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

Deliverable 2.4:
Software V1 report
Due date: 30/04/2014
Delivery date: 25/04/2014
Delivery type: Report, public

Lars Nerger Paul Kirchgessner
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 Sanita Vetra-Carvalho
University of Reading, UK

Pierre Brasseur Jean-Michel Brankart Guillem Candille
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 the intermediate release V1 of the SANGOMA tools, which was replaced by 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 second release of the SANGOMA tools (V1). 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 represented a collection of tools that exist among the partners of SANGOMA. They were not yet adapted to the final standard interface structure, which is documented in the Deliverable D1.5. The new release V1 includes tools that are adapted to the standard data model and interface structure. Both, tools from the V0 release as well as additional tools are included. All tools are independent of the particular data assimilation framework and should be usable with any other implementation of data assimilation algorithms. Next to the documentation in this document, the tool release now also includes examples on how to use a particular tools.

The tools release package can be downloaded from the project web site at `http://www.data-assimilation.net/Tools/`.

# Chapter 2

# Overview of Tools

## 2.1 Diagnostic Tools

### *Fortran*

| | |
|---|---|
| **sangoma_ComputeHistogram** | Compute ensemble rank histograms |
| **sangoma_ComputeEnsStats** | Compute ensemble statistics |
| **sangoma_computeRCRV** | Compute the bias & the dispersion of the RCRV |
| **sangoma_computeCRPS** | Compute the CRPS and its decomposition |
| **sangoma_computeBrier** | Compute the Brier skill score and its decomposition, and the entropy |
| **sangoma_ComputeSensitivity** | Calculate the sensitivity matrix with **H** as matrix |
| **sangoma_ComputeSensitivity_op** | Calculate the sensitivity matrix with **H** as operator |
| **sangoma_computeMI** | Calculate the mutual information |
| **sangoma_computeRE** | Calculate the relative entropy |

### *Matlab/Octave*

| | |
|---|---|
| **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

### *Fortran*

| | |
|---|---|
| **sangoma_pseudornd2D** | Generate random fields with given correlation length |
| **sangoma_MVNormalize** | Perform multivariate normalization |
| **sangoma_EOFCovar** | Initialize covariance matrix from EOF decomposition |

*Matlab/Octave*

| | |
|---|---|
| **Weakly constrained ensemble perturbations** | Create ensemble perturbations that have to satisfy an a priori linear constraint |

## 2.3 Transformation Tools

*Matlab/Octave*

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

## 2.4 Utilities

*Fortran*

| | |
|---|---|
| **sangoma_computepod** | Computes dominant POD modes from an ensemble of snapshots |
| **sangoma_costgrad** | Computes the values of Objective function and Gradient using reduced state dimensions |

*Matlab/Octave*

| | |
|---|---|
| **hfradar_extractf** | Observation operator for HF radar surface currents |

# Chapter 3

# Using the tools

## 3.1 Directory structure of the release

In the code package of the release we distinguish in between tools implemented in Fortran and tools for use with Matlab or Octave. The directory structure is as follows:

```
Fortran/
        ____ diagnostics/
                         _________ examples/
        ____ perturbations/
                           _______ examples/
        ____ transformations/
                             _____ examples/
        ____ utilities/
                       ___________ examples/
Matlab/
       _____ diagnostics/
                         _________ examples/
       _____ perturbations/
                           _______ examples/
       _____ transformations/
                             _____ examples/
       _____ utilities/
                       ___________ examples/
```

The directories are named after the four categories. For each category there is a sub-directory `examples/` in which example implementations are included that show how to use a particular tool.

## 3.2 Fortran-tools

The tools coded in fortran are organized in subdirectories of the directory `Fortran/` according to their category. You can build a library of SANGOMA tools as well as example programs that show how to use a particular tool.

The release uses `make` to build the library of SANGOMA tools. Thus one needs a version of `make` and a Fortran compiler. Some tools also need numerical libraries. The required libraries are **BLAS**, **LAPACK**, and **FFTW**. Please check the documentation in chapter 4, about the requirements of each tool. The Makefiles are configured so that the libraries should be found if they are in a standard path. For fftw3, the program 'pkg-config' is used. If this doesn't work to find the library, one should specify its location manually.

All tools use the Fortran module `sangoma_base.F90` that is stored in the directory `Fortran`. This module provides declarations for real and integer variables that are used to generate a c-compatible interface of the tools.

To build to SANGOMA tools library cd into the directory `Fortran` and execute `make`. This will build the library `Fortran/libsangoma_tools.a` which contains all tools and can be linked to other programs. Next to the library, the files `mod_sangoma_pseudornd.mod` and `sangoma_base.mod` will be generated by the build process. `sangoma_base.mod` is the module binary for a base module, which is only used internally in the tools. `mod_sangoma_pseudornd.mod` is a module binary file holding the Fortran module `mod_sangoma_pseudornd` that provides the tools to generate pseudo random fields following Evensen (1994).

To build the example programs for the tools of a category, one cd's into the corresponding example-dirctory, e.g. `diagnostics/examples/`. Executing `make` in this directory builds all examples contained in this directory. The Makefile also ensures that the SANGOMA tools library is built. The examples are named after the tool it uses. The names of the source code files always begin with `example_`. There are also some other files that are needed in particular examples.

## 3.3 Matlab-tools

The tools coded in for Matlab and Octave are organized in subdirectories of the directory `Matlab/` according to their category. The examples are contained in the sub-directory `examples/` for each category. To use them, one needs to add the path to the tool directory using `addpath`.

# Chapter 4

# Documentation of Tools

## 4.1 Diagnostic Tools

### 4.1.1 `sangoma_ComputeHistogram` — Increment rank histogram (Source File: sangoma_ComputeHistogram.F90)

INTERFACE:

```
SUBROUTINE sangoma_ComputeHistogram(ncall, dim, dim_ens, &
     element, state, ens, hist, delta, status) &
     BIND(C, name="sangoma_computehistogram_")
```

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. Also, the value 'ncall' has to be set. It gives the number of calls used to increment the histogram and is needed to compute the delta-measure that described the deviation from the ideal histogram.

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
2013-11 - L. Nerger - Adaption to SANGOMA data model
2014-02 - L. Nerger - Addition of delta measure
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC

IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in) :: ncall     ! Number of calls to routine
INTEGER(INTPREC), INTENT(in) :: dim       ! State dimension
INTEGER(INTPREC), INTENT(in) :: dim_ens   ! Ensemble size
INTEGER(INTPREC), INTENT(in) :: element   ! Element of vector used for histogram
                                          ! If element=0, all elements are used
REAL(REALPREC), INTENT(in)   :: state(dim)         ! State vector
REAL(REALPREC), INTENT(in)   :: ens(dim, dim_ens)  ! State ensemble
INTEGER(INTPREC), INTENT(inout) :: hist(dim_ens+1) ! Histogram about the state
REAL(REALPREC), INTENT(out)     :: delta           ! value ofr deviation from ideal
INTEGER(INTPREC), INTENT(out)   :: status          ! Status flag (0=success)
```

*COMMENT:*

For an example on using `sangoma_ComputeHistogram` see `example_ComputeHistogram`.

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

INTERFACE:

```
SUBROUTINE sangoma_ComputeEnsStats(dim, dim_ens, element, &
     state, ens, skewness, kurtosis, status) &
     BIND(C, name="sangoma_computeensstats_")
```

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
2013-11 - L. Nerger - Adaption to SANGOMA data model
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC

IMPLICIT NONE
```

*ARGUMENTS:*

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

*COMMENT:*

For an example on using `sangoma_ComputeEnsStats` see `example_ComputeEnsStats`.

### 4.1.3 `sangoma_computeRCRV` — Compute the bias and the dispersion of the RCRV (Source File: sangoma_computeRCRV.F90)

INTERFACE:

```
SUBROUTINE sangoma_computeRCRV(ens, ana, sig0, missing, m, nens, b, d) &
     BIND(C, name="sangoma_computercrv_")
```

DESCRIPTION:

Mean and Standard Deviation of the variable: $y = \frac{(o-m)}{\sqrt{(\sigma^2+\sigma_o^2)}}$
with $o$: observations, $m$: ensemble mean, $\sigma$: ensemble spread, $\sigma_o$: observation error

Check the indistinguishability between the observations and the ensemble members : reliability.

Mean of $y$: `b` = weighted bias of the ensembles
Standard deviation of $y$: `d` = agreement between the mean error and the ensemble spread

A perfectly reliable system gives : b = 0 and d = 1
(see detailed description in deliverable 4.1)

REVISION HISTORY:

```
2014-01 - G. Candille - Initial code for SANGOMA
```

*USES:*

```
  USE, INTRINSIC :: ISO_C_BINDING
  USE sangoma_base, ONLY: REALPREC, INTPREC
  IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in)  :: m           ! Size of the verification set
INTEGER(INTPREC), INTENT(in)  :: nens        ! Ensemble size
REAL(REALPREC), INTENT(in)    :: ens(m,nens) ! Ensemble
REAL(REALPREC), INTENT(in)    :: ana(m)      ! Verification (analysis or observation)
REAL(REALPREC), INTENT(in)    :: sig0        ! observation error
INTEGER(INTPREC), INTENT(in)  :: missing(m)  ! 0 = obs is OK ; 1 = obs is not used
REAL(REALPREC), INTENT(inout) :: b           ! mean of the RCRV (y)
REAL(REALPREC), INTENT(inout) :: d           ! std of the RCRV (y)
```

*COMMENT:*
For an example on using `sangoma_computeRCRV` see `example_ComputeRCRV`.

### 4.1.4 `sangoma_computeCRPS` — Compute the CRPS and its decomposition (Source File: sangoma_computeCRPS.F90)

INTERFACE:

```
  SUBROUTINE sangoma_computecrps(ENS, ANA, MISSING, NCASES, NENS, &
     CRPS, RELI, RESOL, UNCERT, BB, AA, CB_SORT, CB_SORT2) &
     BIND(C, name="sangoma_computecrps_")
```

DESCRIPTION:

Computation of the Continuous Ranked Probability Score and its decomposition:
CRPS = RELI + RESOL
See H. Hersbach (2000) Wea. Forecasting, Vol 15, pp 559-570
(note: RESOL here is equivalent to CRPS_pot = UNCERT - RESOL in Hersbach 2000)

CRPS : global score evaluating both reliability (RELI) and resolution (RESOL)
A perfectly reliable system gives RELI = 0
An informative system gives RESOL $\ll$ UNCERT
(UNCERT is the value of the score based on the verification sample only)
(see detailed description in deliverable 4.1)

REVISION HISTORY:

```
   2014-01 - G. Candille - Initial code for SANGOMA
```

*USES:*

```
  USE, INTRINSIC :: ISO_C_BINDING
  USE sangoma_base, ONLY: REALPREC, INTPREC
  IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in)  :: NCASES      ! Size of the verification set
INTEGER(INTPREC), INTENT(in)  :: NENS        ! Ensemble size
REAL(REALPREC), INTENT(in)    :: ENS(NCASES,NENS) ! Ensemble
REAL(REALPREC), INTENT(in)    :: ANA(NCASES) ! Verification (analysis or observation)
INTEGER(INTPREC), INTENT(in)  :: missing(m)  ! 0 = obs is OK ; 1 = obs is not used
REAL(REALPREC), INTENT(inout) :: CRPS        ! CRPS (global score)
REAL(REALPREC), INTENT(inout) :: RELI        ! reliability part
REAL(REALPREC), INTENT(inout) :: RESOL       ! resolution part
REAL(REALPREC), INTENT(inout) :: UNCERT      ! uncertainty
REAL(REALPREC), INTENT(inout) :: BB(0:NENS)  ! decomposition coefficients of the CRPS
REAL(REALPREC), INTENT(inout) :: AA(0:NENS)  ! decomposition coefficients of the CRPS
```

*CALL-BACK ROUTINES:*

```
! Sorting routine for one vector
INTERFACE
   SUBROUTINE CB_SORT(NENS, V) BIND(C)
     USE sangoma_base, ONLY: REALPREC, INTPREC
     INTEGER(INTPREC),INTENT(in)   :: NENS    ! Size of vector
     REAL(REALPREC),INTENT(inout) :: V(NENS) ! Input/output vector
   END SUBROUTINE CB_SORT
END INTERFACE

! Sorting routine for two vectors
INTERFACE
   SUBROUTINE CB_SORT2(NENS, V1, V2) BIND(C)
     USE sangoma_base, ONLY: REALPREC, INTPREC
     INTEGER(INTPREC),INTENT(in)   :: NENS     ! Size of vector
     REAL(REALPREC),INTENT(inout) :: V1(NENS) ! Input/output vector
     REAL(REALPREC),INTENT(inout) :: V2(NENS) ! 2nd Input/output vector
   END SUBROUTINE CB_SORT2
END INTERFACE
```

*COMMENT:*

For an example on using `sangoma_computeBRIER` see `example_ComputeBRIER`.

### 4.1.5 `sangoma_computeBRIER` — Compute the Brier skill score and its decompostion, and the entropy (Source File: sangoma_computeBRIER.F90)

INTERFACE:

```
  SUBROUTINE sangoma_computeBRIER(m, nens, ens, ana, xth, &
     br, brc, brv, unc, pc, s, sunc, pp, g, pr) &
     BIND(C, name="sangoma_computebrier_")
```

DESCRIPTION:

This subroutine computes scores related to 1 binary event associated
with 1 threshold xth (externally defined by the user).
First, 'predicted' probabilities of occurrence of the event are computed
with their distribution and their associated 'predictable' probabilities
(these can be used to draw reliability and sharpness diagrams).
Then, Brier (skill) score and partition, and entropy, are computed:
$B = E[(p - p'])^2] - E[(p' - pc])^2] + pc(1 - pc)$ and $BS = 1 - B/pc(1 - pc)$
$S = -E[p'log(p')]$
(see details in deliverable 4.1).

REVISION HISTORY:

```
2014-01 - G. Candille - Initial code for SANGOMA
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC
IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in)  :: m             ! Size of the verification set
INTEGER(INTPREC), INTENT(in)  :: nens          ! Ensemble size
REAL(REALPREC), INTENT(in)    :: ens(m,nens)   ! Ensemble
REAL(REALPREC), INTENT(in)    :: ana(m)        ! Verification (analysis or observation)
REAL(REALPREC), INTENT(in)    :: xth           ! threshold
REAL(REALPREC), INTENT(inout) :: br            ! Brier global skill score
REAL(REALPREC), INTENT(inout) :: brc           ! reliability component
REAL(REALPREC), INTENT(inout) :: brv           ! resolution component
REAL(REALPREC), INTENT(inout) :: unc           ! uncertainty
REAL(REALPREC), INTENT(inout) :: pc            ! 'climatological' probability
REAL(REALPREC), INTENT(inout) :: s             ! entropy
REAL(REALPREC), INTENT(inout) :: sunc          ! 'climatological' entropy
REAL(REALPREC), INTENT(inout) :: pp(nens+1)    ! predicted probabilities
REAL(REALPREC), INTENT(inout) :: g(nens+1)     ! distribution of pp
REAL(REALPREC), INTENT(inout) :: pr(nens+1)    ! predictable probabilities
```

*COMMENT:*
For an example on using `sangoma_computeBRIER` see `example_ComputeBRIER`.

### 4.1.6 `sangoma_ComputeSensitivity` — Calculate the sensitivity matrix (Source File: sangoma_ComputeSensitivity.F90)

INTERFACE:

```
SUBROUTINE sangoma_computesensitivity(dim, dim_ens,dim_obs, &
     obs_cov, H, ens, post_w,sensitivity, post_cov_mat, status) &
     BIND(C, name="sangoma_computesensitivity_")
```

DESCRIPTION:

Calculates sensitivity of the posterior mean to the observations (assuming Gaussian observation error and linear observation operator) within a particle filter. The observation operator is given as an input matrix.

To be used after weights have been updated by the observations.

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. See Fowler and van Leeuwen (Tellus 64A (2012) 17192).

This tool uses the libraries `BLAS` and `LAPACK`.

REVISION HISTORY:

```
2014-01 - P. Kirchgessner - Initial Fortran code; based on the
                            Matlab script by Alison Fowler
```

*USES:*

```
  USE, INTRINSIC :: ISO_C_BINDING
  USE sangoma_base, ONLY: REALPREC, INTPREC
  IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in) :: dim                  ! state dimension
INTEGER(INTPREC), INTENT(in) :: dim_ens              ! Ensemble size
INTEGER(INTPREC), INTENT(in) :: dim_obs              ! Number of observations
REAL(REALPREC), INTENT(in)   :: obs_cov(dim_obs,dim_obs) ! Cov. matrix of observations
REAL(REALPREC), INTENT(in)   :: H(dim_obs, dim)      ! Linear observation operator
REAL(REALPREC), INTENT(in)   ::  ens(dim, dim_ens)! ensemble of particles
REAL(REALPREC), INTENT(in)   :: post_w(dim_ens)      ! Posterior weights
                              ! If no weights are given, initialize with 1/dim_ens
REAL(REALPREC), INTENT(out)  :: sensitivity(dim_obs,dim_obs)   ! Sensitivity matrix
REAL(REALPREC), INTENT(out)  :: post_cov_mat(dim,dim) ! Posterior covariance matrix
INTEGER(INTPREC), INTENT(out) :: status              ! Status flag (0=success)
```

*COMMENT:*

For an example on using `sangoma_ComputeSensitivity` see `example_ComputeSensitivity`.

### 4.1.7 `sangoma_ComputeSensitivity_op` — Calculate the sensitivity matrix with H as operator (Source File: sangoma_Compute-Sensitivity_op.F90)

INTERFACE:

```
SUBROUTINE sangoma_computesensitivity_op(dim, dim_ens, dim_obs, &
    obs_cov, ens, post_w, sensitivity, post_cov_mat, CB_obs_op, status) &
    BIND(C, name="sangoma_computesensitivity_op_")
```

DESCRIPTION:

Calculates sensitivity of the posterior mean to the observations (assuming Gaussian observation error and linear observation operator) within a particle filter. The observation operator is given in operator form as a call-back routine.

To be used after weights have been updated by the observations.

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. See Fowler and van Leeuwen (Tellus 64A (2012) 17192).

This tool uses the libraries `BLAS` and `LAPACK`.

REVISION HISTORY:

```
2014-01 - P. Kirchgessner - Initial Code based on sangoma_compute_sensitivity
                            adapted for observation operator as callback routine
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC
IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in) :: dim                ! State dimension
INTEGER(INTPREC), INTENT(in) :: dim_ens            ! Ensemble size
INTEGER(INTPREC), INTENT(in) :: dim_obs            ! Number of observations
REAL(REALPREC), INTENT(in)   :: obs_cov(dim_obs,dim_obs)
                                                   ! Covariance matrix of observations
REAL(REALPREC), INTENT(in)   ::  ens(dim, dim_ens)! ensemble of particles
REAL(REALPREC), INTENT(in)   :: post_w(dim_ens)    ! Posterior weights
                          ! If no weights are given, initialize with 1/dim_ens
REAL(REALPREC), INTENT(out)  :: sensitivity(dim_obs,dim_obs)  !
REAL(REALPREC), INTENT(out)  :: post_cov_mat(dim,dim) ! Posterior covariance matrix
INTEGER(INTPREC), INTENT(out) :: status            ! Status flag (0=success)
```

*CALL-BACK ROUTINE:*

```
! Call-back routine provinding observation operator
INTERFACE
   SUBROUTINE CB_obs_op(step, dim, dim_obs, state, Hstate) BIND(C)
      USE sangoma_base, ONLY: REALPREC, INTPREC
      INTEGER(INTPREC),INTENT(in) :: step            ! Time step
      INTEGER(INTPREC),INTENT(in) :: dim             ! State dimension
      INTEGER(INTPREC),INTENT(in) :: dim_obs         ! Number of observations
      REAL(REALPREC),INTENT(in)  :: state(dim)       ! State ensemble
```

```
        REAL(REALPREC),INTENT(out) :: Hstate(dim_obs) ! Observed ensemble
      END SUBROUTINE
    END INTERFACE
```

*COMMENT:*

For an example on using `sangoma_ComputeSensitivity_op` see `example_ComputeSensitivity`.

### 4.1.8 `sangoma_ComputeRE` — Compute the Relative Entropy (Source File: sangoma_Compute-RE.F90)

INTERFACE:

```
 SUBROUTINE sangoma_computeRE(dim_ens,post_w,prior_w,RE, status) &
      BIND(C, name="sangoma_computere_")
```

DESCRIPTION:

Calculates relative entropy within a particle filter. To be used after weights have been updated by the observations.
OUTPUT: RE - scalar, relative entropy of the posterior given the prior.
METHOD: RE is given by $\int(p(x|y)ln[p(x|y)/p(x)])dx$. 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. $RE \approx \sum(post_w ln(post_w/prior_w))$

REVISION HISTORY:

```
 2014-03 - P. Kirchgessner - Initial Fortran code; based on
                             the Matlab routine by Alison Fowler
```

*USES:*

```
  USE, INTRINSIC :: ISO_C_BINDING
  USE sangoma_base, ONLY: REALPREC, INTPREC
  IMPLICIT NONE
```

*ARGUMENTS:*

```
  INTEGER(INTPREC), INTENT(in) :: dim_ens           ! Ensemble size
  REAL(REALPREC), INTENT(in)   :: post_w(dim_ens)   ! Posterior weights
  REAL(REALPREC), INTENT(inout)   :: prior_w(dim_ens) ! Prior weights
  REAL(REALPREC), INTENT(out)  :: RE                ! Relative Entropy
  INTEGER(INTPREC), INTENT(out) :: status           ! Status flag (0=success)
```

*COMMENT:*

For an example on using `sangoma_ComputeRE` see `example_ComputeRE`.

### 4.1.9 sangoma_computeMutInf — Calculate the Mutual Information (Source File: sangoma_ComputeMutInf.F90)

INTERFACE:

```
sangoma_computeMutInf(dim, dim_ens, dim_obs, dim_sample, R, &
     observations, ens, prior_w, CB_obs_op, MI, status) &
     BIND(C, name="sangoma_computemutinf_")
```

DESCRIPTION:

Calculates sensitivity of the posterior mean to the observations (assuming Gaussian observation error and linear observation operator) within a particle filter. The observation operator has to be provided as an operator in order to use the method. To be used after weights have been updated by the observations. 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. See Fowler and van Leeuwen, 2012.

This tool uses the libraries BLAS and LAPACK.

REVISION HISTORY:

```
2014-01 - P. Kirchgessner - Initial Fortran code; based on the Matlab
                            script by from Alison Fowler
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC
IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in)  :: dim           ! PE-local state dimension
INTEGER(INTPREC), INTENT(in)  :: dim_ens       ! Ensemble size
INTEGER(INTPREC), INTENT(in)  :: dim_obs       ! Number of observations
INTEGER(INTPREC), INTENT(in)  :: dim_sample    ! Sample size for obs. space
REAL(REALPREC), INTENT(in)    :: R(dim_obs,dim_obs)     ! Observation covariance
REAL(REALPREC), INTENT(in)    :: observations(dim_obs)  ! Observations
REAL(REALPREC), INTENT(in)    :: ens(dim, dim_ens)      ! ensemble of particles
REAL(REALPREC), INTENT(in)    :: prior_w(dim_ens)       ! prior weights
REAL(REALPREC), INTENT(out)   :: MI            ! Mutual Information
INTEGER(INTPREC), INTENT(out) :: status        ! Status flag (0=success)
```

*CALL-BACK ROUTINE:*

```
! Call-back routine provinding observation operator
INTERFACE
   SUBROUTINE CB_obs_op(step, dim, dim_obs, state, Hstate) BIND(C)
      USE sangoma_base, ONLY: REALPREC, INTPREC
      INTEGER(INTPREC),INTENT(in) :: step           ! Time step
      INTEGER(INTPREC),INTENT(in) :: dim            ! State dimension
      INTEGER(INTPREC),INTENT(in) :: dim_obs        ! Number of observations
      REAL(REALPREC),INTENT(in)   :: state(dim)     ! State ensemble
      REAL(REALPREC),INTENT(out)  :: Hstate(dim_obs) ! Observed ensemble
   END SUBROUTINE
END INTERFACE
```

*COMMENT:*

For an example on using `sangoma_ComputeMutInf` see `example_ComputeMutInf`.

### 4.1.10 `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:

$$MI = \int (RE * p(y))\, dy \tag{4.1}$$

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.

$$MI = \sum_{i=1}^{M} (RE_i * p(y)_i) * dy. \tag{4.2}$$

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

$$MI = \sum_{i} (RE_i * p(y)_i) \tag{4.3}$$

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.

### 4.1.11 `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

$$\int p(x|y) \ln \left[ \frac{p(x|y)}{p(x)} \right] dx \; . \tag{4.4}$$

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:

$$RE \approx \sum post_w \ln \left( \frac{post\_w}{prior\_w} \right) \tag{4.5}$$

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.

### 4.1.12 `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.

### 4.1.13 `ArM` — Calculate array modes and associated quantities (Source File: sangoma_arm.F90)

INTERFACE:

```
SUBROUTINE sangoma_arm( &
 nstate, nens, nobs, ndof, Af, Df, R, arm_spect, arm, arm_rep, status &
 ) BIND( C, name="sangoma_arm_" )
```

DESCRIPTION:

This routine calculates array modes, the associated spectrum and their representers in state space (modal representers). The problem is scaled in such a way that the observational noise floor is 1, so eigenvalues above 1 in the spectrum denote d.o.f.s of model error detectable above that noise floor.

Input ensemble samples Af and Df are assumed to represent model uncertainties in state-space and data-space, respectively.

Df can be directly generated by the model, or linearly calculated as H*Af, H(nobs,nstate) being the observation operator.

State space and data space are n-dimensional + (optionally) time. Therefore each state-space sample and data-space sample can contain information from several instants if desired. Similarly, modal representers can span space *and* time.

REVISION HISTORY:

```
2011-06: P De Mey arm.f Bologna Summer School 2011
2013-06: P De Mey arm.f Bologna Summer School 2013
2014-03: P De Mey sangoma_arm.F90 SANGOMA release
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC

IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in)  :: nstate          ! state size
INTEGER(INTPREC), INTENT(in)  :: nens            ! Ensemble size
INTEGER(INTPREC), INTENT(in)  :: nobs            ! number of observations
INTEGER(INTPREC), INTENT(in)  :: ndof      ! =MIN(nens,nobs) number of d.o.f.s of problem
REAL(REALPREC), INTENT(in)    :: Af(nstate,nens) ! forecast ensemble anomalies (centered)
REAL(REALPREC), INTENT(in)    :: Df(nobs,nens)   ! same as Af in data space
REAL(REALPREC), INTENT(in)    :: R(nobs,nobs)    ! observation error covariance matrix
REAL(REALPREC), INTENT(out)   :: arm_spect(ndof) ! array mode spectrum
REAL(REALPREC), INTENT(out)   :: arm(nobs,ndof)       ! array modes
REAL(REALPREC), INTENT(out)   :: arm_rep(nstate,ndof) ! modal representers
INTEGER(INTPREC), INTENT(out) :: status               ! Status flag (0=success)
```

*COMMENT:*

For an example on using `sangoma_arm` see `example_arm`.

## 4.2 Perturbation Tools

### 4.2.1 `sangoma_pseudornd2D` — Generate correlated random field (Source File: mod_sangoma_pseudornd.F90)

INTERFACE:

```
sangoma_pseudornd2D(Amat, nx, ny, nens, rh, n1, n2) &
    BIND(C, name="sangoma_pseudornd2d_")
```

DESCRIPTION:

This routine calculates pseudo random 2D fields with presecribed correlation length using the spectral procedure outlined in Evensen (1994).
Include the module with 'use mod_sangoma_pseudornd'

REVISION HISTORY:

```
2014-02 - L. Bertino - Submission for SANGOMA
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC
IMPLICIT NONE
```

*ARGUMENTS:*

```
integer(INTPREC), intent(in) :: nx, ny   ! Horizontal dimensions
integer(INTPREC), intent(in) :: nens     ! Number of fields stored at the time
real(REALPREC), intent(out)  :: Amat(nx,ny,nens) ! Generated random fields
real(REALPREC), intent(in)   :: rh       ! Horizontal decorrelation length
                                         ! in number of horizontal grid cells
integer(INTPREC), intent(in) :: n1, n2   ! horizontal dimensions in fft grid
```

*NOTES:*

1. The tool is provided as a Fortran module. Thus, in Fortran programs, one has to include it with `use mod_sangoma_pseudornd`.
2. The tool needs the FFTW library for the Fourier transformation.

*COMMENT:*

For an example on using `sangoma_pseudornd2D` see `example_pseudornd2D`.

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

INTERFACE:

```
 SUBROUTINE sangoma_MVNormalize(mode, dim_state, dim_field, offset, &
      ncol, states, stddev, status)) &
     BIND(C, name="sangoma_mvnormalize_")
```

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
2013-11 - L. Nerger - Adaption to SANGOMA data model
```

*USES:*

```
IMPLICIT NONE
```

*ARGUMENTS:*

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

*COMMENT:*

For an example on using `sangoma_MVNormalize` see `example_MVNormalize`.

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

INTERFACE:

```
SUBROUTINE sangoma_eofcovar(dim_state, nstates, nfields, dim_fields, &
     offsets, remove_mstate, do_mv, states, stddev, svals, svec, &
     meanstate, status) BIND(C, name="sangoma_eofcovar_")
```

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' holding 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
2013-11 - L. Nerger - Adaption to SANGOMA data model
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC

IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in) :: dim_state        ! Dimension of state vector
INTEGER(INTPREC), INTENT(in) :: nstates          ! Number of state vectors
INTEGER(INTPREC), INTENT(in) :: nfields          ! Number of fields in state vector
INTEGER(INTPREC), INTENT(in) :: dim_fields(nfields) ! Size of each field
INTEGER(INTPREC), INTENT(in) :: offsets(nfields) ! Start position of each field
INTEGER(INTPREC), INTENT(in) :: do_mv            ! 1: Do multivariate scaling
   ! nfields, dim_fields and offsets are only used if do_mv=1
INTEGER(INTPREC), INTENT(in) :: remove_mstate    ! 1: subtract mean state from
                                                 !    states before computing EOFs
REAL(REALPREC), INTENT(inout)  :: states(dim_state, nstates) ! State perturbations
REAL(REALPREC), INTENT(out) :: stddev(nfields)   ! Standard deviation of field
   ! Without multivariate scaling (do_mv=0), it is stddev = 1.0
REAL(REALPREC), INTENT(out) :: svals(nstates)    ! Singular values
```

```
                                                   ! divided by sqrt(nstates-1)
  REAL(REALPREC), INTENT(out) :: svec(dim_state, nstates)   ! Singular vectors
  REAL(REALPREC), INTENT(inout) :: meanstate(dim_state)     ! Mean state
                                                   ! (only changed if remove_mstate=1)
  INTEGER(INTPREC), INTENT(out) :: status          ! Status flag (0=success)
```

*COMMENT:*
For an example on using `sangoma_ComputeEOFCovar` see `example_ComputeEOFCovar`.

### 4.2.4 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 with the following attributes:

- **mask** cell array of land-sea mask for the different variables
- **numels** vector of integers. The ith element is the number of sea grid points in mask i.
- **numels_all** vector of integers. The ith element is the number of all grid points in mask i.
- **size** cell array. The ith element is the size of the ith mask.
- **ind** start index of the ith variable in the packed vector.
- **n** total number of sea grid points

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.

## 4.3 Transformation Tools

### 4.3.1 `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. The structure has the field "method" (same as input parameter) and the field "x" and "y" describing the piecewise transformation function.

### 4.3.2 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.

### 4.3.3 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.

## 4.4 Utilities

### 4.4.1 `sangoma_hfradar_extractf` — Observation operator for HF radar surface currents

INTERFACE:

```
 [velf,velg] = sangoma_hfradar_extractf( ...
                   lon_u,lat_u,z_u,u,...
                   lon_v,lat_v,z_v,v,...
                   nu,res,lon0,lat0, ...
                   lon,lat, ...
                   long,latg)
```

DESCRIPTION:

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

REVISION HISTORY:

```
2012-10 - Alexander Barth - Initial code for SANGOMA
2014-02 - Alexander Barth - Adaption to SANGOMA data model
```

INPUT PARAMETERS:

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

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

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

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

**nu:** the frequency of the HF radar system in Hz.

**res:** the effective azimuthal resolution in degrees.

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

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

**long, latg:** Cartesian grid of the HF radar data. The first dimension is longitude, and the second is the latitude.

OPTIONAL INPUT PARAMETERS:

**long, latg:** Cartesian grid of the HF radar data. The first dimension is longitude, and the second is the latitude.

OUTPUT PARAMETERS:

**velf:** radial velocity on a radial grid.

OPTIONAL OUTPUT PARAMETERS:

**velg:** radial velocity on a Cartesian 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

### 4.4.2 `sangoma_computepod` — Compute dominant POD modes from an ensemble of snapshots. (Source File: sangoma_computepod.F90)

INTERFACE:

```
SUBROUTINE sangoma_computepod(nmodes, dim_state, nstates, nfields, &
     dim_fields, offsets, do_mv, states, stddev, svals, svec, status) &
     BIND(C, name="sangoma_computepod_")
```

DESCRIPTION:

This routine performs an EOF analysis by singular value decomposition. It is used to prepare a dominant eigen vectors and eigenvalues of an ensemble.
Outputs are the leading eigen values (svals) and eigen vectors (svec) based on specified energy.
The routine uses the LAPACK routine 'dgesvd' to compute the singular value decomposition.

REVISION HISTORY:

```
Adopted from sangoma_EOFCovar
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC

IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in) :: nmodes              ! No of leading eigen vectors
INTEGER(INTPREC), INTENT(in) :: dim_state           ! Dimension of state vector
INTEGER(INTPREC), INTENT(in) :: nstates             ! Number of state vectors
INTEGER(INTPREC), INTENT(in) :: nfields             ! Number of fields in state vector
INTEGER(INTPREC), INTENT(in) :: dim_fields(nfields) ! Size of each field
INTEGER(INTPREC), INTENT(in) :: offsets(nfields)    ! Start position of each field
INTEGER(INTPREC), INTENT(in) :: do_mv               ! 1: Do multivariate scaling
   ! nfields, dim_fields and offsets are only used if do_mv=1
REAL(REALPREC), INTENT(in)  :: states(dim_state, nstates) ! State perturbations
REAL(REALPREC), INTENT(out) :: stddev(nfields)      ! Standard deviation of field variability
   ! Without multivariate scaling (do_mv=0), it is stddev = 1.0
REAL(REALPREC), INTENT(out) :: svals(nmodes)       ! Eigen values
REAL(REALPREC), INTENT(out) :: svec(dim_state, nmodes)   ! Eigen Vectors
INTEGER(INTPREC), INTENT(out) :: status             ! Status flag
```

*COMMENT:*

For an example on using `sangoma_computepod` see `example_PODcostgrad`.

### 4.4.3 `sangoma_costgrad` — Compute the values of Objective function and Gradient using reduced state dimensions. (Source File: sangoma_costgrad.F90)

INTERFACE:

```
SUBROUTINE sangoma_costgrad(nmodes, nparam, nsteps, dim_state, &
    nobs, nanalysis, Mx, Malpha, alpha, obs, obsstd, observer, &
    cost, grad, analysis, status) BIND(C, name="sangoma_costgrad_")
```

DESCRIPTION:

This routine computes the objective function and gradient vector using a reduced order model. The reduced order model is obtained by projecting dominant eigen modes in the dynamic operators $M$ and $M\_alpha$ for state and parameters respectively.

To use this routine, one has to generate dominant eigen modes from an ensemble, e.g using `sangoma_computepod` and construct reduced model operators for states and parameters, respectively.

REVISION HISTORY:

```
Adopted from reducedgradient and reducedcost of OpenDA.
```

*USES:*

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC

IMPLICIT NONE
```

*ARGUMENTS:*

```
INTEGER(INTPREC), INTENT(in) :: nmodes     ! Dimension of the leading eigenvector
INTEGER(INTPREC), INTENT(in) :: nparam     ! Dimension of parameters
INTEGER(INTPREC), INTENT(in) :: nsteps     ! Number of time steps
INTEGER(INTPREC), INTENT(in) :: dim_state  ! Dimension of state vector
INTEGER(INTPREC), INTENT(in) :: nobs       ! Number of observations (fixed)
INTEGER(INTPREC), INTENT(in) :: nanalysis  ! Number of analysis steps
REAL(REALPREC), INTENT(in) :: Mx(nsteps, nmodes, nmodes)     ! Reduced state operator
REAL(REALPREC), INTENT(in) :: Malpha(nsteps, nmodes, nparam) ! Reduced param. operator
REAL(REALPREC), INTENT(in) :: alpha(nparam)            ! Initial parameters
REAL(REALPREC), INTENT(in) :: obs(nanalysis, nobs)     ! Observations
REAL(REALPREC), INTENT(in) :: obsstd(nanalysis, nobs)  ! Observation std. deviation
REAL(REALPREC), INTENT(in) :: observer(nobs, nmodes)   ! Observation operator
REAL(REALPREC), INTENT(out)  :: cost       ! Value of the objective function
REAL(REALPREC), INTENT(out)  :: grad(nparam)           ! Gradient vector
INTEGER(INTPREC), INTENT(in) :: analysis(nanalysis)    ! analysis steps
INTEGER(INTPREC), INTENT(out) :: status                ! Status flag
```

*COMMENT:*

For an example on using `sangoma_costgrad` see `example_PODcostgrad`.