SANGOMA: Stochastic Assimilation for the Next Generation Ocean Model Applications

EU FP7 SPACE-2011-1 project 283580

Deliverable 1.5: Documentation of Specifications

Due date: 01/11/2012 Delivery date: 01/11/2012 Delivery type: Report , public

Arnold Heemink, Nils van Velzen, Martin Verlaan, Umer Altaf,

Delft University of Technology, NETHERLANDS

Jean-Marie Beckers, Alexander Barth, University of Liège, BELGIUM

Peter Jan Van Leeuwen, University of Reading, UK

Lars Nerger, Alfred-Wegener-Institute, GERMANY

Pierre Brasseur, Jean-Michel Brankart , CNRS-LEGI, FRANCE

Pierre de Mey, CNRS-LEGOS, FRANCE

Laurent Bertino, NERSC, NORWAY

1. Purpose of this Document

This documents gives an up to date list of software components that are eligible to be shared in SANGOMA with complete specifications. Each software component is elaborated and detailed description of the physical interfaces of these software components is provided. It provides us with a very useful form for describing the individual tools. This includes the sequence of input / output arguments and their types. The modules that are mentioned in this document form an initial set for the initial activities in the SANGOMA project. This document will be updated through out the project duration with information gathered in the collaboration between the partners and whenever a new software module will be adopted.

2. POD calibration tool

- **Description of functionality/purpose:** Variational method for the calibration of dynamical models. This method does not require an adjoint of the dynamical model. The method uses an approximated adjoint, that is computed only using the forward dynamical model. Proper Orthogonal Decomposition POD or Balanced Proper Orthogonal Decomposition (BPOD) is used for the approximation.

  References:
  - M. U. Altaf, A. W. Heemink, and M. Verlaan, *Inverse shallow-water flow modelling using model reduction*, International Journal for Multiscale Computational Engineering, 7 (2009), pp. 577–596.
  - M. U. Altaf, *Model Reduced Variational Data Assimilation for Shallow Water Flow Models,* PhD Thesis. Delft University of Technology (2011).

  The method is implemented in three separate steps (functions). The first two functions are executed to obtain reduced model and the final (main) program computes the values of cost and gradient vector.

1. Function Name: **Compute_POD**

- **Operation:** This function reads an initial ensemble (***nens*** ensemble members) from Netcdef files of model states and store them to an array of vectors ***(nstate, nens)*** of size nstate. Then the eigenvalue problem is solved using this array of ensemble. An example of such a Netcdef file is already given in data model description. Before solving the eigenvalue problem the individual vectors may be normalized. The outputs of the function are truncated eigenvalues and eigenvectors.

- **Inputs:** Netcdef files containing model states (see section 2.2)

- **Output:** A data file containing truncated eigenvectors and eigenvalues. The elements of output file are:

P(nstate,nvals)	truncated eigenvectors
Evals(nvals)	vector containing eigenvalues

2. Function Name: **Compute_RM**

- **Operation:** This function first reads truncated eigenvectors and run model simulations to get model sensitivities with respect to state and parameters. These sensitivities and eigenvectors are then used to construct reduced dynamic operators to perform optimization in reduce space. The outputs of this function are these reduced dynamic operators which are saved to a data file and then use in the main optimization routine.

- **Inputs:** A data file containing truncated eigenvectors and eigenvalues.

  P(nstate,nvals) truncated eigenvectors
  Evals(nvals) vector containing eigenvalues

- **Output:** A data file containing two arrays of vectors containing information of model dynamics and model sensitivities in reduced space. The elements of this file are:

  M(nvals, nvals, nstep) array of vectors of type real
  M_a(nvals, npars, nstep) array of vectors of type real

3. Function Name: **Compute_cost_grad**

- **Operation:** This is the main routine which reads truncated eigenvectors and reduced dynamic operators and then computes the values of cost function and gradient vector. The outputs of this module are the values of the cost function and the gradient vector which are then use by minimization procedure to obtain optimal parameters. Main variables of this functions and their types are defined here.

Variable Name	Type	Description
Nstate	Integer	Size of state vector
Nens	Integer	Size of ensemble
Npar	Integer	Size of parameter
Nvals	Integer	Size of reduced model
Nobs	Integer	Size of observation vector
Nstep	Integer	No. of timesteps
X(nvals)	real array	Real array containing model state at particular time instant
Y(nobs)	real array	Real array containing observations at particular time instance
H(nobs,nvals)	integer array	Vector array containing observation to reduced state mapping
M(nals,nvals)	real array	Vector array containing model dynamic at particular time instance

M_a(nvals,npar)	real array	Vector array containing parameter sensitivities at particular time instance
Nost	Real	Value of cost function
grad(npar)	real array	Real array containing gradient information

- **Inputs:** Truncated eigenvectors, reduced dynamic operators, observations.

P(nstate,nvals)	truncated eigenvectors
Evals(nvals)	vector containing eigenvalues
M(nvals, nvals, nstep)	array of vectors of type real
M_a(nvals, npar, nstep)	array of vectors of type real

- **Output:** Value of objective (cost) function and a vector containing gradient informationThe elements of this file are

cost	value of cost function
grad(npars)	vector containing gradient information

2.1 Pseudo Code

```
1)      program compute_POD
              read ensemble(nens)  // Assumes ensemble already available
              do i = 1 to nens
                     read ensemble(i)
                     set X(ntate,i) = ensemble(i)
              end do
              Normalize X
              Compute X'X
              Compute EVD(X'X)
              Return P(nstate,nvals)
              Return Evals(nvals)
       end compute_POD

2)      program compute_RM
              read P(nstate,nvals)
              read Evals(nvals)
              do i = 1 to nvals
                     Linearize state in P direction //requires model runs
                     Compute M(i,i,step)
                  Compute M_a(i, npar, nstep) // requires model runs
              end do
              Return M(nvals, npar, nstep)
            Return M_a(nvals, npar, nstep)
       end compute_RM

2)      program compute_cost_grad
              cost = 0.0d0
              grad(nvals) is zeros
              read P(nstate,nvals)
              read Evals(nvals)
              read M(nvals, npar, nstep)
            read M_a(nvals, npar, nstep)
              do step = 1 to tstep
                     forward step to find X(step)
                     if observation exists
                            cost = cost + (Y(step) - HX(step))^2
                     end if
              end do

              do step = nstep-1 to 1
                     grad(step) = backward step
              end do
              Return cost
              Return grad
       end compute_cost_grad
```

2.2 Example of Netcdef file as explained in data model description:

netcdf MetO-NWS-PHYS-dm-Agg_1338362551845 {

dimensions:
time = 2 ;
depth = 24 ;
lat = 166 ;
lon = 91 ;

variables:
float lon(lon) ;
lon:standard_name = "longitude" ;
lon:units = "degrees_east" ;
lon:long_name = "longitude" ;
lon:nav_model = "Default grid" ;
lon:axis = "X" ;
lon:_CoordinateAxisType = "Lon" ;
lon:valid_min = -1.000214f ;
lon:valid_max = 8.999739f ;
double time(time) ;
time:standard_name = "time" ;
time:units = "seconds since 2011-04-07 00:00:00" ;
time:calendar = "Gregorian" ;
time:long_name = "Validity time" ;
time:data_time = 86400.f ;
time:axis = "T" ;
time:_CoordinateAxisType = "Time" ;
time:valid_min = 36244800. ;
time:valid_max = 36331200. ;
short vosaline(time, depth, lat, lon) ;
vosaline:_CoordinateAxes = "time depth lat lon " ;
vosaline:_FillValue = -32768s ;
vosaline:missing_value = -32768s ;
vosaline:scale_factor = 0.001f ;
vosaline:add_offset = 30.f ;
vosaline:standard_name = "sea_water_salinity" ;
vosaline:long_name = "Sea Water Salinity" ;
vosaline:units = "1e-3" ;
float lat(lat) ;
lat:standard_name = "latitude" ;
lat:units = "degrees_north" ;
lat:long_name = "latitude" ;
lat:nav_model = "Default grid" ;
lat:axis = "Y" ;
lat:_CoordinateAxisType = "Lat" ;
lat:valid_min = 49.00001f ;
lat:valid_max = 60.00001f ;
float depth(depth) ;
depth:axis = "Z" ;
depth:standard_name = "depth" ;

```
        depth:units                     = "m" ;
        depth:positive                          = "down" ;
        depth:long_name                 = "depth" ;
        depth:_CoordinateAxisType  = "Height" ;
        depth:_CoordinateZisPositive = "down" ;
        depth:valid_min                 = 0.f ;
        depth:valid_max                 = 5000.f ;

// global attributes:
:title              = "North West European Shelf from UK Met Office Model FOAM 7 km" ;
:institution        = "UK Met Office" ;
:references         = "http://www.ncof.co.uk" ;
:source             = "UK Met Office Operational Suite, FOAM 7 km run 2012-05-29" ;
:Conventions  = "CF-1.0" ;
:history            = "Data extracted from dataset http://data.ncof.co.uk/..." ;
:time_min           = 36244800. ;
:time_max           = 36331200. ;
:julian_day_unit = "seconds since 2011-04-07 00:00:00" ;
:z_min                      = 0. ;
:z_max              = 5000. ;
:latitude_min   = 49.0000076293945 ;
:latitude_max  = 60.0000114440918 ;
:longitude_min = -1.00021362304688 ;
:longitude_max = 8.9997386932373 ;
}
```

3. Sangoma_ComputeHistogram

- **Description of functionality/purpose:** 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.

- **Operation:** Input and output is performed in memory via the calling interface of the subroutine.

- **Interface:** The calling interface is defined as follows:

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

Arguments:

```
INTEGER, INTENT(in) :: dim       ! State dimension
INTEGER, INTENT(in) :: dim_ens   ! Ensemble size
INTEGER, INTENT(in) :: element   ! Element of vector 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 state
INTEGER, INTENT(out) :: status             ! Status flag (0=success)
```

4. Sangoma_ComputeEnsStats

- **Description of functionality/purpose:** This routine computes the higher-order ensemble statistics (skewness and kurtosis). Inputes 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.

- **Operation:** Input and output is performed in memory via the calling interface of the subroutine.

- **Interface:** The calling interface is defined as follows:

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

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)
```

5. Sangoma_MVNormalize

- **Description of functionality/purpose:** 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 de viation is computed over all columns if the state vector array. Then, the field is normalized for unit standard deviation by deviding the values by the standard de- viation. 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.

- **Operation:** Input and output is performed in memory via the calling interface of the subroutine.

- **Interface:** The calling interface is defined as follows:

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

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 field in state vec-
tor
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)
```

6. Sangoma_EOFCovar

- **Description of functionality/purpose:** This routine performs an EOF analysis by singular value decomposition. It is used to prepare a covariance matrix for initializing an ensemble. For the decom- position 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.

- **Operation:** Input and output is performed in memory via the calling interface of the subroutine.

- **Interface:** The calling interface is defined as follows:

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

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 perturba-
tions
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
```

7. Weakly Constrained Ensembles (WCE)

- **Description of functionality/purpose:** This Is a Matlab / Octave package. The interface of the package might change in the future. 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.

  References:
  - 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.

1. Function Name: **wce_simple**

- **Operation:** 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.

- **Inputs:**

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

  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.

velocity (opt) 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.

- **Outputs:**

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. x' * WE * 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.

2. Function Name: **wce_tides**

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

- **Inputs:**

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	actor penalizing the total energy (adimensional).
omega	Angular frequency (rad/s).
k	Number of eigenvector and eigenvalues.
nens	Number of ensemble members to generate.
cdrag_u (opt)	Linear drag in the u- and v-momentum equation (no drag is assumed if they are omitted).
cdrag_v (opt)	Linear drag in the u- and v-momentum equation (no drag is assumed if they are omitted).

- **Outputs:**

ezeta, eu, ev	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	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. x' * WE * x is

the total barotropic energy of the vector x.

3. Function Name: **statevector_init**

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

- **Inputs:**

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:**

s	structure to be used with statevector_pack and statevector_unpack.

4. Function Name: **statevector_pack**

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

- **Inputs:**

s	Structure generated by statevector_init .
var1, var2,…	Variables to pack (with the same shape as the corresponding masks).

- **Output:**

x	Vector of packed elements. The size of this vector is 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.

5. Function Name: **statevector_unpack**

- **Operation:** Unpack the vector x into the different variables var1, var2, ...
- **Inputs:**

x	Vector of packed elements. The size of this vector is number of elements of all masks equal to 1.
s	Structure generated by statevector_init.
fillvalue (opt)	The value to fill in var1, var2,... where the masks correspond to a land grid point. The default is zero.

- **Outputs:**

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.

8. Empirical Gaussian Anamorphosis (Anam)

- **Description of functionality/purpose:** This Is a Matlab / Octave package. The interface of the package might change in the future. This package determines the empirical transformation function such that a transformed variable should follow a Gaussian distribution.

1. Function Name: **anam_setup**

- **Operation:** 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.

- **Inputs:**

x	A data sample (vector).
'addnoise', addnoise	Add Guasssian noise level to the data sample.
'method', method (opt)	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.
k	Number of eigenvector and eigenvalues.

- **Output:**

anam	a structure describing the transformation used in anam_transform and anam_inv t .

2. Function Name: **anam_transform**

- **Operation:** Transform the data x according to the transformation anam.

- **Inputs:**

  anam transform (created by anam_setup).

  x Original data.

- **Output:**

  y Transformed data.

3. Function Name: **anam_invtransform**

- **Operation:** Apply inverse transformation to y according to the transformation anam.

- **Inputs:**

  anam transform (created by anam_setup).

  y Transformed data.

- **Output:**

  x Data in original scale.

9. hfradar_extractf

- **Description of functionality/purpose:** This Is a Matlab / Octave package. The interface of the package might change in the future. Observation operator for HF radar surface currents

1. Function Name: **hfradar_extractf**

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

- **Inputs:**

x	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.ion0, 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,	Cartesian grid of the HF radar data. The first

site.grid.lat	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:**

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