SANGOMA: Stochastic Assimilation for the Next Generation Ocean Model Applications EU FP7 SPACE-2011-1 project 283580

Deliverable 2.3:
Software V0 report
Due date: 01/11/2012
Delivery date: 01/11/2012
Delivery type: Report, public

Lars Nerger
Alfred-Wegener-Institute, GERMANY

Arnold Heemink Nils van Velzen
Martin Verlaan M. Umer Altaf
Delft University of Technology, NETHERLANDS

Jean-Marie Beckers Alexander Barth
University of Liège, BELGIUM

Peter Jan Van Leeuwen
University of Reading, UK

Pierre Brasseur Jean-Michel Brankart
CNRS-LEGI, FRANCE

Pierre de Mey
CNRS-LEGOS, FRANCE

Laurent Bertino
NERSC, NORWAY

Contents

Chapter 1

Introduction

Please note: This document only describes the code of release V0 of the SANGOMA tools, which was replaced by the intermediate release V1 and the final code release V2 of the SANGOMA project. Please see the Deliverable 2.5 for the documentation of the final V2 release.

This document provides an overview of the initial release of the SANGOMA tools (V0). Within workpackage 2 of SANGOMA, a collection of tools of general interest for data assimilation applications is prepared. These tools provide additional functionality outside of the core of the analysis steps.

The tools can be categorized as follows:

- **Diagnostic tools**: These tools provide functionality to perform diagnostic operation for data assimilation applications. For example, the efficiency of assimilation techniques can be assessed and statistics significance tests can be performed.
- **Perturbation tools**: These tools allow the user to generate perturbations with prescribed properties in order to generate ensembles of model states or to perform perturbed ensemble integrations.
- **Transformation tools**: Assimilation algorithms that base on the Kalman filter assume Gaussian distributions for optimality. These tools provide functionality to perform preliminary transformations of variables or ensembles to improve the performance with non-Gaussian distributions.
- **Utilities**: Tools of this category provide additional functionality. Examples are tools for efficient manipulations of sparse matrices or the treatment of particular observation types.

The release V0 of the SANGOMA tools represent a collection of tools that exist among the partners of SANGOMA. They are not yet adapted to the final standard interface structure, which will be documented in the Deliverable D1.5.

Chapter 2

Overview of Tools

2.1 Diagnostic Tools

sangoma_ComputeHistogram	Compute ensemble rank histograms
sangoma_ComputeEnsStats	Compute ensemble statistics
mutual_information	Compute mutual information in a particle filter
relative_entropy	Compute relative entropy in a particle filter
sensitivity	Compute sensitivity of posterior mean to observations in a particle filter

2.2 Perturbation Tools

sangoma_MVNormalize	Perform multivariate normalization
sangoma_EOFCovar	Initialize covariance matrix from EOF decomposition
Weakly constrained ensemble perturbations	Create ensemble perturbations that have to satisfy an a priori linear constraint

2.3 Transformation Tools

Empirical Gaussian Anamorphosis	Determine the empirical transformation function such that a transformed variable follows a Gaussian distribution

2.4 Utilities

hfradar_extractf	Observation operator for HF radar surface currents
PodCalibrate	Calibration tool for estimating uncertain model parameters
EnKF	Ensemble Kalman filter as introduced by Evensen and Burgers

Chapter 3

Documentation of Tools

3.1 Diagnostic Tools

3.1.1 sangoma_ComputeHistogram — Increment rank histogram (Source File: sangoma_ComputeHistogram.F90)

INTERFACE:

```
SUBROUTINE sangoma_ComputeHistogram(dim, dim_ens, element, &
     state, ens, hist, status)
```

DESCRIPTION:

This routine increments information on an ensemble rank histogram. Inputs are the ensemble array and a state vector about which the histogram is computed. In addition, the index of the element has to be specified for which the histogram is computed. If this is 0, the histogram information is collected over all elements.

The input/output array 'hist' has to be allocated externally. In addition, it has to be initialized with zeros before the first call.

REVISION HISTORY:

```
2012-08 - Lars Nerger - Initial code for SANGOMA based on PDAF
```

USES:

```
IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER, INTENT(in) :: dim                ! State dimension
INTEGER, INTENT(in) :: dim_ens            ! Ensemble size
INTEGER, INTENT(in) :: element            ! Element used for histogram
     ! If element=0, all elements are used
REAL, INTENT(in)    :: state(dim)         ! State vector
REAL, INTENT(in)    :: ens(dim, dim_ens)  ! State ensemble
INTEGER, INTENT(inout) :: hist(dim_ens+1) ! Histogram about the state
INTEGER, INTENT(out) :: status            ! Status flag (0=success)
```

3.1.2 `sangoma_ComputeEnsStats` — Compute ensemble statistics (Source File: sangoma_ComputeEnsStats.F90)

INTERFACE:

```
SUBROUTINE sangoma_ComputeEnsStats(dim, dim_ens, element, &
     state, ens, skewness, kurtosis, status)
```

DESCRIPTION:

This routine computes the higher-order ensemble statistics (skewness and kurtosis). Inputs are the ensemble array and the state vector about which the histogram is computed (usually the ensemble mean). In addition, the index of the element has to be specified for which the statistics are computed. If this is 0, the mean statistics over all elements are computed.

The definition used for kurtosis follows that used by Lawson and Hansen, Mon. Wea. Rev. 132 (2004) 1966.

REVISION HISTORY:

```
2012-09 - Lars Nerger - Initial code for SANGOMA based on PDAF
```

USES:

```
IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER, INTENT(in) :: dim               ! PE-local state dimension
INTEGER, INTENT(in) :: dim_ens           ! Ensemble size
INTEGER, INTENT(in) :: element           ! ID of element to be used
     ! If element=0, mean values over all elements are computed
REAL, INTENT(in)    :: state(dim)        ! State vector
REAL, INTENT(in)    :: ens(dim, dim_ens) ! State ensemble
REAL, INTENT(out)   :: skewness          ! Skewness of ensemble
REAL, INTENT(out)   :: kurtosis          ! Kurtosis of ensemble
INTEGER, INTENT(out) :: status           ! Status flag (0=success)
```

3.1.3 `mutual_information` (Source file: mutual_information.m)

INTERFACE:

```
MI=mutual_information(M,y,R,H,xp,method,prior_w)
```

DESCRIPTION:

This script calculates mutual information within a particle filter. It uses the assumption that the likelihood is Gaussian. The script is to be used after the

particle weights have been updated by the observations.
The method applied in the script is as follows: Mutual information is the relative entropy (RE) averaged over observation space:

$$MI = \int (RE * p(y))\, dy \tag{3.1}$$

Here $p(y) = \int (p(y|x) * p(x))dy$, given approximately by the sum of the posterior weights. MI can then be approximated in two ways:

1. Quadrature: Discretise the observation space into M points.

$$MI = \sum_{i=1}^{M} (RE_i * p(y)_i) * dy. \tag{3.2}$$

2. Random sampling: Sample M random points from $p(y|x)$.

$$MI = \sum_{i} (RE_i * p(y)_i) \tag{3.3}$$

INPUT PARAMETERS:

M scalar, sample size for observation space.

y vector size p, observations at current time, where p is a number of spatial observations at current time.

R square matrix of size p, the observation error covariance matrix.

H a matrix of size p by n, the (linear) observation operator, where n is size of the state space.

xp matrix of size n by N, the partical values, where N is a number of particles.

method string, you have a choice of method either 'quad' which solves the integral using a quadrature method or 'rndm' which solves the integral using a random sampling method.

prior_w (optional) vector of size N, prior weights. If not explicitly given these are assumed to be 1/N.

OUTPUT PARAMETERS:

MI scalar, mutual information.

3.1.4 `relative_entropy` (Source file: relative_entropy.m)

INTERFACE:

```
RE=relative_entropy(post_w,prior_w)
```

DESCRIPTION:
This script calculates relative entropy within a particle filter. It is to be used after weights have been updated by the observations.
The method applied in the script is as follows: RE is given by

$$\int p(x|y) \ln \left[\frac{p(x|y)}{p(x)} \right] dx \ . \tag{3.4}$$

If the particle positions are unchanged during the assimilation, and only the weights are updated, RE can be approximated in terms of the relative weights:

$$RE \approx \sum post_w \ln \left(\frac{post\_w}{prior\_w} \right) \tag{3.5}$$

INPUT PARAMETERS:

post_w vector of size N, posterior weights, where N is a number of particles.

prior_w (optional) vector of size N, prior weights. If not explicitly given these are assumed to be 1/N.

OUTPUT PARAMETERS:

RE scalar, relative entropy of posterior given the prior.

3.1.5 `sensitivity` (Source file: sensitivity.m)

INTERFACE:

```
[S,Pa]=sensitivity(R,H,post_w,xp)
```

DESCRIPTION:
This function calculates sensitivity of the posterior mean to the observations (assuming Gaussian observation error and a linear observation operator) within a particle filter. It is to be used after weights have been updated by the observations.
The method applied in the script is as follows: The sensitivity of the analysis to the observations can be calculated exactly as the ratio of the posterior variance in observation space to the observation error covariance.
Details of the method are explained in:
A. Fowler and P. J. van Leeuwen. *Measures of observation impact in non-Gaussian data assimilation*. Tellus A 2012, **64**, 17192. doi:10.3402/tellusa.v64i0.17192

INPUT PARAMETERS:

R square matrix of size p, the observation error covariance matrix, where p is the size of observation space.

H a matrix of size p by n, the (linear) observation operator, where n is size of the state space.

post_w vector of size N, posterior weights, where N is a number of particles.

xp matrix of n by N, the particle values.

OUTPUT PARAMETERS:

S square matrix of size p, the sensitivity of the posterior mean to the observations in observation space.

Pa square matrix of size n, the posterior covariance matrix.

3.2 Perturbation Tools

3.2.1 `sangoma_MVNormalize` — Perform multivariate normalization (Source File: sangoma_MVNormalize.F90)

INTERFACE:

```
SUBROUTINE sangoma_MVNormalize(mode, dim_state, dim_field, offset, &
     ncol, states, stddev, status)
```

DESCRIPTION:

This routine performs multivariate normalization and re-scaling. It has two modes:

mode=1: In this case, the routine computes the standard deviation of a field inside the array 'states' holding in each column a state vector. The standard deviation is computed over all columns if the state vector array. Then, the field is normalized for unit standard deviation by dividing the values by the standard deviation. The standard deviation is provided on output together with the scaled array 'states'

mode=2: In this case the input variable 'stddev' is used to rescale the corresponding part of the array 'states'. Usually 'stddev' is obtained by a call with mode=1 before.

REVISION HISTORY:

```
2012-09 - Lars Nerger - Initial code for SANGOMA based on PDAF
```

USES:

```
IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER, INTENT(in) :: mode      ! Mode: (1) normalize, (2) re-scale
INTEGER, INTENT(in) :: dim_state  ! Dimension of state vector
INTEGER, INTENT(in) :: dim_field  ! Dimension of a field in state vector
INTEGER, INTENT(in) :: offset     ! Offset of field in state vector
INTEGER, intent(in) :: ncol       ! Number of columns in array states
REAL, INTENT(inout) :: states(dim_state, ncol)  ! State vector array
REAL, INTENT(inout) :: stddev     ! Standard deviation of field
   ! stddev is output for mode=1 and input for mode=2
INTEGER, INTENT(out) :: status    ! Status flag (0=success)
```

3.2.2 `sangoma_EOFCovar` — Perform EOF decomposition of state trajectory (Source File: sangoma_EOFCovar.F90)

INTERFACE:

```
SUBROUTINE sangoma_EOFCovar(dim_state, nstates, nfields, dim_fields, &
     offsets, do_mv, states, rms, svals, svec, status)
```

DESCRIPTION:

This routine performs an EOF analysis by singular value decomposition. It is used to prepare a covariance matrix for initializing an ensemble. For the decomposition a multivariate scaling can be performed by 'sangoma_MVNormalize' to ensure that all fields in the state vectors have unit variance.

To use this routine, one has to initialize the array 'states' hondling in each column a perturbation vector (state - mean) from a state trajectory. Outputs are the arrays of singular values (svals) and left singular vectors (svec). The singular values are scaled by sqrt(1/(nstates-1)). With this, $svec * svals^2 * svec^T$ is the covariance matrix. In addition, the standard deviation of the fields variance (rms) is an output array. To use the multivariate normalization one has to define the number of different fields in the state (nfields), the dimension of each fields and the offset of field from the start of each state vector.

The routine uses the LAPACK routine 'dgesvd' to compute the singular value decomposition.

REVISION HISTORY:

```
2012-09 - Lars Nerger - Initial code for SANGOMA based on PDAF
```

USES:

```
IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER, INTENT(in) :: dim_state    ! Dimension of state vector
INTEGER, INTENT(in) :: nstates      ! Number of state vectors
INTEGER, INTENT(in) :: nfields      ! Number of fields in state vector
```

```
INTEGER, INTENT(in) :: dim_fields(nfields)  ! Size of each field
INTEGER, INTENT(in) :: offsets(nfields)     ! Start position of each field
INTEGER, INTENT(in) :: do_mv                ! 1: Do multivariate scaling
   ! nfields, dim_fields and offsets are only used if do_mv=1
REAL, INTENT(in)  :: states(dim_state, nstates) ! State perturbations
REAL, INTENT(out) :: rms(nfields)           ! Standard deviation of field
   ! Without multivariate scaling (do_mv=0), it is rms = 1.0
REAL, INTENT(out) :: svals(nstates)         ! Scaled singular values
REAL, INTENT(out) :: svec(dim_state, nstates)   ! Singular vectors
```

3.2.3 Weakly constrained ensemble perturbations

This package creates ensemble perturbations that have to satisfy an a priori linear constraint. It can also be used to create perturbations that are aware of the land-sea mask or that use space- (or time-) dependent correlation length.

Details of the procedure are explained in:

A. Barth, A. Alvera-Azcárate, J.-M. Beckers, R. H. Weisberg, L. Vandenbulcke, F. Lenartz, and M. Rixen. Dynamically constrained ensemble perturbations - application to tides on the West Florida Shelf. Ocean Science, 5(3):259-270, 2009. doi: 10.5194/os-5-259-2009

A. Barth, A. Alvera-Azcárate, K.-W. Gurgel, J. Staneva, A. Port, J.-M. Beckers, and E. V. Stanev. Ensemble perturbation smoother for optimizing tidal boundary conditions by assimilation of High-Frequency radar surface currents - application to the German Bight. Ocean Science, 6(1):161-178, 2010. doi: 10.5194/os-6-161-2010

Function `wce_simple`

INTERFACE:

```
[Ep,info] = wce_simple(mask,pmn,len,Nens,k,...)
```

DESCRIPTION:

Generate ensemble perturbations taking into account: the land-sea mask, correlation length (possibly varying in space) and possibly a vector field (advection constraint). This function works in an arbitrary high dimensional space on an orthogonal curvilinear grid characterized by the metric scale factors.

INPUT PARAMETERS:

mask land-sea mask (true: sea and false: land). This array has n dimensions.

pmn cell array of n arrays (each n-dimensional). The arrays are the metric scale factors for the different dimensions (units are (length-scale)$^{-1}$).

len correlation length. It can be a scalar if the correlation length is constant and the same in all dimensions or a cell array of n arrays. In the later case each array has to be n-dimensional (units are length-scale).

Nens number of ensemble members to generate.

k number of eigenvector and eigenvalues.

OPTIONAL INPUT PARAMETERS:

'velocity', velocity vector field for the advection constraint (units: length-scale). This vector field can be scaled such that the alignment of the perturbation is satisfactory. The array velocity has the same size as mask.

OUTPUT PARAMETERS:

Ep perturbations (same size as mask plus the trailing ensemble dimension).

info structure with some intermediate results:

info.sv structure describing the concatenated state vector.

info.WU eigenvectors. Use info.sv and statevector_unpack to extract the individual variables from WU.

info.lambda eigenvalues.

info.WE weighting matrix related to the total energy. $\mathbf{x}^T\mathbf{W}_E\mathbf{x}$ is the total barotropic energy of the vector x.

NOTE:

The unit "length-scale" can be for example meters or arc degrees. The choice of the unit must be consistent for all parameters.

Function `wce_tides`

INTERFACE:

```
[Ezeta,Eu,Ev,info] = wce_tides(h,pm,pn,g,f,len,alpha,omega,k,Nens,cdragu,cdragv)
```

DESCRIPTION:

Generate ensemble perturbations constrained by the harmonic shallow water equations as a weak constrain. It can be used to create perturbations for tidal parameters.

INPUT PARAMETERS:

h bathymetry (in m, two-dimensional array, positive in water and NaN on land).

pm inverse of the local resolution in the first dimension (in m^{-1}, same size as h).

pn inverse of the local resolution in the second dimension (in m^{-1}, same size as h).

g acceleration due to gravity (scalar, m/s^2).

f Coriolis frequency (scalar, 1/s).

len correlation length (scalar, in m).

alpha factor penalizing the total energy (adimensional).

omega angular frequency (rad/s).

k number of eigenvector and eigenvalues.

Nens number of ensemble members to generate.

OPTIONAL INPUT PARAMETERS:

cdrag_u, cdrag_v linear drag in the u- and v-momentum equation (no drag is assumed if they are omitted).

OUTPUT PARAMETERS:

Ezeta, Eu, Eu perturbation for elevation (in m), u and v (depth averaged velocity in m/s). The shape of these arrays is the same as mask.

info structure with some intermediate results:

info.sv structure describing the concatenated state vector.

info.WU eigenvector. Use info.sv and statevector_unpack to extract the individual variables from WU.

info.lambda eigenvalues.

info.WE weighting matrix related to the total energy. $\mathbf{x}^T\mathbf{W}_E\mathbf{x}$ is the total barotropic energy of the vector x.

NOTE:

see wce_example_tides.m

Function `statevector_init`

INTERFACE:

```
s = statevector_init(mask1, mask2, ...)
```

DESCRIPTION:

Initialize structure for packing and unpacking multiple variables given their corresponding land-sea mask.

INPUT PARAMETERS:

mask1, mask2,... land-sea mask for variable 1,2,... Sea grid points correspond to one and land grid points to zero. Every mask can have a different shape.

OUTPUT PARAMETERS:

s structure to be used with statevector_pack and statevector_unpack.

NOTE:

see also statevector_pack, statevector_unpack

Function `statevector_pack`

INTERFACE:

```
x = statevector_pack(s,var1, var2, ...)
```

DESCRIPTION:

Pack the different variables var1, var2, ... into the vector x. Only sea grid points are retained.

INPUT PARAMETERS:

s structure generated by statevector_init.

var1, var2,... variables to pack (with the same shape as the corresponding masks).

OUTPUT PARAMETERS:

x vector of the packed elements. The size of this vector is the number of elements of all masks equal to 1.

NOTE:

If var1, var2, ... have an additional trailing dimension, then this dimension is assumed to represent the different ensemble members. In this case x is a matrix and its last dimension is the number of ensemble members.

Function `statevector_unpack`

INTERFACE:

```
[var1, var2, ...]  = statevector_unpack(s,x) [var1, var2, ...]
= statevector_unpack(s,x,fillvalue)
```

DESCRIPTION:

Unpack the vector x into the different variables var1, var2, ...

INPUT PARAMETERS:

s structure generated by statevector_init.

x vector of the packed elements. The size of this vector is the number of elements equal to 1 in all masks.

OPTIONAL INPUT PARAMETERS:

fillvalue The value to fill in var1, var2,... where the masks correspond to a land grid point. The default is zero.

OUTPUT PARAMETERS:

var1, var2,... unpacked variables.

NOTE:

If x is a matrix, then the second dimension is assumed to represent the different ensemble members. In this case, var1, var2, ... have also an additional trailing dimension.

3.3 Transformation Tools

3.3.1 `anam_setup` — Empirical Gaussian Anamorphosis (Anam)

INTERFACE:

```
anam = anam_setup(x)
```

DESCRIPTION:

Determine the empirical transformation function (empirical Gaussian anamorphosis). The transformed data (anam_transform(anam,x)) should follow approximately a Gaussian distribution. The transformation function is a piece-wise linear function.

INPUT PARAMETERS:

x a data sample (vector).

OPTIONAL INPUT PARAMETERS:

'addnoise', addnoise add Gaussian noise level to the data sample.

'method', method method can be either 'direct' (i.e. the data sample is mapped directly to Gaussian distributed variable) or 'by_uniform' (i.e. an analytical transformation is first applied to bring the data sample to a bounded interval).

'N', N number of segments of the piece-wise linear function.

OUTPUT PARAMETERS:

anam a structure describing the transformation used in anam_transform and anam_invtransform.

3.3.2 Function `anam_transform`

INTERFACE:

```
y = anam_transform(anam,x)
```

DESCRIPTION:

Transform the data x according to the transformation anam.

INPUT PARAMETERS:

anam transform (created by anam_setup).

x original data.

OUTPUT PARAMETERS:

y transformed data.

3.3.3 Function `anam_invtransform`

INTERFACE:

```
x = anam_invtransform(anam,y)
```

DESCRIPTION:

Apply inverse transformation to y according to the transformation anam.

INPUT PARAMETERS:

anam transform (created by anam_setup).

y transformed data.

OUTPUT PARAMETERS:

x data in original scale.

3.4 Utilities

3.4.1 `hfradar_extractf` - Observation operator for HF radar surface currents

INTERFACE:

```
[velg,velf] = hfradar_extractf(domain,site,u,v)
```

DESCRIPTION:

Extract the model equivalent of HF radar surface currents. The currents u and v are specified on an Arakawa-C grid.

INPUT PARAMETERS:

domain structure describing the model domain with the following fields:

domain.lon_u, domain.lon_v longitude of model grid at u/v points (degrees east).

domain.lat_u, domain.lat_v latitude of model grid at u/v points (degrees north).

domain.z_u, domain.z_v depth of model grid at u/v points (negative in water).

site structure describing the HF radar site with the following fields:

site.nu the frequency of the HF radar system in Hz.

site.res the effective azimuthal resolution in degrees.

site.lon0, site.lat0 longitude and latitude of the HF radar system.

site.lon, site.lat radial grid of the HF radar data. The first dimension is the radial dimension and the second is the azimuth.

site.grid.lon, site.grid.lat Cartesian grid of the HF radar data. The first dimension is longitude, and the second is the latitude.

u,v u- and v-velocity of the model currents on Arakawa-C grid. The order of the dimensions is longitude, latitude and depth.

OUTPUT PARAMETERS:

velg radial velocity on a Cartesian grid.

velf radial velocity on a radial grid.

NOTE:

The sizes of the variable on u- and v-grid are related by size(u,1) + 1 == size(v,1) and size(u,2) == size(v,2) + 1

3.4.2 `PodCalibrate` — Calibration tool for estimating uncertain model parameters (Source File: PodCalibrate.F90 also uses computecost.F90, reducecost.F90, error.F90, reducegradient.F90, lbfgs_rec.F90)

INTERFACE:

```
SUBROUTINE PodCalibrate(modelcls,hsobs_all,rpcalib)
```

DESCRIPTION:

This routine estimates uncertain model parameters. The routine requires OpenDA (COSTA) toolbox. Inputs are class handle of the model (which contains information of model state, input forcing and model parameters) and a handle to stochastic observer containing all observations. The third input is model configuration parameter (an xml input file). The following example shows the main structure of this XML-input file:

```
<?xml version="1.0" encoding="UTF-8"?>
<costa xmlns:xi="http://www.w3.org/2001/XInclude">
   <!-- Observations used for data assimilation or calibration -->
   <CTA_SOBS id="obs_assim" database="obs_lorenz96.sql"/>

   <!-- Model that is used -->
   <CTA_MODELCLASS id="modelclass"  name="CTA_MODBUILD_SP" />

   <!-- Data assimilation or calibration method that is used -->
   <method name="PodCalibrate">
      <!-- Configuration of the method -->
   </method>
</costa>
```

The section `obs_assim` specifies the observations that are used in the calibration process. The section `modelclass` specifies which model is used. The section `method` specifies the data assimilation or calibration method that is used. The tag name specifies the used method.

Details of the procedure are explained in:

M. U. Altaf, M. Verlaan, A. W. Heemink. Efficient identification of uncertain parameters in a large-scale tidal model of the European continental shelf by proper orthogonal decomposition. International Journal for Numerical Methods in Fluids, 68:422-450, 2012. doi: 10.1002/fld.2511

REVISION HISTORY:

```
2012-10 - M. Umer Altaf - Code based on OpenDA (COSTA)
```

USES:

```
IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER, rpcalib   ! (i) CTA_Tree :model configuration (xml-input)
INTEGER, modelcls  ! (i) CTA_ModelClass: Class handle of (deterministic) model
INTEGER, hsobs_all ! (i) CTA_StochObs: Stochastic observer containing all observations
```

bigskip

3.4.3 `EnKF` — Ensemble Kalman filter as introduced by Evensen and Burgers (Class File: EnKF.java, Main Method Analysis also uses AbstractSequentialAlgorithm.java and AbstractSequentialEnsembleAlgorithm.java)

INTERFACE:

```
public void analysis(IStochObserver obs,
                     IVector obsValues,
                     IVector predictions,
                     IStochModelInstance mainModel,
                     ITime analysisTime)
```

DESCRIPTION:

Traditional Ensemble Kalman filter as introduced by Evensen and Burgers. This is a classical implementation processing all observations at once. There is no Schur-product for spatial truncation. The routine requires OpenDA toolbox.

REVISION HISTORY:

```
2012-10 - Martin Verlaan - Code based on OpenDA
```

ARGUMENTS:

```
obs,              !  (i)  The stoch.observer
obsValues,        !  (i)  Values for the observations as a vector
predictions,      !  (i)  Model values corresponding
                  ! to the observations before the analysis step
mainModel,        !  (i)  Main model for output
                  ! (as defined in AbstractSequentialAlgorithm)
analysisTime,     !  (i)  timelabel for the analysis
                  ! (not neccessarily the time of the observations)
```