A Brief Tutorial on GSI Infrastructures & Advanced Features

Ricardo Todling

Global Modeling and Assimilation Office

GSI Tutorial, DTC/NCAR, 21-23 August, 2012

This presentation is a brief guide to some of the basic infrastructure being added to GSI, namely:
- Introducing MetGuess_Bundle
- Interfacing to user-specific applications
- Remarks on adding new observing instruments

Questions and comments to: ricardo.todling@nasa.gov

OUTLINE

- Connecting the code with the Math behind it
- Interfacing user-specific components
  - General concept
  - Illustration 1: timing routines
  - Illustration 2: aerosols
  - Illustration 3: 4D-Var
  - Illustration 4: Hybrid Ensemble
- Adding new instruments (obs. operator)
- Upcoming MetGuess_Bundle

Four-dimensional Variational Approach

The general cost function of the variational formulation

$$J(\mathbf{x}) = \frac{1}{2}(\mathbf{x}_0 - \mathbf{x}^b)^T \mathbf{B}^{-1}(\mathbf{x}_0 - \mathbf{x}^b) + J_x$$

$$+ \frac{1}{2}\sum_{k=0}^{K}[\mathbf{h}(\mathbf{x}_k) - \mathbf{y}_k]^T \mathbf{R}_k^{-1}[\mathbf{h}(\mathbf{x}_k) - \mathbf{y}_k]$$

$$+ \frac{1}{2}\sum_{k=1}^{K}[\mathbf{m}(\mathbf{x}_k) - \mathbf{x}_k]^T \mathbf{Q}_k^{-1}[\mathbf{m}(\mathbf{x}_k) - \mathbf{x}_k]$$

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, $\mathbf{Q} \neq 0$ 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

$$
J_j(\delta\mathbf{x}_j) = \frac{1}{2}(\delta\mathbf{x}_j - \delta\mathbf{x}_j^b)^T \mathbf{B}^{-1}(\delta\mathbf{x}_j - \delta\mathbf{x}_j^b)
$$

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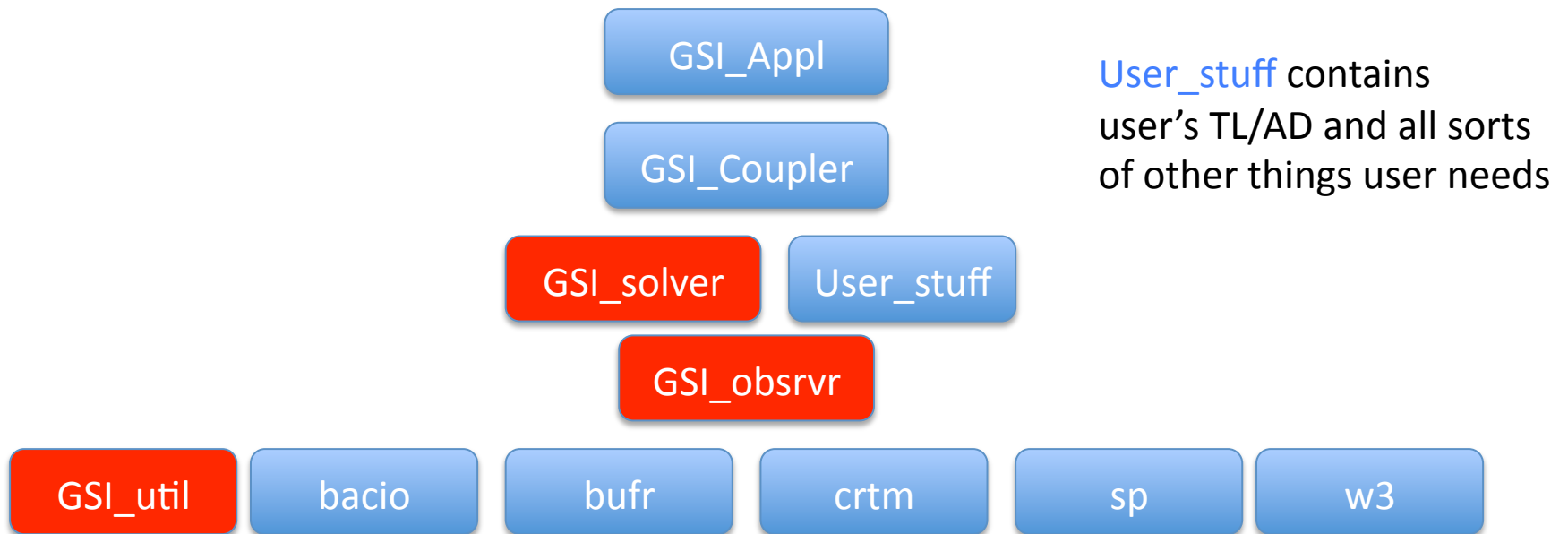$$

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.

GSI Infrastructure:
Split GSI into sub-libraries

- Schematic view of GSI & supporting libraries at GMAO

GSI_Appl

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

GSI_Coupler

GSI_solver   User_stuff

GSI_obsrvr

GSI_util   bacio   bufr   crtm   sp   w3

- Issues:
  - ➤ At present, GSI_solver and GSI_obsrvr cannot be separated
  - ➤ At present, GSI_util cannot sit parallel to supporting libs due to its reliance on some of those

GSI Infrastructure:
General Code Organization

```
! Initialize defaults of vars in modules
  call init_4dvar

! Read in user specification of state and control variables
  call gsi_metguess_init
  call gsi_chemguess_init
  call init_anasv
  call init_anacv

  call init_constants_derived
  call init_oneobmod
  call init_qcvars
  call init_obsmod_dflts
  call init_directories(mype)
  call init_pcp
  call init_rad
       •   •   •
! Initialize values in radinfo
  call init_rad_vars

! Initialize values in aeroinfo
  call init_aero_vars
```

Initialize
gsimain_initialize

```
call gsi_4dcoupler_setservices(rc=ier)
```

SetServices
gsimain_initialize

```
! Call the main gsi driver routine
  call gsisub(mype, init_pass_,last_pass_)
```

Run
gsimain_run

```
  call final_aero_vars
  call final_rad_vars
  call clean_4dvar
  call destroy_obsmod_vars
  call destroy_mpi_vars
  call final_anacv
  call final_anasv
  call gsi_chemguess_final
  call gsi_metguess_final
```

Finalize
gsimain_finalize

GSI Infrastructure:
General Code Organization
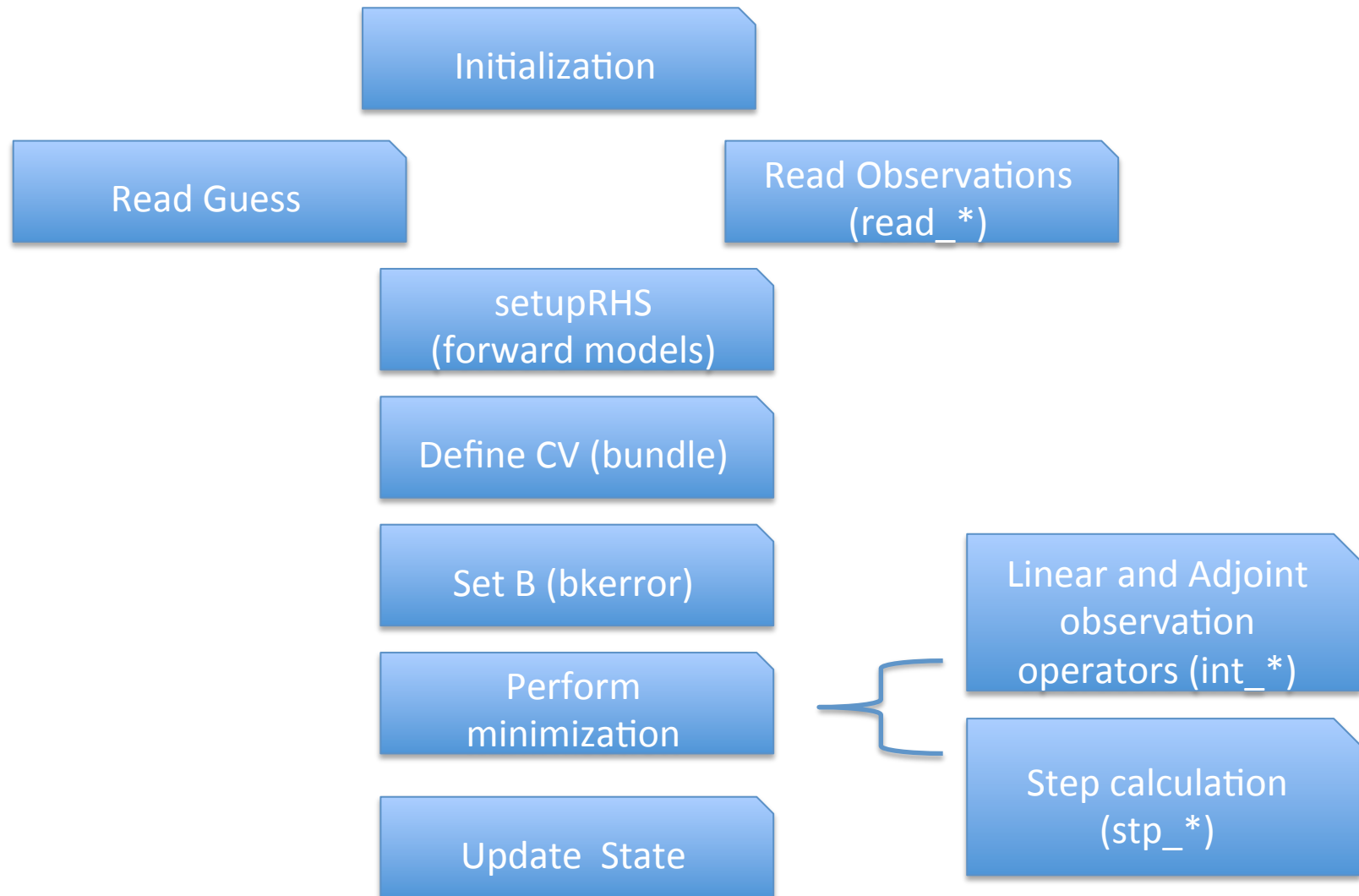
gsisub

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

In 4dvar:

This is part of
in outer loop
During NL model
integration

This becomes
the inner loop

```
! Decide whether to do observer or minimization
  if (lobserver) then
! Do observer only
    call observer_init
    call observer_run
    call observer_finalize
  else
! Complete setup and execute external and internal minimization loops
    call glbsoi(mype)
  endif
```

In 3dvar, this calls the observer

- The observer gets called by the forward model via the ESMF interface

observer

Observer methods

```
public observer_init
public observer_set
public observer_run
public observer_finalize
```

```
!     Read observations from data files !
      call read_obs(ndata,mype)              !
```

$$\mathbf{d}_j = \mathbf{h}(\mathbf{x}^{j-1}) - \mathbf{y}^o$$

```
!   Set up right hand side of analysis equation
    call setuprhsall(ndata,mype,init_pass_,last_pass
```

```
if (obstype == 't'   .or. &
    obstype == 'q'   .or. obstype == 'ps' .or. &
    obstype == 'pw'  .or. obstype == 'spd'.or. &
    obstype == 'gust' .or. obstype == 'vis'.or. &
    obstype == 'mta_cld' .or. obstype == 'gos_ctp'  ) then
    call read_prepbufr(nread,npuse,nouse,infile,obstype,lunout,twind,sis,&
         prsl_full)
    string='READ_PREPBUFR'
Process winds in the prepbufr
else if(obstype == 'uv') then
 Process satellite winds which seperate from prepbufr
    if ( index(infile,'satwnd') /=0 ) then
       call read_satwnd(nread,npuse,nouse,infile,obstype,lunout,gstime,twind,si
            prsl_full)
       string='READ_SATWND'
    else
       call read_prepbufr(nread,npuse,nouse,infile,obstype,lunout,twind,sis,&
            prsl_full)
       string='READ_PREPBUFR'
    endif

    if((obstype == 'amsua' .or. obstype == 'amsub' .or. obstype == 'mhs') .a
        (platid /= 'metop-a' .or. platid /='metop-b' .or. platid /= 'metop-c')
    call read_bufrtovs(mype,val_dat,ithin,isfcalc,rmesh,platid,gstime,&
         infile,lunout,obstype,nread,npuse,nouse,twind,sis, &
         mype_root,mype_sub(mm1,i),npe_sub(i),mpi_comm_sub(i),llb,lll)
    string='READ_BUFRTOVS'

Process atms data
else if (obstype == 'atms') then
    llb=1
    lll=1
    call read_atms(mype,val_dat,ithin,isfcalc,rmesh,platid,gstime,&
         infile,lunout,obstype,nread,npuse,nouse,twind,sis, &
         mype_root,mype_sub(mm1,i),npe_sub(i),mpi_comm_sub(i),llb,lll)
    string='READ_ATMS'

Process airs data
else if(platid == 'aqua' .and. (obstype == 'airs' .or.   &
        obstype == 'amsua'  .or.  obstype == 'hsb' ))then
    call read_airs(mype,val_dat,ithin,isfcalc,rmesh,platid,gstime,&
         infile,lunout,obstype,nread,npuse,nouse,twind,sis,&
         mype_root,mype_sub(mm1,i),npe_sub(i),mpi_comm_sub(i))
    string='READ_AIRS'
```

glbsoi

$$(\mathbf{B}^{-1} + \mathbf{H}_j^T \mathbf{R}^{-1} \mathbf{H}_j)\delta\mathbf{x}_j = \mathbf{H}_j^T \mathbf{R}^{-1} \mathbf{d}_j + \mathbf{B}^{-1} \mathbf{b}_{j-1}$$

```
!    Set up right hand side of analysis equation
     call setuprhsall(ndata,mype,.true.,.true.)
!
```

$$\mathbf{d}_j = \mathbf{h}(\mathbf{x}^{j-1}) - \mathbf{y}^o$$

Prepare RHS of Eq

$$\mathbf{b}_{j-1} = \mathbf{x}^b - \mathbf{x}_{j-1}$$

```
!    Set up right hand side of adjoint of analysis equation
     if (lsensrecompute) lobsensfc=(jiter==jiterend)
     if (lobsensfc.or.iobsconv>0) call init_fc_sens
```

Only in Adjoint Mode: Replace RHS with model sensitivity

```
!    Call inner minimization loop
     if (laltmin) then
        if (lsqrtb) then
           if (mype==0) write(6,*)'GLBSOI: Using sqrt(B), jiter=',jiter
           call sqrtmin
        endif
        if (lbicg) then
           if (mype==0) write(6,*)'GLBSOI: Using bicg, jiter=',jiter
           call bicg
        endif
     else
!       Standard run
        if (mype==0) write(6,*)'GLBSOI:  START pcgsoi jiter=',jiter
        call pcgsoi
     end if
```

$$(\mathbf{I} + \mathbf{B}^{1/2}\mathbf{H}_j^T \mathbf{R}^{-1} \mathbf{H}_j \mathbf{B}^{1/2})\mathbf{z}_j = \mathbf{B}^{1/2}\mathbf{H}_j^T \mathbf{R}^{-1} \mathbf{d}_j + \mathbf{B}^{-1/2}\mathbf{b}_{j-1}$$
$$\mathbf{z}_j = \mathbf{B}^{-1/2}\delta\mathbf{x}_j$$

$$(\mathbf{I} + \mathbf{B}\mathbf{H}_j^T \mathbf{R}^{-1} \mathbf{H}_j)\delta\mathbf{x}_j = \mathbf{B}\mathbf{H}_j^T \mathbf{R}^{-1} \mathbf{d}_j + \mathbf{b}_{j-1}$$

$$\mathbf{z} + \mathbf{H}_j^T \mathbf{R}^{-1} \mathbf{H}_j \delta\mathbf{x}_j = \mathbf{H}_j^T \mathbf{R}^{-1} \mathbf{d}_j + \mathbf{B}^{-1} \mathbf{b}_{j-1}$$
$$\mathbf{z}_j = \mathbf{B}^{-1}\delta\mathbf{x}_j$$

sqrtmin

Ideal minimization scheme when running 4dvar.

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

```fortran
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
```

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

```fortran
! 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

```
! Contribution from background term
gradx = xhat
```

going backwards

```
if (lsaveobsens) then
! Observation sensitivity right hand side
  do ii=1,gradx%lencv
    gradx%values(ii) = gradx%values(ii) - fcsens%values(ii)
  enddo
else
```

going forwards

```
! Contribution from previous background term
  do ii=1,gradx%lencv
    gradx%values(ii) = gradx%values(ii) + xhatsave%values(ii)
  enddo
endif
```

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

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

control2model

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

(*)

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

! 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(:),workst,workvp, &
              sval(jj)%t,sval(jj)%p,workrh,sval(jj)%oz, &
              sval(jj)%sst,sval(jj)%cw,nnn)

! Balance equation
  call balance(sval(jj)%t,sval(jj)%p,workst,workvp,fpsproj)

! Apply strong balance constraint
  call strong_bk(workst,workvp,sval(jj)%p,sval(jj)%t,sval(jj)%oz,sval(jj)%cw)

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

! Get 3d pressure
  call getprs_tl(sval(jj)%p,sval(jj)%t,sval(jj)%p3d)

! Convert input normalized RH to q
  call normal_rh_to_q(workrh,sval(jj)%t,sval(jj)%p3d,sval(jj)%q)

! Calculate sensible temperature
  call tv_to_tsen(sval(jj)%t,sval(jj)%q,sval(jj)%tsen)

! Convert streamfunction and velocity potential to u,v
  call getuv(sval(jj)%u,sval(jj)%v,workst,workvp)

end do

! Bias correction terms
do ii=1,nsclen
  bval%predr(ii)=xhat%predr(ii)*sqrt(varprd(ii))
enddo
```

NOTE: Same remark as from model2control applies here: this is a convolution of variable transformations and the application of a part $\mathbf{L}$ of the square-root decomposition of B (see *)

model_tl

j-th iteration propagation
with the tangent linear model:

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

```
! 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) call abor1('model_tl: error xini')
    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) call abor1('model_tl: error xobs')
    xobs(ii) = xx
    d0=d0+dot_product(xx,xx)
  endif

! Apply TL model
#ifdef GEOS_PERT
  if (.not.idmodel) then
    call gsi2pgcm ( xx, xpert, 'tlm', ierr )        ! T
    call amodel_tl (xpert,nymdi=nymdi,nhmsi=nhmsi,ntsteps=ndt,g5pert=.true.)! M
    call pgcm2gsi ( xpert, xx, 'tlm', ierr )        ! T(-1)
  endif
#endif /* GEOS_PERT */

  call tick (nymdi,nhmsi,dt)
enddo

! Post-process final state
if (nobs_bins>1) then
  xobs(nobs_bins) = xx
  d0=d0+dot_product(xx,xx)
endif
if(ldprt) print *, myname, ': total (gsi) dot product ', d0
```

```
! RHS for conventional temperatures
  call intt(yobs%t, &
            rval%tsen,sval%tsen,rval%t,sval%t,rval%q,sval%q, &
            rval%u,sval%u,rval%v,sval%v,rval%p,sval%p, &
            rval%sst,sval%sst )

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

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

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

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

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

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

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

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

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

! RHS for GPS local observations
  call intgps(yobs%gps, &
              rval%t,rval%q,rval%p3d,sval%t,sval%q,sval%p3d)

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

```
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

Evaluate cost due to various constraints

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

```fortran
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
```

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

```fortran
! 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
```

evaljgrad

Evaluate cost due to various constraints

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

```fortran
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
```

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

```fortran
! 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_ad

j-th iteration propagation with the adjoint model:

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

```fortran
! Post-process final state
if (nobs_bins>1) then
  call self_add(xx,xobs(nobs_bins))
  if (.not.idmodel) then
    d0=dot_product(xobs(1),xobs(1))
  endif
endif

! Run AD model
do istep=nstep-1,0,-1
  call tick(nymdi,nhmsi,-dt)

! Apply AD model
#ifdef GEOS_PERT
  if (.not.idmodel) then
    call gsi2pgcm  (xx,xpert,'adm',ierr)                       ! T'(-1)
    call amodel_ad (xpert,nymdi,nhmsi,ntsteps=ndt,g5pert=.true.)  ! M'
    call pgcm2gsi  (xpert,xx,'adm',ierr)                       ! T'
  endif
#endif /* GEOS_PERT */

! 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
```

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

  workst(:,:,:)=zero
  workvp(:,:,:)=zero
  workrh(:,:,:)=zero

! Convert RHS calculations for u,v to st/vp for application of
! background error
  call getstvp(rval(jj)%u,rval(jj)%v,workst,workvp)

! Calculate sensible temperature
  call tv_to_tsen_ad(rval(jj)%t,rval(jj)%q,rval(jj)%tsen)

! Adjoint of convert input normalized RH to q to add contribution of moisture
! to t, p , and normalized rh
  call normal_rh_to_q_ad(workrh,rval(jj)%t,rval(jj)%p3d,rval(jj)%q)

! Adjoint to convert ps to 3-d pressure
  call getprs_ad(rval(jj)%p,rval(jj)%t,rval(jj)%p3d)

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

! Apply transpose of strong balance constraint
  call strong_bk_ad(workst,workvp,rval(jj)%p, &
                    rval(jj)%t,rval(jj)%oz,rval(jj)%cw)

! Transpose of balance equation
  call tbalance(rval(jj)%t,rval(jj)%p,workst,workvp,fpsproj)

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

  call ckgcov_ad(gradz,workst,workvp,rval(jj)%t,rval(jj)%p,workrh,&
                 rval(jj)%oz,rval(jj)%sst,rval(jj)%cw,nnn)

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

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

end do
```

(*)

model2control

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

NOTE: This procedure is indeed a convolution of variable transformations and the application of a part of $\mathbf{L}^T$ the square-root decomposition of B (see *)

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

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:

$$\delta \mathbf{x} = \mathbf{Kd} \;\; = \;\; (\mathbf{B}^{-1} + \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{R}^{-1} \mathbf{d} \quad \boxed{\text{phy-space}}$$

$$= \;\; \mathbf{B} \mathbf{H}^T (\mathbf{H} \mathbf{B} \mathbf{H}^T + \mathbf{R}) \mathbf{d} \quad \boxed{\text{obs-space}}$$

and its adjoint calculates:

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

$$= \;\; (\mathbf{H} \mathbf{B} \mathbf{H}^T + \mathbf{R}) \mathbf{H} \mathbf{B} \mathbf{g} \quad \boxed{\text{obs-space}}$$

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

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

```
!         read and convert output of GCM adjoint
          call allocate_state(fcgrad)
          call allocate_preds(zbias)
          zbias=zero
#ifdef GEOS_PERT
          call model_init(ierr,skiptraj=idmodel)
          call pgcm2gsi(fcgrad,'adm',ierr)
          call model_clean()
#else /* GEOS_PERT */
          fcgrad = zero ! not yet implemented
#endif /* GEOS_PERT */
          call model2control(fcgrad,zbias,fcsens)
          call deallocate_state(fcgrad)
          call deallocate_preds(zbias)
```

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

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.

Interfacing to user-specific components

- Simplest possible paradigm: FORTRAN-77-like
  - No "use" statements allowed at interface level
  - No "ifdef's" (preferred)
  - GSI Convention: package name stub_XXX.F90
  - User Convention: (suggested) name cplr_XXX.F90
- Current available GSI interfaces:
  - timermod.F90
  - set_crtm_aerosolmod.F90
  - gsi_4dcouplermod.F90
  - gsi_ensmod.F90
  - gsi_nstmod.F90
- Each of the interfaces is associated with a stub, respectively:
  - stub_timermod.F90
  - stub_set_crtm_aerosol.F90
  - stub_pertmod.F90
  - stub_ensmod.F90
  - stub_nstmod.F90

Interfacing to user-specific components
Illustration I: **timermod**

- More often than not, timing routines are user and machine specific. **timermod** allows for the possibility of a user to supply its own timing mechanism.

Actual interface: **timermod.F90**

Stub routines: **stub_timermod.F90**

```
!
public timer_ini
public timer_fnl
public timer_pri

interface timer_ini
  subroutine timer_init_ (str)
  implicit none
  character(len=*),intent(in   ) ::
  end subroutine timer_init_
end interface

interface timer_fnl
  subroutine timer_final_ (str)
  implicit none
  character(len=*),intent(in   ) :: str
  end subroutine timer_final_
end interface

interface timer_pri
  subroutine timer_pri_ (lu)
  use kinds, only : i_kind
  implicit none
  integer(i_kind),intent(in   ) :: lu
  end subroutine timer_pri_
end interface
```

This is a module; this what routines in GSI call

```
subroutine timer_init_ (str)
!$$$  subprogram documentation block
!              .      .    .
! subprogram:     timer_init_         initialize procedure
!
!   prgmmr: todling            org: gmao
!
! abstract: initializes timer
!
! program history log:
!   2007-10-01  todling
!
!   input argument list:
!      str - string designation for process to be t
!
!   output argument list:
!
! attributes:
!   language: f90
!   machine:
!
!$$$ end documentation block

implicit none
character(len=*),intent(in   ) :: str
end subroutine timer_init_
```

This is a NOT module; ONLY called by timermod

Similar for other two routines, i.e., they are empty subroutines that do nothing

Interfacing to user-specific components
Illustration I: **timermod**

- If a user wants to specify its own timings, it should provide a Coupler for the timing routines, as in **cplr_timermod** below:

User-specific routines: **cplr_timermod.F90**

These are user functions GSI knows nothing of

```fortran
subroutine timer_init_ (str)
use m_zeit, only: zeit_ci  ! A GMAO module for timing
implicit none
character(len=*),intent(in  ) :: str
call zeit_ci(str)          ! start GMAO timing for str
end subroutine timer_init_

subroutine timer_final_ (str)
use m_zeit, only: zeit_co  ! A GMAO module for timing
implicit none
character(len=*),intent(in  ) :: str
call zeit_co(str)          ! stop GMAO timing for str
end subroutine timer_final_

subroutine timer_pri_ (lu)
use kinds, only : i_kind
use mpimod, only : mype
use m_zeit, only : zeit_flush  ! A GMAO module for timing
implicit none
integer(i_kind),intent(in  ) :: lu
if(mype==0) call zeit_flush(lu,subname_at_end=.true.)
end subroutine timer_pri_
```

Prologues stripped off for display only.

Interfacing to user-specific components
Illustration I: **timermod**

- The implications of adding user-specific functions/routines to GSI are the following:
  - The Make procedure in the GSI directory can no-longer create the GSI executable.
  - The Make procedure in the GSI directory must instead create a GSI library.
  - The corresponding dummy stub must be removed from the GSI library before the executable is created. This is easily accomplished by the flags of the archiving command. For example, in Linux, to remove the **stub_timermod.o** object file that would be in the GSI library (called it libgsi.a for the time being), one can simply add the following line to the Makefile that creates the executable:
    - ➢ ar –d libgsi.a **stub_timermod.o**
  - The Make procedure creating the executable can then load the GSI library, together with the user-library containing the Coupler, that is, in the example above cplr_timermod.o, and whatever else is needed, plus the main program from GSI (**gsimain.F90**).
  - This means the **gsimain.F90** should be placed outside of GSI. For the time being, the GSI directory could still keep a copy of this program, but only for reference.

Interfacing to user-specific components
Illustration II: **set_crtm_aerosolmod**

- Aerosols can be brought into GSI via the ChemBundle. For example, to bring the 15 GOCART aerosols GMAO sets the chem_guess table as

```
chem_guess::
#      GOCART Aerosols
#   ------ Dust ------
 du001   72   1     10    dust              DU001
 du002   72   1     10    dust              DU002
 du003   72   1     10    dust
 du004   72   1     10    dust              DU004
 du005   72   1     10    dust              DU005
#      ------ Sea-salt ------
 ss001   72   1     10    ssam              SS001
 ss002   72   1     10    sscm1             SS002
 ss003   72   1     10    sscm2             SS003
 ss004   72   1     10    sscm3        |    SS004
 ss005   72   1     10    sea_salt          SS005
#      ------ Sulfates ------
 so4     72   1     10    sulfate           SO4
#      ------ Carbonaceous (main) ------
 bcphobic 72  1     10    dry_black_carbon    BCPHOBIC
 bcphilic 72  1     10    wet_black_carbon    BCPHILIC
 ocphobic 72  1     10    dry_organic_carbon  OCPHOBIC
 ocphilic 72  1     10    wet_organic_carbon  OCPHILIC
::
```

These are the internal GSI names

In ChemBundle a value of 10 means aerosol.

These are the names of the fields in file

- The settings above allow aerosols to be passed to CRTM, but the GSI Jacobians do not take into account the sensitivity of fields to the aerosols – only the radiance feel the aerosols, but not the conventional fields. It's very simple to have the Jacobian augmented to take such sensitivities into account.

Interfacing to user-specific components
Illustration II: **set_crtm_aerosolmod**

- When having aerosols passed to CRTM one thing necessary is specification of the particle sizes. This is done via a model-specific Mie calculation that requires the environment relative humidity and aerosol type. This is where the **aerosol interface** comes into play.

Stub routines: **stub_set_crtm_aerosol.F90**

Actual interface: **set_CRTM_aerosolmod.F90**

```
module set_crtm_aerosolmod

implicit none

private
public Set_CRTM_Aerosol

interface Set_CRTM_Aerosol
subroutine Set_CRTM_Aerosol_ ( km, na, aero_name, aero_conc, rh, aerosol)

use kinds, only: i_kind,r_kind
use mpimod, only: mype
use CRTM_Aerosol_Define, only: CRTM_Aerosol_type

implicit none

integer(i_kind) , intent(in)    :: km            ! number of levels
integer(i_kind) , intent(in)    :: na            ! number of aerosols
character(len=*), intent(in)    :: aero_name(na) ! [na]    GOCART aerosol names: du0001, etc.
real(r_kind),     intent(in)    :: aero_conc(km,na) ! [km,na] aerosol concentration (Kg/m2)
real(r_kind),     intent(in)    :: rh(km)        ! [km]    relative humdity [0,1]

type(CRTM_Aerosol_type), intent(inout) :: aerosol(na)! [na]   CRTM Aerosol object

end subroutine Set_CRTM_Aerosol_
end interface

end module set_crtm_aerosolmod
```

This provides a general interface

```
subroutine Set_CRTM_Aerosol ( km, na, aero_name, aero_conc, rh, aerosol)

! USES:

  use kinds, only: i_kind,r_kind
  use mpimod, only: mype
  use CRTM_Aerosol_Define, only: CRTM_Aerosol_type

implicit none

! !ARGUMENTS:

  integer(i_kind) , intent(in)    :: km            ! number of levels
  integer(i_kind) , intent(in)    :: na            ! number of aerosols
  character(len=*), intent(in)    :: aero_name(na) ! [na]    GOCART aero
  real(r_kind),     intent(in)    :: aero_conc(km,n      ... ol con
  real(r_kind),     intent(in)    :: rh(km)                     hu

  type(CRTM_Aerosol_type), intent(inout) :: 

  if(mype==0) then
    print*, 'Stub Set_CRTM_Aerosol: Co
    print*, 'Stub Set_CRTM_Aero
  endif

end subroutine Set_CRTM_Aerosol
```

This doesn't do anything.

Prologues stripped off for display only.

Interfacing to user-specific components
Illustration II: **set_crtm_aerosolmod**

- A user wanting to exercise the aerosol capability should provide it's own coupler. In the case of GMAO, the coupler looks something like this:

Actual GMAO coupler: **cplr_set_CRTM_aerosolmod.F90**

This is where the work starts to happen

f77-like Coupler calls user-specific f90 routine

Prologues stripped off for display only.

Interfacing to user-specific components Illustration II: **set_crtm_aerosolmod**

- As with timermod when we wanted to replace the stub with the real thing, we need to remove the stub from the GSI library and add our own (user-specific) library containing, in this case, the Coupler with the aerosol-specific calculations.

- This must follow the same mechanism through the Makefiles as discussed in Illustration I.

Interfacing to user-specific components Illustration III: **gsi_4dcouplermod**

- This provides the coupling mechanism to user-specific TL and AD models (will eventually be renamed gsi_pertmod.F90)

- The companion stub file is stub_pertmod.F90, that, as with other stubs, must be removed from the GSI library to allow the user to specify it's own coupler.

- This interface is more complex than those of previous illustrations. Only a sketchy illustration follows.

Actual interface:

Methods in **gsi_4dcouplermod.F90**

```
interface GSI_4dCoupler_init_traj
  subroutine pertmod_initialize_ (idmodel,rc)
  use kinds, only: i_kind
  implicit none
  logical,optional,intent(in):: idmodel
  integer(i_kind),optional,intent(out):: rc
  end subroutine pertmod_initialize_
end interface
```

Trajectory initialization

```
interface GSI_4dCoupler_model_ad
  subroutine pertmod_ADrun_(xini,xobs,iymd,ihms,ndt,rc)
  use kinds, only: i_kind
  use gsi_bundlemod, only: gsi_bundle
  implicit none
  type(gsi_bundle),intent(inout):: xini ! inout: adjoint increment perturbation
  type(gsi_bundle),       pointer:: xobs ! input: adjoint perturbation state
  integer(i_kind ),intent(in   ):: iymd ! starting date (YYYYMMDD) of the adjoint perturbation state
  integer(i_kind ),intent(in   ):: ihms ! starting time (HHMMSS) of the adjoint perturbation state
  integer(i_kind ),intent(in   ):: ndt  ! Number of time steps to integrate TLM for
  integer(i_kind ),optional,intent(out):: rc    ! return status code
  end subroutine pertmod_ADrun_
end interface
```

Run ADM

```
!
! !PUBLIC MEMBER FUNCTIONS:
!
public GSI_4dCoupler_parallel_init
public GSI_4dCoupler_setServices
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
```

Interfacing to user-specific components Illustration III: **gsi_4dcouplermod**

- Both GMAO and NCEP have interfaced their TLM/ADM to GSI. The former has interfaced two different models, the most recent one being fully ESMF-capable; the latter has interfaced a perturbation model based on integrating the tendencies originally available in GSI.

- As illustration we show some of the NCEP perturbation model interface. This is composed mainly of two components:
  - cplr_pertmod: An f77-like coupler providing a replacement of stub_pertmod
  - ncep_permod: A f90 module providing the entry point to the perturbation model, and it's TL and AD counterparts.
  - For now, a specific feature of the perturbation model implementation is that the observer must "run the non-linear model" (that is the perturbation model). Though quite unusual, the interface is general to easily accommodate this case.

Interfacing to user-specific components
Illustration III: **gsi_4dcouplermod**

Actual interface: **cplr_pertmod.F90**

```fortran
subroutine pertmod_initialize_(idmodel,rc)

  use kinds, only: i_kind
  use mpimod, only: mype
  use nonlinmod, only: ncep_model_nl_init
  use nonlinmod, only: ncep_model_nl
  use mpeu_util, only: tell,perr,die
  use obsmod, only: lobserver
  implicit none

  logical,optional,intent(in):: idmodel
  integer(i_kind),optional,intent(out):: rc      ! return status code
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  character(len=*),parameter :: myname_=MYNAME//'::pertmod_initialize_'
  logical:: idmodel_
  integer(i_kind):: ier

  if(present(rc)) rc=0
  idmodel_=.true.
  if(present(idmodel)) idmodel_=idmodel
  call ncep_model_nl_init(ier)
       if(ier/=0) then
         call perr(myname_,'pertmod_initialize(), rc =',ier)
         if(.not.present(rc)) call die(myname_)
         rc=ier
         return
       endif
  if(idmodel_) return
! For now, the observer runs the non-linear trajectory model
  if(.not.lobserver) return ! _rt must be another param to control NL_call
  call ncep_model_nl

end subroutine pertmod_initialize_
```

Typically, this initializes the trajectory

Relies on nonlimod.F90, a module driving the pert model

In this case, it actually runs the perturbation to generate the trajectory – in the conventional case, the trajectory would simply be read-in

Prologues stripped off for display only.

Interfacing to user-specific applications
Illustration III: **gsi_4dcouplermod**

Actual interface: **cplr_pertmod.F90**

```fortran
subroutine pertmod_TLrun_(p_xini,xobs,iymd,ihms,ntstep,rc)

  use kinds, only: i_kind
  use gsi_bundlemod, only: gsi_bundle
  use ncep_pertmod, only: ncep_4dmodel_tl
  use mpeu_util, only: tell,perr,die
  implicit none

  type(gsi_bundle),        pointer:: p_xini       ! input: increment perturbation prop
  type(gsi_bundle),intent(inout):: xobs           ! inout: TL perturbation state
  integer(i_kind ),intent(in )::  iymd    ! staring date (YYYYMMDD) of the perturbatio
  integer(i_kind ),intent(in )::  ihms    ! staring time (HHMMSS) of the perturbation
  integer(i_kind ),intent(in ):: ntstep ! Number of time steps to integrate TLM for
  integer(i_kind ),optional,intent(out):: rc    ! return status code

      !! t := (nymdi,nhmsi); n:=ntstep; xi:=xini; yo:=xobs
      !! e(t) = A(t)*xi(t)
      !! z(t+n) = M(t+n,t)*[z(t)+e(t)]
      !! yo(t+n) = G(t+n)*z(t)
!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  character(len=*),parameter :: myname_=MYNAME//'::pertmod_TLrun_'
  integer(i_kind):: ier

  if(present(rc)) rc=0
  call ncep_4dmodel_tl(p_xini,xobs,iymd,ihms,ntstep,ier)
      if(ier/=0) then
        call perr(myname_,'pertmod_TLrun(), rc =',ier)
        if(.not.present(rc)) call die(myname_)
        rc=ier
        return
      endif

end subroutine pertmod_TLrun_
```

Typically, this runs the TLM

Relies on ncep_pertmod.F90, a module driving the TL/AD pert model

NOTE: ncep_permod.F90, and nonlinmod.F90 do not live inside GSI – they are part of the so-called NCEP_Coupler library. This also includes various other codes specific to the perturbation model.

Procedure from ncep_pertmod.F90 that actually integrations TLM

Prologues stripped off for display only.

Interfacing to user-specific applications
Illustration III: **gsi_4dpertmod**

Actual interface: **cplr_pertmod.F90**

The handy put method: GSI_4dCoupler_putpert allows for writing of the Increment as it evolves within the assimilation window, see routine view_st

```
! write out perturbation vector
mydate = ibdate
do ii=1,nobs_bins
   nymd = 10000*mydate(1)+mydate(2)*100+mydate(3)
   nhms = 10000*mydate(4)
   ! iwrtinc ...

   if(mype==0) then
     write(6,'(2a,i8.8,2x,i6.6)')trim(myname_),': start writing state on ', nymd
   endif
   call gsi_4dcoupler_putpert (sval(ii),nymd,nhms,'tlm',filename)

   ! increment mydate ...
   fha(:)=0.0; ida=0; jda=0
   fha(2)=nhr_obsbin! relative time interval in hours
   ida(1)=mydate(1) ! year
   ida(2)=mydate(2) ! month
   ida(3)=mydate(3) ! day
   ida(4)=0         ! time zone
   ida(5)=mydate(4) ! hour
   ! Move date-time forward by nhr_assimilation hours
   call w3movdat(fha,ida,jda)
   mydate(1)=jda(1)
   mydate(2)=jda(2)
   mydate(3)=jda(3)
   mydate(4)=jda(5)
enddo
```

The PUT from the coupler
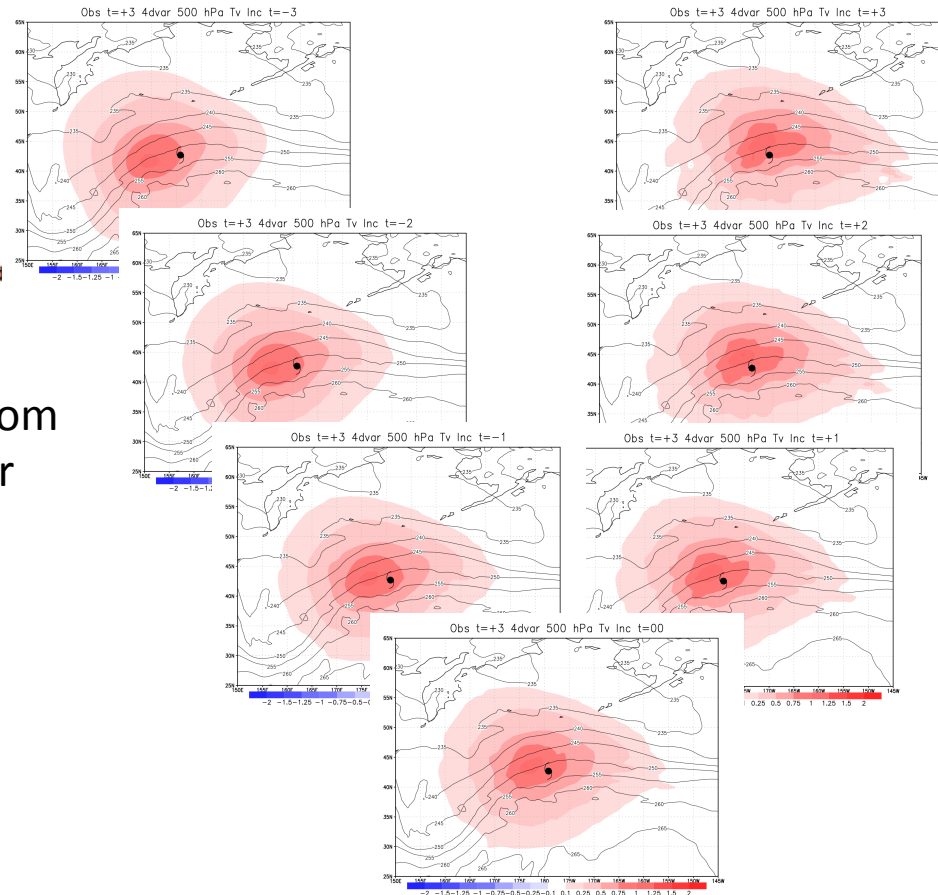
Prologues stripped off for display only.

Interfacing to user-specific applications
# Illustration IV: **gsi_enscouplermod**

- At the moment this interface applies only to the Global option in GSI. In the future, a general interface will accommodate the regional option as well.

- The idea here is to allow users to inject their ensemble members in to GSI and its hybrid-ensemble component. For this, only a grid definition and a reader-like interface are needed. The bulk of what happens in these can be fully hidden from GSI; e.g., the GMAO interface does the read of the ensemble member through is ESMF-compliant reader.

- As illustration we show the general interface and a little detail of how the GMAO-coupling takes place. In GSI, the interfaces are defined through:

  - gsi_enscouplermod: An f90 interfacing defining available methods (see next page); and providing a replacement of stub_ensmod providing the f77 interface to allow GSI to build without the user-specific routines.

  - cplr_ensmod: is the set of programs defined by the user, that replace stub_ensmod at compilation (library build time).

Interfacing to user-specific applications
Illustration IV: **gsi_enscouplermod**

Actual interface: **cplr_ensmod.F90**

```
!
! !PUBLIC MEMBER FUNCTIONS:
!
public GSI_EnsCoupler_localization_grid
public GSI_EnsCoupler_get_user_ens
public GSI_EnsCoupler_put_gsi_ens
```
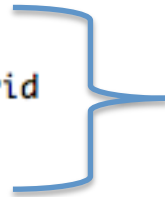
The only methods currently needed are:
- a grid definition
- a get – to retrieve user's members
- a put to allow writing of ensemble perturbations

NOTE: none are these are yet general enough, in particular, the get is tied up to the variables currently participating in the hybrid covariance. Some time in the near future we'll make this general so the bundle user to feed the necessary fields for hybrid can carry whatever the users desires (e.g., aerosol members, or CO, etc).

MetGuess/ChemGuess_Bundle

- Presently, ChemGuess_Bundle allows flexible input of Chem-related fields (tracers and aerosols) to GSI.

- A desirable similar flexibility to handle all of the other (meteorological) guess fields, motivates introduction of MetGuess_Bundle.

- Just as with ChemGuess, MetGuess_Bundle is controlled by a table named met_guess added to the anavinfo resource file. Examples are given below:

GMAO

```
met_guess::
!var   level  crtm_use  desc                    orig_name
  cw    72     10        cloud_condensate        qctot
  ql    72     -1        Water                  |qltot
  qi    72     -1        Ice                     qitot
#qr     72     10        Rain                    qrtot
#qs     72     10        Snow                    qstot
#qg     72     10        Graupel                 qg
#qh     72     10        Hail                    qh
#cf     72      2        cloud_frac4rad(fcld) cloud
::
```

NCEP Global

```
met_guess::
!var   level  crtm_use  desc                    orig_name
  cw    64     10        cloud_condensate        qctot
#ql     64     -1        Water                   qltot
#qi     64     -1        Ice                     qitot
#qr     64     10        Rain                    qrtot
#qs     64     10        Snow                    qstot
#qg     64     10        Graupel                 qg
#qh     64     10        Hail                    qh
#cf     64      2        cloud_frac4rad(fcld) cloud
::
```

MetGuess_Bundle: Methods

- As with ChemGuess, MetGuess does not handle parallelization; e.g., fields are on subdomains. This means filling up this bundle must be done by the user after reading the guess and distributing it onto subdomains.

- Presently, the available methods in MetGuess are:

```
public :: gsi_metguess_create_grids
public :: gsi_metguess_destroy_grids
public :: gsi_metguess_init
public :: gsi_metguess_get
public :: gsi_metguess_final
```

- The trickiest of the Methods is the GET. It's easy to use but has multiple capability. Examples of the GET function are given in the ProTex documentation available in the source code. Here a couple of simple examples follow:

MetGuess_Bundle: Methods

- Examples of using the GET Method:
  - Say a routine wants to know whether or not the variable cw is in MetGuess_Bundle. This can be done simply with the call

    call gsi_metguess_get ('var::cw', ivar, ier)

    if ivar is grater than zero, the variable is present in the bundle.
  - Say a routine wants to get the number of all 3d cloud fields in the MetGuess_Bundle, this can be done by use the tag clouds::3d, as in:

    call gsi_metguess_get ('clouds::3d',n,ier)

    notice this uses the same interface as in the example above, but returns

    information about something else.
  - Say a routine wants the name of all 3d cloud-fields

    call gsi_metguess_get ('clouds::3d',cld3dnames,ier)

     now the returned variable cld3dnames is a character array with the names of all 3d-cloud-guess. Notice it is important to inquire before hand about the number of 3d-cloud fields available and to properly allocate space for the character arrays cld3dnames, and only then make the call above.
  - Other functionalities and inquire modes are available.

MetGuess/Chem_Guess_Bundle Remarks and Work in Progress

- As ChemGuess_Bundle, MetGuess_Bundle is treated as a common block.
  - This means you cannot, for the time being, instantiate it.
- As ChemGuess_Bundle, MetGuess_Bundle is an almost opaque object.
  - This means only methods are available to the outside world, and the bundle itself (common block, for now)
- In the present (upcoming, June 2011) version of the GSI NCEP trunk, only clouds are being handled by MetGuess. That is, winds, temperature, specific humidity, and all other meteorological fields are still handled in guess_grids (as the ges_X variables). This will change in the near future.

General Remarks: Adding new observing instruments

- We all know adding new instruments involves changes to the following files/procedures:
  - obsmod
  - read_obs
  - setuprhsall (and addition of corresponding new setupNEW)
  - intjo, stpjo
  - addition of intNEW and stpNEW

General Remarks: Adding new observing instruments

- What many may not know is that adding new obs-instruments also requires changes to:
  - read_obsdiags/write_obsdiags
  - setupyobs
  - obs_sensitivity
  - m_rhs (possibly, when stats involved)
- Invariably when these are not changed the basic mode of running GSI may work, but hardly any of the advanced features will.

General Remarks: Others

- Use general **intrinsic math functions**, instead of specific (only) functions, that is:
    - Sqrt() rather than Dsqrt()
    - Abs() rather than Dabs()
    - Etc
- Bundle supports both single and double precision. It is important to specify the bundle KIND when creating a bundle, as in for example:

    write(bname,'(a)') 'State Vector'

    call GSI_BundleCreate(yst,grid,bname,ierror, &
    names2d=svars2d,names3d=svars3d,edges=edges, &
    bundle_kind=r_kind)

Closing Remarks

- Please know MetGuess and ChemGuess are still under development.

- Comments and concerns are always welcome.