# TRAJECTORY MANAGEMENT IN THE CYCLE 42 OF ARPEGE/IFS.

YESSAD K. (METEO-FRANCE/CNRM/GMAP/ALGO)

July 7, 2015

*Abstract:*

*In ARPEGE/IFS, some applications (for example a 4DVAR minimisation) use a tangent linear code and an adjoint code. Such codes require to compute a trajectory, this trajectory is computed by running the direct code. The present documentation has for purpose to describe how is managed this trajectory, and how the information contained in this trajectory is passed from the direct code to the tangent linear and the adjoint codes. Some additional applications are briefly described, for example the possibility to use a trajectory coming from another direct job done with a higher resolution. A specific list of routines doing trajectory management is provided.*

*Résumé:*

*Dans ARPEGE/IFS, il y a des applications (par exemple une minimisation 4DVAR) qui nécessitent d'appeler du code tangent linéaire (TL) et du code adjoint (AD). De tels codes demandent le calcul d'une trajectoire, laquelle est obtenue via le code direct. Cette documentation a pour but de décrire comment est gérée cette trajectoire, et comment l'information contenue dans cette trajectoire est transmise depuis le code direct vers les codes TL et AD. On décrit brièvement quelques applications supplémentaires, par exemple la possibilité d'utiliser une trajectoire calculée depuis un autre modèle à résolution plus fine. On fournit une liste de routines utilisées dans la gestion de trajectoire.*

## Contents

# 1 Introduction.

In ARPEGE/IFS, some applications (for example a 4DVAR minimisation) use a tangent linear code and an adjoint code. Such codes require to compute a trajectory, this trajectory is computed by running the direct code. The present documentation has for purpose to describe how is managed this trajectory, and how the information contained in this trajectory is passed from the direct code to the tangent linear and the adjoint codes. Some additional applications are briefly described, for example the possibility to use a trajectory coming from another direct job done with a higher resolution. We stick currently to the 3D primitive equation model.

### ∗ Modifications since cycle 41:

- Minor modifications in call-tree.

# 2 Management of internal trajectory.

## 2.1 The configurations involved.

The case is encountered in all configurations using at least the tangent linear code or the adjoint code. Such codes are linearised along the trajectory, and we have to describe how the content of this trajectory is transmitted from the direct integration to the tangent linear and adjoint integrations. The configurations involved are currently configurations 131, but also 401, 501, 601, 801 for the primitive equation model. The architecture which is generally encountered for these configurations is the following one:

- Integration of the direct code to compute the trajectory.
- One or several integrations of the tangent linear code.
- One or several integrations of the adjoint code.

We can study for example the case where there is one tangent linear integration and one adjoint integration.

## 2.2 Saving the trajectory in the direct code.

The direct code computes $X(t_0 + k\Delta t)$ at each timestep ($k$ between 0 and **NSTOP**). $X$ is a generic notation for each prognostic variable or each surface variable. When it is necessary to save the trajectory, the different $X$ are stored in specific buffers at each timestep.

Variables $X$ can be separated into several classes:

- The GMV upper-air fields: the RHS of the equation giving the temporal evolution of such a variable is non-zero (example: wind components, temperature). In ARPEGE/IFS, such fields have always a spectral representation and also a grid-point representation.
- The GMVS surface fields: the RHS of the equation giving the temporal evolution of such a variable is non-zero (example: logarithm of surface hydrostatic pressure). In ARPEGE/IFS, such fields have always a spectral representation and also a grid-point representation.
- The GFL upper-air fields: the RHS of the equation giving the temporal evolution of such a variable is zero (example: humidity). In ARPEGE/IFS, such fields have always a grid-point representation. They can have a spectral representation.
- The surface grid-point evolving fields (example: surface temperature and surface water content): they are used mainly in the physics. In ARPEGE/IFS, such fields have always a grid-point representation. They are never spectral ones.
- The surface grid-point constants (example: land-sea mask): they are used mainly in the physics. In ARPEGE/IFS, such fields have always a grid-point representation. They are never spectral ones.
- The surface spectral constants (example: surface orography): In ARPEGE/IFS, such fields have always a spectral representation and also a grid-point representation.

Since there are several classes of variables, there are also several buffers to store the trajectory.

Additionally, some intermediate calculations done in the direct code can be saved in the trajectory (this is the case for example in the semi-Lagrangian scheme): such additional storage increases the size of the trajectory buffer, and reduces the amount of trajectory recalculations which must be done in the tangent linear and in the adjoint codes. The amount of such additional storage must respect a good compromise, and it generally remains a lot of intermediate trajectory calculations which must be done in the tangent linear and in the adjoint codes.

According to the value of variable **LTRAJGP**, it is possible for variables with a spectral representation to store them only in spectral space (**LTRAJGP**=.F.) or to store them both in spectral and grid-point space (**LTRAJGP**=.T.). The first solution reduces the size of the trajectory buffers, but increases the number of inverse spectral transforms when calling the tangent linear and the adjoint codes. The second solution avoids to do inverse spectral transforms on the trajectory when calling the tangent linear and the adjoint codes, but increases the size of the trajectory buffers.

On a theoretical point of view, the trajectory savings and calculations have to be the same ones for the tangent linear code and for the adjoint code, but this is not always the case; sometimes the adjoint code requires more trajectory calculations than the tangent linear code. In the tangent linear code and the adjoint code, variables containing the trajectory have generally a name ending by 5. For example, when **PTT0** is stored in the direct code (temperature at instant $t$ in grid-point space), it is read in the tangent linear code in the array **PTT5**. The correspondences for the name appendices are generally:

```
direct         trajectory in TL, AD

 --- Variables at t ---
 X0              X5
 X0L             X5L
 X0M             X5M
 X0F,XF0         X5F,XF5
 X0H,XH0         X5H,XH5

 --- Variables at t-dt ---
 X9              X95
 X9L             X95L
 X9M             X95M
 X9F,XF9         X95F,XF95
 X9H,XH9         X95H,XH95

 --- Variables at t+dt ---
 X1              X15
 X1L             X15L
 X1M             X15M
 X1F,XF1         X15F,XF15
 X1H,XH1         X15H,XH15

 --- "Timeless" variables ---
 X               X5
 XL              X5L
 XM              X5M
 XF              X5F or XF5
 XH              X5H or XH5
```

In some applications requiring tricky calculations for the trajectory (for example in the adjoint code of the semi-lagrangian scheme), there are sometimes trajectory arrays, the name of ones ends by 6, 7.

Most of saving in trajectory buffers is now done by in-lined code. The code still has a limited number of routines saving data in trajectory buffers (examples: **STORE_TRAJ_PHYS_LAYER**, **STORE_MAIN_TRAJ**, **WRPHTRAJT**, **WRPHTRAJM**, **WRPHTRAJTM**, **WRPHTRAJTM_NL**, **WRITE_GRID_TRAJ**).

Where trajectory saving is done in the direct code (simplified organigramme):

```
CNT4 -> STEP0 ->
 * Inverse transforms
 * SCAN2M ->
   - STORE_MAIN_TRAJ
   - in-lined writing on trajectory buffer PTRAJ.
   - GP_MODEL_HEAP or GP_MODEL_STACK -> GP_MODEL (grid-point calculations) ->
     * CPG_DRV -> CPG ->
       - CPG_GP
       - MF_PHYS (MF physics) ->
         * in-lined writing on trajectory buffer PTRAJ_PHYS.
         * WRPHTRAJM
       - CPG_DYN ->
         * LACDYN or CPEULDYN
         * in-lined writing on trajectory buffer PTRAJ_SLAG (SL calculations for NTRSLTYPE=2 only)
       - CPG_END ->
         * in-lined writing on trajectory buffer PTRAJ_PHYS.
 * Direct transforms
 * Spectral calculations
```

## 2.3 Getting the trajectory in the tangent linear and the adjoint codes.

Most of reading from trajectory buffers is now done by in-lined code. The code still has a limited number of routines reading data in trajectory buffers (examples: **GET_TRAJ_SPEC**, **GET_TRAJ_GRID**,

**GET_TRAJ_PHYS**, **RDPHTRAJT**, **RDPHTRAJM**, **RDPHTRAJTM**, **RDPHTRAJTM_NL**).

Where trajectory recoverage is done in the tangent linear code (simplified organigramme):

```
CNT4TL ->
 - GET_TRAJ_SPEC then STEPO (inverse transforms only) if LTRAJGP=F: fills GMVS, GMV5S, GFL5
 - STEPOTL ->
   * Inverse transforms
   * SCAN2MTL ->
     - GET_TRAJ_GRID if LTRAJGP=T: fills GMVS, GMV5S, GFL5
     - GP_MODEL_TL (grid-point calculations) ->
       * CPG_DRV_TL -> CPGTL ->
         - CPG5_GP ->
           * RDPHTRAJM
           * in-lined reading from some trajectory buffers PTRAJ...
         - CPG_GP_TL
         - MF_PHYSTL (MF physics)
         - CPG_DYN_TL ->
           * LACDYNTL or CPEULDYNTL
           * in-lined reading from trajectory buffers PTRAJ_SLAG (SL calculations for NTRSLTYPE=2 only).
         - CPG_END_TL
   * Direct transforms
   * TL of spectral calculations
```

Remark: no trajectory is required in the spectral calculations because they are linear calculations (TL = direct code).

Where trajectory recoverage is done in the adjoint code (simplified organigramme):

```
CNT4AD ->
 - GET_TRAJ_SPEC then STEPO (inverse transforms only) if LTRAJGP=F: fills GMVS, GMV5S, GFL5
 - STEPOAD ->
   * AD of spectral calculations
   * Inverse transforms (AD of direct transforms)
   * SCAN2MAD ->
     - in-lined reading from trajectory buffer PTRAJEC, if LTRAJGP=T: fills GMVS, GMV5S, GFL5
     - GP_MODEL_AD (grid-point calculations) ->
       * CPG5 ->
         - in-lined reading from trajectory buffers PTRAJ_SLAG (SL calculations for NTRSLTYPE=2 only).
       * CPG_DRV_AD -> CPGAD ->
         - CPG5_GP ->
           * RDPHTRAJM
           * in-lined reading from some trajectory buffers PTRAJ...
         - CPG_END_AD
         - CPG_DYN_AD
         - MF_PHYSAD (MF physics)
         - CPG_GP_AD
   * Direct transforms (AD of inverse transforms)
```

Remarks:

- No trajectory is required in the spectral calculations because they are linear calculations (TL = direct code).
- Calling **CPG5** could be a possible option in the TL code also (we cannot avoid it in the AD code).

## 2.4 Buffers used for trajectory.

They are now in module **YOMTRAJ**, in specific structures. The different buffers are attributes of **TRAJEC**.

Modules **trajectory_mod.F90**, **traj_main_mod.F90**, **traj_physics_mod.F90**, **traj_semilag_mod.F90** and **traj_surface_mod.F90** contains routines allocating or deallocating attributes of **YOMTRAJ** structures; they also contain some routines reading or writing on buffers.

**It is important to say that, under STEPO/TL/AD, there should be no access to the trajectory structure through the module YOMTRAJ. The only allowed way to use the trajectory there is through the dummy arguments, in order to be compliant with the OOPS design.**

# 3 Management of external trajectory: the option LTRAJHR.

## 3.1 Writing the trajectory on a file.

For some applications it is useful to write the trajectory on a file, in the current resolution or in a lower resolution. This application is possible in a forecast (configuration 1), for example with screening or with trajectory update (in a 4DVAR assimilation). In this case not only the model variables are computed at each instant, but they are saved on a file (currently a GRIB file with a specific structure, one file per layer) via the routines **STORE_MAIN_TRAJ** (upper air fields and surface hydrostatic pressure) and some in-lined code (surface fields). In this case variables **LIFSTRAJ** and **LTRAJHR** are set to .T. . These routines call **WRITE_GRID_TRAJ** to write grid-point data and **WRITE_SPEC** to write spectral data. Some memory transfers are done by calling **COPY_SPA2SPEC** and **COPY_SPEC2SPA**.

Data is written at the resolution **NSMAX_TRAJ** which is assumed to be lower or equal to the model resolution **NSMAX**. **NSMAX_TRAJ** is assumed to be the truncation of the model which will read these data. The timestep of the model writing the data **TSTEP** is assumed to divide the timestep of the model reading the data **TSTEP_TRAJ**.

The way of converting data from resolution **NSMAX** to **NSMAX_TRAJ** is simply done by removing coefficients in spectral data (and doing some simple interpolations for grid-point data which have no spectral representation): this technique can work only in a non stretched and a non tilted geometry.

An alternate way, applicable to a subset of fields, is to do horizontal interpolations in grid-point space (call to routines **GRID_BILINEAR**, **GRID_BICUBIC**, **GRID_BICONSERV**).

The following trajectory fields can be interpolated:

- Upper air spectral fields.
- Upper air grid-point fields (**GFL**, **GMV**, **GMVS**).
- Surface fields: **CST** (constant fields), **SRFC** (surface variables).

The other trajectory fields (for example: fields for SL scheme (SLAG), fields for physics (**PHYS**, **PHYS_TLAD**, **RAINGG_TLAD**)) cannot be interpolated.

Where trajectory saving and writing is done in the direct code (simplified organigramme):

```
CNT4 -> STEPO ->
 * Inverse transforms
 * SCAN2M ->
   - (writing on PTRAJEC%CST)
   - STORE_MAIN_TRAJ ->
     * WRITE_GRID_TRAJ
     * COPY_SPA2SPEC
     * WRITE_SPEC
   - writing on PTRAJEC%SRFC
   - GP_MODEL_HEAP or GP_MODEL_STACK -> GP_MODEL (grid-point calculations) ->
     * CPG_DRV -> CPG ->
       - CPG_GP
       - MF_PHYS (MF physics) ->
         * (writing on PTRAJ_PHYS)
         * (WRPHTRAJM)
       - CPG_DYN ->
         * LACDYN or CPEULDYN
         * (writing on PTRAJ_SLAG) (SL calculations for NTRSLTYPE=2 only)
       - CPG_END ->
         * (writing on PTRAJ_PHYS)
 * Direct transforms
 * Spectral calculations
```

In parenthesis we have put the trajectory savings which are required when a TL or an adjoint code is called after that, but which are not done in our case. We can see that all the trajectory savings are now concentrated under **SCAN2M**.

## 3.2 Reading the trajectory on a file.

For some configurations using TL and adjoint code it is possible to replace the trajectory computed by a trajectory read on a file (GRIB or ARPEGE file), at all timesteps or for a subset of timesteps. This is possible at least for configuration 131 (4DVAR minimisation), but such an option could be implemented also in configurations 601 or 801. In this case variable **LTRAJHR** is set to .T. . Readings on this file are done by calling the routines **READ_TRAJ_GRID** (grid-point upper air fields and surface hydrostatic pressure), **READ_TRAJ_SPEC** (spectral upper air fields and surface hydrostatic pressure), and some in-lined code (surface fields) which put the data read in the **TRAJEC** buffer. Additionally the corresponding **GET_TRAJ....** routines are called to put data in **SPA5**, **GMV5**, **GMV5S**, **GFL5**, etc. These routines call **READ_GRID_TRAJ** to read grid-point data (reads GRIB files, the ARPEGE file version of this routine (**READ_GRID_TRAJ_FROMFA**) is not yet coded, and for surface fields the ARPEGE file version of this routine (**READ_SURFGRID_TRAJ_FROMFA**) is currently incomplete) and **READ_SPEC** (GRIB files, if **LARPEGEF_TRAJHR**=.F.) or **READ_SPEC_FROMFA** (ARPEGE files, if **LARPEGEF_TRAJHR**=.T.) to read spectral data. Some memory transfers are done by calling **COPY_SPA2SPEC** and **COPY_SPEC2SPA**.
This is equivalent to run the direct model with a resolution, then the TL or the adjoint model with a lower resolution, but currently it is not possible to do that in the same job!

Where trajectory reading is done in the direct code (simplified organigramme):

```
CNT4 ->
 * Part 3.6:
   - READ_TRAJECTORY ->
     * READ_SURFGRID_TRAJ_FROMFA (FA, case LARPEGEF_TRAJHR=T)
     * READ_SURFGRID_TRAJ (GRIB, case LARPEGEF_TRAJHR=F)
     * READ_TRAJ_GRID ->
       - READ_GRID_TRAJ if LTRAJGP=T (GRIB)
       - READ_SPEC (!!!) if LTRAJGP=F (GRIB, case LARPEGEF_TRAJHR=F)
       - READ_SPEC_FROMFA (!!!) if LTRAJGP=F (FA, case LARPEGEF_TRAJHR=T)
     * READ_TRAJ_SPEC ->
       - READ_SPEC (GRIB, case LARPEGEF_TRAJHR=F)
       - READ_SPEC_FROMFA (FA, case LARPEGEF_TRAJHR=T)
       - COPY_SPEC2SPA
   - GET_TRAJ_SPEC
 * Part 3.10: STEPO ->
   - Inverse transforms
   - SCAN2M ->
     * read from PTRAJEC%SRFC
     * read from PTRAJEC%MAIN (only to get some grid-point GFL fields)
     * GP_MODEL_HEAP or GP_MODEL_STACK -> GP_MODEL (grid-point calculations)
   - Direct transforms
   - Spectral calculations
 * Part 4:
   - READ_TRAJECTORY ->
     * READ_SURFGRID_TRAJ (GRIB)
     * READ_TRAJ_GRID ->
       - READ_GRID_TRAJ if LTRAJGP=T (GRIB)
       - READ_SPEC (!!!) if LTRAJGP=F (GRIB, case LARPEGEF_TRAJHR=F)
       - READ_SPEC_FROMFA (!!!) if LTRAJGP=F (FA, case LARPEGEF_TRAJHR=T)
     * READ_TRAJ_SPEC ->
       - READ_SPEC (GRIB, case LARPEGEF_TRAJHR=F)
       - READ_SPEC_FROMFA (FA, case LARPEGEF_TRAJHR=T)
       - COPY_SPEC2SPA
```

Additional remarks:

- At METEO-FRANCE, the trajectory file at instant 0 should not be read.
- Trajectory for GFL: assumes that all GFL are spectral ones (**LREADGPTRAJ**=F) or grid-point ones (**LREADGPTRAJ**=T). In the trajectory there is currently no possibility to mix spectral and grid-point GFL.
- Developments for **LTRAJHR_SURF** are currently not complete for METEO-FRANCE applications.

# 4 Modules and namelists.

These modules are auto-documented so description of each variable is provided in the code source. We can recall here the most important variables to know for each module:

- **YOMCT0** (0-level control):
  - LIFSTRAJ.
  - LARPEGEF_TRAJHR, LARPEGEF_TRAJBG, LARPEGEF_RDGP_TRAJHR, LARPEGEF_RDGP_TRAJBG.

  Some of these variables are in namelist **NAMCT0**.
- **YOMIOP** (trajectory features which have not yet been updated according to the new design). Some of these variables are in namelist **NAMTRAJP**.
- **YOPHNC**: LETRAJP and LETRAJPT. Some of these variables are in namelist **NAMTRAJP**.
- **YOMSIMPHL** (simplified MF physics). In particular LTRAJPS, LTRAJPST, LPROCLDTL. Some of these variables are in namelist **NAMSIMPHL**.
- **YOMTRAJ** (buffer TRAJEC and variables to control trajectory). Some of these variables are in namelist **NAMVAR**. Some variables linked with timestep and resolution are currently in **NAMDYN**.
- **YOMVWRK** and **NAMVWRK**: NTRSLTYPE
- **YOMGMV5** (trajectory for GMV grid-point arrays).
- **YOMGFL5** (trajectory for GFL grid-point arrays).
- **YOMSP5** (trajectory for spectral arrays).