PDAF Tutorial

Implementation of the analysis step

for variant of 3D-Var

http://pdaf.awi.de

PDAF Parallel Data Assimilation Framework

V1.2 – 2024-09-19

Implementation Tutorial for 3D-Var in PDAF

We discuss the implementation
of the 3D-Var variants with PDAF

This bases on the tutorial for the implementation
of ensemble filters

The focus is on explaining the main code features

The example code is part of the PDAF source code package
downloadable at http://pdaf.awi.de

(This tutorial is compatible with PDAF V2.3 and later)

Implementation Tutorial for PDAF online / serial model

This is just an example!

For the complete documentation of PDAF's interface

see the documentation

at http://pdaf.awi.de

Overview

The implementation of 3D-Var methods in PDAF follows
R. Bannister, Q. J. Roy. Meteorol. Soc. 143 (2017) 607-633

3 variants

a) 3D-Var (with parameterized covariance matrix)
b) 3D Ensemble Var (using ensemble to represent covariances)
c) Hybrid 3D-Var (combining parameterized and ensemble covariances)

Variants involving an ensemble need to transform ensemble perturbations

The methods use the

- global ESTKF or
- local LESTKF

Contents

3D-Var

Overview

3D-Var Method

Cost function at fixed time:

$$J(\mathbf{x}) = \underbrace{(\mathbf{x}-\mathbf{x}^b)^T\mathbf{B}^{-1}(\mathbf{x}-\mathbf{x}^b)}_{\text{background term}} + \underbrace{(\mathbf{y}-H[\mathbf{x}])^T\mathbf{R}^{-1}(\mathbf{y}-H[\mathbf{x}])}_{\textit{observation term}}$$

3D-Var method:

At a given time minimize $J(\mathbf{x})$ iteratively or solve for

$$\nabla_{\mathbf{x}} J(\mathbf{x}) = 0$$

Gradient provides direction for minimization

$$\nabla_{\mathbf{x}} J(\mathbf{x}) = 2\mathbf{B}^{-1}(\mathbf{x}-\mathbf{x}_b) - 2\mathbf{H}^T\mathbf{R}^{-1}(\mathbf{y}-H[\mathbf{x}])$$

$\mathbf{H}$: linearization of H
(derivative of H at value x)

PDAF Parallel Data Assimilation Framework

Incremental 3D-Var

Replace the cost function by a quadratic cost function in terms of increments:

Use: $\mathbf{x} = \mathbf{x}^b + \delta\mathbf{x}$ ($\delta\mathbf{x}$ will be small)

$H(\mathbf{x}) = H(\mathbf{x}^b) + \mathbf{H}\ \delta\mathbf{x}$ write $\mathbf{d} = \mathbf{y} - H(\mathbf{x}^b)$

Then we have

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

linearized **H**!

Control Vector Transformation

Use a change of variable

factorization $\mathbf{B} \approx \mathbf{L}\mathbf{L}^T$

L should be a simple matrix or covariance operator(s)

Control variable **v** with: $\delta\mathbf{x} = \mathbf{L}\mathbf{v}$

(Size of **v** *and* $\delta\mathbf{x}$ *usually different)*

Modified cost function

$$\tilde{J}(\mathbf{v}) = \frac{1}{2}\mathbf{v}^T\mathbf{v} + \frac{1}{2}(\mathbf{H}\mathbf{L}\mathbf{v} - \delta\mathbf{d})^T\mathbf{R}^{-1}(\mathbf{H}\mathbf{L}\mathbf{v} - \delta\mathbf{d})$$

→ Mathematically: Preconditioning by matrix **B**

Implementing the minimizations

Provided by subroutine (as in EnKFs)

1. Start with $\delta \mathbf{x} = 0$ which implies $\mathbf{v}^0 = 0$
2. Compute background innovation $\delta \mathbf{d}_k = \boxed{\mathbf{y}_k} - \boxed{H_k(\mathbf{x}_k^b)}$

Operation implemented as subroutine (as in EnKFs)

3. Iterations $i = 0, \ldots, i_{max}$

Direct vector difference

Compute

$$\nabla \tilde{J}(\mathbf{v}^i) = \mathbf{v}^i + \mathbf{L}^T \mathbf{H}^T \mathbf{R}^{-1} (\mathbf{H}\mathbf{L}\mathbf{v}^i - \delta \mathbf{d})$$

Step-wise calculation of the terms

Intermediate result is always a vector

*Each step for **L** and **H** implemented as subroutine ('operation')*

Computation of cost function is analogous

$(\mathbf{H}\mathbf{L}\mathbf{v}^i - \delta \mathbf{d})$ and $\mathbf{R}^{-1}(\mathbf{H}\mathbf{L}\mathbf{v}^i - \delta \mathbf{d})$ are only computed once

PDAF Parallel Data Assimilation Framework

Variational methods in PDAF

5 variants of incremental 3D-Var:

➜ Difference in representation of $\mathbf{B}^{1/2}$

- 3D-Var (parameterized covariances) $\mathbf{B} = \mathbf{L}\mathbf{L}^T$
- Ensemble 3D-Var (ensemble covariances) $\mathbf{B} = \mathbf{Z}\mathbf{Z}^T$ with $\mathbf{Z} = \frac{1}{N_e - 1}(\mathbf{X} - \overline{\mathbf{X}})$
  - ensemble transformation by global ETKF or localized LETKF
- hybrid 3D-Var (parameterized + ensemble covariances) $\mathbf{B} = [\mathbf{Z}, \mathbf{L}][\mathbf{Z}, \mathbf{L}]^T$
  - ensemble transformation by global ETKF or localized LETKF

Implementations follow Bannister, QJRMS, 2017

→ Incremental 3D-Var with control variable transform

1a) Files for the Tutorial

Tutorial implementations

Files are in the PDAF package

Directory:

```
/tutorial/3dvar/
          offline_2D_serial
          online_2D_serialmodel
          online_2D_parallelmodel
```

- Fully working implementations of user codes
- PDAF core files are in `/src`
  Makefile refers to it and compiles the PDAF library
- Only need to specify the compile settings (compiler, etc.) by environment variable PDAF_ARCH. Then compile with 'make'.

PDAF Parallel Data Assimilation Framework

Code template files

Code template files in

```
/templates/3dvar/
```

- Set of files as add-on to other template files
- Focused on 3D-Var methods
- To use the template
  - First copy files from e.g. online_2D_serialmodel
  - Second copy files for 3D-Var overwriting some of the previously copied files

PDAF library

Directory: `/src`

- PDAF library is not part of the template
- PDAF is compiled separately as a library
  and linked when the assimilation program is compiled
- Makefile includes a compile step for the PDAF library
- One can also cd to `/src` and run 'make' there
  (requires setting of PDAF_ARCH)

`$PDAF_ARCH`

- Environment variable to specify the compile specifications
- Definition files in `/make.arch`
- Define by, e.g.

  `setenv PDAF_ARCH linux_gfortran_openmpi` (tcsh/csh)

  `export PDAF_ARCH=linux_gfortran_openmpi` (bash)

PDAF Parallel Data Assimilation Framework

1b) The model and the forecast phase

Model and Forecast Phase

Model

- identical to that used for the ensemble filters
  - ➜ See tutorials on ensemble filters for details

Forecast phase

- Implementation of forecast phase is identical to that in ensemble filters
  - ➜ See tutorials on ensemble filters for details
  - ➜ Particularity
    - 3D-Var with parameterized covariances runs with ensemble size = 1
    - 3D Ensemble Var and Hybrid 3D-Var run with full ensemble size

This tutorial does not not distinguish offline and online:

➜ analysis step essentially the same for both

1c) init_PDAF

init_pdaf.F90

Routine sets parameters for PDAF and calls `PDAF_init` to initialize the data assimilation:

Particular settings for 3D-Var methods (showing the default values):

```
1. filtertype = 200        ! all 3D-Var methods
2. subtype = 0             ! Select 3D-Var method:
                           !   (0) parameterized 3D-Var
                           !   (1) 3D Ensemble Var using LESTKF for ensemble update
                           !   (4) 3D Ensemble Var using ESTKF for ensemble update
                           !   (6) hybrid 3D-Var using LESTKF for ensemble update
                           !   (7) hybrid 3D-Var using ESTKF for ensemble update
3. type_opt = 1            ! Type of minimizer for 3DVar
                           !   (1) LBFGS, (2) CG+, (3) plain CG
                           !   (12) CG+ parallel, (13) plain CG parallel
4. dim_cvec = dim_ens      ! dimension of control vector (parameterized part)
5. mcols_cvec_ens = 1      ! Multiplication factor for ensemble control vector
                           ! (to simulate localization)
6. beta_3dvar = 0.5        ! Hybrid weight for hybrid 3D-Var
```

init_ens_pdaf.F90 and init_3dvar_pdaf.F90

Routines are called through PDAF_init

`init_ens_pdaf.F90`

- Contains the usual ensemble initialization
- Used with 3D Ensemble Var and hybrid 3D-Var

`init_3dvar_pdaf.F90`

- Initialization for 3D-Var with parameterized covariance matrix
- 3D-Var uses dim_ens = 1!
    - Initialize single state vector
- Need to initialize covariance matrix information
    - $B^{1/2}$ is simulated by scaled ensemble perturbations at initial time

assimilate_pdaf.F90

5 different calls to PDAF_assimilate_*

- different 3D-Var methods
- select according to `subtype`
  (this selection is coded by the user, not done internally to DPAF because different variants of 3D-Var need different call-back routines)

PDAF Parallel Data Assimilation Framework

2a) 3D-Var

with parameterized covariance matrix

Files particular or modified for 3D-Var

Template (templates/3dvar) contains required files for 3D-Var

- just need to be filled with functionality
- Use in combination with templates for ensemble filters

`init_pdaf.F90` `init_3dvar_pdaf.F90`	initialization
`prepoststep_3dvar_pdaf.F90`	post step
`callback_obs_pdafomi.F90` `obs_*_pdafomi.F90`	analysis step
`cvt_pdaf.F90` `cvt_ens_pdaf.F90`	Control vector transformation

PDAF Parallel Data Assimilation Framework

3D-Var initialization and pre/poststep

Parameterized 3D-Var

- run with `dim_ens=1`
- Set dimension of control vector by `dim_cvec` (in init_pdaf)

Initialization:

`init_3dvar_pdaf.F90`

- fill initial state estimate - as `ens_p(:,1)`
- initialize matrix $\mathbf{B}^{1/2}$ from initial ensemble - array `Vmat`

Prepoststep:

`prepostep_3dvar_pdaf.F90`

- Adaption of prepoststep_ens_pdaf for dim_ens=1

callback_obs_pdafomi.F90

Observation handling with PDAF-OMI – calling observation modules

Need 2 additional routines (compared to ensemble filters):

`obs_op_lin_pdafomi`

`obs_op_adj_pdafomi`

`obs_op_lin_pdafomi`

- linearized observation operator (forward: $\mathbf{y} = \mathbf{H}\,\mathbf{x}$)
- same calling interface as `obs_op_pdafomi`
- in tutorial examples identical to `obs_op_pdafomi` since full operator is linear

`obs_op_adj_pdafomi`

- adjoint operation: $\mathbf{x} = \mathbf{H}^T\,\mathbf{y}$
- calling interface swaps **x** and **y** (`state_p` and `ostate`)

PDAF Parallel Data Assimilation Framework

obs_*_pdafomi.F90

PDAF-OMI observation modules

Need 2 additional routines (compared to ensemble filters):

`obs_op_lin_OBSTYPE` with `OBSTYPE=A`, `B`, or `C`

`obs_op_adj_OBSTYPE`

`obs_op_lin_OBSTYPE`

- Not present in example since full operator (obs_op_OBSTYPE) is linear
- `obs_op_lin_pdafomi` in `callback_obs_pdafomi` directly calls `obs_op_OBSTYPE`

`obs_op_adj_OBSTYPE`

- Additional routine
- Just call adjoint observation operator provided by PDAF-OMI:

  `PDAFomi_obs_op_adj_gridpoint` for OBSTYPE=A or B

  `PDAFomi_obs_op_adj_interp_lin` for OBSTYPE=C

cvt_pdaf.F90

Control vector transformation: x = L v

input: control vector **v** – in example codes: vector `v_p`

output: state vector **x** – in example codes: vector `Vv_p`

Required operation

- Multiply control vector with square root **L** of covariance matrix
- **L** was initialized in `init_3dvar_pdaf` (variable name `Vmat_p`)

  → use direct multiplication Vv_p = Vmap_p v_p

Note:

Real cases usually more complicated:

→ **L** could involve balance operations, distributions of increments over different variables, use of decorrelation lengths, use of EOFs, etc.

→ Would be implemented in form of covariance operators

cvt_adj_pdaf.F90

Adjoint control vector transformation: $\mathbf{v} = \mathbf{L}^T \mathbf{x}$

input: state vector **x** – in example codes: vector `Vv_p`

output: control vector **v** – in example codes: vector `v_p`

Required operation

- Multiply state vector with adjoint of square root **L** of covariance matrix (usually $\mathbf{L}^T$)
- **L** was initialized in init_3dvar_pdaf (variable name Vmat_p)

  ➜ Use direct multiplication $v\_p = Vmat\_p^T \, Vv\_p$

PDAF Parallel Data Assimilation Framework

Running 3D-Var – options for call to PDAF_init

Choose the type of 3D-Var (variable `subtype`)

- 0 parameterized 3D-Var
- 1 ensemble 3D-Var using local LESTKF for ensemble transformation
- 4 ensemble 3D-Var using global ESTKF for ensemble transformation
- 6 hybrid 3D-Var using local LESTKF for ensemble transformation
- 7 hybrid 3D-Var using global ESTKF for ensemble transformation

Choose type of optimizer (variable `type_opt`)

- 1 LBFGS
- 2 CG+
- 3 plain CG
- 12 CG+ parallelized (decomposed control vector)
- 13 plain CG parallelized (decomposed control vector)

Set length of control vector (number of columns in covariance operator)

- `dim_cvec` for 3D-Var cases 0, 6, or 7
- `dim_ens` for 3D-Var cases 1, 6, or 7

Set hybrid weight of hybrid 3D-Var

- `beta_3dvar` between 1=ensemble and 0=parameterized 3D-Var

Running 3D-Var

In **offline_2D_serial:**

Run 3D-Var with CG+ solver, size of control vector =4:

./PDAF_offline -dim_ens 1 -filtertype 200 **-type_3dvar 0** -type_opt 2 -dim_cvec 4

Note: The result is almost identical to running the ESTKF in /tutorial/offline_2D_serial/ with './PDAF_offline -dim_ens 4 - filtertype 6' (same problem is solved with different methods)

In **online_2D_serialmodel:**

Run 3D-Var with LBFGS solver, size of control vector =4:

./model_pdaf -dim_ens 1 -filtertype 200 **-subtype** 0 -type_opt **1** -dim_cvec 4

The result differs from ESTKF in /tutorial/online_2D_serialmodel/
because of ensemble integration (and LBFGS solver)

(Depending on your MPI library you might need '`mpirun -np 1`' to run these cases)

2b) 3D Ensemble Var

use ensemble covariance matrix

Files particular or modified for 3D Ensemble Var

`init_pdaf.F90`	initialization
`callback_obs_pdafomi.F90` `obs_*_pdafomi.F90`	analysis step
`cvt_ens_pdaf.F90` `cvt_adj_ens_pdaf.F90`	control vector transformation

PDAF Parallel Data Assimilation Framework

3D Ensemble Var initialization in init_pdaf

Ensemble 3D-Var

run with actual ensemble of size `dim_ens>1`

- Call to PDAF_init needs specification of size of control vector (`dim_cvec_ens` or `filter_param_i(5)`)
- Determine `dim_cvec_ens` as
  - `dim_cvec_ens = dim_ens * mcols_cvec_ens`
  - `mcols_cvec_ens` is motivated from localization:
    - Without localization: dim_ens columns of ensemble perturbations
    - With localization: append column sets of each dim_ens columns
    - Each set of dim_ens columns is tapered differently

- In tutorial: No localization applied, but multiple sets of dim_ens columns are supported
- **Note:** dim_cvec_ens can be freely chosen, not necessarily based on mcols_cvec_ens

callback_obs_pdafomi.F90

Identical to 3D-Var

Observation handling with PDAF-OMI – calling observation modules

Need 2 additional routines (compared to ensemble filters):

`obs_op_lin_pdafomi`

`obs_op_adj_pdafomi`

`obs_op_lin_pdafomi`

- linearized observation operator (forward: $\mathbf{y} = \mathbf{H}\,\mathbf{x}$)
- same calling interface as `obs_op_pdafomi`
- in tutorial examples identical to `obs_op_pdafomi` since full operator is linear

`obs_op_adj_pdafomi`

- adjoint operation: $\mathbf{x} = \mathbf{H}^T\,\mathbf{y}$
- calling interface swaps **x** and **y** (`state_p` and `ostate`)

obs_*_pdafomi.F90

Identical to 3D-Var

PDAF-OMI observation modules

Need 2 additional routines (compared to ensemble filters):

`obs_op_lin_OBSTYPE` with `OBSTYPE=A, B`, or `C`

`obs_op_adj_OBSTYPE`

`obs_op_lin_OBSTYPE`

- Not present in example since full operator (obs_op_OBSTYPE) is linear
- `obs_op_lin_pdafomi` in `callback_obs_pdafomi` directly calls `obs_op_OBSTYPE`

`obs_op_adj_OBSTYPE`

- Additional routine
- Just call adjoint observation operator provided by PDAF-OMI:

  `PDAFomi_obs_op_adj_gridpoint` for OBSTYPE=A or B

  `PDAFomi_obs_op_adj_interp_lin` for OBSTYPE=C

PDAF Parallel Data Assimilation Framework

cvt_ens_pdaf.F90

Different from 3D-Var

Control vector transformation with *ensemble* information: x = Z v

input: Control vector `v_p`

output: state vector `Vv_p`

Required operation

- Multiply control vector with square root **Z** of ensemble covariance matrix
- At beginning of iterations: Initialize **Z** for use in all iterations (array `Vmat_ens_p`)
- During iterative optimization:

  → use direct multiplication `Vv_p = Vmat_ens_p v_p`

Note:

Real cases are usually more complicated:

→ **Z** would include localization, e.g. by multiple sets of columns and tapering

→ Variable `mcols_cvec_ens` prepares for this; but no localization implemented (columns are just reproduced without tapering)

cvt_adj_ens pdaf.F90

Adjoint control vector transformation with *ensemble* information: v = $\mathbf{Z}^T$ x

input: state vector `Vv_p`

output: control vector `v_p`

Different from 3D-Var

Required operation

- Multiply state vector with adjoint of square root **Z** of covariance matrix (usually $\mathbf{Z}^T$)
- **Z** was initialized in `cvt_ens_pdaf` (variable name `Vmat_ens_p`)

→ Use direct multiplication `v_p = Vmat_ens_p`^{T} `Vv_p`

PDAF Parallel Data Assimilation Framework

Filter analysis step to transform ensemble perturbations

- 3D-Var part only computed analysis state vector
- Ensemble perturbations are transformed by ensemble filter (ESTKF or LESTKF)

  ➔ all user routines for ESTKF(global) or LESTKF (localized) need to be implemented

  ➔ Recommendation: to first implement and test analysis for ETKF/LESTKF before use in Ensemble or Hybrid 3D-Vars

Running 3D Ensemble Var

In **offline_2D_serial:**

Run ensemble 3D-Var/LESKTF with LBFGS, size of control vector (ensemble) =4:

./PDAF_offline -dim_ens 4 -filtertype 200 -type_3dvar 1 -type_opt 1

Run ensemble 3D-Var/ESTKF with CG+, size of control vector (ensemble) =4:

./PDAF_offline -dim_ens 4 -filtertype 200 -type_3dvar 4 -type_opt 2

In **online_2D_serialmodel:**

Run ensemble 3D-Var/ESKTF with CG+, size of control vector (ensemble) =4;

mpirun -np 4 ./model_pdaf -dim_ens **4** -filtertype 200 -subtype 4 -type_opt 2

State estimate at step 02 is almost identical to running the ESTKF in /tutorial/online_2D_serialmodel/ with 'mpirun –np 4 ./model_pdaf -dim_ens 4 - filtertype 6'
(Note: 3D-Var/LESTKF only uses localization to update ensemble perturbations, not state)

(Depending on your MPI library you might need '`mpirun -np 1`' in offline_2D_serial)

PDAF Parallel Data Assimilation Framework

2c) Hybrid 3D-Var

Combine ensemble and parameterized covariance matrix

Files particular or modified for hybrid 3D-Var

init_pdaf.F90
init_ens_pdaf.F90

initialization

callback_obs_pdafomi.F90
obs_*_pdafomi.F90

analysis step

cvt_pdaf.F90
cvt_ens_pdaf.F90
cvt_adj_pdaf.F90
cvt_adj_ens_pdaf.F90

control vector transformation

PDAF Parallel Data Assimilation Framework

Hybrid 3D-Var initialization in init_pdaf

Hybrid 3D-Var

- run with actual ensemble of size `dim_ens>1`
- represent square root by combination of ensemble and parameterized covariances: $\mathbf{B}^{1/2} = [\mathbf{L}\ \mathbf{Z}]$
  - Call to `PDAF_init` needs specification of size of control vector (dim_cvec & `dim_cvec_ens` or `filter_param_i(4)` & `filter_param_i(5)`)
  - Determine `dim_cvec_ens` as in 3D Ensemble Var to allow for localization

- **Notes:**
  - Hybrid 3D-Var implementation of PDAF strictly separates parameterized and ensemble covariance parts
  - `cvt_pdaf/cvt_adj_pdaf` and `cvt_ens_pdaf/cvt_adj_ens_pdaf` are all used
  - Separation might be too restrictive if aim is to mix ensemble information into parameterized part

    ➜ Flexible combinations possible with 3D Ensemble Var using `cvt_ens_pdaf/cvt_adj_ens_pdaf` for combined operations

PDAF Parallel Data Assimilation Framework

Hybrid 3D-Var initialization and pre/poststep

Initialization:

`init_ens_pdaf.F90`

- Usual ensemble initialization for dim_ens
- In addition:
  - Initialize square root of parameterized background covariance matrix (`Vmat_p`)
  - Same initialization as in `init_3dvar_pdaf.F90`

Prepoststep:

`prepostep_ens_pdaf.F90`

- Identical to routine for ensemble filters!

PDAF Parallel Data Assimilation Framework

callback_obs_pdafomi.F90

Identical to 3D-Var

Observation handling with PDAF-OMI – calling observation modules

Need 2 additional routines (compared to ensemble filters):

`obs_op_lin_pdafomi`

`obs_op_adj_pdafomi`

`obs_op_lin_pdafomi`

- linearized observation operator (forward: $\mathbf{y} = \mathbf{H}\,\mathbf{x}$)
- same calling interface as `obs_op_pdafomi`
- in tutorial examples identical to `obs_op_pdafomi` since full operator is linear

`obs_op_adj_pdafomi`

- adjoint operation: $\mathbf{x} = \mathbf{H}^T\,\mathbf{y}$
- calling interface swaps **x** and **y** (`state_p` and `ostate`)

PDAF Parallel Data Assimilation Framework

obs_*_pdafomi.F90

Identical to 3D-Var

PDAF-OMI observation modules

Need 2 additional routines (compared to ensemble filters):

`obs_op_lin_OBSTYPE` with `OBSTYPE=A, B`, or `C`

`obs_op_adj_OBSTYPE`

`obs_op_lin_OBSTYPE`

- Not present in example since full operator (obs_op_OBSTYPE) is linear
- `obs_op_lin_pdafomi` in `callback_obs_pdafomi` directly calls `obs_op_OBSTYPE`

`obs_op_adj_OBSTYPE`

- Additional routine
- Just call adjoint observation operator provided by PDAF-OMI:

  `PDAFomi_obs_op_adj_gridpoint` for OBSTYPE=A or B

  `PDAFomi_obs_op_adj_interp_lin` for OBSTYPE=C

cvt_pdaf.F90

Control vector transformation: x = L v

input: control vector **v** – in example codes: vector `v_p`

output: state vector **x** – in example codes: vector `Vv_p`

Identical to 3D-Var

Required operation

- Multiply control vector with square root **L** of covariance matrix
- **L** was initialized in `init_3dvar_pdaf` (variable name `Vmat_p`)

  → use direct multiplication Vv_p = Vmap_p v_p

Note:

Real cases usually more complicated:

→ **L** could involve balance operations, distributions of increments over different variables, use of decorrelation lengths, use of EOFs, etc.

→ Would be implemented in form of covariance operators

cvt_adj_pdaf.F90

Identical to 3D-Var

Adjoint control vector transformation: v = $\mathbf{L}^T$ x

input: state vector **x** – in example codes: vector `Vv_p`

output: control vector **v** – in example codes: vector `v_p`

Required operation

- Multiply state vector with adjoint of square root **L** of covariance matrix (usually $\mathbf{L}^T$)
- **L** was initialized in init_3dvar_pdaf (variable name Vmat_p)

➜ Use direct multiplication v_p = Vmat_p^T Vv_p

PDAF Parallel Data Assimilation Framework

cvt_ens_pdaf.F90

Identical to 3D Ens Var

Control vector transformation with ensemble information: x = Z v

input: Control vector `v_p`

output: state vector `Vv_p`

Required operation

- Multiply control vector with square root **Z** of ensemble covariance matrix
- At beginning of iterations: Initialize **Z** for use in all iterations (array `Vmat_ens_p`)
- During iterative optimization:

  ➜ use direct multiplication `Vv_p = Vmat_ens_p v_p`

Note:

Real cases are usually more complicated:

➜ **Z** would include localization, e.g. by multiple sets of columns and tapering

➜ Variable `mcols_cvec_ens` prepares for this; but no localization implemented (columns are just reproduced without tapering)

cvt_adj_ens pdaf.F90

Identical to 3D Ens Var

Adjoint control vector transformation with ensemble information: v = $\mathbf{Z}^T$ x

input: state vector `Vv_p`

output: control vector `v_p`

Required operation

- Multiply state vector with adjoint of square root **Z** of covariance matrix (usually $\mathbf{Z}^T$)
- **Z** was initialized in `cvt_ens_pdaf` (variable name `Vmat_ens_p`)

➜ Use direct multiplication `v_p = Vmat_ens_p`^{T} `Vv_p`

PDAF Parallel Data Assimilation Framework

Running Hybrid 3D-Var

In **offline_2D_serial:**

Run hybrid 3D-Var/LESTKF with CG+,
size of control vector 8: ensemble part =4 and parameterized part =4:

./PDAF_offline -filtertype 200 -type_3dvar 6 -type_opt 2 **-dim_ens 4 -dim_cvec 4**

Need to specify both dim_ens and dim_cvec!

In **online_2D_serialmodel:**

Run ensemble 3D-Var/LESKTF with LBFGS,

size of control vector 8: ensemble part =4 and parameterized part =4:

mpirun -np 4 ./model_pdaf -filtertype 200 -subtype 6 -type_opt 1 \
-dim_ens 4 -dim_cvec 4 –beta_3dvar 0.7

beta_3dvar: determines hybrid weight (here 70% for ensemble/30% for parameterized)

(Depending on your MPI library you might need '`mpirun -np 1`' in offline_2D_serial)

PDAF Parallel Data Assimilation Framework

3) Parallelization of 3D-Var analysis

online_2D_parallelmodel

Parallelization: decompose state vector and covariances

Handling of domain decomposed state vectors

- State vector follows domain decomposition
- Ensemble decomposed according to domain decomposition
  - Thus also **L** is decomposed (each process holds `dim_p` rows)

  ➜ Adjoint operation $\mathbf{v} = \mathbf{L}^T \mathbf{x}$ results in incomplete sums

  ➜ Need global sum over all processors
    - Implementation in cvt_adj_pdaf/cvt_adj_ens_pdaf:
      1. Apply multiplication for process-local part getting partial sum
      2. Apply MPI_Allreduce to obtain vector **v** holding global sums

PDAF Parallel Data Assimilation Framework

Parallelization: decomposed control vectors

Handling of decomposed control vector

- Use `type_opt=12` or `type_opt=13`

  ➜ parallelized solvers using decomposed control vectors

  - Now **v** is decomposed
  - Forward control vector transformation **x** = **L** **v** results in incomplete sums
    - Implemention in cvt_pdaf/cvt_ens_pdaf:
      1. first gather the global vector **v**
      2. multiply for process-local rows of **x**
  - Result of adjoint operation **v** = $\mathbf{L}^T$ **x**: only some rows of **v** required on a process
    - Implementation in cvt_adj_pdaf/cvt_adj_ens_pdaf:
      1. apply MPI_Allreduce (for incomplete sum if **x** and **L** are decomposed)
      2. then select process-specific part of **v**

PDAF Parallel Data Assimilation Framework

The End!

Tutorial described example implementations

- Online mode of PDAF parallelized over ensemble members
- Simple 2D model without parallelization and with OpenMP parallelization
- Implementation supports different 3D-Var methods
  - Parameterized, ensemble, hybrid
  - Ensemble transformation with global and localized filters
- Extension to more realistic cases possible with limited coding
- Applicable also for large-scale problems

For full documentation of PDAF
and the user-implemented routines
see http://pdaf.awi.de

PDAF Parallel Data Assimilation Framework