PDAF Tutorial

Implementation of the analysis step

in offline mode

http://pdaf.awi.de

PDAF Parallel Data Assimilation Framework

V1.5 – 2018-04-05

Implementation Tutorial for PDAF offline

We demonstrate the implementation

of an offline analysis step with PDAF

using the template routines provided by PDAF

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

PDAF Parallel Data Assimilation Framework

Implementation Tutorial for PDAF offline

This is just an example!

For the complete documentation of PDAF's interface

see the documentation

at http://pdaf.awi.de

PDAF Parallel Data Assimilation Framework

Overview

Focus on Error Subspace Transform Kalman Filter
(ESTKF, Nerger et al., Mon. Wea. Rev. 2012)

4 Parts

1. Without parallelization
   a) Global filter
   b) Localized filter
   (and OpenMP-parallelization)
2. With MPI-parallelization
   a) Global filter
   b) Localized filter

We recommend to first implement the global filter. The localized filter re-uses routines of the global filter.

We assume that 1a is implemented before 1b and 1a is implemented before 2a (1b before 2b).

Contents

0a) Files for the Tutorial

Tutorial implementation

Files are in the PDAF package

Directories:

`/tutorial/offline_2D_serial`	(without parallelization)
`/tutorial/offline_2D_openmp`	(with OpenMP-parallelization)
`/tutorial/offline_2D_parallel`	(with MPI parallelization)

- 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’.

Templates for offline mode

Directory: `/templates/offline`

- Contains all required files
- Contains also
  command line parser, memory counting, timers
  (convenient but not required)

To generate your own implementation:

1. Copy directory to a new name
2. Complete routines for your model
3. Set base directory (`BASEDIR`) in Makefile
4. Set `$PDAF_ARCH`
5. Compile

PDAF library

Directory: `/src`

- The 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` (tcsh/csh)
  `export PDAF_ARCH=linux_gfortran` (bash)

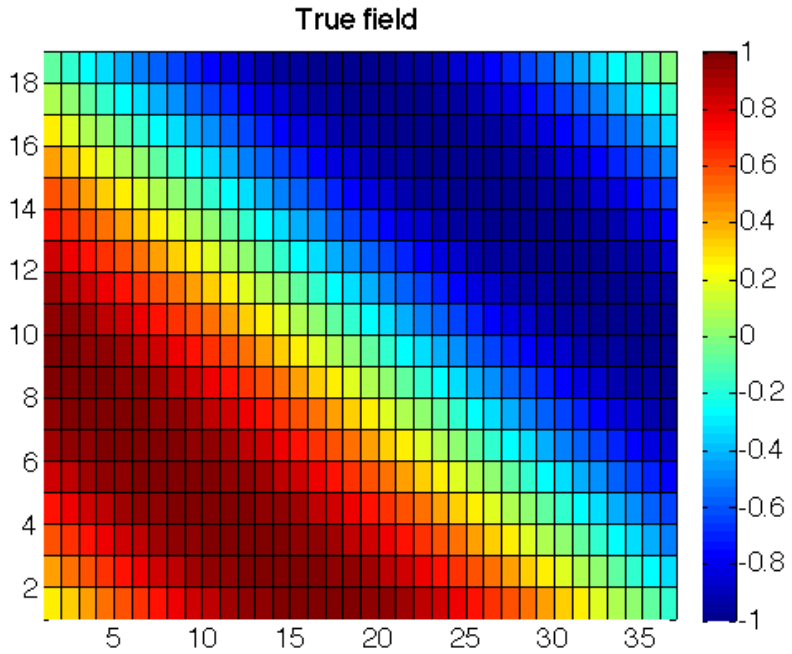
0b) The Model

Simple assimilation problem

- 2-dimensional model domain
- One single field (like temperature)
- Direct measurements of the field
- Data gaps (i.e. data at selected grid points)
- Same error estimate for all observations
- Observation errors are not correlated
  (diagonal observation error covariance matrix)
- Perform a single analysis step using input files for the ensemble and observations (offline mode: No time stepping in the assimilation program)

2D „Model“

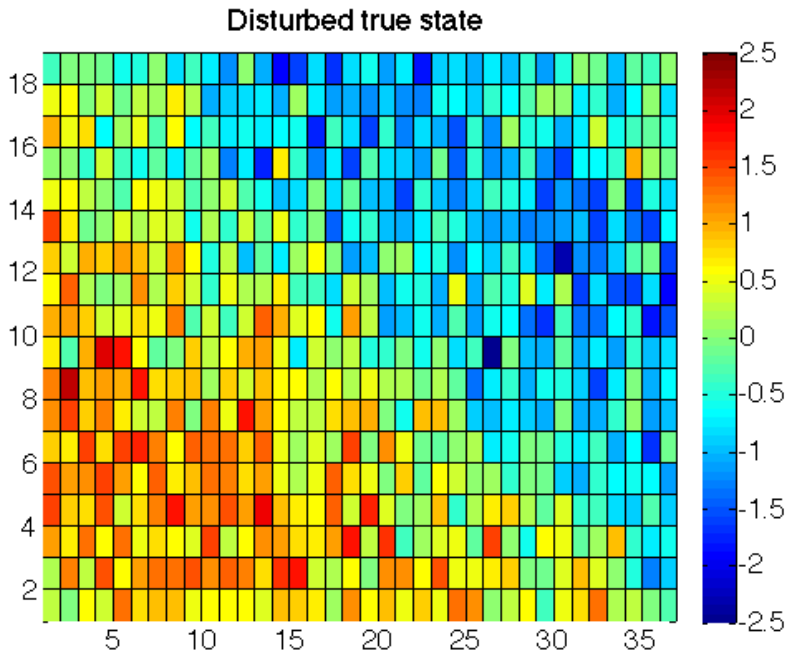
- Simple 2-dimensional grid domain
- 36 x 18 grid points (longitude x latitude)
- True state: sine wave in diagonal direction
- No dynamics for offline mode
- Stored in text file (18 rows) – `true.txt`

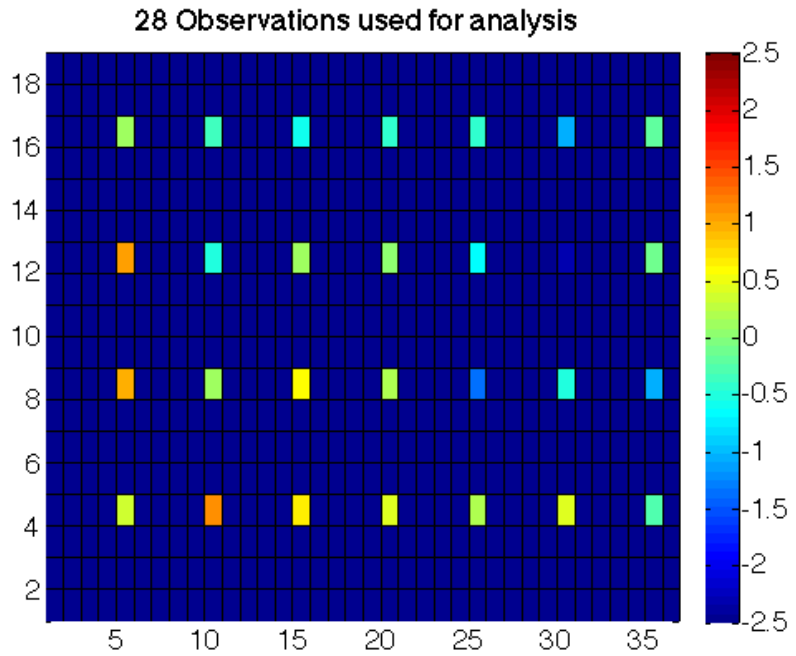


PDAF Parallel Data Assimilation Framework

Observations

- Add random error to true state (standard deviation 0.5)
- Select a set of observations at 28 grid points
- File storage:
  text file, full 2D field, -999 marks 'no data' – `obs.txt`

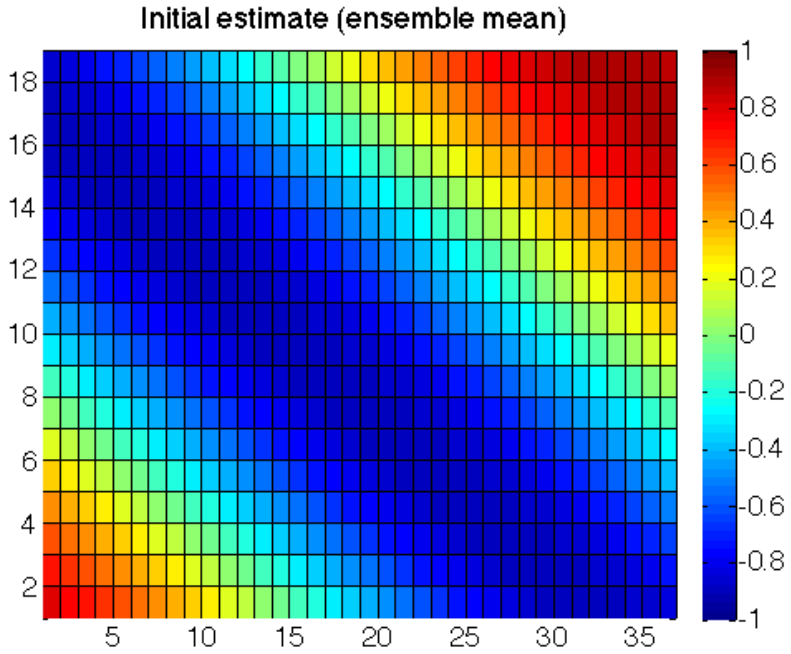


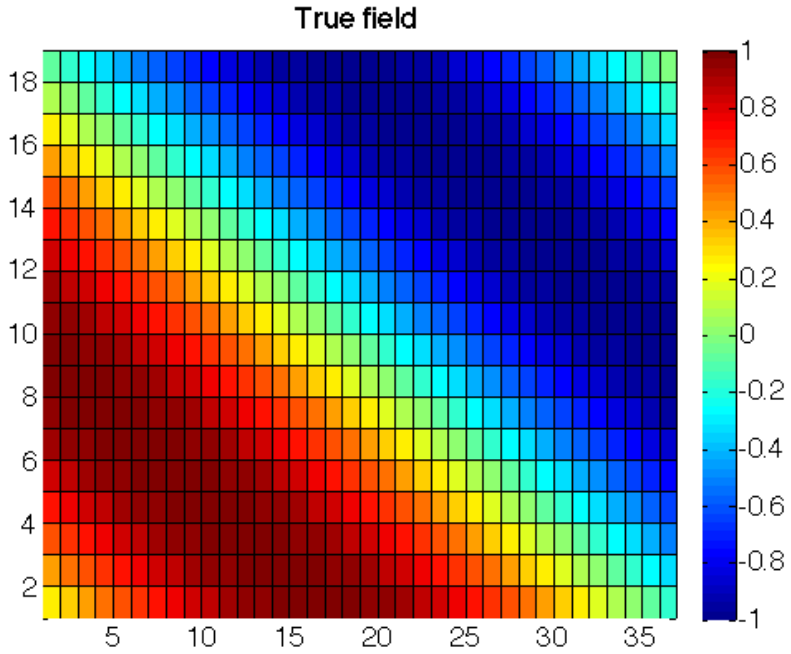


PDAF Parallel Data Assimilation Framework

Ensemble

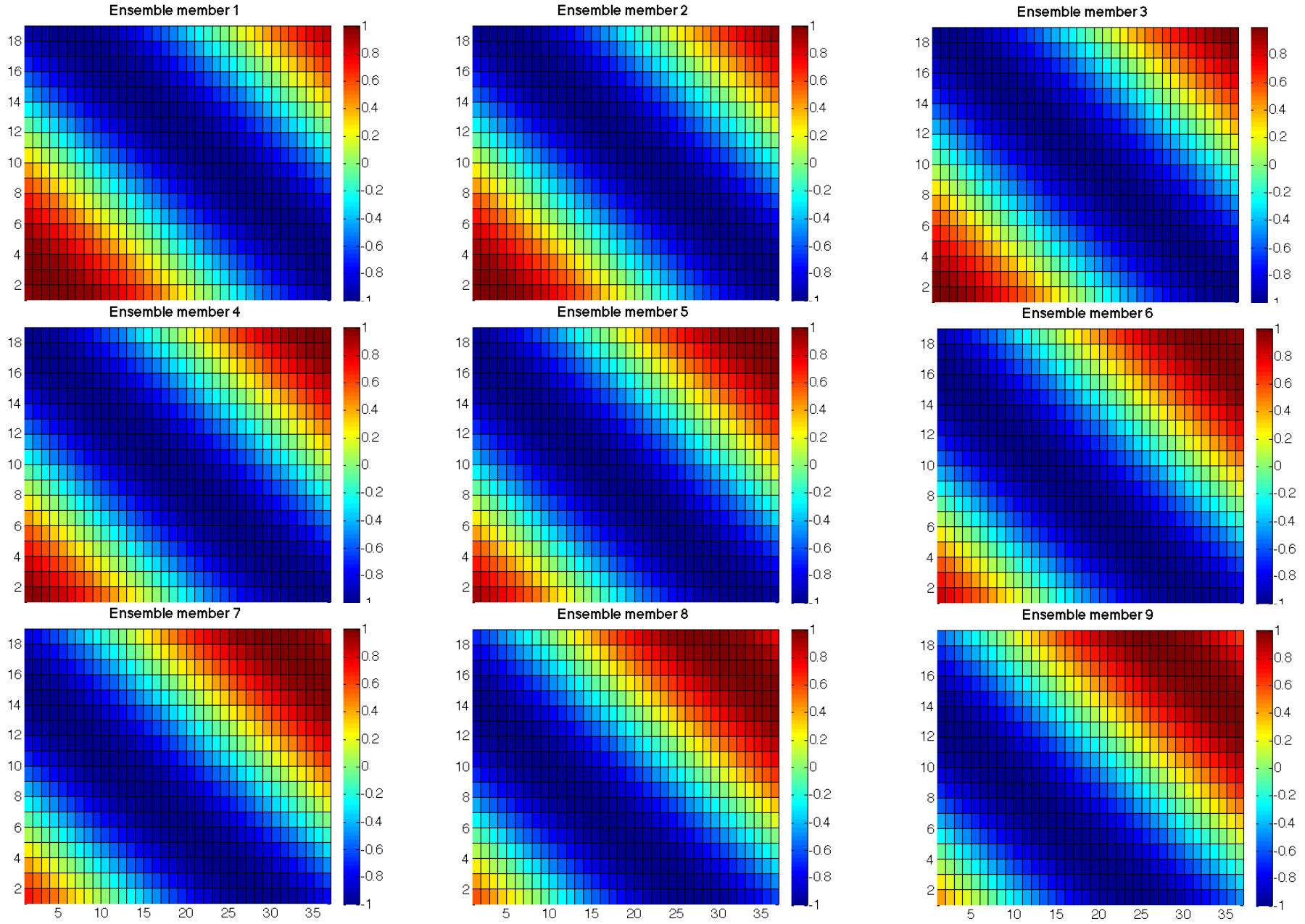
- Ensemble size 9
- Sine waves shifted along diagonal (truth not included)
- One text file per ensemble member – `ens*.txt`





PDAF Parallel Data Assimilation Framework

Ensemble states



0c) state vector and observation vector

State vector – some terminology used later

- PDAF performs computations on state vectors
- **State vector**
  - Stores model fields in a single vector
  - Tutorial shows this for one 2-dimensional field
  - Multiple fields are just concatenated into the vector
  - All fields that should be modified by the assimilation have to be in the state vector
- **State dimension**
  - Is the length of the state vector
    (the sum of the sizes of the model fields in the vector)
- **Ensemble array**
  - Rank-2 array which stores state vectors in its columns

Observation vector

- **Observation vector**
  - Stores all observations in a single vector
  - Tutorial shows this for one 2-dimensional field
  - Multiple observed fields are just concatenated into the vector
- **Observation dimension**
  - Is the length of the observation vector
    (sum of the observations over all observed fields in the vector)
- **Observation operator**
  - Operation that computes the observed part of a state vector
  - Tutorial only selects observed grid points
  - The operation can involve interpolation or integration
    depending on type of observation

0d) PDAF offline mode

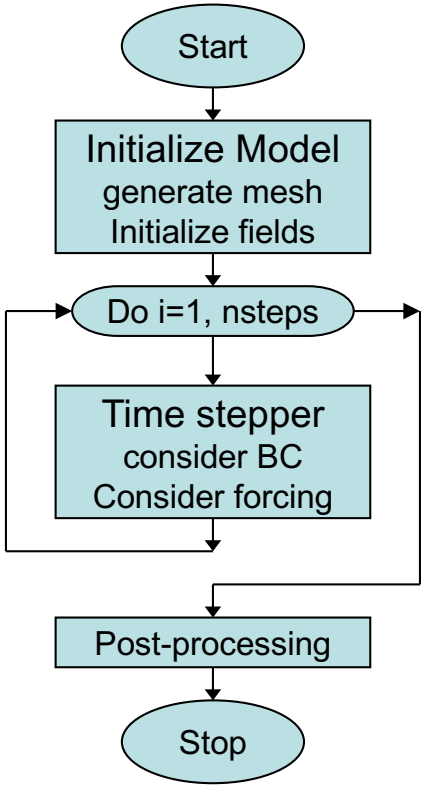
Offline mode

- Two separate programs
  - `"Model"` – performs ensemble integrations
  - `"PDAF_offline"` – perform analysis step

- Couple both programs through files
  1. `"PDAF_offline"` reads ensemble forecast files
  2. Performs analysis step
  3. Writes analysis ensemble files (restart files for "Model")
  4. `"Model"` reads restart files and performs ensemble integration

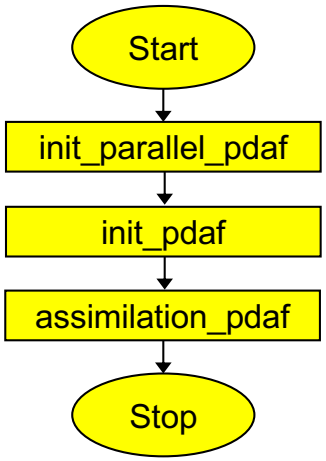
Programs in offline mode

Model



- Run for each ensemble member
- Write restart files

Assimilation program



- Read restart files (ensemble)
- Compute analysis step
- Write new restart files

PDAF Parallel Data Assimilation Framework

PDAF_offline: General program structure

```
program main_offline

   init_parallel_pdaf
                        initialize communicators
                        (not relevant without parallelization)

   initialize
                        initialize model information

   init_pdaf
                        initialize parameters for PDAF
                        and read ensemble

   assimilation_pdaf
                        perform analysis
                        (by call to PDAF_put_state_X)

end program
```

1a) Global filter without parallelization

Running the tutorial program

- cd to `/tutorial/offline_2D_serial`
- Set environment variable `PDAF_ARCH` or set it in Makefile (e.g. `linux_gfortran`)
- Compile by running `'make'` (next slide will discuss possible compile issues)
- Run the program with `./PDAF_offline`

- Inputs are read in from `/tutorial/inputs_offline`
- Outputs are written in `/tutorial/offline_2D_serial`
- Plot result, e.g with 'octave':

```
load state_ana.txt

pcolor(state_ana)
```

Requirements for compiling PDAF

PDAF requires libraries for BLAS and LAPACK

- Libraries to be linked are specified in the include file for make in `/make.arch` (file according to `PDAF_ARCH`)
- For `$PDAF_ARCH=linux_gfortran` the specification is

```
LINK_LIBS =-L/usr/lib -llapack  -lblas   -lm
```

- If the libraries are at another non-default location, one has to change the directory name (`/usr/lib`)
- Some systems or compilers have special libraries (e.g. MKL for ifort compiler, or ESSL on IBM/AIX)

PDAF needs to be compiled for double precision

- Needs to be set at compiler time in the include file for make:
- For gfortran: `OPT = -O3 -fdefault-real-8`

Files in the tutorial implementation

`/tutorial/inputs_offline`

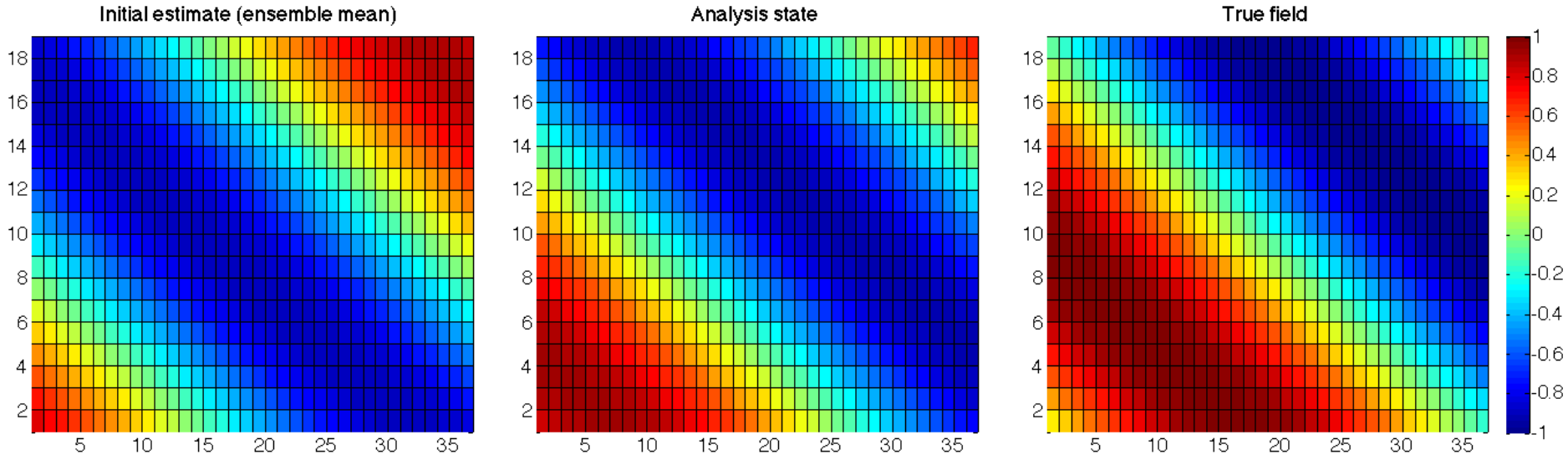
- `true.txt` — true state
- `state_ini.txt` — initial estimate (ensemble mean)
- `obs.txt` — observations
- `ens_X.txt` (X=1,..., 9) — ensemble members

`/tutorial/offline_2D_serial` (after running PDAF_offline)

- `state_ana.txt` — analysis state estimate
- `ens_X_ana.txt` (X=1,...,9) — analysis ensemble members

Result of the global assimilation

- The analysis state is closer to the true field than the initial estimate
- Truth and analysis are not identical
  (the ensemble does not allow it)



PDAF Parallel Data Assimilation Framework

Files to be changed

Template contains all required files

- just need to be filled with functionality

`mod_assimilation.F90`	Fortran module
`initialize.F90`	initialization
`init_pdaf_offline.F90`	
`init_ens_offline.F90`	
`init_dim_obs_pdaf.F90`	analysis step
`obs_op_pdaf.F90`	
`init_obs_pdaf.F90`	
`prodrinva_pdaf.F90`	
`prepoststep_ens_offline.F90`	post step

mod_assimilation.F90

Fortran module

- Declares the parameters used to configure PDAF
- Add model-specific variables here
  (see next slides)
- Will be included (with '`use`') in the user-written routines

initialize.F90

Routine initializes the model information

1. Define 2D mesh in mod_assimilation.F90

```
integer :: nx, ny
```

2. In initialize.F90 include `nx, ny,` and `dim_state_p` with `use mod_assimilation`

3. Define mesh size in initialize.F90

```
nx = 36
ny = 18
```

4. Define state dimension in initialize.F90

```
dim_state_p = nx * ny
```

Note: Some variables end with `_p`.
It means that the variable is specific for a process.
(Not relevant until we do parallelization)

init_pdaf_offline.F90

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

Template contains list of available parameters
(declared in and used from `mod_assimilation`)

For the example set :

1. `dim_ens = 9`
2. `rms_obs = sqrt(0.5)`
3. `filtertype = 6` (for ESTKF)

In call to PDAF_init, the name of the ensemble initialization routine is specified:

`init_ens_offline`

init_ens_offline.F90

A *call-back* routine called by PDAF_init:

- Implemented by the user
- Its name is specified in the call to PDAF_init
- It is called by PDAF through a defined interface:

```
SUBROUTINE init_ens_offline(filtertype, dim_p,
       dim_ens, state_p, Uinv, ens_p, flag)
```

Declarations in header of the routine shows “intent” (input, output):

```
REAL, INTENT(out)   :: ens_p(dim_p, dim_ens)
```

Note:
All call-back routines have a defined interface and show the intent of the variables. Their header comment explains what is to be done in the routine.

init_ens_offline.F90 (2)

Initialize ensemble matrix `ens_p`

1. Include `nx, ny` with `use mod_assimilation`
2. Declare and allocate `real :: field(ny, nx)`
3. Loop over ensemble files `(i=1,dim_ens)`

   for each file:

   - read ensemble state into `field`
   - store contents of `field` in column `i` of `ens_p`

Note:
Columns of `ens_p` are state vectors.
Store following storage of field in memory (column-wise in Fortran)

The analysis step

At this point the initialization of PDAF is complete:

- Forecast ensemble is initialized
- Filter algorithm and its parameters are chosen

Next:

- Implement user-routines for analysis step
- All are call-back routines:
  - User-written, but called by PDAF

Note:
Some variables end with _p.
It means that the variable is specific for a process.
(Not relevant until we do parallelization)

init_dim_obs_pdaf.F90

Routine to

- read observation file
- count number of available observations
  (direct output to PDAF: **`dim_obs_p`**)

Optional, also

- initialize array holding available observations
- initialize index array telling index of observation point
  in full state vector

The most complicated routine in the example!
(but less than 100 lines)

init_dim_obs_pdaf.F90 (2)

Preparations and reading of observation file:

1. Include `nx, ny` with `use mod_assimilation`
2. declare and allocate real array `obs_field(ny, nx)`
3. read observation file:

```
OPEN (12, file='inputs_offline/obs.txt', &
    status='old')
DO i = 1, ny
   READ (12, *) obs_field(i, :)
END DO
CLOSE (12)
```

init_dim_obs_pdaf.F90 (3)

Count available observations (**dim_obs_p**):

1. Declare `integer :: cnt, cnt0`
2. Now count

```
cnt = 0
DO j = 1, nx
  DO i= 1, ny
     IF (obs_field(i,j) > -999.0) cnt = cnt + 1
  END DO
END DO
dim_obs_p = cnt
```

init_dim_obs_pdaf.F90 (4)

Initialize observation vector (`obs`)
and index array (`obs_index`):

1. In `mod_assimilation it is` declared
   `real, allocatable :: obs_p(:), obs_index_p(:)`
   Include these variable with `use mod_assimilation`

2. Allocate
   `obs_p(dim_obs_p), obs_index_p(dim_obs_p)`
   (If already allocated, deallocate first)

3. Now initialize …

Note:
The arrays only contain information about valid observations;
one could store observations already in files in this way.

init_dim_obs_pdaf.F90 (5)

3. Now initialize

```
cnt0 = 0                    ! Count grid points
cnt = 0                     ! Count observations
DO j = 1, nx
  DO i= 1, ny
    cnt0 = cnt0 + 1
    IF (obs_field(i,j) > -999.0) THEN
      cnt = cnt + 1
      obs_index_p(cnt) = cnt0        ! Index
      obs_p(cnt) = obs_field(i, j) ! observations
    END IF
  END DO
END DO
```

obs_op_pdaf.F90

Implementation of observation operator
acting one some state vector

Input: state vector `state_p`

Output: observed state vector `m_state_p`

1. Include `obs_index_p` by `use mod_assimilation`
2. Select observed grid points from state vector:

```
DO i = 1, dim_obs_p
  m_state_p(i) = state_p(obs_index_p(i))
END DO
```

Note:
`dim_obs_p` is an input argument of the routine

init_obs_pdaf.F90

Fill PDAF's observation vector

Output: vector of observations `observation_p`

1. Include `obs` by `use mod_assimilation`
2. Initialize `observation_p`:

```
observation_p = obs_p
```

Note:
This is trivial, because of the preparations in `init_dim_obs_pdaf`!

(However, the operations needed to be separate, because PDAF allocates `observations_p` after the call to `init_dim_obs_pdaf`)

prodrinva_pdaf.F90

Compute the product of the inverse observation error covariance matrix with some other matrix

- Input: Matrix `A_p(dim_obs_p, rank)`
- Output: Product matrix **`C_p`**`(dim_obs_p, rank)`
  (`rank` is typically `dim_ens-1`)

1. Declare and initialize inverse observation error variance

```
ivariance_obs = 1.0 / rms_obs**2
```

2. Compute product:

```
DO j = 1, rank
  DO i = 1, dim_obs_p
     C_p(i, j) = ivariance_obs * A_p(i, j)
  END DO
END DO
```

prepoststep_ens_offline.F90

Post-step routine for the offline mode:

Already there in the template:

1. Compute ensemble mean state `state_p`
2. Compute estimated variance vector `variance`
3. Compute estimated root mean square error `rmserror_est`

Required extension:

4. Write analysis ensemble into files used for model restart
   (Analogous to reading in `init_ens_pdaf_offline`)

Possible (useful) extension:

5. Write analysis state (ensemble mean, `state_ana.txt`)

Done!

The analysis step in offline mode is fully implemented now

The implementation allows you now to use the global filters ESTKF, ETKF, and SEIK

Not usable are EnKF and SEEK (The EnKF needs some other user files und SEEK a different ensemble initialization)

A complete analysis step

We now have a fully functional analysis step
- if no localization is required!

Possible extensions for a real application:

Adapt routines for

- Multiple model fields
  - → Store full fields consecutively in state vector
- Third dimension
  - → Extend state vector
- Different observation types
  - → Store different types consecutively in observation vector
- Other file type (e.g. binary or NetCDF)
  - → Adapt reading/writing routines