The transformation package for ALADIN

G. Radnoti, April 2001

For the CY24 of ARPEGE/IFS a new transformation package has been created for the global model (TFL). We tried to follow this structure and we created an ALADIN counterpart of the package (TAL). The adaptation has been done and cycle AL15 of ALADIN has been created in a "transform package-conform" manner. I see two main advantages of the new structure:

- The package can be called independently from the model to perform the desired spectral transformations;

- All the spectral transformations related setup, inter-processor communicators, internal array structures and internal functions have been separated from the model; a model developper only has to know the interfaces to the package.

In some sense one can imagine the transformation package similar to the XRD auxiliary library. Indesd, the package is put to separate clearcase project (TAL24 for ALADIN as a counterpart of TFL24 of ARPEGE) and binary library. The ALADIN version is complete in the sense that the libray contains all elements, even those that are identical to the ARPEGE version. The convention that we followed is that if a TFL routine had been modified for ALADIN, this got a new name (e.g. PRFI2_MOD.F90==>EPRFI2_MOD.F90). If a routine had not been modified, it was copied from TFL to TAL without change on the same name. Therefore the loading of TFL is necessary only for configuration e927, for every other ALADIN configuration TAL is complete.

The clearcase project of the package contains directories external, interface, module and programs. For a developper who simply wants to use the package as it is from ALADIN only external directory is to be studied: this contains the routines through which the package is available, so it is the interface for plugging the package to the model. Directory programs contain two example programs to illustrate how the package can be activated independently from the model. This is at the same time a separate test environment of the package. Therefore in this guide i will mostly concentrate on the plug-in interfaces of external directories. I will also mention some new features in the module procedures, but it is rather for those who may in the future develop the package itself.

To study how the package can be used one may take a look at of the two test programs, one performing pure inverse and direct transformations and the other performing adjoint transformations. One can also take a look at how the package is activated in the ALADIN model. The example programs are very easy to understand so i will rather concentrate on the model environment: As mentioned above, only the routines of directory external are to be called from "outside". The first thing to do is to to call the setup routines of the package. In the model it is done in rotine SUETRANS. SETUP_TRANS0 and ESETUP_TRANS are the two setup routines to be called from the model to setup the transformation environment. All the arguments are input ones and most of them are optional (if an optional argument is not passed, some default setting will be used). It is important to note that the package is extensively using optinal dummy arguments.

OPTIONAL ARGUMENTS:

Since it was so far not very widely used, here i summarize some rules of using optional arguments: the syntax is very simple, e.g: LOGICAL ,OPTIONAL,INTENT(IN) :: LDALLOPERM

When one calls a routine which has optional argument one can use the traditional way of calling by listing all argument:

for example we have the routine EFTDIR_CTL that has (some) optional arguments. The following example shows the traditional way mixed with the way using the optional feautre:

```
CALL EFTDIR_CTL(PGP,IF_UV_G,IF_SCALARS_G,IF_GP,IF_FS,&
 & KVSETUV=IVSETUV(ISTUV_G:IENUV_G),&
 & KVSETSC=IVSETSC(ISTSC_G:IENSC_G),KPTRGP=IPTRGP,CPL_PROC=CPL_PROC)
```

Going from the left to the right in the argument list as long as we dont skip any optional argument we can use the traditional way. As soon as we skip an optinoal argument we have to explicitly identify which passed argument corresponds to which argument in the called routine: KPTRGP=IPTRGP means for example that we pass to the argument KPTRGP of EFTDIR_CTL the variable IPTRGP. Inside the routine having optional arguments

IF (PRESENT(CPL_PROC)) ...is used to decide whether the optional argument CPL_PROC has been passed to the routie or not. There is one more interesting feature of optional argument to be mentioned here: still using the sam example, if EFTDIR_CTL is passing furhter down its optional argument KPTRGP : CALL TRGTOL (ZGTF,PGP,KF_FS,KF_GP,IVSET,KPTRGP), and if EFTDIR_CTL was called without KPTRGP, the optional argument KPTRGP will not be present in TRGTOL even if formally the calling syntax contained it.

By executing the tranformation package setup, the package initialized its basic variables and the message passing environment. As compared to earlier model cycles, subroutines SUSTAONL,SU(E)MPLAT were fully moved to the package and also some parts of SU(E)MP. The modules and variables of the package are not in the scope of the model. However, the model still needs some variables that are initialized in the package. The model can get this information through the routine ETRANS_INQ, that passes back the required variables by optional dummy arguments. (E)TRANS_INQ is typically called from SU(E)MP.

If the setup has been done, one can call the main transformation routines. These are EDIR_TRANS, EINV_TRANS and their adjoint counterparts. (Inside the package fullpos is not treated directly, since fullpos transformations are done with the same geometry as model transformations.) The two main transformation routines (and their adjoint versions) can be called directly from the model. However in the transformation directory of the model there are embedding routines that customize these calls to the model environment.Therefore for example in STEPO the calls to the package are arranged through these model routines under directory transformation (e.g. ETRANSDIRH-->ETRANSDIR_MDL-->EDIR_TRANS). One is not obliged to do always like this and indeed in some cases we call EDIR_TRANS, EINV_TRANS "directly". I have the feeling that these are the routines that an ALADIN developper will most likely use. They are called at each time-step from STEPO either for the model variables or for fullpos fields. Additionally, transformations are called at setup (orography fitting, derivatives of orography, wind-->vorticity, divergence conversion, map factor fitting) and EINV_TRANS is called for creating gridpoint space coupling buffer from ELSWA3 (some other occurences: ECHKEVO gridpoint evolution diagnostics, history file creation: vor,div-->u,v, ...).

The ALADIN package differs from the ARPEGE version basically in the following features:

- somewhat different setup of parallelization;

- FFT instead of Legendre transformation in meridional direction;

- Elliptic truncation;

- Coupling

The first three components were always present and they are coming from the nature of the problem. Coupling is a new issue in the sense that in principle it has not much to do with spectral transformations. However the structure of the code brings this issue close to the package because it would be difficult to perform coupling on normal gridpoint space GPP(NPROMA,IFIELDS,NGPBLKS) type arrays. With the use of the package all the other gridpoint data structures remain hidden for the model, the transformation package on input gets the blocked data structure and on output provides (NFLEV,NSPEC2) structure spectral fields (at inverse transforms obviously it is the opposite). From this on can also guess that the TRGTOL,TRLTOG, TRLTOM, TRMTOL transposition routines had to become package modules and they are not allowed to be directly called by the model. We had two options to solve this problem (in a bit inconsistent way we have implemented both options, one for the coupling itself and the other one for the creation of the coupling buffer and in future it can be decided which approach should stay in the code)

- We pass a caller of the coupling routine to the package as an optional external procedure and at the right place the package calls this external in the case if it was passed down. Then we can achive that the package calls coupling after TRLTOG, i.e. after the gridpoint array is transposed from the (NPROMA,IFIELDS,NGPBKS) structure to (NGPTOT,IFIELDS) structure. This solution has been chosen for coupling (see in STEPO that ECOUPL1 is passed down to the package, as an external procedure name);

- The second option is simpler but less general: before getting into the labirynth of the package we mimic the functionality of TRLTOG with a simple loop. This works perfectly as long as there is no latitude splitting and B-level parallelization. This option was made for creating GT3BUF coupling buffer.

Let us see how we should activate EDIR_TRANS or EINV_TRANS:

In the simplest case one can do it by the use of ESPEREE,EREESPE. The interface to these routines has not changed, so this works as before. However, EINV_TRANS is more clever than ESPEREE and EDIR_TRANS is more clever than EREESPE and we may benefit from this. Direct access to these package routines provides the possibility of computing derivatives, uv<-->vordiv conversions simultaneously. As an impressive example i will use below SUEOROG (initialization of the grid-point orography fields of the model) to illustrate how the life can become easier with the package. This routine basically starts from spectral orography, makes invesre transform to initialize gridpoint orography, then computes derivatives from spectral orography, transforms them into gridpoint space and in the case of non-hydrostatic it does the sam for second gerivatives. The old code looked like

```
.

CALL ESPEREE(1,1,PSPOR,ZGPOR) ; OROG(1:NGPTOT) = ZGPOR

!* 1.2 INITIALIZE GRIDPOINT 1ST ORDER DERIVATIVES OF OROGRAPHY

! ---------------------------------------------------------

DO JMLOC=1,NUMP
  IM=MYMS(JMLOC)
  ZFFTUI(:,:)=_ZERO_ ; ZFFTV(:,:)=_ZERO_ ; ZFFTT(:,:)=_ZERO_ ; ZFFTD(:,:)= &
   & _ZERO_
  CALL EPRFI1B(IM,ZFFTU,1,2,ZSPEC0,1,1)
  CALL EPRFI1B(IM,ZFFTV,1,2,ZSPOR,1,1)
  CALL EOROGGRAD(1,IM,JMLOC,1,ZFFTU,ZFFTV,ZFFTT,ZFFTD,ZU,ZV,.FALSE.)
  CALL EUPDSPB(IM,1,1,ZFFTT,1,2,ZSPVOR)
  CALL EUPDSPB(IM,1,1,ZFFTD,1,2,ZSPDIV)
ENDDO
CALL ESPEREE(1,1,ZSPVOR,ZGPORL) ; OROGL(1:NGPTOT) = ZGPORL
CALL ESPEREE(1,1,ZSPDIV,ZGPORM) ; OROGM(1:NGPTOT) = ZGPORM

!* 1.3 INITIALIZE GRIDPOINT 2ND ORDER DERIVATIVES OF OROGRAPHY

! ---------------------------------------------------------

IF (LNHDYN) THEN

  CALL EREESPE(1,1,ZSPVOR,ZGPORL)
  DO JMLOC=1,NUMP
    IM=MYMS(JMLOC)
    ZFFTU(:,:)=_ZERO_ ; ZFFTV(:,:)=_ZERO_ ; ZFFTT(:,:)=_ZERO_ ; ZFFTD(:,:)= &
     &_ZERO_
    CALL EPRFI1B(IM,ZFFTU,1,2,ZSPEC0,1,1)
    CALL EPRFI1B(IM,ZFFTV,1,2,ZSPVOR,1,1)
    CALL EOROGGRAD(1,IM,JMLOC,1,ZFFTU,ZFFTV,ZFFTT,ZFFTD,ZU,ZV,.FALSE.)
    CALL EUPDSPB(IM,1,1,ZFFTT,1,2,ZSPDXX)
  ENDDO
  CALL ESPEREE(1,1,ZSPDXX,ZGPORL) ; OROGLL(1:NGPTOT) = ZGPORL

  CALL EREESPE(1,1,ZSPDIV,ZGPORM)
  DO JMLOC=1,NUMP
    IM=MYMS(JMLOC)
    ZFFTU(:,:)=_ZERO_ ; ZFFTV(:,:)=_ZERO_ ; ZFFTT(:,:)=_ZERO_ ; ZFFTD(:,:)= &
     &_ZERO_
    CALL EPRFI1B(IM,ZFFTU,1,2,ZSPEC0,1,1)
    CALL EPRFI1B(IM,ZFFTV,1,2,ZSPDIV,1,1)
    CALL EOROGGRAD(1,IM,JMLOC,1,ZFFTU,ZFFTV,ZFFTT,ZFFTD,ZU,ZV,.FALSE.)
    CALL EUPDSPB(IM,1,1,ZFFTT,1,2,ZSPDXX)
    CALL EUPDSPB(IM,1,1,ZFFTD,1,2,ZSPDYY)
  ENDDO
  CALL ESPEREE(1,1,ZSPDXX,ZGPORL) ; OROGLM(1:NGPTOT) = ZGPOR
  CALL ESPEREE(1,1,ZSPDYY,ZGPORM) ; OROGMM(1:NGPTOT) = ZGPORM
```

And here it is how the same functionality can be reached by the package:

```
CALL EINV_TRANS(PSPSCALAR=ZSPOR,PGP=ZRMOUT,LDSCDERS=.TRUE.,KPROMA=NGPTOT)
OROG(1:NGPTOT)=ZRMOUT(1:NGPTOT,1,1)
OROGM(1:NGPTOT)=ZRMOUT(1:NGPTOT,2,1)
OROGL(1:NGPTOT)=ZRMOUT(1:NGPTOT,3,1)

IF (LNHDYN) THEN
  ZRMIN2(1:NGPTOT,1,1)=OROGM(1:NGPTOT)
  ZRMIN2(1:NGPTOT,2,1)=OROGL(1:NGPTOT)
  ZSPOR2=_ZERO_
  CALL EDIR_TRANS(PSPSCALAR=ZSPOR2,PGP=ZRMIN2,KPROMA=NGPTOT)
  ZRMOUT2=_ZERO_
  CALL EINV_TRANS(PSPSCALAR=ZSPOR2,PGP=ZRMOUT2,KPROMA=NGPTOT,LDSCDERS=.TRUE.)
  OROGMM(1:NGPTOT)=ZRMOUT2(1:NGPTOT,3,1)
  OROGLM(1:NGPTOT)=ZRMOUT2(1:NGPTOT,4,1)
  OROGLL(1:NGPTOT)=ZRMOUT2(1:NGPTOT,6,1)
ENDIF
```

One third ofthe code lines, more expressive function names, etc.... But one has to know a few things.

Why KPROMA=NGPTOT? ???

Because EINV_TRANS gives you back an (NPROMA,IFIELDS,NGPBLKS) array. But NPROMA is not fixed in the package, you can pass as KPROMA anything, and with the choice KPROMA = NGPTOT you will get back the most conviniently shaped (NGPTOT,IFIELDS,1) array.

Why for example OROGLL(1:NGPTOT)=ZRMOUT2(1:NGPTOT,6,1), what is the magic index 6????

You just have to take a look at of the comment lines of EINV_TRANS and EDIR_TRANS that explain you very clearly (by Mats Hamrud) the order of gridpoint fields as requested by EDIR_TRANS and as provided by EINV_TRANS.

What else to be careful with????

You can see from the above example that number of fileds and things like that are not passed to the package. Such things are often computed in the package by the use of UBOUND intrinsic function. For example UBOUND(PGP,2) is the size of PGP array in its second dimension. It is important to know it because you MAY NOT pass to the package for example a ZSPVOR(NSPEC2) array instead of a ZSPVOR(NSPEC2,1) array. The package requires exact shape matching of the passed arguments (you fail already at compilation level if you dont keep this rule).

Directory external contains furthermore a spectral norm diagnostic routine (this is not a replacement of the model spectral diagnostics, but a useful tool for validating the package and external applications). Also there are 4 routines for distributing and gathering gridpoint and spectral data. These 4 routines are neither used by the model, but they are useful in extenral applications (see how they are applied by the two test programs).