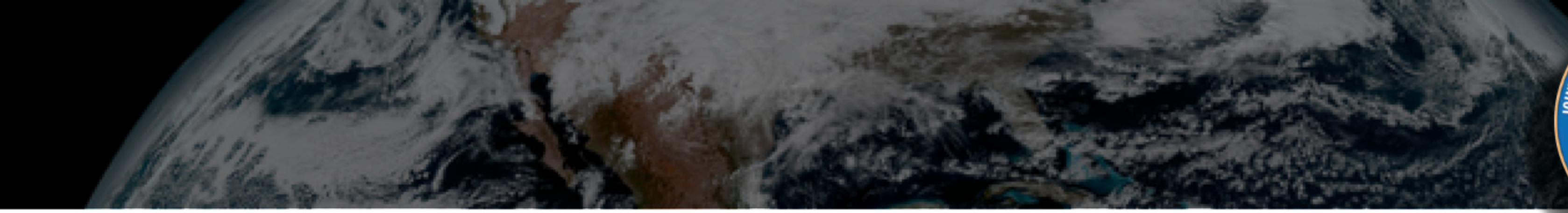

The Joint Effort for Data assimilation Integration

Observation Operators
(Unified Forward Operators, UFO)

Joint Center for Satellite Data Assimilation (JCSDA)

Unified Forward Operators (UFO)

- Main idea: have forward operators as independent from the models as possible, so the "unified" forward operators can be easily shared between different models
- Grid/model-specific part of the observation operator is decoupled from the rest of the observation operator
- Example: "full" observation operator H_{full} (that takes full state on input) that can be written as

$$H_{full}(x_{full}) = H\left(Int(x_{full})\right) = H(x_{loc})$$

where Int is horizontal and time interpolation (to observation lat-lon-time) operator

Forward operator: interpolation and UFO

$$H_{full}(x_{full}) = H\left(Int(x_{full})\right) = H(x_{loc})$$

x_{full} is full model state (State)

Int : horizontal interpolator; called getValues; implemented in the model (Dan's talk tomorrow)

$x_{loc} = Int(x_{full})$ is model state interpolated horizontally and in time to observation locations (Geographic Values at Locations; GeoVaLs)

H : observation operator (after horizontal interpolation);
implemented in UFO, used by different models

$H(x_{loc}) = H_{full}(x_{full})$ is model equivalent in the observation space (ObsVector)

Interpolated model state (GeoVaLs)

- GeoVaLs are vertical profiles of requested model variables at observation x-y-t location. Forward operator defines which variables it needs from the model to compute H(x)).
- Examples:
  - radiances: vertical profiles of t, q, ozone, pressure; surface variables: wind, SST, land properties, etc.
  - radiosondes/aircrafts: vertical profiles of pressure (to do vertical interpolation), t, u, v, q
  - sea ice concentration retrieval: sea ice concentrations for different ice thickness categories
  - SST retrieval: SST (observation operator becomes an identity in this case)

Implementing new observation operator in UFO

One needs to implement:

- Setup routine:
  - define which variables observation operator needs from the model;
  - define which observation “variables” will be calculated in H(x) (channels list for radiances, “variables” list for conventional observations (e.g. t, u, v))
- Observation operator routine
  - Input: GeoVaLs x_{loc} (interpolated model vertical profiles)
  - Output: ObsVector $H(x_{loc})$ (model equivalent in the observation space)
  - Observation operator also has access to information from ObsSpace (metadata: e.g., observation pressure for radiosonde, scan angle for radiances, etc)

Radiosonde simple example: setup routine

```
subroutine conventional_profile_setup_(self, c_conf)
  class(ufo_conventional_profile), intent(inout) :: self
  ...

  !> "variables" in obsvector: just t
  self%varout(1) = 'air_temperature'
  !> variables we need from the model: t and pressure for vertical interpolation
  self%varin(1) = "virtual_temperature"
  self%varin(2) = "atmosphere_ln_pressure_coordinate"
  ...
end subroutine conventional_profile_setup_
```

Radiosonde simple example: observation operator

```
subroutine conventional_profile_simobs_(self, geovals, hofx, obss)
  type(ufo_geovals), intent(in)    :: geovals
  real(c_double),    intent(inout)    :: hofx(:)
  real(kind_real), dimension(:), allocatable   :: obspressure
  type(ufo_geoval), pointer                    :: presprofile, profile

  ! Get pressure profiles from geovals
  call ufo_geovals_get_var(geovals, "atmosphere_ln_pressure_coordinate", &
                           presprofile)
  ! Get the observation vertical coordinates
  call obsspace_get_db(obss, "MetaData", "air_pressure", obspressure)
  ! Calculate the vert interpolation weights, and interpolate from
  ! geovals to observational vert location into hofx
  do iobs = 1, nlocs
    call vert_interp_weights(presprofile%nval, log(obspressure(iobs)/10.), &
                             presprofile%vals(:,iobs), wi, wf)
    call vert_interp_apply(profile%nval, profile%vals(:,iobs), hofx(iobs), wi, wf)
  enddo
end subroutine conventional_profile_simobs_
```

Implementing tangent-linear and adjoint observation operator

One needs to implement:

- Setup routine
- Set trajectory routine: calculates the Jacobian $\mathbf{H} = \left.\frac{\partial H}{\partial x}\right|_{x=x_0}$
  - Input: GeoVaLs x_0
- TL observation operator routine
  - Input: GeoVaLs dx (interpolated model vertical profiles)
  - Output: ObsVector $\mathbf{H}dx$ (model equivalent in the observation space)
- AD observation operator routine
  - Input: ObsVector dy (model equivalent in the observation space)
  - Output: GeoVals $\mathbf{H}^{\boldsymbol{T}}dy$ (interpolated model vertical profiles)

Setup of TL/AD observation operator

- TL/AD setup: as nonlinear observation operator, needs to define model variables that are needed to compute tangent linear and adjoint.
- Model variables can differ from the ones in nonlinear observation operator. For trajectory GeoVaLs, nonlinear observation operator variables are used. For TL/AD GeoVaLs one needs to specify variables that need to be changed in assimilation.
- Example: radiosonde observations don’t change model pressure. Pressure needs to be part of nonlinear observation operator variables so interpolation can be performed, but don’t need to be perturbed as part of an adjoint or passed in tangent-linear.

Implementing tangent-linear and adjoint observation operator

- Set trajectory routine: calculates the Jacobian $\mathbf{H} = \left.\frac{\partial H}{\partial x}\right|_{x=x_0}$
  - Input: GeoVaLs x_0 (variables specified in nonlinear obs operator)
- TL observation operator routine
  - Input: GeoVaLs dx (variables specified in TL/AD)
  - Output: ObsVector $\mathbf{H}dx$
- AD observation operator routine
  - Input: ObsVector dy
  - Output: GeoVals $\mathbf{H}^{T}dy$ (variables specified in TL/AD)

Radiosonde simple example: datatype to store trajectory

```
type, extends(ufo_basis_tlad) :: ufo_conventional_profile_tlad
  private
  real(kind_real), allocatable :: wf(:)      ! for interpolation weights
  integer, allocatable :: wi(:)
     ...
end type ufo_conventional_profile_tlad
```

Radiosonde simple example: set trajectory

```
subroutine conventional_profile_tlad_settraj_(self, geovals, obss)
  class(ufo_conventional_profile_tlad), intent(inout) :: self

  ! Get pressure profiles from geovals
  call ufo_geovals_get_var(geovals, "atmosphere_ln_pressure_coordinate", &
                           presprofile)
  ! Get the observation vertical coordinates
  call obsspace_get_db(obss, "MetaData", "air_pressure", obspressure)
  ! Calculate the interpolation weights
  do iobs = 1, self%nlocs
    call vert_interp_weights(presprofile%nval, log(obspressure(iobs)/10.), &
                             presprofile%vals(:,iobs), self%wi(iobs),      &
                             self%wf(iobs))
  enddo
end subroutine conventional_profile_tlad_settraj_
```

Radiosonde simple example: TL operator

```
subroutine conventional_profile_simobs_tl_(self, geovals, hofx, obss)
  class(ufo_conventional_profile_tlad), intent(in) :: self

  ! Get profile for temperature geovals
  call ufo_geovals_get_var(geovals, "virtual_temperature", profile)

  ! Interpolate from geovals to observational location into hofx
  do iobs = 1, self%nlocs
    call vert_interp_apply_tl(profile%nval, profile%vals(:,iobs), &
                              hofx(iobs), self%wi(iobs), self%wf(iobs))
  enddo

end subroutine conventional_profile_simobs_tl_
```


JOINT CENTER FOR SATELLITE DATA ASSIMILATION
NOAA NASA JCSDA Navy Air Force

In **ufo** repository (to get the latest update):

```
git checkout develop
git pull
git checkout <your-branch-name>
git merge develop
```

Practical 2: Improving radiosonde UFO

- TODO: improve “conventional profile” (we’ll test on radiosonde) code to be able to assimilate multiple observation “variables” (t, u, v).

- The code you’re provided only assimilates one “variable” (t).

- You’ll need to change conventional_profile_simobs_ routine in ufo/src/ufo/basis/ufo_conventional_profile_mod.F90 to calculate hofx for multiple variables.

Adding new variables to nonlinear radiosonde operator

- Setup routine is already updated to read all variables that need to be assimilated from the config file.
- You can test your code by running ctest --VV -R ufo_radiosonde_opr
- For the config used in this test see ufo/test/testinput/radiosonde.yaml
  - Section "variables" currently only specifies air_temperature, you'd need to add eastward_wind and northward_wind to the list of variables when you test your code on multiple variables
  - This test compares norm $\sqrt{mean(H(x))}$ of H(x) computed by UFO to the norm specified in the config file (section "rmsequiv"). For benchmark, we use GSI computed H(x) that you can find in file ioda/test/testinput/atmosphere/sondes_obs_2018041500_m.nc4. If you change the list of variables to be assimilated, the norm of H(x) will change too. Python script ufo/tools/print_gsi_norm.py might be useful to compute the GSI norm for t, u, v.

```yaml
ObsTypes:
  - ObsType: Radiosonde
    ObsData:
      ObsDataIn:
        obsfile: Data/sondes_obs_2018041500_m.nc4
      ObsDataOut:
        obsfile: Data/sondes_obs_2018041500_m_out.nc4
    variables:
    - air_temperature
    GeoVaLs:
      norm: 8471.883687854357
      random: 0
      filename: Data/sondes_geoval_2018041500_m.nc4
    ObsFilters:
    - Filter: Background Check
      variable: air_temperature
      threshold: 3.0
    rmsequiv: 242.21818
    tolerance: 1.0e-03  # in % so that corresponds to 10^-5
```

Adding new variables to TL/AD observation operator

- Once you've implemented nonlinear observation operator for multiple variables and changed your config, all UFO tests (including TL/AD) should pass,
- However, you will still need (similar) changes for TL/AD to make variational assimilation work with all observations. The changes would be in routines conventional_profile_simobs_tl_ and conventional_profile_simobs_ad_ in ufo/src/ufo/basis/ufo_conventional_profile_tlad_mod.F90