OOPS developments at ECMWF

VarBC and MFLA

Roel Stappers

[1]Norwegian Meteorological Institute
roels@met.no

27th ALADIN Wk & HIRLAM ASM
Helsinki
3–6 April 2017

[0]With many helpful discussions with Alan, Deborah, Mats, Marcin, Olivier,...

Overview

1. Introduction
2. Multi-Resolution (Mats Hamrud)
3. IFS bugfix in balance operator (Sebastien Masssart)
4. Spamming (Olivier Marsden)
5. VarBC
6. MFLA

For more information see presentations of the second scalability day at https://software.ecmwf.int/wiki/display/OOPS/Talks+and+seminars

Introduction (why OOPS)

Formulations of DA and flexibility in OOPS

Primal formulation ($\mathbf{d} = \mathbf{y} - \mathcal{H}(x_0^g)$, $b = x_0^b - x_0^g$)

$$(\mathbf{B}^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})\delta x_0 = \mathbf{B}^{-1}b + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{d}$$

Formulations of DA and flexibility in OOPS

Primal formulation ($\mathbf{d} = \mathbf{y} - \mathcal{H}(x_0^g)$, $b = x_0^b - x_0^g$)

$$(\mathbf{B}^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})\delta x_0 = \mathbf{B}^{-1}b + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{d}$$

Saddle point formulation

$$\begin{bmatrix} \mathbf{B}^{-1} & \mathbf{H}^T \\ \mathbf{H} & -\mathbf{R} \end{bmatrix} \begin{bmatrix} \delta x \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{B}^{-1}b \\ \mathbf{d} \end{bmatrix}$$

Dual formulation (3D/4D-PSAS)

$$(\mathbf{H}\mathbf{B}\mathbf{H}^T + \mathbf{R})\lambda = -\mathbf{d} + \mathbf{H}b$$

$$\delta x = -\mathbf{B}\mathbf{H}^T\lambda + b$$

Formulations of DA and flexibility in OOPS

Primal formulation ($\mathbf{d} = \mathbf{y} - \mathcal{H}(x_0^g)$, $b = x_0^b - x_0^g$)

$$(\mathbf{B}^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})\delta x_0 = \mathbf{B}^{-1}b + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{d}$$

Saddle point formulation

$$\begin{bmatrix} \mathbf{B}^{-1} & \mathbf{H}^T \\ \mathbf{H} & -\mathbf{R} \end{bmatrix} \begin{bmatrix} \delta x \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{B}^{-1}b \\ \mathbf{d} \end{bmatrix}$$

Dual formulation (3D/4D-PSAS)

$$(\mathbf{HBH}^T + \mathbf{R})\lambda = -\mathbf{d} + \mathbf{H}b$$
$$\delta x = -\mathbf{BH}^T\lambda + b$$

Weak constraint 4D-VAR

$$(\mathbf{L}^T\mathbf{D}^{-1}\mathbf{L} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})\delta\mathbf{x} = \mathbf{L}^T\mathbf{D}^{-1}\mathbf{b} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{d}$$

- Saddle point weak constraint 4D-VAR etc. EDA, EnKF, ETKF
- Flexibility to change linear equation solvers (PCG, MINRES, RPCG, GMRES)

OOPS planning

Q1-2 2017

- Model field containers
- Model refactoring
- Multi-resolution interpolations
- VarBC
- VarQC
- Singular vectors
- Restart Mechanism
- Integration and testing at all stages

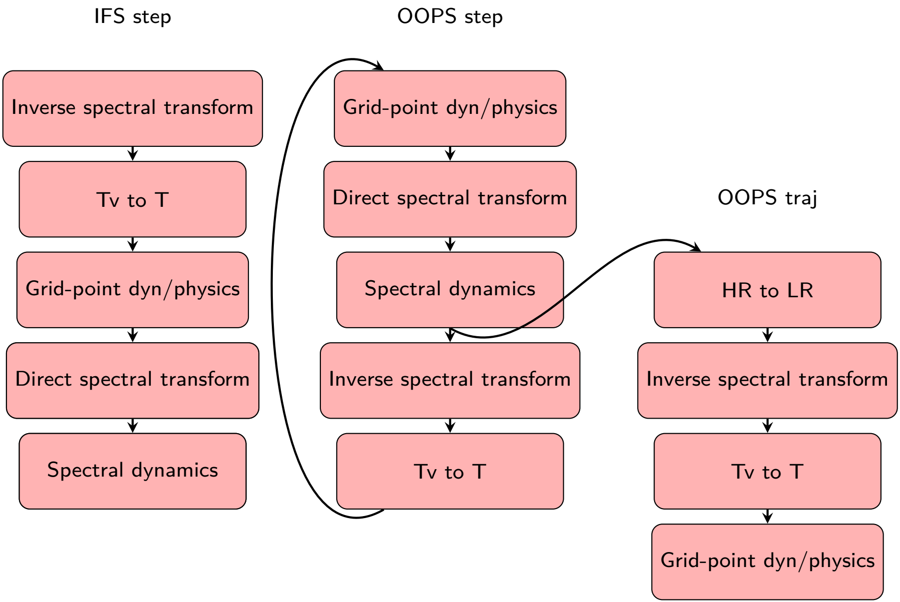
Q3-4 2017

- Optimisation and parallelisation
- IFS QC and data selection screening for OOPS
- Cycling multiple outer loop incremental 4D-Var
- Physics (model, TL, AD)
- TOVSCV
- Observation error covariance
- Fullpos + DDH processing
- Integration and testing at all stages

[1]

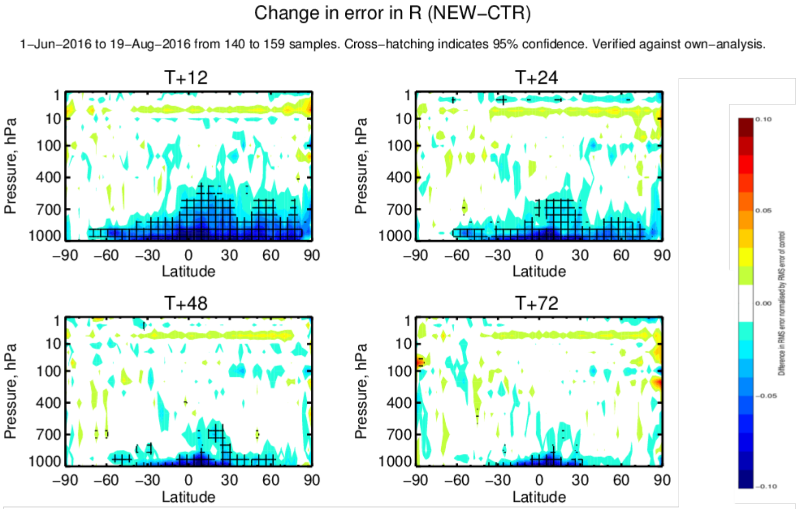
[1] For more see https://software.ecmwf.int/wiki/display/OOPS/Project+Documentation

Multi-Resolution (from Mats)

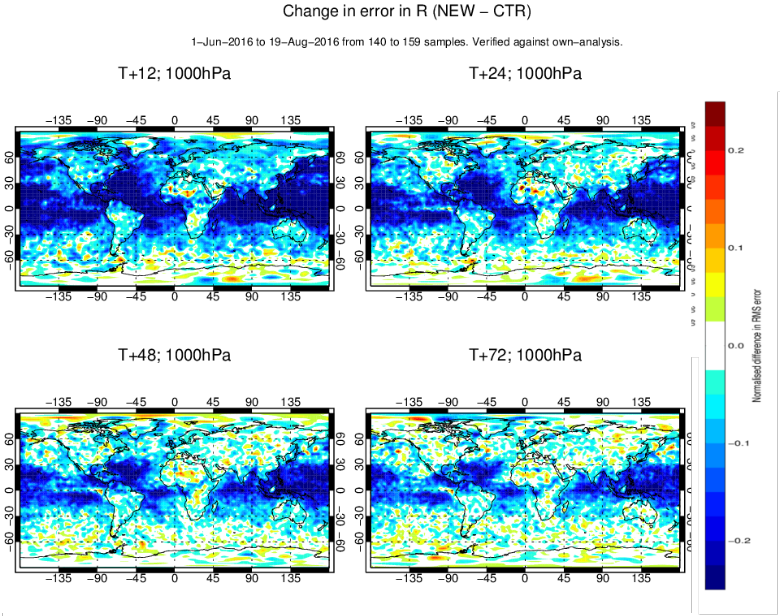


IFS Bugfix missing Tv to T conversion in balance operator (Sebastien)

Tv to T (JBCHVARI.F90). Reduction in RMSE for humidity



Tv to T (JBCHVARI.F90). Reduction in RMSE for humidity



Spamming (Olivier)

Spamming

- For cycles ≤ 41 , the IFS stored most of its mutable data in modules, and routines accessed this data implicitly via `USE MODULE, ONLY : DATA`
- A necessary but not sufficient step towards functional OOPS has been to modify all IFS routines to work on data passed as arguments. This was done "automatically".

Spamming

- For cycles ≤ 41 , the IFS stored most of its mutable data in modules, and routines accessed this data implicitly via `USE MODULE, ONLY : DATA`
- A necessary but not sufficient step towards functional OOPS has been to modify all IFS routines to work on data passed as arguments. This was done "automatically".
- From cycle 43 onwards, argument passing is implemented for geometry-related data flow field data
- From cycle 45 onwards, model-related information will be passed around.

Spamming

- For cycles ≤ 41 , the IFS stored most of its mutable data in modules, and routines accessed this data implicitly via `USE MODULE, ONLY : DATA`
- A necessary but not sufficient step towards functional OOPS has been to modify all IFS routines to work on data passed as arguments. This was done "automatically".
- From cycle 43 onwards, argument passing is implemented for geometry-related data flow field data
- From cycle 45 onwards, model-related information will be passed around.
-
```
SUBROUTINE GP_MODEL(YDGEOM,YDFLDS,YDDIMF, YDDPHY, YDSLPHY, YGFL, YDDYN, YDRIP,
YDERIP, YDTLSCAW, YDTRSCAW, YDTSCO, YDTCCO, YDCHEM, YDECLD, YDECND, YDCUMFS,
YDEGWD, YDEGWWMS, YDRADF, YDERAD, YDENEUR, YDELWRAD, YDEAERD, YDAERD15,
YDEAERATM, YDEUVRAD, YDECLDP, YDEWCOU, YDEPHLI, YDPHNC, YDPHLC, YDEAERLID,
YDEAERMAP, YDEAERSNK, YDEAERSRC, YDEAERVOL, YDEDBUG, YDEPHY, YDMCC, YDCOM, YDCOU,
YDSLREP, YDNCL, YDEGWDWMS, YDSRFTLAD, YDRCOEF, YDTNH, YDWM , YDCDDH, YDLDDH,
YDMDDH, YDSDDH, YDTDDH, YDGPDDH, YDPADDH, YDSPDDH, YDPTRSLB1, YDPTRSLB2,
YDPTRSLB15, RADGRID, YDERDI, YDCFU, YDXFU, YDSTOPH, YDECUCONVCA, YDCOAPHY, YDTRC,
GPHIST, YDSPNG, YDEDYN, YDEGEO, YDEGSL, YDEMP, YDCVMNH, YDPHY, YDPHY0, YDPHY1,
YDPHY2, YDPHY3, YDPHYDS, YDTOPH, YDVDOZ, YDSIMPHL, YDARPHY, YDMSE, CDCONF)
```

Grouping of modules

An additional level of derived types has been added, to attempt some grouping of concerns :

```
type model_general_conf_type     GEOM, YRDIMF, YGFL, YRRIP
type model_atmos_ocean_coupling_type YRMCC, YRCOM, YRCOU
type model_wave_coupling_typed  YREWCOU
type model_lam_coupling_type    YRELBC0B, YRELBC0C

type model_dynamics_type        YRDYN, YRSPNG, YRPTRGPPC, YYTLSCAW, YYTLSCAWH,
                                YYTRSCAW,YYTRSCAWH, YYTSCO, YYTCCO, YRSLREP,
                                YRPTRSLB1,YRPTRSLB2,YRPTRSLB15, YRTNH
type model_physics_general_type YRDPHY YRSLPHY YRCOAPHY
type model_physics_ecmwf_type   YREPHY YRECLD YRECLDP YRECND YRECUMF YRECUCONV
                                YREGWD YREGWWMS
type model_physics_simplinear_type YREPHLI,YRCUMFS,YREGWDWMS, YRECUMF2,YRPHLC,
                                 YRPHNC,YRNCL, YRSRFTLAD, GPHIST
```

Grouping of modules

An additional level of derived types has been added, to attempt some grouping of concerns :

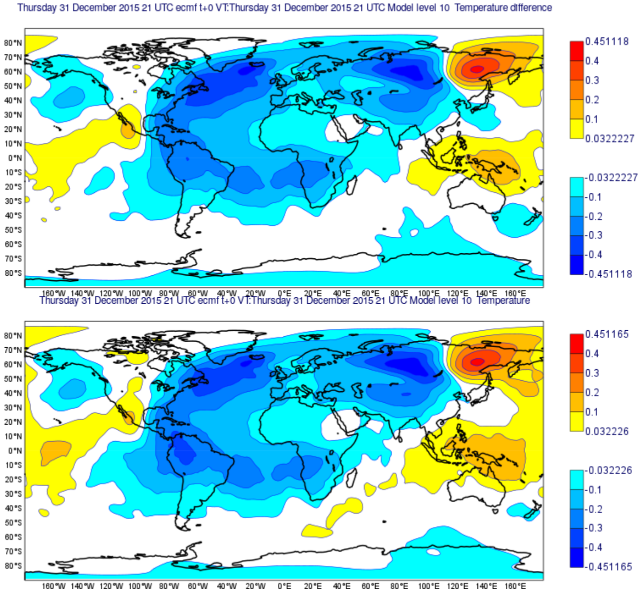
```
type model_general_conf_type     GEOM, YRDIMF, YGFL, YRRIP
type model_atmos_ocean_coupling_type YRMCC, YRCOM, YRCOU
type model_wave_coupling_typed  YREWCOU
type model_lam_coupling_type    YRELBC0B, YRELBC0C

type model_dynamics_type        YRDYN, YRSPNG, YRPTRGPPC, YYTLSCAW, YYTLSCAWH,
                                YYTRSCAW,YYTRSCAWH, YYTSCO, YYTCCO, YRSLREP,
                                YRPTRSLB1,YRPTRSLB2,YRPTRSLB15, YRTNH
type model_physics_general_type YRDPHY YRSLPHY YRCOAPHY
type model_physics_ecmwf_type   YREPHY YRECLD YRECLDP YRECND YRECUMF YRECUCONV
                                YREGWD YREGWWMS
type model_physics_simplinear_type YREPHLI,YRCUMFS,YREGWDWMS, YRECUMF2,YRPHLC,
                                 YRPHNC,YRNCL, YRSRFTLAD, GPHIST
```

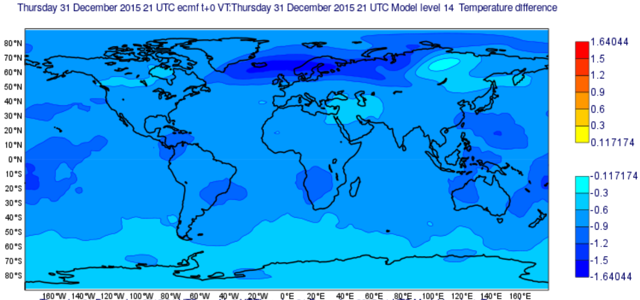
Approximately 1800 files will change.

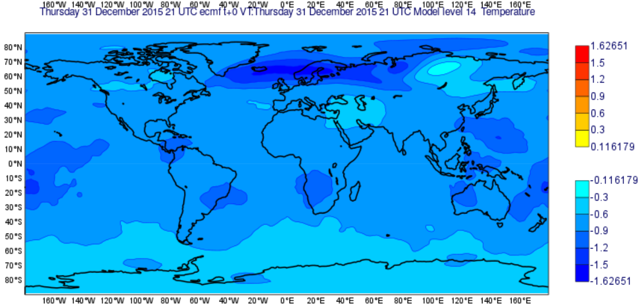
VarBC

Temperature level 10 IFS (top) versus OOPS, only ATOVS

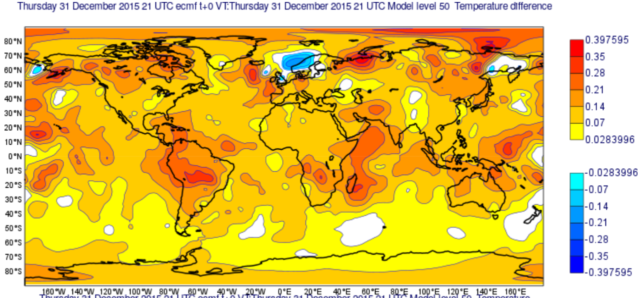


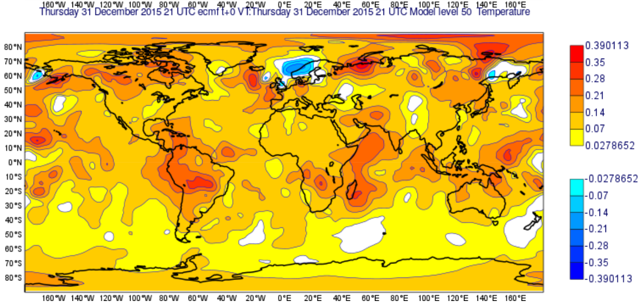
Temperature level 14 IFS (top) versus OOPS, only ATOVS



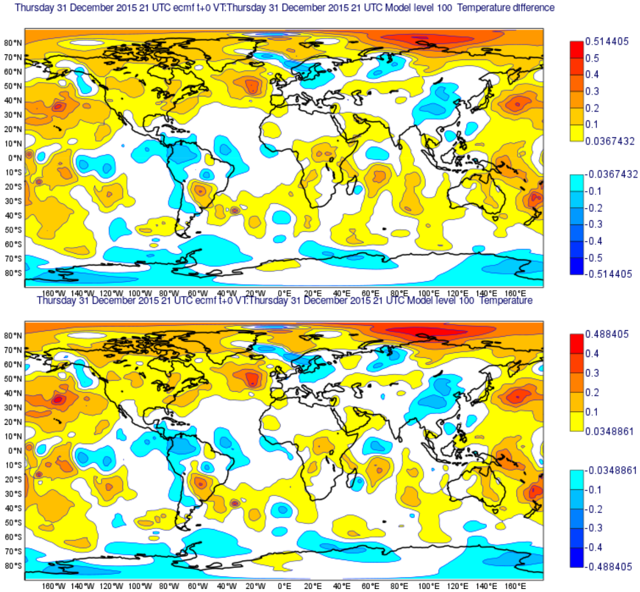


Temperature level 50 IFS (top) versus OOPS, only ATOVS

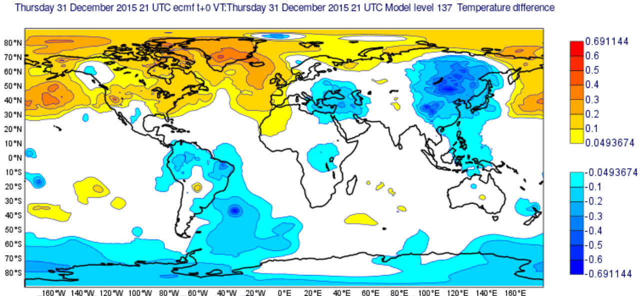


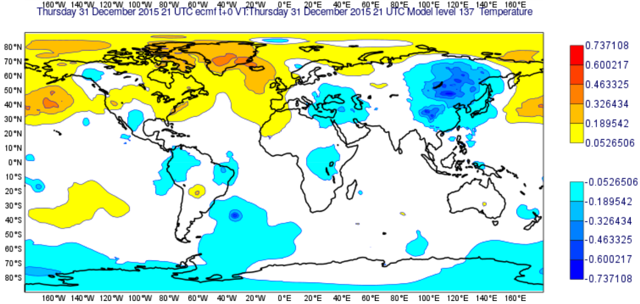


Temperature level 100 IFS (top) versus OOPS, only ATOVS



Temperature level 137 IFS (top) versus OOPS, only ATOVS





Explanation of the differences

$$J(\delta x, \delta\beta) = \frac{1}{2} \left\| \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} - \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{H}_x & \mathbf{H}_\beta \end{bmatrix} \begin{bmatrix} \delta x_0 \\ \delta\beta \end{bmatrix} \right\|^2_{\tilde{\mathbf{B}}^{-1}}$$

Hessian

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{H}_x^T \\ \mathbf{0} & \mathbf{I} & \mathbf{H}_\beta^T \end{bmatrix} \begin{bmatrix} \mathbf{B}_0 & 0 & 0 \\ 0 & \mathbf{B}_\beta & 0 \\ 0 & 0 & \mathbf{R} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{H}_x & \mathbf{H}_\beta \end{bmatrix} = \begin{bmatrix} \mathbf{B}_0^{-1} + \mathbf{H}_x\mathbf{R}^{-1}\mathbf{H}_x & \mathbf{H}_x^T\mathbf{R}^{-1}\mathbf{H}_\beta \\ \mathbf{H}_\beta^T\mathbf{R}^{-1}\mathbf{H}_x & \mathbf{B}_\beta^{-1} + \mathbf{H}_\beta^T\mathbf{R}^{-1}\mathbf{H}_\beta \end{bmatrix}$$

- Dick Dee (2004): Preconditioning with $\mathbf{B}_\beta$ is not effective for the lower right block. Instead in the IFS an approximation of the inverse of $\mathbf{B}_\beta^{-1} + \mathbf{H}_\beta^T\mathbf{R}^{-1}\mathbf{H}_\beta$ is used for the preconditioning.
- The control variable transform uses $L = \left(B_\beta^{-1} + \frac{m}{\sigma_o^2} C\right)^{1/2}$ Where C is a globally averaged covariance of predictors.
- The control variable transform is not based on the background error covariance. The current OOPS code can't handle such preconditioners. It is open what is the best way to proceed.

Explanation of the differences

$$J(\delta x, \delta\beta) = \frac{1}{2} \left\| \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} - \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{H}_x & \mathbf{H}_\beta \end{bmatrix} \begin{bmatrix} \delta x_0 \\ \delta\beta \end{bmatrix} \right\|^2_{\tilde{\mathbf{B}}^{-1}}$$

Hessian

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{H}_x^T \\ \mathbf{0} & \mathbf{I} & \mathbf{H}_\beta^T \end{bmatrix} \begin{bmatrix} \mathbf{B}_0 & 0 & 0 \\ 0 & \mathbf{B}_\beta & 0 \\ 0 & 0 & \mathbf{R} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{H}_x & \mathbf{H}_\beta \end{bmatrix} = \begin{bmatrix} \mathbf{B}_0^{-1} + \mathbf{H}_x\mathbf{R}^{-1}\mathbf{H}_x & \mathbf{H}_x^T\mathbf{R}^{-1}\mathbf{H}_\beta \\ \mathbf{H}_\beta^T\mathbf{R}^{-1}\mathbf{H}_x & \mathbf{B}_\beta^{-1} + \mathbf{H}_\beta^T\mathbf{R}^{-1}\mathbf{H}_\beta \end{bmatrix}$$

- Dick Dee (2004): Preconditioning with $\mathbf{B}_\beta$ is not effective for the lower right block. Instead in the IFS an approximation of the inverse of $\mathbf{B}_\beta^{-1} + \mathbf{H}_\beta^T\mathbf{R}^{-1}\mathbf{H}_\beta$ is used for the preconditioning.
- The control variable transform uses $L = \left(B_\beta^{-1} + \frac{m}{\sigma_o^2} C\right)^{1/2}$ Where C is a globally averaged covariance of predictors.
- The control variable transform is not based on the background error covariance. The current OOPS code can't handle such preconditioners. It is open what is the best way to proceed.
- Can we cycle $(\mathbf{B}_\beta)_{k+1} = \left((\mathbf{B}_\beta^{-1})_k + \mathbf{H}_\beta^T\mathbf{R}^{-1}\mathbf{H}_\beta\right)^{-1}$, i.e. have an extended Kalman filter? Can CG compute the eigen vectors of the lower right block, exploiting the special block diagonal structure (if observations are ordered by bias group)?

varbc_class.F90

```
  TYPE, PUBLIC :: CLASS_VARBC
    PRIVATE
    INTEGER(KIND=JPIM)         :: NOBGROUP    ! number of bias groups
    INTEGER(KIND=JPIM), PUBLIC :: NOBPARAM    ! total number of bias parameters (dimension of augmented control
    INTEGER(KIND=JPIM), PUBLIC :: MXNPRED     ! number of predictors
    INTEGER(KIND=JPIM)         :: MXNPARAM    ! number of params (just MXNPRED+1)
    INTEGER(KIND=JPIM), PUBLIC :: MXBODYPRED ! number of predictors with body-dependence
    INTEGER(KIND=JPIM)         :: BODYPREDSTART ! start ID of predictors with body-dependence
    REAL(KIND=JPRB),    PUBLIC :: AFJPCOST = 0.0_JPRB  ! VarBC cost function; initialised to initial contribut
    TYPE(TYPE_VARBC), POINTER :: YVARBC(:)=>NULL() ! main VarBC table
    CHARACTER(LEN=JPREDNAME), POINTER :: CPREDDESC(:)=>NULL() ! predictor descriptions
    REAL(KIND=JPRB), POINTER :: PPREDNORM(:,:)=>NULL() ! predictor normalization: global mean, stdv
    INTEGER(KIND=JPIM), POINTER :: IHSTFGDEP_COMP(:,:,:)=>NULL() ! nhstfgdep partial sum
   CONTAINS
    PROCEDURE :: SETUP_TRAJ
    PROCEDURE :: SETUP_MIN
    PROCEDURE :: PREDICTORS
    PROCEDURE :: BODY_PREDICTORS
    PROCEDURE :: ACCUM_STATS
    PROCEDURE :: BIAS
    PROCEDURE :: BIASAD
    PROCEDURE :: HIST_INITIALISE
    PROCEDURE :: HIST
    PROCEDURE :: HIST_COMPLETE
    PROCEDURE :: SUM_INITIALISE
    PROCEDURE :: SUM_COMPLETE
    PROCEDURE :: GRADSTATS
    PROCEDURE :: PARAM_INIT
    PROCEDURE :: PARAM_ZERO
    PROCEDURE :: PARAM_ONE
    PROCEDURE :: PARAM_GET
    PROCEDURE :: PARAM_SET
    PROCEDURE :: COMPUTE_JP
    PROCEDURE :: FINALISE_TRAJ
    PROCEDURE :: FINALISE_MIN
    PROCEDURE :: DELETE
   #if !defined (__GFORTRAN__)
    FINAL :: VARBC_FINAL
   #endif
   END TYPE
```

varbc_table.F90

```
TYPE TYPE_VARBC
    INTEGER(KIND=JPIM)                  :: PREVDATE     ! previous date when group was encountered
    INTEGER(KIND=JPIM)                  :: NPARAM       ! number of bias parameters
    INTEGER(KIND=JPIM)                  :: NCOUNT       ! data count
    CHARACTER(LEN=8)                    :: OBSCLASS     ! observation class
    CHARACTER(LEN=80)                   :: GROUPKEY     ! class-dependent group description
    INTEGER(KIND=JPIM), ALLOCATABLE     :: NPREDCS(:)   ! list of predictors
    REAL(KIND=JPRB),    ALLOCATABLE     :: APARAMS(:)   ! bias parameters - latest estimate
    REAL(KIND=JPRB),    ALLOCATABLE     :: ZPARAMS(:)   ! bias parameters - prescribed values
    INTEGER(KIND=JPIM)                  :: NCSTART      ! coldstart option
    LOGICAL                             :: LLBKGCON     ! flag for background constraint
    LOGICAL                             :: LLMASKRS     ! flag for radiosonde masking
    LOGICAL                             :: LLMASKCLD    ! flag for cloud-cover masking
    LOGICAL                             :: LLMODE       ! flag for mode correction
    LOGICAL                             :: LLINCR       ! flag for incremental solution
    LOGICAL,            ALLOCATABLE     :: LLCONST(:)   ! flag for constant parameters
    REAL(KIND=JPRB)                     :: OBSERR       ! rms obs error
  ! Parameters for CVARBC
    LOGICAL                             :: LCVARBC      ! flag for constrained VarBC
    REAL(KIND=JPRB)                     :: BIASCOR_CON  ! prescribed bias value for CVarBC
    REAL(KIND=JPRB)                     :: BIASERR_CON  ! error for the size of the bias correction
    REAL(KIND=JPRB)                     :: ALPHA_CON    ! tuning factor for CVARBC cost function
  ! End parameters for CVARBC
    REAL(KIND=JPRB),    ALLOCATABLE     :: BKGERR(:)    ! background error std dev
    REAL(KIND=JPRB),    ALLOCATABLE     :: APARAM0(:)   ! background parameter values
    REAL(KIND=JPRB),    ALLOCATABLE     :: APARAM5(:)   ! trajectory parameter valuess
    REAL(KIND=JPRB),    ALLOCATABLE     :: ACHGVAR(:,:) ! change-of-variable operator
    REAL(KIND=JPRB),    ALLOCATABLE     :: ACVARIN(:,:) ! inverse change-of-variable
    REAL(KIND=JPRB),    ALLOCATABLE     :: AGRAD(:)     ! gradient w/r to bias parameters
    REAL(KIND=JPRB),    ALLOCATABLE     :: AGRAD0(:)    ! initial gradient w/r to bias parameters
    INTEGER(KIND=JPIM), ALLOCATABLE     :: NHSTFGDEP(:) ! histogram of background departures
    REAL(KIND=JPRB)                     :: DFGDEP       ! histogram range
    REAL(KIND=JPRB)                     :: QCPARMS(JPMXNQCPARMS)   ! QC parameters
    INTEGER(KIND=JPIM), ALLOCATABLE     :: MPREDXCNT(:,:) ! number of observations per predictor covariance
    REAL(KIND=JPRB),    ALLOCATABLE     :: APREDMEAN(:)   ! mean predictor
    REAL(KIND=JPRB),    ALLOCATABLE     :: APREDXCOV(:,:) ! predictor covariance
    INTEGER(KIND=JPIM)                  :: NCOMP          ! local number of predictor contributions
    INTEGER(KIND=JPIM)                  :: MCOMP          ! local current predictor contributions
    REAL(KIND=JPRB),    ALLOCATABLE     :: APREDMEAN_COMP(:,:)   ! mean predictor contributions
    REAL(KIND=JPRB),    ALLOCATABLE     :: APREDXCOV_COMP(:,:,:) ! predictor covariance contributions
```

IFS Varbc

- How to implement copy construction and copy assignment for such object?

IFS Varbc

- How to implement copy construction and copy assignment for such object?
- Make shallow copy of background and first guess related fields using non-owning pointers in `class_varbc`? Very ticky to get right especially when we are updating the first guess related fiels.
- Splitting `class_varbc` and `varbc_table` might be possible but the object is used in 103 subroutines (called 400 times). Want to avoid passing e.g. background and increment, covariance information separately for each of these.
- Probably a hacky solution will be implemented.

To do for VarBC

- Radiosondes
- Fix the IFS implementation of VarBC for AIREP (Lars CY45R1)
- Copy constructor for `class_varbc`
- Check convergence for BIAS PARAM (write out VarBC.cycle file from OOPS)
- Compare with OZONE obs only.
- Preconditioning of VarBC. Kalman Filter in IFS? Change minimization algorithms in OOPS?

Matrix free linear algebra

Hessian in OOPS

```
  void multiply(const CtrlInc_ & dx, CtrlInc_ & dz) const {
//  Setup TL terms of cost function
    PostProcessorTL<Increment_> costtl;
    JqTermTL_ * jqtl = j_.jb().initializeTL();
    costtl.enrollProcessor(jqtl);
    unsigned iq = 0;
    if (jqtl) iq = 1;
    for (unsigned jj = 0; jj < j_.nterms(); ++jj) {
      costtl.enrollProcessor(j_.jterm(jj).setupTL(dx));
    }

//  Run TLM
    j_.runTLM(dx, costtl);

//  Finalize Jb+Jq
//  Get TLM outputs, multiply by covariance inverses and setup ADJ forcing terms
    PostProcessorAD<Increment_> costad;
    dz.zero();
    CtrlInc_ dw(j_.jb());

//  Jb
    CtrlInc_ tmp(j_.jb());
    j_.jb().finalizeTL(jqtl, dx, dw);
    j_.jb().multiplyBinv(dw, tmp);
    JqTermAD_ * jqad = j_.jb().initializeAD(dz, tmp);
    costad.enrollProcessor(jqad);
    j_.zeroAD(dw);

//  Jo + Jc
    for (unsigned jj = 0; jj < j_.nterms(); ++jj) {
      boost::scoped_ptr<GeneralizedDepartures> ww(costtl.releaseOutputFromTL(iq+jj));
      boost::shared_ptr<GeneralizedDepartures> zz(j_.jterm(jj).multiplyCoInv(*ww));
      costad.enrollProcessor(j_.jterm(jj).setupAD(zz, dw));
    }

//  Run ADJ
    j_.runADJ(dw, costad);
    dz += dw;
    j_.jb().finalizeAD(jqad);
  }
```

Matrix free linear algebra in oops

Using MFLA we can now write this as

```
auto hessian   = Binv + Ht*Rinv*H;
```

Matrix free linear algebra in oops

Using MFLA we can now write this as

```
auto hessian   = Binv + Ht*Rinv*H;
```

In IncrementalAssimilation.h

```
// Define the matrices
      HMatrix<MODEL>      H(J);      // Generalized H. Refactor this
      HtMatrix<MODEL>     Ht(J);
      BMatrix<MODEL>      B(J);
      BinvMatrix<MODEL>   Binv(J);
      RinvMatrix<MODEL>   Rinv(J); // Generalized Rinv. Refactor this

  // Compute RHS.
      CtrlInc_ rhs(J.jb());
      J.computeGradientFG(rhs);
      J.jb().addGradientFG(rhs);
      rhs *= -1.0;
      CtrlInc_ dx(J.jb());

      using namespace mfla; // provides operator* and operator+
      auto hessian   = Binv + Ht*Rinv*H;

      auto dx = pcg(rhs, hessian, B, ninner);
```

Matrix free linear algebra in oops

Would like

```
// Define the matrices
      HMatrix<MODEL>     H(J);     // Generalized H. Refactor this
      BMatrix<MODEL>     B(J);
      BinvMatrix<MODEL>  Binv(J);
      RinvMatrix<MODEL>  Rinv(J); // Generalized Rinv. Refactor this

      using namespace mfla; // provides operator* and operator+
      auto rhs       = ~H*Rinv*d;
      auto hessian   = Binv + ~H*Rinv*H;

      auto dx  = pcg(rhs, hessian, B, ninner);
```

Matrix free linear algebra in oops

Would like

```
// Define the matrices
      HMatrix<MODEL>      H(J);      // Generalized H. Refactor this
      BMatrix<MODEL>      B(J);
      BinvMatrix<MODEL>  Binv(J);
      RinvMatrix<MODEL>  Rinv(J); // Generalized Rinv. Refactor this

      using namespace mfla; // provides operator* and operator+
      auto rhs        = ~H*Rinv*d;
      auto hessian    = Binv + ~H*Rinv*H;

      auto dx  = pcg(rhs, hessian, B, ninner);
```

- Note `HMatrix` here is a generalized HMatrix, this should be split, the dependency on the cost function object `J` removed. Try to let HMatrix be an interface to `SUBROUTINE OBS_EQUIV_TL`. (In ifs/oops/allobs_oper_mod.F90)
- Want an explicit class for the model propagator and operator L in weak constraint 4D-VAR making sure that applying the Hessian consists of a single TL and AD integration.
- Also remove the template on `<MODEL>`.

Prod.h

```
template<class S, class T>
class Prod {
 private:
  typedef typename std::remove_reference<S>::type::domain_type dom1;
  typedef typename std::remove_reference<T>::type::codomain_type cod2;
  static_assert(std::is_same<dom1, cod2>::value, "domain1␣!=␣codomain2");
 public:
  typedef typename std::remove_reference<T>::type::domain_type   domain_type;
  typedef typename std::remove_reference<S>::type::codomain_type codomain_type;

  Prod(S && s, T && t) : _s(std::forward<S>(s)), _t(std::forward<T>(t)) { }
  Prod(const Prod&)              = delete;
  Prod& operator=(const Prod&)   = delete;
  Prod& operator=(Prod&&)        = delete;
  Prod(Prod&& )                  = default;

  void multiply(const domain_type & v, codomain_type & w )  const {
      cod2 t;
     _t.multiply(v,t);
     _s.multiply(t,w);
  }

//  codomain_type operator*(const domain_type & v)  const {return _s*(_t*v); }

 private:
  S _s;
  T _t;
};

// Creator function
template<class S, class T>
Prod<S, T> operator*(S&& s,T&& t) {
  return Prod<S, T>(std::forward<S>(s),std::forward<T>(t));
}
```

Thank you for your attention! Questions?