A Brief Tutorial on 4DVAR and Adjoint Capabilities in GSI (June2010)

Ricardo Todling,
Yannick Trémolet*, and Jing Guo
Global Modeling and Assimilation Office

GSI Tutorial, NCAR, 28-30 June 2010

This presentation is a brief guide to the modifications that took place in GSI to accommodate implementation of both 4DVAR and the direct adjoint of GSI.

But nothing comes for free:
- 4DVAR-capable means: TLM and ADM codes for GCM must be available
- Use of GSI-adjoint beyond an analysis sensitivity tool requires GCM ADM

*While on leave from ECMWF in 2007.

Introductory Remarks

- This presentation reports no the status of GSI 4DVAR and GSI-Adjoint as implemented in its upgraded June-2010 version.
- The equations appearing along snapshots of code are for most part cryptic and meant to simply serve as a guide to finding where things happen – not a detailed formulation.
- The excerpt of codes appearing here have been slightly manipulated for presentation clarity; the notation ... indicates code bypassed for clarity

4DVAR- and Adjoint-Related Features

- Observer capability
- Observation time binning
- Separation between control and state spaces
- Digital filter
- Various sqrt(B)-based minimization options
  - Vanilla conjugate gradient
  - Quasi-Newton (L-BFGS, m1qn3)
  - Lanczos
- Adjoint analysis capability

Code-Related Components

- state- and control-vectors
- obsdiags structure
- analysis time counted from beginning of time window
- split intall routine into int3dvar & intjo
- split stpcalc routine into stp3dvar & stpjo
- add evalJ: evalgrad/evaljo/evaljc/evaljcdfi/evalqlim
- add obs_sensitivity to handle GSI adjoint

Four-dimensional Variational Approach

The general cost function of the variational formulation

$$\begin{aligned} J(\mathbf{x}) &= \frac{1}{2}(\mathbf{x}_0 - \mathbf{x}^b)^T \mathbf{B}^{-1}(\mathbf{x}_0 - \mathbf{x}^b) + J_x \\ &+ \frac{1}{2}\sum_{k=0}^{K}[\mathbf{h}(\mathbf{x}_k) - \mathbf{y}_k]^T \mathbf{R}_k^{-1}[\mathbf{h}(\mathbf{x}_k) - \mathbf{y}_k] \\ &+ \frac{1}{2}\sum_{k=1}^{K}[\mathbf{m}(\mathbf{x}_k) - \mathbf{x}_k]^T \mathbf{Q}_k^{-1}[\mathbf{m}(\mathbf{x}_k) - \mathbf{x}_k] \end{aligned}$$

where

- $\mathbf{x} \equiv [\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_K]^T$ is a 4d state vector;
- $\mathbf{h}_k$ and $\mathbf{m}_k$ are the nonlinear observation and dynamical model operators, respectively;
- $\mathbf{B}$, $\mathbf{Q}_k$, and $\mathbf{R}_k$ are the background, model, and observation error covariances, respectively.
- Strong constraint formulation: $\mathbf{Q}_k \to \infty$;
- Weak constraint formulation, $\mathbf{Q} \neq \mathbf{0}$ accounts for imperfections in the model $\mathbf{m}$;
- J_x represents extra constraint (e.g., balance).

Strong Constraint Incremental 4DVAR

For simplicity consider now the strong constraint case. In incremental 4DVAR the cost function at the j-th iteration is

$$\begin{aligned} J_j(\delta\mathbf{x}_j) &= \frac{1}{2}(\delta\mathbf{x}_j - \delta\mathbf{x}_j^b)^T\mathbf{B}^{-1}(\delta\mathbf{x}_j - \delta\mathbf{x}_j^b) \\ &+ \frac{1}{2}\sum_{k=0}^{K}(\mathbf{H}_{j,k}\mathbf{M}_{j,k}\delta\mathbf{x}_j - \mathbf{d}_{j,k})^T\mathbf{R}^{-1}(\mathbf{H}_{j,k}\mathbf{M}_{j,k}\delta\mathbf{x}_j - \mathbf{d}_{j,k}) \end{aligned}$$

where $\mathbf{d}_{j,k} \equiv \mathbf{y}_k - \mathbf{h}_k(\mathbf{m}_k(\mathbf{x}^b))$, $\delta\mathbf{x}_j^b \equiv \mathbf{x}^b - \mathbf{x}_{j-1}$, and

- $\delta\mathbf{x}_j \equiv \mathbf{x}_j - \mathbf{x}_{j-1}$ is the control variable;
- The inner loop minimization of J_j can be solved by
  - Conjugate gradient
  - Quasi-Newton (such as L-BFGS)
  - Lanczos
- Conditioning of the J_j minimization is determined by the Hessian $\nabla^2 J_j = \mathbf{B}^{-1} + \sum_k \mathbf{M}_{j,k}^T\mathbf{H}_{j,k}^T\mathbf{R}^{-1}\mathbf{M}_{j,k}\mathbf{H}_{j,k}$, which spectrum is such that a good preconditioning is essential, particularly in 4DVAR.

gsisub

- The main entry point of GSI, deciding whether to run the observer or the minimization

In 4dvar:

This is part of outer l during NL model integration

This becomes the inner loop

```
! Complete setup and execute external and internal minimization loops
if (lobserver) then
  if(init_pass) call observer_init()
  call observer_run(init_pass=init_pass,last_pass=last_pass)
  if(last_pass) call observer_finalize()
else
   call glbsoi(mype)
endif
```

- The observer can be called by the forward model (GCM) via the available ESMF interface (or some other user-designed interface)
- The observer:
  1. called at the frequency of background availability
  2. in the initial pass, observer reads observations for the entire time window
  3. once two time slots of the background are available, the observer calculates the residuals for observations falling within the time interval of two consecutive background fields present in memory
  4. the calculation in (3) continues until the last pass, when last background fields is loaded to memory and the clock hits the end of the assimilation interval

Where do things happen in the 4DVAR-capable GSI?

▷ Define a suitable square-root of $\mathbf{B} = \mathbf{L}\mathbf{L}^T$

- `ckgcov`
- `ckgcov_ad`

(from David Parrish)

▷ Introduce a control variable change: $\delta\mathbf{x} = \mathbf{L}\delta\lambda$

- `control2model`
- `model2control`

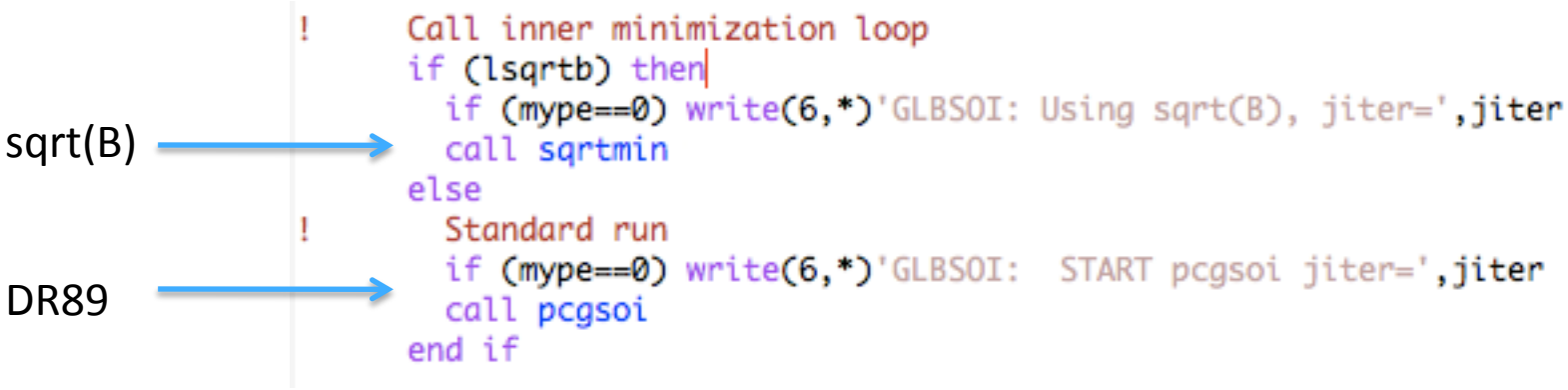
▷ The analysis solver:

1. The j-th inner loop of the incremental analysis solves $(\mathbf{I} + \mathbf{L}^T\mathbf{M}_j^T\mathbf{H}_j^T\mathbf{R}^{-1}\mathbf{H}_j\mathbf{M}_j\mathbf{L})\lambda_j = \mathbf{L}^T\mathbf{M}_j^T\mathbf{H}_j^T\mathbf{R}^{-1}\mathbf{d}_j + \mathbf{b}_j$
2. Cost function at the j-th iteration: $\hat{J}_j = \frac{1}{2}\lambda_j^T\lambda_j + \frac{1}{2}(\mathbf{H}_j\mathbf{M}_j\mathbf{L}\lambda_j - \mathbf{d}_j)^T\mathbf{R}^{-1}(\mathbf{H}_j\mathbf{M}_j\mathbf{L}\lambda_j - \mathbf{d}_j)$
3. The ideal preconditioning is given by the sqrt of the inverse Hessian, $\nabla^2\hat{J}_j = \mathbf{I} + \mathbf{L}^T\mathbf{M}_j^T\mathbf{H}_j^T\mathbf{R}^{-1}\mathbf{M}_j\mathbf{H}_j\mathbf{L}$, operator form of rhs allows getting estimate of this matrix.

- `evaljgrad`

glbsoi

- The main change here is a split between two preconditioning strategies:
  I. the B-precond of Derber and Rosati (1989)
  II. the square-root(B)-precond



GSI Gridded Structures

The original implementation of GSI-4DVAR includes clear separation of the state and control vectors; the first, relates to what the observation operators use, H(x); the second, relates to the definition of the cost function J(z)

State Vector (Q1FY10)

```
type state_vector
  real(r_kind), pointer :: values(:) => NULL()

  real(r_kind), pointer :: u(:)   => NULL()
  real(r_kind), pointer :: v(:)   => NULL()
  real(r_kind), pointer :: t(:)   => NULL()
  real(r_kind), pointer :: q(:)   => NULL()
  real(r_kind), pointer :: oz(:)  => NULL()
  real(r_kind), pointer :: cw(:)  => NULL()
  real(r_kind), pointer :: p(:)   => NULL()
  real(r_kind), pointer :: sst(:) => NULL()

  logical :: lallocated = .false.
end type state_vector
```

Control Vector (Q1FY10)

```
type control_state
  real(r_kind), pointer :: values(:) => NULL()
  real(r_kind), pointer :: st(:)  => NULL()
  real(r_kind), pointer :: vp(:)  => NULL()
  real(r_kind), pointer :: t(:)   => NULL()
  real(r_kind), pointer :: rh(:)  => NULL()
  real(r_kind), pointer :: oz(:)  => NULL()
  real(r_kind), pointer :: cw(:)  => NULL()
  real(r_kind), pointer :: p(:)   => NULL()
  real(r_kind), pointer :: sst(:) => NULL()
end type control_state

type control_vector
  integer(i_kind) :: lencv
  real(r_kind), pointer :: values(:) => NULL()
  type(control_state), allocatable :: step(:)
  real(r_kind), pointer :: predr(:) => NULL()
  real(r_kind), pointer :: predp(:) => NULL()
  logical :: lallocated = .false.
end type control_vector
```

The desire to have GSI handle various scenarios motivated a generalization of these vectors and introduction of the so-called GSI_Bundle. Before, talking about the present form of the state and control vectors we give a brief introduction to the GSI_Bundle next.

The GSI_Bundle

- Concept follows from discussion with Arlindo da Silva and the ESMF concept of a Bundle, i.e., a collection of fields on a grid.
- Aims at being a general (for now) regular grid structure that can be used for multiple purposes: control, state, and guess vectors.
- Bundle has only very weak dependence on GSI (and should remain this way): kinds, constants, and for the time being, m_rerank.F90 (by Jing Guo).
- In the June-2010 version, GSI_Bundle is used to defined the state and control vectors, as well as the GSI_ChemBundle, which is a structure devoted to chemistry-related fields, including aerosols.
- In the future, we might consider using GSI_Bundle to handle the background (guess) fields as well.

The GSI_Bundle: Definition

```
type GSI_Bundle
   character(len=MAXSTR) :: name
   integer(i_kind) :: n1d=-1     ! number of 1-d variables
   integer(i_kind) :: n2d=-1     ! number of 2-d variables
   integer(i_kind) :: n3d=-1     ! number of 3-d variables
   integer(i_kind) :: NumVars=-1 ! total number of variables (n1d+n2d+n3d)
   integer(i_kind) :: ndim=-1    ! size of pointer values
   type(GSI_Grid)  :: grid
   type(GSI_1D),    pointer :: r1(:) => null()
   type(GSI_2D),    pointer :: r2(:) => null()
   type(GSI_3D),    pointer :: r3(:) => null()
   integer(i_kind), pointer :: ival1(:)  => null()
   integer(i_kind), pointer :: ival2(:)  => null()
   integer(i_kind), pointer :: ival3(:)  => null()
   real(r_kind),    pointer :: values(:) => null()
end type GSI_Bundle
```

This is what a more general grid may look like

```
type GSI_Grid                 ! simple regular grid for now
   integer(i_kind) :: im=-1   ! dim of 1st rank
   integer(i_kind) :: jm=-1   ! dim of 2nd rank
   integer(i_kind) :: km=-1   ! dim of 3nd rank
                              ! A more general grid would include
!!    integer(i_kind) :: ihalo  ! halo of 1st dim
!!    integer(i_kind) :: jhalo  ! halo of 2nd dim
!!    integer(i_kind) :: khalo  ! halo of 3nd dim (usually not needed)
!!    real(r_kind), pointer :: lat(:,:)   ! field of latitudes
!!    real(r_kind), pointer :: lon(:,:)   ! field of longitudes
!!    real(r_kind), pointer :: pm (:,:,:) ! field of mid-layer pressures
!!    real(r_kind), pointer :: pe (:,:,:) ! field of edge pressures
end type GSI_Grid
```

Very simple grid; should eventually come from outside of GSI_Bundle

Private types of Bundle: never to be made public

```
type GSI_1D
   character(len=MAXSTR) :: shortname          ! name, e.g., 'ps'
   character(len=MAXSTR) :: longname           ! longname, e.g., 'Surface Pressure'
   character(len=MAXSTR) :: units              ! units, e.g. 'hPa'
   real(r_kind), pointer :: q(:) => null()     ! rank-1 field
end type GSI_1D

type GSI_2D
   character(len=MAXSTR) :: shortname
   character(len=MAXSTR) :: longname
   character(len=MAXSTR) :: units
   real(r_kind), pointer :: q(:,:) => null()   ! rank-2 field
end type GSI_2D

type GSI_3D
   character(len=MAXSTR) :: shortname
   character(len=MAXSTR) :: longname
   character(len=MAXSTR) :: units
   logical               :: edge               ! edge field: 3rd dim is km+1
   real(r_kind), pointer :: q(:,:,:) => null() ! rank-3 field
end type GSI_3D
```

The GSI_Bundle: Particulars

```
type GSI_Bundle
   character(len=MAXSTR) :: name
   integer(i_kind) :: n1d=-1     ! number of 1-d variables
   integer(i_kind) :: n2d=-1     ! number of 2-d variables
   integer(i_kind) :: n3d=-1     ! number of 3-d variables
   integer(i_kind) :: NumVars=-1 ! total number of variables (n1d+n2d+n3d)
   integer(i_kind) :: ndim=-1    ! size of pointer values
   type(GSI_Grid)  :: grid
   type(GSI_1D),    pointer :: r1(:) => null()
   type(GSI_2D),    pointer :: r2(:) => null()
   type(GSI_3D),    pointer :: r3(:) => null()
   integer(i_kind), pointer :: ival1(:)  => null()
   integer(i_kind), pointer :: ival2(:)  => null()
   integer(i_kind), pointer :: ival3(:)  => null()
   real(r_kind),    pointer :: values(:) => null()
end type GSI_Bundle
```

Redundant part of GSI_Bundle

Fields in the bundle type(GSI_Bundle) :: Bundle can be referred to:

- e.g., as Bundle%r2%q
- or redundantly, e.g., as Bundle%values(IPNT) where IPNT relates to ival2, as explained in what follows (see also unit-tester)

The size of the Bundle is either the total number of variables NumVars and/or the total size of NumVars*grid, ndim, that is, size of Bundle%values

The GSI_Bundle: Functions

```
public GSI_Bundle            ! Bundle
public GSI_BundleCreate      ! Create a Bundle
public GSI_BundleDPlevs      ! dot product w/ possible "halo"
public GSI_BundleSum         ! dot product w/ possible "halo"
public GSI_BundleSet         ! Set Bundle
public GSI_BundleInquire     ! Inquire about Bundle contents
public GSI_BundleMerge       ! Merge two Bundles
public GSI_BundlePrint       ! Print contents of Bundle
public GSI_BundleGetPointer  ! Get pointer to variable
public GSI_BundleGetVar      ! Get contents of variable
public GSI_BundlePutVar      ! Put contents in variable
public GSI_BundleUnset       ! Unset Bundle
public GSI_BundleDestroy     ! Destroy Bundle
public assignment(=)         ! Assign to Bundle contents
public self_add              ! Add contents of bundles
public self_mul              ! Add contents of bundles
public gsi_bundlehadamard    ! Hadamard product of contents of two bundles
```

- Reference to the Bundle should be made via these functions
- Most these functions are overloaded, e.g., GSI_BundleCreate

```
interface GSI_BundleCreate
       module procedure create1_
       module procedure create2_
       module procedure create3_
end interface
```

→ General create (from scratch)
→ Create new Bundle from existing Bundle
→ Create new Bundle after Merging two Bundles

that is, it allows creation of Bundles in various ways

GSI Gridded Structures

State Vector

- The state vector is simply a GSI_Bundle, that is, a collection of fields defined via a resource file
- For example, it's initialization in allocate_state is

```
call GSI_GridCreate(grid,lat2,lon2,nsig)
write(bname,'(a)') 'State Vector'
call GSI_BundleCreate(yst,grid,bname,ierror, &
                      names2d=svars2d,&
                      names3d=svars3d,edges=edges)
```

where 2d- and 3d-fields are defined via a resource file, viz. table in anavinfo file:

```
state_vector::
!var     level  itracer amedge  source     funcof
 u         72      0      no     met_guess    u
 v         72      0      no     met_guess    vi
 tv        72      0      no     met_guess    tv
 tsen      72      0      no     met_guess    tv,q
 q         72      1      no     met_guess    q
 oz        72      1      no     met_guess    oz
 cw        72      1      no     met_guess    cw
 p3d       73      0      yes    met_guess    p3d
 ps         1      0      no     met_guess    p3d
 sst        1      0      no     met_guess    sst
::
```

Control Vector

- The control vectors is a type that collects various GSI_Bundles, Grids, and ancillary fields required by the cost function

```
type control_vector
   integer(i_kind) :: lencv
   real(r_kind), pointer :: values(:) => NULL()
   type(GSI_Grid)  :: grid_step
   type(GSI_Bundle), pointer :: step(:)
   type(GSI_Bundle), pointer :: motley(:)
   type(GSI_Grid)  :: grid_aens
   type(GSI_Bundle), pointer :: aens(:,:)
   real(r_kind), pointer :: predr(:) => NULL()
   real(r_kind), pointer :: predp(:) => NULL()
   logical :: lallocated = .false.
end type control_vector
```

- Depending on the CV definition, some pointers are never defined

GSI Ungridded (Obs) Structures

Obs_diag Structure

```
type obs_diag
   sequence
   type(obs_diag), pointer :: next => NULL()
   real(r_kind), pointer :: nldepart(:)    ! (miter+1)
   real(r_kind), pointer :: tldepart(:)    ! (miter)
   real(r_kind), pointer :: obssen(:)      ! (miter)
   real(r_kind) :: wgtjo
   integer(i_kind) :: indxglb
   integer(i_kind) :: nchnperobs           ! number of channels per observations
                                           !  (dummy, expect for radiances)
   logical, pointer :: muse(:)             ! (miter)
   logical :: luse

   integer(i_kind) :: idv,iob,ich     ! device id and obs index for verification
end type obs_diag
```

e.g., w_ob_type

```
type w_ob_type
   sequence
   type(w_ob_type),pointer :: llpoint => NULL()
   type(obs_diag), pointer :: diagu => NULL()
   type(obs_diag), pointer :: diagv => NULL()
   real(r_kind)    :: ures          !  u component residual
   real(r_kind)    :: vres          !  v component residual
   real(r_kind)    :: err2          !  surface pressure error squared
   real(r_kind)    :: raterr2       !  square of ratio of final obs error
                                    !  to original obs error
   real(r_kind)    :: time          !  observation time in sec
   real(r_kind)    :: b             !  variational quality control parameter
   real(r_kind)    :: pg            !  variational quality control parameter
   real(r_kind)    :: wij(8)        !  horizontal interpolation weights
   real(r_kind)    :: upertb        !  random number adding to the obs
   real(r_kind)    :: vpertb        !  random number adding to the obs
   integer(i_kind) :: ij(8)         !  horizontal locations
   integer(i_kind) :: k1            !  level of errtable 1-33
   integer(i_kind) :: kx            !  ob type
   logical         :: luse          !  flag indicating if ob is used in pen.

   integer(i_kind) :: idv,iob       ! device id and obs index for sorting
end type w_ob_type
```

Append obs_diag structure to keep info needed for adj-gsi

```
! Loop over control steps
do jj=1,nsubwin

   workst(:,:,:)=zero
   workvp(:,:,:)=zero
   workrh(:,:,:)=zero
!  Get pointers to require state variables
   call gsi_bundlegetpointer (rval(jj),'u'    ,rv_u,    istatus)
      • • •

!  Convert RHS calculations for u,v to st/vp for application of
!  background error
   if(do_getuv) call getuv(rv_u,rv_v,workst,workvp,1)

!  Calculate sensible temperature
   if(do_tv_to_tsen_ad) call tv_to_tsen_ad(rv_tv,rv_q,rv_tsen)

!  Adjoint of convert input normalized RH to q to add contribution of moisture
!  to t, p , and normalized rh
   if(do_normal_rh_to_q_ad) call normal_rh_to_q_ad(workrh,rv_tv,rv_p3d,rv_q)

!  Adjoint to convert ps to 3-d pressure
   if(do_getprs_ad) call getprs_ad(rv_ps,rv_tv,rv_p3d)

!  Multiply by sqrt of background error adjoint (ckerror_ad)
!  ----------------------------------------------------------------------

!  Transpose of balance equation
   if(do_tbalance) call tbalance(rv_tv,rv_ps,workst,workvp,fpsproj)

!  Apply variances, as well as vertical & horizontal parts of background error
   gradz(:)=zero

!  create an internal structure w/ the same vars as those in the control vector
   call gsi_bundlecreate (cstate,grad%step(1),'model2control work',istatus)
   if (istatus/=0) then
      write(6,*)trim(myname),': trouble creating work bundle'
      call stop2(999)
   endif
!  Adjoint of control to initial state
   call gsi_bundleputvar ( cstate, 'sf', workst, istatus )
      • • •

!  Apply adjoint of sqrt-B
   call ckgcov_ad(gradz,cstate,nnnn1o)

!  Clean up
   call gsi_bundledestroy(cstate,istatus)
   if (istatus/=0) then
      write(6,*)trim(myname),': trouble destroying work bundle'
      call stop2(999)
   endif

   do ii=1,nval_lenz
      grad%step(jj)%values(ii)=grad%step(jj)%values(ii)+gradz(ii)
   enddo

end do
```

(*)

model2control

$$\delta\lambda = \mathbf{L}^T \delta\mathbf{x}$$

model2control is a convolution of variable transformations and application of the square-root decomposition of B (see *)

```
! Bias predictors are duplicated
do ii=1,nsclen
   grad%predr(ii)=grad%predr(ii)+bval%predr(ii)*sqrt(varprd(ii))
enddo
do ii=1,npclen
   grad%predp(ii)=grad%predp(ii)+bval%predp(ii)*sqrt(varprd(nsclen+ii))
enddo
```

control2model

$$\delta \mathbf{x} = \mathbf{L}\delta\lambda$$

(*)

```
! Loop over control steps
do jj=1,nsubwin

!  create an internal structure w/ the same vars as those in the control vector
   call gsi_bundlecreate (cvec,xhat%step(jj),'control2model work',istatus)
   if(istatus/=0) then
      write(6,*) trim(myname), ': trouble creating work bundle'
      call stop2(999)
   endif

!  Multiply by sqrt of background error (ckerror)
!  ----------------------------------------------------------------------
!  Apply sqrt of variance, as well as vertical & horizontal parts of background
!  error

   call ckgcov(xhat%step(jj)%values(:),cvec,nnnn1o)

!  Get pointers to require state variables
   call gsi_bundlegetpointer (sval(jj),'u'    ,sv_u,    istatus)
        • • •
!  Copy variables from CV to SV
   call gsi_bundlegetvar ( cvec, 'sf' , workst, istatus )
        • • •
!  destroy temporary bundle
   call gsi_bundledestroy(cvec,istatus)
   if(istatus/=0) then
      write(6,*) trim(myname), ': trouble destroying work bundle'
      call stop2(999)
   endif

!  Balance equation
   if(do_balance) call balance(sv_tv,sv_ps,workst,workvp,fpsproj)

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

!  Get 3d pressure
   if(do_getprs_tl) call getprs_tl(sv_ps,sv_tv,sv_p3d)

!  Convert input normalized RH to q
   if(do_normal_rh_to_q) call normal_rh_to_q(workrh,sv_tv,sv_p3d,sv_q)

!  Calculate sensible temperature
   if(do_tv_to_tsen) call tv_to_tsen(sv_tv,sv_q,sv_tsen)

!  Convert streamfunction and velocity potential to u,v
   if(do_getuv) call getuv(sv_u,sv_v,workst,workvp,0)

end do
```

NOTE: Same remark as from model2control applies here: this is a convolution of variable transformations and Application the square-root decomposition of B (see *)

sqrtmin

```
if (lbfgsmin) then
! call lbfgs  (xhat,costf,gradx,eps,itermax,nprt,nwrvecs)
  call m1qn3  (xhat,costf,gradx,eps,itermax,nprt,nwrvecs)

elseif (lcongrad) then
! Setup CONGRAD
  if (.not.ltlint) call abor1('sqrtmin: congrad requires ltlint')
  call setup_congrad(mype,nprt,jiter,jiterstart,itermax,nwrvecs, &
                     l4dvar,lanczosave)
  lsavev=(.not.lobsensfc).and.(jiter<miter)

  if (lobsensfc.and.lobsensadj) then
!   Get Lanczos vectors
    if ( llancdone ) then        ! Lanczos vecs already computed, read them in
      call read_lanczos(itermax)
    else                         ! Do forward min to get Lanczos vectors
      call congrad(xhat,costf,gradx,eps,itermax,iobsconv,lsavev)
    endif

!   Compute sensitivity
    zgg=dot_product(fcsens,fcsens,r_quad)
    if (mype==0) write(6,888)trim(myname),': Norm fcsens=',sqrt(zgg)
    xhat=fcsens
    call congrad_ad(xhat,itermax)
  else
!   Compute increment
    call congrad(xhat,costf,gradx,eps,itermax,iobsconv,lsavev)
  endif

! Finish CONGRAD
  call save_precond(lsavev)

else  ! plain conjugate gradient
  if (.not.ltlint) call abor1('sqrtmin: pcgsqrt requires ltlint')
  call pcgsqrt(xhat,costf,gradx,eps,itermax,nprt)
endif
```

Ideal minimization scheme when running 4dvar.

When inner loop is linear, this is equivalent to original min, but not identical.

pcgsqrt

Vanilla CG to mimic original minimization scheme, but now using the sqrt(B) pre-conditioning

Calculating ∇J is at the core of the CG

```
! Perform inner iteration
inner_iteration: do iter=1,itermax
  if (mype==0) write(6,*)trim(myname),': Minimization iteration',iter

! Search direction
  do ii=1,dirx%lencv
    dirx%values(ii)=-gradx%values(ii)+beta*dirx%values(ii)
  end do

! Estimate
  do ii=1,xtry%lencv
    xtry%values(ii)=xhat%values(ii)+dirx%values(ii)
  end do

! Evaluate cost and gradient
  call evaljgrad(xtry,zfk,grtry,lsavinc,nprt,myname)

! Get A q_k
  do ii=1,grtry%lencv
    grtry%values(ii)=grtry%values(ii)-grad0%values(ii)
  end do

! Calculate stepsize
  dkqk=dot_product(dirx,grtry,r_quad)
  alpha=zero_quad
  if(abs(dkqk)>tiny_r_kind) alpha = zgk/dkqk

! Update estimates
  do ii=1,xhat%lencv
    xhat%values(ii) =xhat%values(ii) +alpha* dirx%values(ii)
    gradx%values(ii)=gradx%values(ii)+alpha*grtry%values(ii)
  end do

  zgnew=dot_product(gradx,gradx,r_quad)
  beta=zero_quad
  if(abs(zgk)>tiny_r_kind) beta=zgnew/zgk
  zgk=zgnew

end do inner_iteration
costf=zfk
```

evaljgrad

going backwards

going forwards

$$\nabla \hat{J}_j = \lambda + \mathbf{L}_j^T \mathbf{M}_j^T \mathbf{H}_j^T \mathbf{R}^{-1} (\mathbf{H}_j \mathbf{M}_j \mathbf{L} \lambda_j - \mathbf{d}_j)$$

```
! Contribution from background term
gradx = xhat

if (lsaveobsens) then
! Observation sensitivity right hand side
  do ii=1,gradx%lencv
    gradx%values(ii) = gradx%values(ii) - fcsens%values(ii)
  enddo
else
! Contribution from previous background term
  do ii=1,gradx%lencv
    gradx%values(ii) = gradx%values(ii) + xhatsave%values(ii)
  enddo
endif
```

```
! Convert from control space to model space
call control2model(xhat,mval,sbias)

! Run TL model to fill sval
if (l4dvar) then
  call model_tl(mval,sval,llprt)
else
  do ii=1,nobs_bins
    sval(ii)=mval(1)
  enddo
end if
```

```
! Compare obs to solution and transpose back to grid (H^T R^{-1} H)
do ibin=1,nobs_bins
   call intjo(yobs(ibin),rval(ibin),rbias,sval(ibin),sbias,ibin)
end do

! Evaluate Jo
call evaljo(zjo,iobs,nprt,llouter)
```

intjo

```
! RHS for conventional temperatures
  call intt(yobs%t,rval,sval)

! RHS for precipitable water
  call intpw(yobs%pw,rval,sval)

! RHS for conventional moisture
  call intq(yobs%q,rval,sval)

! RHS for conventional winds
  call intw(yobs%w,rval,sval)

! RHS for radar superob winds
  call intsrw(yobs%srw,rval,sval)

! RHS for lidar winds
  call intdw(yobs%dw,rval,sval)

! RHS for radar winds
  call intrw(yobs%rw,rval,sval)

! RHS for wind speed observations
  call intspd(yobs%spd,rval,sval)

! RHS for ozone observations
  call intoz(yobs%oz,yobs%o3l,rval,sval)

! RHS for carbon monoxide
  call intco(yobs%co3l,rval,sval)

! RHS for surface pressure observations
  call intps(yobs%ps,rval,sval)

! RHS for MSLP obs for TCs
  call inttcp(yobs%tcp,rval,sval)

! RHS for conventional sst observations
  call intsst(yobs%sst,rval,sval)

! RHS for GPS local observations
  call intgps(yobs%gps,rval,sval)
```

intq

```
    if(do_foto) time_q=qptr%time*r3600
!   Forward model
    val=w1* sq(j1)+w2* sq(j2)+w3* sq(j3)+w4* sq(j4)+ &
        w5* sq(j5)+w6* sq(j6)+w7* sq(j7)+w8* sq(j8)
    if ( do_foto ) then
      val=val+&
       (w1*dsq(j1)+w2*dsq(j2)+w3*dsq(j3)+w4*dsq(j4)+ &
        w5*dsq(j5)+w6*dsq(j6)+w7*dsq(j7)+w8*dsq(j8))*time_q
    endif

    if (lsaveobsens) then
      qptr%diags%obssen(jiter) = val*qptr%raterr2*qptr%err2
    else
      if (qptr%luse) qptr%diags%tldepart(jiter)=val
    endif

    if (l_do_adjoint) then
      if (lsaveobsens) then
        grad = qptr%diags%obssen(jiter)

      else
        val=val-qptr%res
!     ..... cut for illustration purpose
      endif
    endif
```

Notice now the int routines can apply the usual full operator:

$$\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}$$

as well as simple:

$$\mathbf{R}^{-1}\mathbf{H}$$

evaljgrad

Similarly to int routines, evaljgrad can be used to apply only part of operator … $\mathbf{R}^{-1}\mathbf{H}_j\mathbf{M}_j\mathbf{L}\lambda_j$

Evaluate cost due to various constraints

```
if (l_do_adjoint) then
! Moisture constraint
  zjl=zero_quad
  if (.not.ltlint) then
    do ibin=1,nobs_bins
      call evalqlim(sval(ibin)%q,zjl,rval(ibin)%q)
    enddo
  endif

  if (ljcdfi) then
    call evaljcdfi(sval,zjc,rval)
  else
! Jc and other 3D-Var terms
! Don't know how to deal with Jc term so comment for now...
! call eval3dvar(sval,zjc,rval,zdummy)
    zjc=zero_quad
  endif
```

$$\nabla \hat{J}_j = \lambda + \mathbf{L}_j^T \mathbf{M}_j^T \mathbf{H}_j^T \mathbf{R}^{-1}(\mathbf{H}_j \mathbf{M}_j \mathbf{L}\lambda_j - \mathbf{d}_j)$$

```
! Run adjoint model
  if (l4dvar) then
    call model_ad(mval,rval,llprt)
  else
    mval(1)=rval(1)
    do ii=2,nobs_bins
      call self_add(mval(1),rval(ii))
    enddo
  end if

! Adjoint of convert control var to physical space
  call model2control(mval,rbias,gradx)

! Cost function
  fjcost=zjb+zjo+zjc+zjl

endif
```

model_tl

j-th iteration propagation with the tangent linear model:

$$\delta \mathbf{x}_{k;j} = \mathbf{M}_{k,k-1;j} \delta \mathbf{x}_{k-1;j}$$

```
! Initialize TL model
call gsi_4dcoupler_init_model_tl(ndtpert)
          • • •
! Run TL model
do istep=0,nstep-1

!  Apply control vector to x_{istep}
   if (MOD(istep,nfrctl)==0) then
      ii=istep/nfrctl+1
      if (ii<1.or.ii>nsubwin) then
         write(6,*)'model_tl: error xini',ii,nsubwin
         call stop2(151)
      end if
      call self_add(xx,xini(ii))
   endif

!  Post-process x_{istep}
   if (MOD(istep,nfrobs)==0) then
      ii=istep/nfrobs+1
      if (ii<1.or.ii>nobs_bins) then
         write(6,*)'model_tl: error xobs',ii,nobs_bins
         call stop2(152)
      end if
      xobs(ii) = xx
      d0=d0+dot_product(xx,xx)
   endif

!  Apply TL model
   call gsi_4dcoupler_model_tl(xx,nymdi,nhmsi,ndt)

   call tick (nymdi,nhmsi,dt)
enddo
          • • •
! Finalize TL model
call gsi_4dcoupler_final_model_tl()
```

General interface to forward perturbation model

Note: some editing done here to facilitate discussion; handling of Lagrangian component removed for simplicity

model_ad

j-th iteration propagation with the adjoint model:

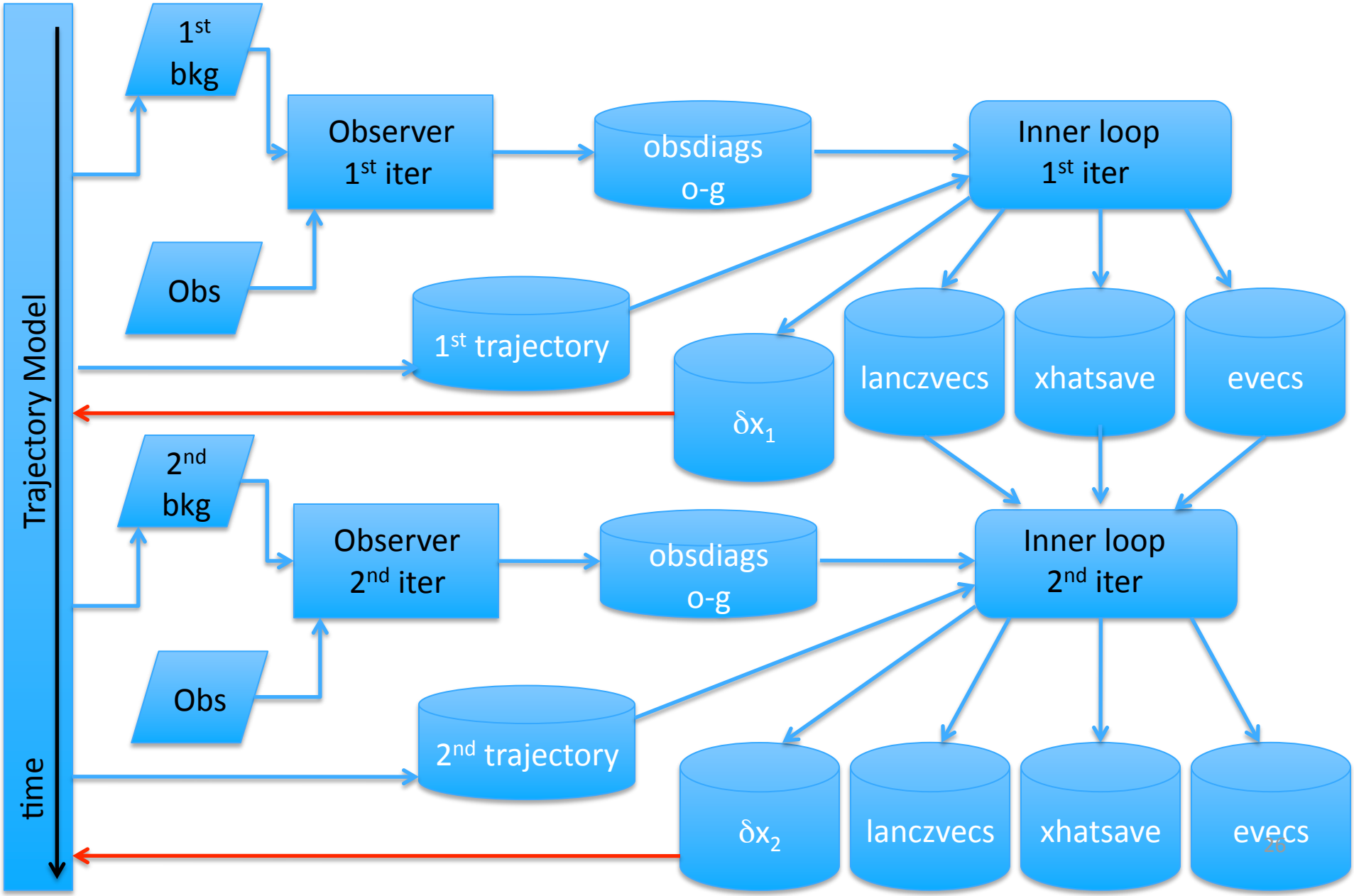
$$\delta \mathbf{x}_{k-1;j} = \mathbf{M}^{T}_{k,k-1;j} \delta \mathbf{x}_{k;j}$$

```
! Initialize AD model
call gsi_4dcoupler_init_model_ad(ndtpert)
                  . . .
! Run AD model
do istep=nstep-1,0,-1
   call tick(nymdi,nhmsi,-dt)

!  Apply AD model
   call gsi_4dcoupler_model_ad(xx,nymdi,nhmsi,ndt)

!  Post-process x_{istep}
   if (MOD(istep,nfrobs)==0) then
      ii=istep/nfrobs+1
      call self_add(xx,xobs(ii))
   endif

!  Apply control vector to x_{istep}
   if (MOD(istep,nfrctl)==0) then
      ii=istep/nfrctl+1
      xini(ii)=xx
      d0=dot_product(xobs(ii),xx)
   endif

enddo
                  . . .
! Finalize AD model
call gsi_4dcoupler_final_model_ad()
```

General interface to backward perturbation model

Note: some editing done here to facilitate discussion; handling of Lagrangian component removed for simplicity

What happens when 4DVAR iterates?



3DVAR namelist settings

- Default CG

```
&SETUP
miter=2,niter(1)=100,niter(2)=150,
niter_no_qc(1)=50,niter_no_qc(2)=0,
write_diag(1)=.true.,write_diag(2)=.false.,
write_diag(3)=.true.,
gencode=82,qoption=2,
 factqmin=0.005,factqmax=0.005,deltim=
$DELTIM,
 ndat=62,npred=5,iguess=-1,
oneobtest=.false.,retrieval=.false.,
l_foto=.false.,use_pbl=.false.,
 /
```

- Square-root(B) CG

```
&SETUP
miter=2,niter(1)=100,niter(2)=150,
niter_no_qc(1)=999,niter_no_qc(2)=999,
write_diag(1)=.true.,write_diag(2)=.false.,
write_diag(3)=.true.,
gencode=82,qoption=2,
 factqmin=0.005,factqmax=0.005,deltim=
$DELTIM,
 ndat=62,npred=5,iguess=-1,
oneobtest=.false.,retrieval=.false.,
l_foto=.false.,use_pbl=.false.,
lsqrtb=.true.,ltlint=.true.,
 /
```

Note: new opts are minimal set suggested opts, that have been tested.

4DVAR namelist settings

- Observer (1st outer loop)

```
&SETUP
miter=2,niter(1)=100,niter(2)=150,
niter_no_qc(1)=999,niter_no_qc(2)=999,
write_diag(1)=.true.,write_diag(2)=.false.,
write_diag(3)=.true.,
gencode=82,qoption=2,
 factqmin=0.005,factqmax=0.005,deltim=
$DELTIM,
 ndat=62,npred=5,iguess=-1,
oneobtest=.false.,retrieval=.false.,
l_foto=.false.,use_pbl=.false.,
l4dvar=.true., jiterstart=1,
lobserver=.true.,
 /
```

- 1st inner loop

```
&SETUP
miter=2,niter(1)=100,niter(2)=150,
niter_no_qc(1)=999,niter_no_qc(2)=999,
write_diag(1)=.true.,write_diag(2)=.false.,
write_diag(3)=.true.,
gencode=82,qoption=2,
 factqmin=0.005,factqmax=0.005,deltim=
$DELTIM,
 ndat=62,npred=5,iguess=-1,
oneobtest=.false.,retrieval=.false.,
l_foto=.false.,use_pbl=.false.,
l4dvar=.true., jiterstart=1,
lsqrtb=.true.,ltlint=.true., lcongrad=.true.,
nhr_assimilation=6,nhr_obsbin=1,
idmodel=.false., lwrtinc=.true.,
 /
```

The Linear 4d-Analysis Adjoint

A linear analysis system calculates:

$$\begin{aligned}\delta\mathbf{x} = \mathbf{K}\mathbf{d} &= (\mathbf{B}^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{d} \quad \boxed{\text{phy-space}} \\ &= \mathbf{B}\mathbf{H}^T(\mathbf{H}\mathbf{B}\mathbf{H}^T + \mathbf{R})\mathbf{d} \quad \boxed{\text{obs-space}}\end{aligned}$$

and its adjoint calculates:

$$\begin{aligned}\delta\mathbf{z} = \mathbf{K}^T\mathbf{g} &= \mathbf{R}^{-1}\mathbf{H}(\mathbf{B}^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{g} \quad \boxed{\text{phy-space}} \\ &= (\mathbf{H}\mathbf{B}\mathbf{H}^T + \mathbf{R})\mathbf{H}\mathbf{B}\mathbf{g} \quad \boxed{\text{obs-space}}\end{aligned}$$

Obtaining the adjoint in practice:

- ▷ Direct, line-by-line, adjoint (Zhu & Gelaro 2007)
- ▷ Operator manipulation:
  - Observation space (Baker & Daley 2000):

$$(\mathbf{H}\mathbf{B}\mathbf{H}^T + \mathbf{R})\delta\mathbf{z} = \mathbf{H}\mathbf{B}\mathbf{g}$$

Either one of these can be done with GSI

  - Physical scape (Trémolet 2008): $\delta\mathbf{z} = \mathbf{R}^{-1}\mathbf{H}\delta\mathbf{g}$

$$(\mathbf{B}^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})\delta\mathbf{g} = \mathbf{g}$$

  - Approximate Hessian: $\tilde{\mathbf{A}}^{-1} = \mathbf{U}\Lambda\mathbf{U}^T \sim \sqrt{\mathbf{B}^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}}$

$$\delta\mathbf{z} = \mathbf{R}^{-1}\mathbf{H}\tilde{\mathbf{A}}\mathbf{g}$$

obs_sensitivity

```
!       read and convert output of GCM adjoint
        do ii=1,nsubwin
           call allocate_state(fcgrad(ii))
        end do
        call allocate_preds(zbias)
        zbias=zero
        call gsi_4dcoupler_getpert(fcgrad,nsubwin,'adm')
        call model2control(fcgrad,zbias,fcsens)
        do ii=1,nsubwin
           call deallocate_state(fcgrad(ii))
        end do
        call deallocate_preds(zbias)
      endif
```

General interface to input gradient vector

Read GCM sensitivity

Convert to GSI control vec

- The above prepares the right-hand-side of the equation to be solved:

$$(\mathbf{I} + \mathbf{L}^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\mathbf{L})\delta\mathbf{g} = \mathbf{L}\mathbf{g}$$

- From what we saw a few slides back, sqrtmin, through multiple call to evaljgrad will solve the equation above

- To finally get the sensitivity

$$\delta\mathbf{z} = \mathbf{R}^{-1}\mathbf{H}\mathbf{L}\delta\mathbf{g}$$

one needs to simply call evaljgrad making sure not to apply the adjoint part

sqrtmin

GSI adjoint when forward analysis uses Lanczos CG; i.e., Hessian-based adjoint.

Get **U** and Λ

$\delta \mathbf{z} = \mathbf{R}^{-1}\mathbf{H}\tilde{\mathbf{A}}\mathbf{g}$

```
if (lbfgsmin) then
! call lbfgs  (xhat,costf,gradx,eps,itermax,nprt,nwrvecs)
  call m1qn3  (xhat,costf,gradx,eps,itermax,nprt,nwrvecs)

elseif (lcongrad) then
! Setup CONGRAD
  if (.not.ltlint) call abor1('sqrtmin: congrad requires ltlint')
  call setup_congrad(mype,nprt,jiter,jiterstart,itermax,nwrvecs, &
                     l4dvar,lanczosave)
  lsavev=(.not.lobsensfc).and.(jiter<miter)

  if (lobsensfc.and.lobsensadj) then
!   Get Lanczos vectors
    if ( llancdone ) then        ! Lanczos vecs already computed, read them in
      call read_lanczos(itermax)
    else                         ! Do forward min to get Lanczos vectors
      call congrad(xhat,costf,gradx,eps,itermax,iobsconv,lsavev)
    endif

!   Compute sensitivity
    zgg=dot_product(fcsens,fcsens,r_quad)
    if (mype==0) write(6,888)trim(myname),': Norm fcsens=',sqrt(zgg)
    xhat=fcsens
    call congrad_ad(xhat,itermax)
  else
!   Compute increment
    call congrad(xhat,costf,gradx,eps,itermax,iobsconv,lsavev)
  endif

! Finish CONGRAD
  call save_precond(lsavev)

else  ! plain conjugate gradient
  if (.not.ltlint) call abor1('sqrtmin: pcgsqrt requires ltlint')
  call pcgsqrt(xhat,costf,gradx,eps,itermax,nprt)
endif
```

e.g., bottom of setupw, optionally appending

The setup routines now have the option to save additional quantities to the diag files so one can calculate observation impact

```
rdiagbuf(20,ii) = data(ivob,i)        ! v wind component observation (m/s)
rdiagbuf(21,ii) = dvdiff              ! v obs-ges used in analysis (m/s)
rdiagbuf(22,ii) = vob-vgesin          ! v obs-ges w/o bias correction (m/s) (future slot)

rdiagbuf(23,ii) = factw               ! 10m wind reduction factor

if (lobsdiagsave) then
   ioff=23
   do jj=1,miter
      ioff=ioff+1
      if (obsdiags(i_w_ob_type,ibin)%tail%muse(jj)) then
         rdiagbuf(ioff,ii) = one
      else
         rdiagbuf(ioff,ii) = -one
      endif
   enddo
   do jj=1,miter+1
      ioff=ioff+1
      rdiagbuf(ioff,ii) = obsptr%nldepart(jj)
      ioff=ioff+1
      rdiagbuf(ioff,ii) = obsdiags(i_w_ob_type,ibin)%tail%nldepart(jj)
   enddo
   do jj=1,miter
      ioff=ioff+1
      rdiagbuf(ioff,ii) = obsptr%tldepart(jj)
      ioff=ioff+1
      rdiagbuf(ioff,ii) = obsdiags(i_w_ob_type,ibin)%tail%tldepart(jj)
   enddo
   do jj=1,miter
      ioff=ioff+1
      rdiagbuf(ioff,ii) = obsptr%obssen(jj)
      ioff=ioff+1
      rdiagbuf(ioff,ii) = obsdiags(i_w_ob_type,ibin)%tail%obssen(jj)
   enddo
endif
```

Usage flag

Nonlinear departures

Linear departures

Observation sensitivities

3DVAR-ADJ namelist settings

- Square-root(B) CG

```
&SETUP
miter=2,niter(1)=100,niter(2)=150,
niter_no_qc(1)=999,niter_no_qc(2)=999,
write_diag(1)=.true.,write_diag(2)=.false.,
write_diag(3)=.true.,
gencode=82,qoption=2,
 factqmin=0.005,factqmax=0.005,deltim=
$DELTIM,
 ndat=62,npred=5,iguess=-1,
oneobtest=.false.,retrieval=.false.,
l_foto=.false.,use_pbl=.false.,
lsqrtb=.true.,ltlint=.true.,
 /
```

- ADJ square-root(B) CG

```
&SETUP
miter=2,niter(1)=100,niter(2)=150,
....
lsqrtb=.true.,ltlint=.true.,
jiterstart=1,jiterend=2,
lobsensmin=.true.,
lsensrecompute=.true.,lobsensfc=.true.,
lobsdiagsave=.true.,
 /
```

```
&SETUP
miter=2,niter(1)=100,niter(2)=150,
....
lsqrtb=.true.,ltlint=.true.,
jiterstart=1,jiterend=1,
lobsensmin=.true.,
lsensrecompute=.true.,lobsensfc=.true.,
lobsdiagsave=.true.,
 /
```

Note: new opts are minimal set suggested opts, that have been tested.

A Framework for General User-specific Functionalities in GSI

- The idea is to provide a general framework to allow users to replace certain GSI-functionalities with alternative user-specific ones
- The present framework requires simply that users of GSI provide a library with their own routines to overwrite GSI's
- The present framework handles two such functionalities:

  (i) timing: timermod.F90

  (ii) perturbation models: pertmod.F90
- Generalizing other GSI functions is planed as well, e.g.:
  - Calculation of saturation specific humidity (genqsat)
  - Precipitation physics (and it's adjoint; pcp_k)
  - Aerosol particle size (Mie calculation; GOCART_Aerosol_Size)
  - Background fields

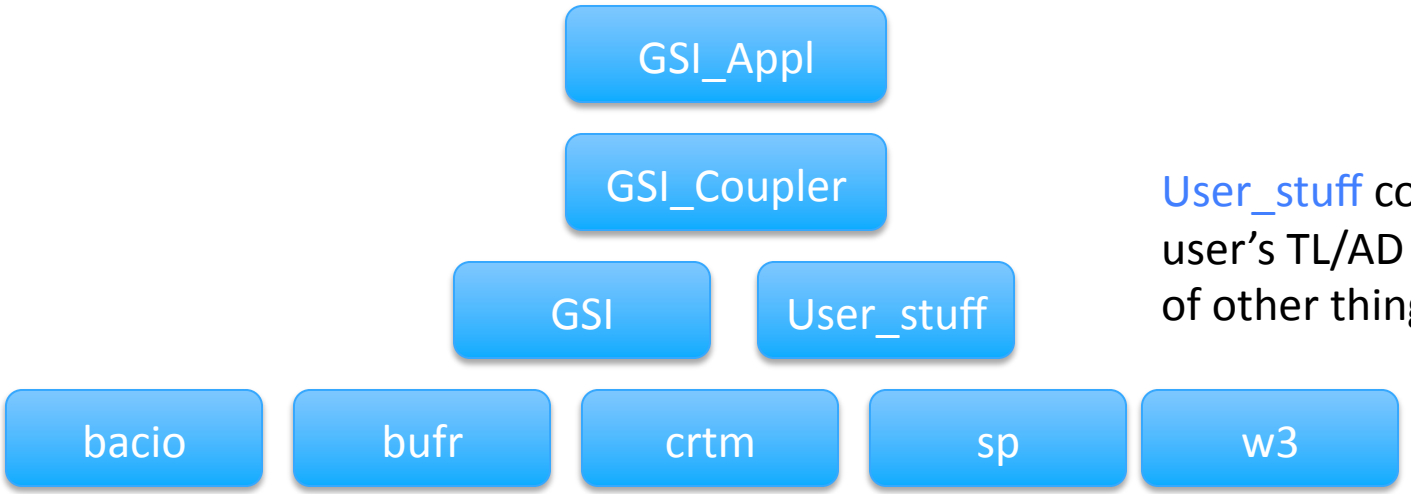
A Framework for General User-specific Functionalities in GSI

- GSI library contains the following internal module procedures:
  - timermod.F90
  - gsi_4dcouplermod.F90 (to be renamed to pertmod.F90)

  and corresponding straight FORTRAN (non-module) packages:
  - stub_timermod.F90
  - stub_pertmod.F90
- When user is not concerned with the timing and perturbation model routines the GSI library is fully self-contained
- When user has its own timer and/or perturbation model routines, user is expected to provide these corresponding functionalities as a separate library, completely outside of GSI.
- For the sake of argument, lets assume the user library lives in a directory called GSI_Coupler. Furthermore, let's assume the user provided corresponding packages in GSI_Coupler are:
  - cplr_timermod.F90
  - cplr_pertmod.F90

A Framework for General User-specific Functionalities in GSI

- Schematic view of hierarchy of various libraries involved in creating a GSI executable



User_stuff contains user's TL/AD and all sorts of other things user needs

- A consequence of this hierarchy is that, when users want to overwrite GSI procedures, they are also expected to generate the executable outside of GSI (directory), say, in GSI_Appl

A Framework for General User-specific Functionalities in GSI

- Before the executable is built the stubs inside GSI must be removed from the GSI library. This can easily be done at Makefile level either when the Coupler is built or in the GSI_Appl right before the GSI library is fed to the link line
- How does the perturbation interface look like?

gsi_4dcouplermod.F90

```
public GSI_4dCoupler_parallel_init
public GSI_4dCoupler_init_traj
public GSI_4dCoupler_init_model_tl
public GSI_4dCoupler_model_tl
public GSI_4dCoupler_final_model_tl
public GSI_4dCoupler_init_model_ad
public GSI_4dCoupler_model_ad
public GSI_4dCoupler_final_model_ad
public GSI_4dCoupler_grtests
public GSI_4dCoupler_getpert
public GSI_4dCoupler_putpert
public GSI_4dCoupler_final_traj
```

cplr_pertmod.F90

```
 subroutine parallel_init_
 subroutine init_traj_(idmodel)
 subroutine init_pertmod_tl_ (ndt_tl)
 subroutine pertmod_tl_ (xx,nymdi,nhmsi,ndt)
 subroutine final_pertmod_tl_ ()
 subroutine init_pertmod_ad_ (ndt_ad)
 subroutine pertmod_ad_(xx,nymdi,nhmsi,ndt)
 subroutine final_pertmod_ad_ ()
 subroutine grtests_(mval,sval,nsubwin,nobs_bins)
!subroutine get_pert
   subroutine get_1pert_(xx,what)
   subroutine get_Npert_(xx,n,what)
!subroutine put_pert
   subroutine put_Npert_(xx,n,what)
   subroutine put_1pert_(xx,nymd,nhms,what,label)
 subroutine final_traj_()
```

Related GSI Trac Tickets

- Ticket #38: general interfaces for TL/AD models for 4dvar
- Ticket #85: running the GSI Adjoint
- Ticket #86: updating observation diagnostic tools to handle observation sensitivity and impact

General Issues

- Details to allow for multiple resolution outer and inner loops are being worked out.
- I/O and memory usage in Lanczos-based 4DVAR is large – some thought must be given for optimization.
- All hooks for weak constraint 4DVAR are in place; now it's deciding on Q and an affordable strategy.
- It should soon be possible to combine 4dvar with the hybrid-ensemble strategy.

Questions: ricardo.todling@nasa.gov

Info on GMAO 4DVAR to appear: http://geos5.org/wiki/index.php?title=Main_Page