 tfl/module/FTDIR_MOD.F90 : skip redundant transformation (NLOEN=1)
- wrmlppadm.F90 : ??

Appendix C

Correspondence of ARPEGE/IFS and ALADIN cycles

The table below summarizes the numbering of ARPEGE/IFS and ALADIN cycles, since 1997. We also briefly stress their main "historical" achievement, which can sometimes be ... a bug !

ARP/IFS	ALD	Date	Notice
CY16T1	AL07	fall 97	last stable cycle on Cray C90
CY18T1	AL08	spring 98	first (uncomplete) ported cycle to VPP
CY18T3	AL08T3	early summer 98	oper cycle on VPP
CY19T1	AL09	fall 98	ISBA and bug of the 8th coupling file
CY20/CY21	AL10/CY21_al11	late 98	code cleaning
CY21T1	AL11	spring 99	regular phasing
CY21T2	AL11T2	fall 99	3D-VAR runs !
CY22T1	AL12	early 00	regular phasing
CY22T2	AL13	early summer 00	short phasing (missed CY23)
CY23	AL14	fall 00	non oper ("bad") cycle
CY24T1	AL15	fall 01	ODB works at last !
CY25T1	AL25T1	spring 02	new geometry MAKDO
CY25T2	AL25T2	fall 02	NHS P/C scheme
CY26T1	AL26T1	spring 03	new distribution of control vector
CY27	AL27	fall 03	GMV/GFL dataflow (non oper)
CY28	AL28	late fall 03	new coding rules (interfaces) and new ODB file structure
CY28T0	AL28T0	winter 04	Aladin explicit interfaces in Arpge
CY28T1	AL28T1	late spring 04	MSG data and β-plane J_b
CY28T2	AL28T2	July 04	operational version of CY28
CY28T3	AL28T3	September 04	bugfixes from Aladin partners
CY29	AL29	November 04	RTTOV8 and VarBC
CY29T1	AL29T1	February 05	rationalized setup for horizontal diffusion, SLHD, SEVIRI radiances and radar J_o
CY29T2	AL29T2	April 05	Arome prototype, 3TL P/C, J_k, rotated Mercator projection
CY29T3	AL29T3	April 05	miscellaneous bugfixes
CY30	AL30	July 05	new GOM structure, wavelet J_b (IFS)
CY30T1	AL30T1	December 05	NH Cleaning
CY31	AL31	April 06	Surfex under NPROMA loop, rewritin of GFL attributes
CY31T1	AL31T1	July 06	Hirlam Physics, surfex in ALADIN
CY31T2	AL31T2	September 06	SL-TL in ALADIN
CY32	AL32	December 06	new surface data structure, ALARO0 physics

Appendix D

Table of ARPEGE/IFS routines with ALADIN counterparts or additions

Legend: D - duplicated routines: ALADIN routine can perform ARPEGE operations if LELAM=.F. or ALADIN operations if LELAM=.T. C - ALADIN routine is a counterpart to an ARPEGE/IFS one, performing the same operations under ALADIN conditions, A - ALADIN routine is called in addition to the ARPEGE/IFS one, thus completing it for ALADIN conditions, NO - no correspondence whatsoever although the name could lead us to think so, NC - not called for the time being. Directories of location are defined as in the ARPEGE library (and can be different from ALADIN locations).

```
======
adiab:
======
       larche.F90        elarche.F90  C
      larche5.F90       elarche5.F90  C
     larchetl.F90      elarchetl.F90  C
       larmes.F90        elarmes.F90  C
      larmes5.F90       elarmes5.F90  C
     larmestl.F90      elarmestl.F90  C
       lascaw.F90        elascaw.F90  C
     lascawtl.F90      elascawtl.F90  C
     specrtdm.F90        especrt.F90  C
        spcsi.F90         espcsi.F90  C
       spnhsi.F90        espnhsi.F90  C
       spchor.F90        espchor.F90  C
      spcsiad.F90       espcsiad.F90  C
     spnhsiad.F90      espnhsiad.F90  C
     spchorad.F90      espchorad.F90  C

=====
c9xx:
=====
       chk923.F90        echk923.F90  C
       chkrel.F90        echkrel.F90  C
       coptra.F90        ecoptra.F90  A
       ganiso.F90        eganiso.F90  C
       geo923.F90        egeo923.F90  C
       incli1.F90        eincli1.F90  C
       incli2.F90        eincli2.F90  C
       incli3.F90        eincli3.F90  C
       incli4.F90        eincli4.F90  C
       incli5.F90        eincli5.F90  C
       incli6.F90        eincli6.F90  C
       incli7.F90        eincli7.F90  C
       inclib.F90        einclib.F90  C
       inclir.F90        einclir.F90  C
        inirp.F90         einirp.F90  C
       inter0.F90        einter0.F90  C
       inter1.F90        einter1.F90  C
      inter10.F90       einter10.F90  C
       inter2.F90        einter2.F90  C
       inter6.F90        einter6.F90  C
       inter8.F90        einter8.F90  C
       lislap.F90        elislap.F90  C
       relspe.F90        erelspe.F90  C

========
control:
========
   control/spch.F90     adiab/espch.F90  C
 control/spchad.F90   adiab/espchad.F90  C
   control/spcm.F90     adiab/espcm.F90  C
 control/spcmad.F90   adiab/espcmad.F90  C

====
dia:
====
      spnormave.F90       espnormave.F90  C
        spnormb.F90         espnormb.F90  C
       wmoviedm.F90        ewmoviedm.F90  C
         wmovph.F90          ewmovph.F90  C
        wrimova.F90         ewrimova.F90  C
         wrplpp.F90        ewrplppdm.F90  C

=======
kalman:
=======
  kalman/balads.F90     var/ebalads.F90  C

=======
pp_obs:
=======
    pp_obs/spos.F90        dia/espos.F90  C

======
setup:
======
         suallo.F90           suello.F90  A
          suct0.F90           suect0.F90  A
          sudim.F90           suedim.F90  A
          sudyn.F90           suedyn.F90  A
          sufft.F90           suefft.F90  A
          sufpg.F90          suefpg3.F90  A
        sugem1a.F90          suegeo1.F90  C
```

```
        sugem2.F90         suegeo2.F90  C
         suhdf.F90          suehdf.F90  C
         suheg.F90          suehel.F90  C
         suhdu.F90                   .
        sutric.F90                   .
        suhow1.F90         suehow1.F90  C
        suhow2.F90         suehow2.F90  C
      suhowlsm.F90       suehowlsm.F90  C
         suios.F90          sueios.F90  A
         sulap.F90          suelap.F90  A
      sumetric.F90       suemetric.F90  C
          sump.F90           suemp.F90  C
         suoph.F90          sueoph.F90  C
        suorog.F90         sueorog.F90  C
        suspe0.F90         suespe0.F90  A
    susc2(a,b).F90          suesc2.F90  A

========
sinvect:
========
        chnorm.F90         echnorm.F90  C
       rdtllcz.F90        erdtllcz.F90  C
       sptrlcz.F90        esptrlcz.F90  C
         sulcz.F90          suelcz.F90  C
       wrtllcz.F90        ewrtllcz.F90  C
         wrtsv.F90          ewrtsv.F90  C
      lcztoifs.F90       lcztoald.F90  C

==========
transform:
==========
        reespe.F90         ereespe.F90  C
        speree.F90         esperee.F90  C
          spuv.F90           espuv.F90  C
   transdir_fp.F90    etransdir_fp.F90  C
  transdir_mdl.F90   etransdir_mdl.F90  C
transdir_mdlad.F90 etransdir_mdlad.F90  C
transdir_nhconv.F90 etransdir_nhconv.F90  C
     transdirh.F90      etransdirh.F90  C
   transdirhad.F90    etransdirhad.F90  C
   transinv_fp.F90    etransinv_fp.F90  C
  transinv_mdl.F90   etransinv_mdl.F90  C
transinv_mdlad.F90 etransinv_mdlad.F90  C
transinv_nhconv.F90 etransinv_nhconv.F90  C
     transinvh.F90      etransinvh.F90  C
   transinvhad.F90    etransinvhad.F90  C

========
utility:
========
        deallo.F90          deello.F90  A
      spareord.F90       espareord.F90  C
     spconvert.F90      espconvert.F90  C

====
```

```
var:
====
      balnolin.F90      ebalnonlin.F90  C
    balnolinad.F90    ebalnonlinad.F90  C
    balnolintl.F90    ebalnonlintl.F90  C
      balomega.F90       ebalomega.F90  C
    balomegaad.F90     ebalomegaad.F90  C
    balomegatl.F90     ebalomegatl.F90  C
       balstat.F90        ebalstat.F90  C
     balstatad.F90      ebalstatad.F90  C
       balvert.F90        ebalvert.F90  C
     balvertad.F90      ebalvertad.F90  C
      balverti.F90       ebalverti.F90  C
    balvertiad.F90     ebalvertiad.F90  C
         cosjr.F90          ecosjr.F90  C
       cvaru2i.F90        ecvaru2i.F90  C
     cvaru2iad.F90      ecvaru2iad.F90  C
        jghcor.F90         ejghcor.F90  C
       jghcori.F90        ejghcori.F90  C
          jgnr.F90         ejgnrgg.F90  C
        jgnrad.F90       ejgnrggad.F90  C
         jgnri.F90        ejgnrggi.F90  C
       jgnriad.F90      ejgnrggiad.F90  C
        monvar.F90          moevar.F90  A
       scaljgs.F90        escaljgs.F90  C
       sualges.F90         suelges.F90  A
        suallr.F90          suellr.F90  A
        suallt.F90          suellt.F90  A
       suallt7.F90         suellt7.F90  A
       sujbbal.F90        suejbbal.F90  C
       sujbcor.F90        suejbcor.F90  C
      sujbcosu.F90       suejbcosu.F90  C
       sujbcov.F90        suejbcov.F90  C
       sujbdat.F90      suejbdat96.F90  C
       sujbstd.F90        suejbstd.F90  C
      sujbtest.F90       suejbtest.F90  C
        suscal.F90         suescal.F90  C

=========
parallel:
=========
     commjbbal.F90      ecommjbbal.F90  C
   gathereigmd.F90    egathereigmd.F90  C
     gatherspa.F90      egatherspa.F90  C
      slextpol.F90       eslextpol.F90  C
     slextpol1.F90      eslextpol1.F90  C
    slextpol1a.F90     eslextpol1a.F90  C
     slextpol2.F90      eslextpol2.F90  C
        trmtos.F90         etrmtos.F90  C
        trstom.F90         etrstom.F90  C

==========================================

 utility/chien.F90    c9xx/cchien.F90  C
 setup/suinif.F90   inidata/elsac.F90  C
```

```
==========================================

False friends:

==========
transform:
==========
        sperad.F90         esperad.F90  NO

=======
pp_obs:
=======
           pos.F90            epos.F90  NO
          hpos.F90            epos.F90  NO
          vpos.F90            epos.F90  NO
          spos.F90            epos.F90  NO
```

Appendix E

Tests of the variational model configurations: what to look for

A repeated exercise in every phasing is to run and validate the model versions which use the tangent linear and adjoint models. These runs also test the basic usage of the control vector structure, which is otherwise of utmost importance in the variational analysis configuration (e131). We describe here what the tests are doing, and how to check easily for completion and correctness. We illustrate the numerical results with outputs from `mitraillette`. These variational model configurations are supposed to work properly both in mono and multi-processor mode, but only Eulerian advection is coded for the time being.

e501 Test of the tangent linear model: We test the order 1 of the Taylor expansion of the non-linear model (taken as the functional to be verified):

$$TL = \frac{M(x + 10^{-\lambda}\epsilon) - M(x)}{10^{-\lambda}\mathcal{M}\epsilon}$$

The test loops over about 10 values for λ. The bigger the exponent, the smaller the size of the perturbation, and the closer to 1 the TL term. You can grep for 'LAMBDA' to scan all printouts for any given exponent in the return listing. TL values are printed out for a restricted number of spectral coefficients (taken by chance). All should converge to 1 with increasing exponent, *but due to numerical precision error on any computer platform*, there is a limit from where the result again starts to diverge from 1. If you grep for any specific spectral coefficient, you can visualize the quality of the Taylor test with λ:

```
VORT.  I =    21856 RAT = 0.9998223417410624E+00  -.64789E-16 -.13325E-05 -.10209E-05 -.10209E-05
VORT.  I =    21856 RAT = 0.1000034097889765E+01  -.64803E-15 -.13325E-05 -.10209E-05 -.10209E-05
VORT.  I =    21856 RAT = 0.1000004033745197E+01  -.64801E-14 -.13325E-05 -.10209E-05 -.10209E-05
VORT.  I =    21856 RAT = 0.1000000301869860E+01  -.64801E-13 -.13325E-05 -.10209E-05 -.10209E-05
VORT.  I =    21856 RAT = 0.1000000012012292E+01  -.64801E-12 -.13325E-05 -.10209E-05 -.10209E-05
VORT.  I =    21856 RAT = 0.1000000315790866E+01  -.64801E-11 -.13325E-05 -.10209E-05 -.10209E-05
VORT.  I =    21856 RAT = 0.1000003161976504E+01  -.64801E-10 -.13325E-05 -.10209E-05 -.10209E-05
VORT.  I =    21856 RAT = 0.1000031623509691E+01  -.64803E-09 -.13331E-05 -.10209E-05 -.10215E-05
VORT.  I =    21856 RAT = 0.1000316188032933E+01  -.64821E-08 -.13390E-05 -.10209E-05 -.10273E-05
VORT.  I =    21856 RAT = 0.1003157095282599E+01  -.65005E-07 -.13975E-05 -.10209E-05 -.10856E-05
VORT.  I =    21856 RAT = 0.1031004851759863E+01  -.66810E-06 -.20006E-05 -.10209E-05 -.16679E-05
```

An interesting by-side feature is that, as long as the test converges above machine precision, the first 2 or 3 non-0 or non-9 digits in TL ("RAT") are stable, and give the second order error of the Taylor expansion.

e401 Test of the adjoint model. We check for the adjoint equality:

$$< \mathcal{M}x, y >=< x, \mathcal{M}^{*}y >$$

The scalar inner product is the one defined in `suescal.F90`, and stored in SCALP. The two vectors x and y are random (be careful: their definition can change from one cycle to the next, if ECMWF changes the random vector generator ...). $\mathcal{M}$ and $\mathcal{M}^{*}$ are the tangent linear and the adjoint models, respectively. You also have run the direct non-linear model to compute the incore trajectory. A typical (good) output looks like:

```
TEST OF THE ADJOINT
```

```
                    12345678901234567890
< F(X) , Y >  = -.76693013885752860000E-02
< X , F*(Y) > = -.76693013885752800000E-02
THE DIFFERENCE IS     3.565 TIMES THE ZERO OF THE MACHINE
```

12 or 13 equal digits should be obtained for the adiabatic eulerian model. A bit less can be accepted for the Semi-Lagrangian models, in adiabatic mode. With simplified physics, the more physics are switched on, the less valid the test (only about 5-6 digits).

e801 In this configuration, the principle is to compute a cost function as the distance between a model forecast x^b and a reference (analysis) state x^a:

$$J = < x^b - x^a, x^b - x^a >$$

A small difference is added, with the purpose to decrease the distance J:

$$J' = < x^b - x^a + \delta, x^b - x^a + \delta > = J + 2 < x^b - x^a, \delta > + < \delta, \delta >$$

Ignoring the second order term, and considering that δ is valid at the forecast time, one retrieves the gradient of J at initial time by applying the adjoint equation:

$$2 < x^b - x^a, \delta > = 2 < x^b - x^a, \mathcal{M}\delta_0 > = 2 < \mathcal{M}^*(x^b - x^a), \delta_0 >$$

For validation, the test of the gradient of the cost function is checked by setting $\delta_0 = 10^{-\lambda}\epsilon$:

$$TG = \frac{-2.10^{-\lambda} < \mathcal{M}^*(x^b - x^a), \epsilon >}{J - J'}$$

Like for e501, there should be convergence to 1 with increasing exponent, unless machine precision is reached. A classical output is given here:

```
GRTEST gradient T1= 0.9992162954680948
GRTEST gradient T1= 1.000065303983853
GRTEST gradient T1= 0.9999274208188891
GRTEST gradient T1= 0.9999982412286191
GRTEST gradient T1= 0.9999996403552931
GRTEST gradient T1= 0.9999970236809124
GRTEST gradient T1= 0.9999704656398133
GRTEST gradient T1= 0.9997046927658785
GRTEST gradient T1= 0.9970469249118699
GRTEST gradient T1= 0.9704690051107859
GRTEST gradient T1= 0.7046659011193949
GRTEST gradient T1= -1.955432667564963
```

The convergence always occurs from below 1, with 9's as representative digits, because the term $< \delta, \delta >$ is omitted in the numerator. In the numerically stable phase, the first 2-3 digits after the 9's give the second order error of the expansion.

e601 This configuration computes the singular values and singular vectors for the TL/AD models, which are the eigenvalues and eigenvectors of the equation:

$$< \mathcal{M}x, \mathcal{M}x > = < x, \mathcal{M}^*\mathcal{M}x > = \mu^2 < x, x >$$

The eigenvalues are listed in the return listing below the "RITZ" character sequence. Eigenvectors are written to an FA file "SVARPE+000$n" for the n-th singular vector. An example of printout (adiabatic e601, "AH6E"):

```
TQLB exited with IERR =    0

TQLB delivered... J =  1 FIRST =  2 LAST =  1
                   ID1 =  1 L =  1 I =  1 |BND|^2 =  1.00000000000000D+00
  I         RITZ(I)        (        BND(I)        )       ALF(I)      BET(I)
  1   4.79781347552814D+00 (  1.00000000000000D+00)      4.798D+00   5.321D-05

DSORT2 delivered...
  1   4.79781347552814D+00 (  1.81486221380826D+00)

J ** RITZ ** EIG=SQRT(ABS(RITZ)) ** BND
   1   0.479781347553E+01   0.219039116952E+01   0.181486221381E+01 ** OK **  0.3783E-03
```

Another example of printout (trajectory with some old physics, e601 "AG6E"):

```
TQLB exited with IERR =    0

TQLB delivered... J =  1 FIRST =  2 LAST =  1
                   ID1 =  1 L =  1 I =  1 |BND|^2 =  1.00000000000000D+00
  I         RITZ(I)        (        BND(I)        )       ALF(I)      BET(I)
  1   4.58908925041394D+00 (  1.00000000000000D+00)      4.589D+00   0.000D+00

DSORT2 delivered...
  1   4.58908925041394D+00 (  1.74341388189764D+00)
J ** RITZ ** EIG=SQRT(ABS(RITZ)) ** BND
   1   0.458908925041E+01   0.214221596727E+01   0.174341388190E+01 ** OK **  0.3799E-03
```

The structure of the singular vector can actually be plotted, for instance using a simple "spectral chagal".