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

Deliverable 2.5:
Software V2 report
Due date: 31/10/2015
Delivery date: 9/11/2015
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

1 Introduction **6**

2 Overview of Tools **8**
2.1 Diagnostic Tools 8
2.2 Perturbation Tools 9
2.3 Transformation Tools 9
2.4 Utilities 10
2.5 Analysis 10

3 Using the tools **12**
3.1 Directory structure of the release 12
3.2 Fortran-tools 13
3.3 Matlab-tools 13

4 Documentation of Fortran-Tools **14**
4.1 Diagnostic Tools 14
4.1.1 `sangoma_CheckNormality` – Anderson-Darling Test (Source File: sangoma_CheckNormality.F90) 14
4.1.2 `sangoma_CheckWhiteness` — Check whiteness of innovations (Source File: sangoma_CheckWhiteness.F90) 15
4.1.3 `sangoma_CompareObsDiag` — Compare observation-space diagnostics (Source File: sangoma_CompareObsDiag.F90) 15
4.1.4 `sangoma_ComputeBRIER` — Compute Brier score & its decomposition & the entropy (Source File: sangoma_ComputeBRIER.F90) 17
4.1.5 `sangoma_computeCRIGN` — Computes the CRPS and CRIGN scores (Source File: sangoma_ComputeCRIGN.F90) 18
4.1.6 `sangoma_ComputeCRPS` — Compute the CRPS and its decomposition (Source File: sangoma_ComputeCRPS.F90) 19
4.1.7 `sangoma_CheckEnsSpread` — Check whether ensemble spread is realistic (Source File: sangoma_CheckEnsSpread.F90) 20
4.1.8 `sangoma_ComputeEnsStats` — Compute ensemble statistics (Source File: sangoma_ComputeEnsStats.F90) 21
4.1.9 `sangoma_ComputeEffSample` — Compute effective sample size (Source File: sangoma_ComputeEffSample.F90) 22
4.1.10 `sangoma_ComputeHistogram` — Increment rank histogram (Source File: sangoma_ComputeHistogram.F90) 23
4.1.11 `sangoma_ComputeInvStats` —- Computes the innnovation statistics 24

4.1.12 sangoma_ComputeMutInf — Calculate the Mutual Information (Source File: sangoma_ComputeMutInf.F90) . . . 25
4.1.13 sangoma_ComputeRCRV — Compute the bias and the dispersion of the RCRV (Source File: sangoma_ComputeRCRV.F90) . . . 26
4.1.14 sangoma_ComputeRE — Compute the Relative Entropy (Source File: sangoma_Compute-RE.F90) . . . 27
4.1.15 sangoma_ComputeSMatrix - Computes scaled ensemble observation anomalies matrix . . . 28
4.1.16 sangoma_ComputeSensitivity — Calculate the sensitivity matrix (Source File: sangoma_ComputeSensitivity.F90) . . . 28
4.1.17 sangoma_ComputeSensitivity_op — Calculate sensitivity matrix with H as operator (Source File: sangoma_ComputeSensitivity_op.F90) . . . 29
4.1.18 sangoma_ArM — Calculate array modes and associated quantities (Source File: sangoma_arm.F90) . . . 31
4.1.19 sangoma_ArM-CA — Calculate array modes and use them to verify the consistency of a forecast ensemble given an ensemble of innovations (Source File: sangoma_armca.F90) . . . 32
4.1.20 sangoma_ObsDiag — Compute sampled observation-space statistics (Source File: sangoma_ObsDiag.F90) . . . 33
4.2 Perturbation Tools . . . 34
4.2.1 sangoma_pseudornd2D — Generate correlated random field (Source File: sangoma_pseudornd.F90) . . . 34
4.2.2 sangoma_MVNormalize — Perform multivariate normalization (Source File: sangoma_MVNormalize.F90) . . . 35
4.2.3 sangoma_EOFCovar — Perform EOF decomposition of state trajectory (Source File: sangoma_EOFCovar.F90) . . . 36
4.2.4 sangoma_SampleEns — Generate ensemble by 2nd order exact sampling of EOF modes (Source File: sangoma_SampleEns.F90) 37
4.3 Transformation Tools . . . 38
4.3.1 sangoma_Anamorphosis — Computes local Gaussian anamorphosis . . . 38
4.3.2 sangoma_ComputeQuantiles — Computes ensemble quantiles as input for anamorphosis . . . 39
4.4 Utilities . . . 40
4.4.1 sangoma_computepod — Compute dominant POD modes from an ensemble of snapshots. (Source File: sangoma_computepod.F90) 40
4.4.2 sangoma_costgrad — Compute the values of Objective function and Gradient using reduced state dimensions. (Source File: sangoma_costgrad.F90) . . . 41
4.5 Analysis Steps . . . 42
4.5.1 sangoma_ens_analysis – Computes the analysis ensemble using the ETKF scheme (Source File: mod_sangoma_ensemble_analysis.F90) 42
4.5.2 sangoma_local_ens_analysis – Computes the local analysis ensemble using the ETKF scheme (Source File: mod_sangoma_ensemble_analysis.F90) . . . 44
4.5.3 sangoma_enkf_analysis – EnKF Analysis Step (Source File: sangoma_enkf_analysis.F90) . . . 45

4.5.4 sangoma_ensrf_analysis – EnSRF Serial Analysis Step (Source File: sangoma_ensrf_analysis.F90) . . . 46
4.5.5 sangoma_estkf_analysis – ESTKF Analysis Step (Source File: sangoma_estkf_analysis.F90) . . . 47
4.5.6 sangoma_etkf_analysis – ETKF Analysis Step (Source File: sangoma_etkf_analysis.F90) . . . 49
4.5.7 sangoma_lestkf_analysis – LESTKF Analysis Step (Source File: sangoma_lestkf_analysis.F90) . . . 50
4.5.8 sangoma_letkf_analysis – LETKF Analysis Step (Source File: sangoma_letkf_analysis.F90) . . . 52
4.5.9 sangoma_netf_analysis – NETF Analysis Step (Source File: sangoma_etkf_analysis.F90) . . . 53

5 Documentation of Matlab-Tools **55**
5.1 Diagnostic Tools . . . 55
5.1.1 computeBRIER — Compute the Brier skill score and its decomposition, and the entropy (Source File: computeBRIER.m) . . . 55
5.1.2 computeCRPS — Compute the CRPS and its decomposition (Source File: computeCRPS.m) . . . 56
5.1.3 computehistogram — Increment rank histogram (Source File: computehistogram.m) . . . 57
5.1.4 computeRCRV — Compute the bias and the dispersion of the RCRV (Source File: computeRCRV.m) . . . 57
5.1.5 mutual_information (Source file: mutual_information.m) . . . 58
5.1.6 relative_entropy (Source file: relative_entropy.m) . . . 60
5.1.7 sensitivity (Source file: sensitivity.m) . . . 60
5.2 Perturbation Tools . . . 61
5.2.1 Weakly constrained ensemble perturbations . . . 61
5.3 Transformation Tools . . . 66
5.3.1 anam_setup — Empirical Gaussian Anamorphosis (Anam) . . . 66
5.3.2 Function anam_transform . . . 67
5.3.3 Function anam_invtransform . . . 67
5.4 Utilities . . . 68
5.4.1 sangoma_hfradar_extractf — Observation operator for HF radar surface currents . . . 68
5.5 Analysis Steps . . . 69
5.5.1 sangoma_ensemble_analysis – Computes the analysis ensemble using various ensemble schemes (Source File: mod_ensemble_analysis.m) . . . 69
5.5.2 sangoma_local_ensemble_analysis – Computes the local analysis ensemble using various ensemble schemes (Source File: sangoma_local_ensemble_analysis.m) . . . 70
5.5.3 sangoma_local_EnKF – Computes the local analysis ensemble using the EnKF (Source File: sangoma_local_EnKF.m) . . . 71

Chapter 1

Introduction

This document provides an overview of the third and final release of the SANGOMA tools (V2). Within work package 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 different tool boxes for data assimilation.

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.
- **Analysis steps**: Tools to compute the analysis step of an ensemble data assimilation method.

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 following release V1 included tools that are adapted to the standard data model and interface structure. Both, tools from the V0 release as well as additional tools were included. All tools were 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 V1 also included examples on how to use a particular tool.

The final release V2 of the SANGOMA tools extends the collection of tools. The implementation work focused on diagnostic tools. However, it was also decided to add

the analysis step of several variants of ensemble Kalman filters. With this, a user can also compute an analysis step using some ensemble of model states and a set of observation. These analysis steps are simplified compared to those implemented in the different available tools boxes. For example, they don't provide parallelization and dynamic allocation of internal arrays. With these simplifications, the tools are usable for moderately-sized problems (e.g. state dimensions of $O(10^5)$), but for larger systems and for higher efficiency, we recommend to use one of the data assimilation toolboxes that have been described in Deliverable 1.1 of SANGOMA.

The tools release package can be downloaded from the project web site at `http://www.data-assimilation.net/Tools/`. In addition, the collection of tools is available at Source Forge at `http://sourceforge.net/projects/sangoma/` where it will remain available after the end of the SANGOMA project.

Chapter 2

Overview of Tools

2.1 Diagnostic Tools

Fortran

sangoma_CheckEnsSpread	Compute ensemble spread and deviation of ensemble mean from an input state
sangoma_CheckNormality	Anderson-Darling Test to check normality of a sample
sangoma_CheckWhiteness	Check whiteness of innovations
sangoma_CompareObsDiag	Compare observation-space diagnostics
sangoma_ComputeBRIER	Compute the Brier skill score and its decomposition, and the entropy
sangoma_ComputeCRIGN	Compute CRPF and CRIGN scores
sangoma_ComputeCRPS	Compute the CRPS and its decomposition
sangoma_ComputeEffSample	Compute the effective sample size of a particle filter
sangoma_ComputeEnsStats	Compute ensemble statistics
sangoma_ComputeHistogram	Compute ensemble rank histograms
sangoma_ComputeInvStats	Compute innovation statistics
sangoma_ComputeMutInf	Compute the mutual information
sangoma_ComputeRCRV	Compute the bias & the dispersion of the RCRV
sangoma_ComputeRE	Calculate the relative entropy
sangoma_ComputeSMatrix	Compute scaled ensemble observation anomalies
sangoma_ComputeSensitivity	Calculate the sensitivity matrix with **H** as matrix
sangoma_ComputeSensitivity_op	Calculate the sensitivity matrix with **H** as operator
sangoma_arm	Calculate array modes
sangoma_armca	Check the consistency of an ensemble using array modes
sangoma_ObsDiag	Compute sampled observation-space diagnostics

Matlab/Octave

computeBRIER	Compute the Brier skill score and its decomposition, and the entropy
computeCRPS	Compute the CRPS and its decomposition
computeRCRV	Compute the bias & the dispersion of the RCRV
computeHistogram	Compute ensemble rank histograms
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
sangoma_SampleEns	Generate an ensemble by 2nd order exact sampling of EOF modes

Matlab/Octave

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

2.3 Transformation Tools

Fortran

sangoma_Anamorphosis	Computes local Gaussian anamorphosis
sangoma_ComputeQuantiles	Computes ensemble quantiles as input for anamorphosis

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
mod_sangoma_utils	Module of utilities for easy porting from Matlab

Matlab/Octave

hfradar_extractf	Observation operator for HF radar surface currents

2.5 Analysis

Fortran

sangoma_ens_analysis	Computes the analysis ensemble using the ETKF scheme
sangoma_local_ensemble_analysis	Computes the local analysis ensemble using the ETKF scheme
sangoma_enkf_analysis	Compute analysis ensemble using the EnKF with perturbed observations (globally or with covariance localization)
sangoma_ensrf_analysis	Compute analysis ensemble using the EnSRF with serial observation processing (globally or with covariance localization)
sangoma_estkf_analysis	Compute analysis ensemble using the global ESTKF method
sangoma_etkf_analysis	Compute analysis ensemble using the global ETKF method
sangoma_lestkf_analysis	Compute analysis ensemble using the ESTKF method with observation localization
sangoma_letkf_analysis	Compute analysis ensemble using the ETKF method with observation localization
sangoma_netf_analysis	Compute analysis ensemble using the NETF method

Matlab/Octave

sangoma_ensemble_analysis	Computes the analysis ensemble using the EnSRF, EAKF, ETKF, ETKF2, SEIK, ESTKF or EnKF scheme
sangoma_local_ensemble_analysis	Computes the local analysis ensemble using the EnSRF, EAKF, ETKF, ETKF2, SEIK, ESTKF or EnKF scheme (domain localization)
sangoma_local_EnKF	Computes the local analysis ensemble using the EnKF (covariance localization)

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/
        ____ analysis/
                       ___________ examples/
Matlab/
       _____ diagnostics/
                         _________ examples/
       _____ perturbations/
                          _______ examples/
       _____ transformations/
                            _____ examples/
       _____ utilities/
                       ___________ examples/
        ____ analysis/
                       ___________ 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 `sangoma_base.mod`, `mod_sangoma_pseudornd.mod`, `mod_sangoma_utils.mod`, and `mod_sangoma_analysis.mod` will be generated by the build process. The file `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 that provides the functions for the routine generating pseudo random fields following Evensen (1994). `mod_sangoma_utils.mod` provides functions to easy porting from Matlab and `mod_sangoma_analysis.mod` provides helper functions for the analysis steps.

To build the example programs for the tools of a category, one cd's into the corresponding example-directory, 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, in particular files holding callback routines.

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 Fortran-Tools

4.1 Diagnostic Tools

4.1.1 `sangoma_CheckNormality` – Anderson-Darling Test (Source File: sangoma_CheckNormality.F90)

INTERFACE:

```
subroutine sangoma_CheckNormality(n,x,alpha,H,pvalue,adstat)  &
   bind(C, name="sangoma_checknormality_")
```

DESCRIPTION:

Performs an Anderson-Darling Test to test if a sample is from a Gaussian distribution. The null hypothesis H0 is that the sample x follows a Gaussian distribution. If H is 1, then this null hypothesis can be rejected at the signficance level of alpha. It also returns the p-values (probability for a wrong H0 rejection) and the Anderson-Darling test statistic.

USES:

```
use, intrinsic :: iso_c_binding
use sangoma_base, only: realprec, intprec
implicit none
```

ARGUMENTS:

```
integer(intprec), intent(in) :: n       ! number of samples
real(realprec),   intent(in) :: x(n)    ! sample value to check
real(realprec),   intent(in) :: alpha   ! significance level (fraction of 1)
integer(intprec), intent(out) :: H      ! 1 if we reject H0 (otherwise 0)
real(realprec),   intent(out) :: pvalue ! p-value for the test
real(realprec),   intent(out) :: adstat ! Anderson-Darling test statistic
```

COMMENT:

For an example on using this function see `example_CheckNormality.F90`.

4.1.2 sangoma_CheckWhiteness — Check whiteness of innovations (Source File: sangoma_CheckWhiteness.F90)

INTERFACE:

```
SUBROUTINE sangoma_checkwhiteness(dim, dimt, inno, Qval, pval) &
    BIND(C, name="sangoma_checkwhiteness_")
```

DESCRIPTION:

This routines checks whether the a set of innovations is white in time. The innovations are stored as columns of the array 'inno'. The check is done using the Ljung-Box test.

This tool uses the library cdflib (included in the SANGOMA tools package).

REVISION HISTORY:

```
2015-04 - Lars Nerger - Initial code
```

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) :: dimt          ! number innovation times
REAL(REALPREC), INTENT(in) :: inno(dim, dimt) ! array of innovations
REAL(REALPREC), INTENT(out) :: Qval           ! Q-value of Ljung-Box test
REAL(REALPREC), INTENT(out) :: Pval           ! P-value of significance
```

COMMENT:

For an example on using sangoma_CheckWhiteness see example_CheckWhiteness.

4.1.3 sangoma_CompareObsDiag — Compare observation-space diagnostics (Source File: sangoma_CompareObsDiag.F90)

INTERFACE:

```
SUBROUTINE sangoma_CompareObsDiag (dim_obs, dim_ens, out1, out2, &
    out3, out4, typeOut, Hens, CB_diag_R, diff1, diff2, diff3, diff4, &
    m_diff, quot1, quot2, quot3, quot4, m_quot, status) &
    BIND(C, name="sangoma_compareobsdiag_")
```

DESCRIPTION:

This routine computes the different diagnostics on the observations and the ensemble projections on the observations space based on the inputs. It takes the inputs compute from `sangoma_ObsDiag` and computes the diagnostics described in Desroziers et al. (2005). Outputs are the differences, quotients and their mean.

This tool uses the library `BLAS`.

REVISION HISTORY:

```
2015-04 - Paul Kirchgessner - Initial code
```

USES:

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

IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)    :: dim_obs ! Observation dimension
INTEGER(INTPREC), INTENT(in)    :: dim_ens ! Ensemble dimension
INTEGER(INTPREC),   INTENT(inout)   :: typeOut(4) ! Which diagnostics to compute
     ! Out1-4 are inputs computed from sangoma_obsdiag
REAL(REALPREC), INTENT(inout)   :: out1(dim_obs) ! E[d^of (d^of)^T]
REAL(REALPREC), INTENT(inout)   :: out2(dim_obs) ! E[d^af (d^of)^T]
REAL(REALPREC), INTENT(inout)   :: out3(dim_obs) ! E[d^oa d^of]
REAL(REALPREC), INTENT(inout)   :: out4(dim_obs) ! E[d^af d^oa]
REAL(REALPREC), INTENT(in)   :: Hens(dim_obs,dim_ens) ! Projection of ensemble
                                                      ! onto observation space
REAL(REALPREC) , INTENT(out) :: m_diff(4)      ! mean difference
REAL(REALPREC) , INTENT(out) :: m_quot(4)      ! mean quotients
REAL(REALPREC) , INTENT(out) :: diff1(dim_obs) ! difference for diag. 1
REAL(REALPREC) , INTENT(out) :: diff2(dim_obs) ! difference for diag. 2
REAL(REALPREC) , INTENT(out) :: diff3(dim_obs) ! difference for diag. 3
REAL(REALPREC) , INTENT(out) :: diff4(dim_obs) ! difference for diag. 4
REAL(REALPREC) , INTENT(out) :: quot1(dim_obs) ! quotient for diag. 1
REAL(REALPREC) , INTENT(out) :: quot2(dim_obs) ! quotient for diag. 2
REAL(REALPREC) , INTENT(out) :: quot3(dim_obs) ! quotient for diag. 3
REAL(REALPREC) , INTENT(out) :: quot4(dim_obs) ! quotient for diag. 4
INTEGER(INTPREC), INTENT(out) :: status        ! Status flag (0= success)


! Call-back routine providing diagonal elements of R
INTERFACE
   SUBROUTINE CB_diag_R(step, dim_obs, diag_R) BIND(C)
      USE sangoma_base, ONLY: REALPREC, INTPREC
      INTEGER(INTPREC),INTENT(in) :: step              ! Time step
      INTEGER(INTPREC),INTENT(in) :: dim_obs           ! State dimension
      REAL(REALPREC)  ,INTENT(out) :: diag_R(dim_obs)  ! Number of observations
```

```
  END SUBROUTINE
END INTERFACE
```

COMMENT:

For an example on using this function see `example_Obs_Diag`.

4.1.4 `sangoma_ComputeBRIER` — Compute Brier score & its decomposition & 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 one binary event associated with the 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 its partition as well as the 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`.

4.1.5 `sangoma_computeCRIGN` — Computes the CRPS and CRIGN scores (Source File: sangoma_ComputeCRIGN.F90)

and the CRIGN according to Toedter and Ahrens (MWR, 2012)

INTERFACE:

```
SUBROUTINE sangoma_computeCRIGN(dim_ens, fc_ens, obs, crps, crign, &
    CB_SORT) BIND(C, name="sangoma_computecrign_")
```

DESCRIPTION:

Computes the CRPS according to Hersbach (2000) and the CRIGN according to Toedter and Ahrens (MWR, 2012)

REVISION HISTORY:

```
2015    - Julian Toedter    - Initial Code
2015-04 - Paul Kirchgessner - Revision for SANGOMA data model
```

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)   :: fc_ens(dim_ens) ! Ensemble forecast values
real(REALPREC), intent(in)   :: obs             ! Observation to verify the forecast
real(REALPREC), intent(out)  :: crps, crign     ! Scores

! 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
```

COMMENT:
For an example on using this function see `example_compute_CRIGN`.

4.1.6 `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_computeCRPS` see `example_ComputeCRPS`.

4.1.7 `sangoma_CheckEnsSpread` — Check whether ensemble spread is realistic (Source File: sangoma_CheckEnsSpread.F90)

INTERFACE:

```
SUBROUTINE sangoma_checkensspread(dim, dim_ens, state, ens, &
     std_ens, std_state) &
     BIND(C, name="sangoma_checkensspread_")
```

DESCRIPTION:

This routine computes the ensemble spread (standard deviation) and the RMS deviation of the provided state (representing the truth) from the ensemble mean state. Both values should be similar for a representative ensemble.

REVISION HISTORY:

```
2015-04 - Lars Nerger - Initial code
```

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
REAL(REALPREC), INTENT(in) :: state(dim)        ! state vector (the 'truth')
REAL(REALPREC), INTENT(in) :: ens(dim, dim_ens) ! Ensemble array
REAL(REALPREC), INTENT(out) :: std_ens   ! standard deviation of ensemble spread
REAL(REALPREC), INTENT(out) :: std_state ! RMS deviation of mean state from input state
```

COMMENT:

For an example on using `sangoma_CheckEnsSpread` see `example_CheckEnsSpread`.

4.1.8 `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`.

4.1.9 `sangoma_ComputeEffSample` — Compute effective sample size (Source File: sangoma_ComputeEffSample.F90)

INTERFACE:

```
SUBROUTINE sangoma_computeEffSample(dim_sample, weights, effSample, &
     status) BIND(C, name="sangoma_computeeffsample_")
```

DESCRIPTION:

This routine computes the effective sample size of a particle filter as defined in Doucet et al. (2001) p. 333

REVISION HISTORY:

```
2014-06 - Paul Kirchgessner - Initial code for SANGOMA
```

USES:

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

IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim_sample          ! Sample size
REAL(REALPREC), INTENT(in)    :: weights(dim_sample) ! Weights of the samples
REAL(REALPREC), INTENT(out)   :: effsample           ! Effective sample size
INTEGER(INTPREC), INTENT(out) :: status              ! Status flag (0=success)
```

COMMENT:

For an example on using this function see `example_compute_EffSample`.

4.1.10 `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`.

4.1.11 `sangoma_ComputeInvStats` —- Computes the innnovation statistics

INTERFACE:

```
SUBROUTINE sangoma_computeinvstats(dim1,dim2,innovation_f,rms_f, &
    bias_f,min_f,max_f,Pvalue,status) &
    BIND(C, name="sangoma_computeinvstats_")
```

DESCRIPTION:

This routine calculates BIAS, RMS, and Min/Max Errors of the innovation vector (y-Hx). It assumes that the observation operator does not change in between observation time.

OUTPUT: Vectors returning RMS, BIAS, min / max errors, and statistics over space and time by Pvalue test.

This tool uses the library `cdflib` (included in the SANGOMA tools package).

REVISION HISTORY:

```
2015-04 - M. Umer Altaf - Initial code
```

USES:

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

IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in) :: dim1         ! size of dim1
INTEGER(INTPREC), INTENT(in) :: dim2         ! size of dim2
REAL(REALPREC), INTENT(in)   :: innovation_f(dim1,dim2)  ! 2D innovation array
REAL(REALPREC), INTENT(out)  :: rms_f(dim1)  ! RMS
REAL(REALPREC), INTENT(out)  :: bias_f(dim1) ! BIAS
REAL(REALPREC), INTENT(out)  :: min_f(dim1)  ! Min errors
REAL(REALPREC), INTENT(out)  :: max_f(dim1)  ! Max Errors
REAL(REALPREC), INTENT(out)  :: Pvalue       ! Statistics for space or time
INTEGER(INTPREC), INTENT(out) :: status      ! Status flag (0=success)
```

COMMENT:

For an example on using `sangoma_ComputeInvStats` see `example_ComputeInvStats`.

4.1.12 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 providing the 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`.

4.1.13 `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, σ: ensemble spread, σ_o: observation error

The tool checks 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`.

4.1.14 `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 `RE` 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 as $RE \approx \sum(post_w \ln(post_w/prior_w))$

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`.

4.1.15 sangoma_ComputeSMatrix - Computes scaled ensemble observation anomalies matrix

INTERFACE:

```
SUBROUTINE sangoma_computesmatrix(dim_ens,dim_obs,dim_state,obs_opr, &
    ens_anom,obs_cov,mat_S,status) &
    BIND(C, name="sangoma_computesmatrix_")
```

DESCRIPTION:

Computes the analysis step diagnostic in terms of performance of the observational array at detecting prior errors (Sakov et al. 2010).
OUTPUT: Scaled ensemble observation anomalies Matrix (S).
This tool uses the libraries BLAS and LAPACK.

REVISION HISTORY:

```
2015-08 - M. Umer Altaf - Initial code
```

USES:

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

IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in) :: dim_ens          ! size of ensemble
INTEGER(INTPREC), INTENT(in) :: dim_obs          ! number of observations
INTEGER(INTPREC), INTENT(in) :: dim_state        ! state dimensions
REAL(REALPREC), intent(in)   :: obs_opr(dim_obs,dim_state)  ! observation operator
REAL(REALPREC), INTENT(in)   :: ens_anom(dim_state,dim_ens) ! ensemble anomalies
REAL(REALPREC), INTENT(in)   :: obs_cov(dim_obs,dim_obs)    ! observation cov. matrix
REAL(REALPREC), INTENT(out)  :: mat_S(dim_obs,dim_ens) ! S Matrix
INTEGER(INTPREC), INTENT(out) :: status               ! Status flag (0=success)
```

COMMENT:
For an example on using sangoma_ComputeSMatrix see example_ComputeSMatrix.

4.1.16 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`.

4.1.17 `sangoma_ComputeSensitivity_op` — Calculate sensitivity matrix with H as operator (Source File: sangoma_ComputeSensitivity_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 an observation operator provided
                            as a 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`.

4.1.18 sangoma_ArM — Calculate array modes and associated quantities (Source File: sangoma_arm.F90)

INTERFACE:

```
SUBROUTINE sangoma_arm(nstate, nens, nobs, ndof, Af, Yf, &
    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 the degrees of freedom of signal (dofs) of model error detectable above that noise floor.

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

`Yf` 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.

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

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
2015-10: P De Mey sangoma_arm.F90 adjusted some comments,
         changed name of variable Df to Yf
```

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 dofs of problem
REAL(REALPREC), INTENT(in)    :: Af(nstate,nens) ! forecast ensemble anomalies
```

```
REAL(REALPREC), INTENT(in)    :: Yf(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`.

4.1.19 `sangoma_ArM-CA` — Calculate array modes and use them to verify the consistency of a forecast ensemble given an ensemble of innovations (Source File: sangoma_armca.F90)

INTERFACE:

```
SUBROUTINE sangoma_armca(nstate, nens, nobs, ndof, Af, Yf, Y0, &
    R, arm_spect, arm, fcst_dist, inno_dist, fcst_mean, &
    inno_mean, fcst_med, inno_med, fcst_var, inno_var, tol, &
    nranks, status) BIND(C, name="sangoma_armca_")
```

DESCRIPTION:

This routine calculates array modes and ensemble consistency diagnostics given an ensemble of forecasts and an ensemble of observations.

Some of the arguments and definitions are common with the `sangoma_arm` tool.

We work in the space of array modes: (1) in that space, the representer matrix and observational error covariance matrix (assumed full) take a simple form; (2) the consistency can be checked simultaneously for the whole array, and not only for individual observations as is usually done; (3) the consistency criterion is hierarchized from most significant (along the dominant array mode) to least significant.

A simple implementation of the criterion is proposed, counting the number of array-space ranks for which the consistency is verified within an under-defined tolerance.

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

REVISION HISTORY:

```
2015-10: P De Mey sangoma_armca.F90 initial 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
REAL(REALPREC), INTENT(in)    :: Yf(nobs,nens)   ! same as Af in data space
REAL(REALPREC), INTENT(inout) :: Y0(nobs,nens)   ! INPUT: the (column-wise) vectors
                ! of observations for all ensemble members;
                ! OUTPUT: the (column-wise) innovations for all ensemble members
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)   :: fcst_dist(ndof,nens) ! array-space samples
                                              !  (distribution) from Ensemble forecast
REAL(REALPREC), INTENT(out)   :: inno_dist(ndof,nens) ! array-space samples
                                              !  (distribution) from Ensemble innovation
REAL(REALPREC), INTENT(out)   :: fcst_mean(ndof) ! mean of array-space Ensemble
                                              !   forecast distribution
REAL(REALPREC), INTENT(out)   :: inno_mean(ndof) ! mean of array-space Ensemble
                                              !   innovation distribution
REAL(REALPREC), INTENT(out)   :: fcst_med(ndof)  ! median of array-space Ensemble
                                              !  forecast distribution
REAL(REALPREC), INTENT(out)   :: inno_med(ndof)  ! median of array-space Ensemble
                                              !   innovation distribution
REAL(REALPREC), INTENT(out)   :: fcst_var(ndof)  ! variance of array-space Ensemble
                                              !   forecast distribution
REAL(REALPREC), INTENT(out)   :: inno_var(ndof)  ! variance of array-space Ensemble
                                              !   innovation distribution
REAL(REALPREC), INTENT(in)    :: tol             ! tolerance for the calculation
                                              !  of nranks, with 0 < tol < 1
INTEGER(INTPREC), INTENT(out) :: nranks          ! number of array-space ranks
                        ! for which 1-tol <= inno_var(k)/(fcst_var(k)+1) <= 1+tol
INTEGER(INTPREC), INTENT(out) :: status          ! Status flag (0=success)
```

COMMENT:

For an example on using `sangoma_armca` see `example_armca`.

4.1.20 `sangoma_ObsDiag` — Compute sampled observation-space statistics (Source File: sangoma_ObsDiag.F90)

INTERFACE:

```
SUBROUTINE sangoma_ObsDiag(dim_obs,dim_ens,innovation, residual,&
                  increment, typeOut,out1,out2,out3,out4) &
                  BIND(C, name="sangoma_obsdiag_")
```

DESCRIPTION:

This routine computes the left side of the statistics of the observation covariance given in Desrozier et al. (2005)

The following diagnostics are considered:

$$E[d^{of}(d^{of})^T] = R + HP^fH^T$$
$$E[d^{af}(d^{of})^T] = HP^fH^T$$
$$E[d^{oa}d^{of}] = R$$
$$E[d^{af}d^{oa}] = HP^aH^T$$

This tool uses the library `BLAS`.

REVISION HISTORY:

`2015-04 - Paul Kirchgessner - Initial code`

USES:

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

IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in) :: dim_obs ! Observation dimension
INTEGER(INTPREC), INTENT(in) :: dim_ens ! dimension of ensemble
REAL(REALPREC), INTENT(in)   :: innovation(dim_obs,dim_ens) ! y-Hx^f
REAL(REALPREC), INTENT(in)   :: residual(dim_obs,dim_ens)   ! y-Hx^a
REAL(REALPREC), INTENT(in)   :: increment(dim_obs,dim_ens)  ! Hx^f-Hx^a
INTEGER(INTPREC),   INTENT(in)   :: typeOut(4)
 ! Integer array(4) (1 compute; 0 do not compute)
 ! (1) E[d^of (d^of)^T] = R + HP^fH^T
 ! (2) E[d^af (d^of)^T] = HP^fH^T
 ! (3) E[d^oa d^of] = R
 ! (4) E[d^af d^oa] = HP^aH^T
REAL(REALPREC), INTENT(out)   :: out1(dim_obs) ! E[d^of (d^of)^T]
REAL(REALPREC), INTENT(out)   :: out2(dim_obs) ! E[d^af (d^of)^T]
REAL(REALPREC), INTENT(out)   :: out3(dim_obs) ! E[d^af (d^of)^T]
REAL(REALPREC), INTENT(out)   :: out4(dim_obs) ! E[d^af d^oa]
```

COMMENT:

For an example on using this function see `example_Obs_Diag`.

4.2 Perturbation Tools

4.2.1 `sangoma_pseudornd2D` — Generate correlated random field (Source File: 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).
This tool uses the library `FFTW3`.

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 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 `sangoma_SampleEns` — Generate ensemble by 2nd order exact sampling of EOF modes (Source File: sangoma_SampleEns.F90)

INTERFACE:

```
SUBROUTINE sangoma_SampleEns(dim, dim_ens, modes, svals, state, &
     ens, flag) BIND(C, name="sangoma_sampleens_")
```

DESCRIPTION:

This routine generates an ensemble of model states from a provided mean state and EOF modes (singular vectors of a peturbation matrix) and singular values. The resulting ensemble is the 2nd order-exact sample covariance matrix and mean state. The ensemble state vectors are computed as
$ens_i = state + sqrt(dim_ens - 1)modes(\Omega C)^T$
where C holds in its diagonal the singular values ($svals$). Ω is an orthogonal transformation matrix that preserves the mean state. The generated ensemble fulfills the condition for the state error covariance matrix
$P = 1/(sqrt(dim_ens - 1)\sum_{i=1}^{dim\_ens}(ens_i - state)(ens_i - state)^T$.
The routine uses the library 'BLAS'.

REVISION HISTORY:

```
2014-05 - Lars Nerger - Initial code based on PDAF example code
```

USES:

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

IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in) :: dim                    ! Size of state vector
INTEGER(INTPREC), INTENT(in) :: dim_ens                ! Size of ensemble
REAL(REALPREC), INTENT(inout) :: modes(dim, dim_ens-1) ! Array of EOF modes
REAL(REALPREC), INTENT(in)    :: svals(dim_ens-1)      ! Vector of singular values
REAL(REALPREC), INTENT(inout) :: state(dim)            ! PE-local model state
REAL(REALPREC), INTENT(out)   :: ens(dim, dim_ens)     ! State ensemble
INTEGER(INTPREC), INTENT(inout) :: flag                ! Status flag
```

COMMENT:
For an example on using `sangoma_SampleEns` see `example_SampleEns`.

4.3 Transformation Tools

4.3.1 `sangoma_Anamorphosis` — Computes local Gaussian anamorphosis

INTERFACE:

```
SUBROUTINE sangoma_anamorphosis(dim, dim_qua, dir, &
    qua_ref, qua, vct, status) &
    BIND(C, name="sangoma_anamorphosis_")
```

DESCRIPTION:

This routine performs local anamorphosis transformation of a state or observation vector using a set of quantiles of the input ensemble (as provided by `sangoma_ComputeQuantiles`).

This is an implementation of the specific algorithm described in Brankart et al. (2012). Ocean Science, 8, 121-142.

Inputs are the vector to transform, the quantiles of the ensemble, and the corresponding quantiles of the target distribution (assumed all distinct).

REVISION HISTORY:

```
2015-02 - J.-M. Brankart - Initial SANGOMA code
```

USES:

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

IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim       ! Vector (state or obs) dimension
INTEGER(INTPREC), INTENT(in)  :: dim_qua   ! Number of quantiles
INTEGER(INTPREC), INTENT(in)  :: dir       ! Forward (+) or backward (-) transformation
REAL(REALPREC), INTENT(in)    :: qua_ref(dim_qua)  ! Quantiles of the target distribution
REAL(REALPREC), INTENT(in)    :: qua(dim, dim_qua) ! Ensemble quantiles
REAL(REALPREC), INTENT(inout) :: vct(dim)          ! Vector to transform
INTEGER(INTPREC), INTENT(out) :: status            ! Status flag (0=success)
```

COMMENT:

For an example on using `sangoma_Anamorphosis` see `example_Anamorphosis`.

4.3.2 `sangoma_ComputeQuantiles` — Computes ensemble quantiles as input for anamorphosis

INTERFACE:

```
SUBROUTINE sangoma_computequantiles(dim, dim_ens, dim_qua, &
    qua_def, ens, qua, CB_SORT) &
    BIND(C, name="sangoma_computequantiles_")
```

DESCRIPTION:

This routine computes the quantiles of an ensemble in state or observation space.

Inputs are the ensemble array and the quantile definition (between 0 and 1)

For instance: 0 for minimum of the ensemble, 0.25 for the first quartile, 0.5 for the median, 0.75 for the last quartile, 1 for the maximum of the ensemble.

REVISION HISTORY:

```
2015-02 - J.-M. Brankart - Initial SANGOMA code
```

USES:

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

IMPLICIT NONE
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim              ! Vector (state or obs) dimension
INTEGER(INTPREC), INTENT(in)  :: dim_ens          ! Ensemble size
INTEGER(INTPREC), INTENT(in)  :: dim_qua          ! Number of quantiles
REAL(REALPREC), INTENT(in)    :: qua_def(dim_qua) ! Quantile definition
REAL(REALPREC), INTENT(in)    :: ens(dim, dim_ens) ! Ensemble
REAL(REALPREC), INTENT(out)   :: qua(dim, dim_qua) ! Ensemble quantiles
```

COMMENT:

For an example on using `sangoma_Anamorphosis` see `example_Anamorphosis`.

4.4 Utilities

4.4.1 `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
USE mod_sangoma_analysis, ONLY: generate_rndmat

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`.

4.4.2 `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, &
    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) :: 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`.

4.5 Analysis Steps

4.5.1 `sangoma_ens_analysis` – Computes the analysis ensemble using the ETKF scheme (Source File: mod_sangoma_ensemble_analysis.F90)

INTERFACE:

```
subroutine sangoma_ens_analysis(dim,dim_obs,dim_ens,Ef,HEf,y,method,Ea,cbR) &
  bind(C, name="sangoma_ensemble_analysis_")
```

DESCRIPTION:

Computes the analysis ensemble `Ea` based on forecast ensemble `Ef` using the ETKF scheme. Currently, the method has to be ETKF2 (see Hunt et al., 2007) cbR is a callback-routine to compute the product of observation error covariance matrix R if mode is 1 with a given vector or the inverse of R times a vector if mode is -1.

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_obs            ! number of observations
integer(intprec), intent(in) :: dim_ens            ! ensemble size

real(realprec), intent(in) :: Ef(dim,dim_ens)      ! forecast ensemble
real(realprec), intent(in) :: HEf(dim_obs,dim_ens) ! obs. forecast ensemble
real(realprec), intent(in) :: y(dim_obs)           ! observation
character(kind=c_char, len=1), intent(in) :: method(*)
                                                   ! name of the method
                                                   ! (null-terminated c-string)
real(realprec), intent(out) :: Ea(dim,dim_ens)     ! analysed ensemble

 interface
   subroutine cbR(m,x,mode,Rx) bind(C)
    use, intrinsic :: iso_c_binding
    use sangoma_base, only: realprec, intprec
    implicit none
    integer(intprec), intent(in) :: m     ! number of observations
    real(realprec), intent(in) :: x(:)    ! a vector in observation space
    integer(intprec), intent(in) :: mode  ! determines if R (mode=1) or its
                                          ! inverse (mode=-1) is applied to x
    real(realprec), intent(out) :: Rx(:)  ! The result of the operation
   end subroutine cbR
 end interface
```

COMMENT:

For an example on using this function see `example_ensemble_analysis`. A code for checking the consistency of the analysis is provided by `test_ensemble_analysis`.

4.5.2 `sangoma_local_ens_analysis` – Computes the local analysis ensemble using the ETKF scheme (Source File: mod_sangoma_ensemble_analysis.F90)

INTERFACE:

```
subroutine sangoma_local_ensemble_analysis(dim, dim_obs, dim_ens, &
    Ef, HEf, y, diagR, part,selectObs,method,Ea) &
    bind(C, name="sangoma_local_ensemble_analysis_")
```

DESCRIPTION:

Computes local ensemble analysis `Ea` based on forecast ensemble `Ef` using the ETKF scheme. Currently, method has to be ETKF2 (see Hunt et al., 2007) selectObs is a callback-routine which returns the weight for each observation relative to a state vector index i.

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_obs              ! number of observations
integer(intprec), intent(in) :: dim_ens              ! ensemble size

real(realprec),   intent(in) :: Ef(dim,dim_ens)      ! forecast ensemble
real(realprec),   intent(in) :: HEf(dim_obs,dim_ens) ! obs. forecast ensemble
real(realprec),   intent(in) :: y(dim_obs)           ! observation

real(realprec),   intent(in) :: diagR(dim_obs)       ! diagonal elements of R
integer(intprec), intent(in) :: part(dim)            ! partition vector
character(kind=c_char, len=1), intent(in) :: method(*)
                                                     ! name of the method
                                                     ! (null-terminated c-string)
real(realprec), intent(out) :: Ea(dim,dim_ens)       ! analysed ensemble

interface
  subroutine selectObs(dim_obs,i,w) bind(C)
   use, intrinsic :: iso_c_binding
   use sangoma_base, only: realprec, intprec
   implicit none
   integer(intprec), intent(in) :: dim_obs   ! number of observations
   integer(intprec), intent(in) :: i         ! index in the state vector
   real(realprec), intent(out) :: w(dim_obs) ! resulting weight relative to the
                                             ! i-th element of the state vector
  end subroutine selectObs
end interface
```

COMMENT:

For an example on using this function see `example_local_ensemble_analysis`.
A code for checking the consistency of the analysis is provided by `test_local_ensemble_analysis`.

4.5.3 `sangoma_enkf_analysis` – EnKF Analysis Step (Source File: sangoma_enkf_analysis.F90)

INTERFACE:

```
SUBROUTINE sangoma_enkf_analysis(dim, dim_obs, dim_ens, ens, &
     Hens, obs, covarR, inflate, cb_localize_enkf, flag) &
     BIND(C, name="sangoma_enkf_analysis_")
```

DESCRIPTION:

Analysis step of the ensemble Kalman filter with perturbed observations. The analysis is implemented with the representer-type formulation. In this version the matrix HP is explicitly computed.

This code is a simplified version of the EnKF implemented in PDAF. E.g. the parallelization, memory counting and timers have been removed. Also the interface is shortened and automatic arrays are used instead of dynamic allocation. The observation operator is applied outside of the analysis routine, which leads to a restriction to linear observation operators. The routine uses a blocked ensemble transformation as in PDAF to avoid the need for a second ensemble array.

Covariance localization can be applied in the call-back routine cb_localize_enkf.

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

USES:

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

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim       ! Dimension of model state
INTEGER(INTPREC), INTENT(in)  :: dim_obs   ! Dimension of observation vector
INTEGER(INTPREC), INTENT(in)  :: dim_ens   ! Size of state ensemble
REAL(REALPREC), INTENT(inout) :: ens(dim, dim_ens)   ! State ensemble - overwritten
                                          ! In: forecast; Out: analysis ensemble
REAL(REALPREC), INTENT(inout) :: Hens(dim_obs, dim_ens)   ! Observed ensemble
REAL(REALPREC), INTENT(in)    :: obs(dim_obs)             ! Vector of observations
REAL(REALPREC), INTENT(inout) :: covarR(dim_obs, dim_obs) ! Observation covariance
REAL(REALPREC), INTENT(in)    :: inflate   ! Inflation factor
INTEGER(INTPREC), INTENT(inout) :: flag    ! Status flag
```

```
! External  call-back subroutine
INTERFACE
   ! Routine applying localization to the matrices HP and HPH^T
   SUBROUTINE cb_localize_enkf(dim, dim_obs, HP, HPH) BIND(C)
     USE sangoma_base, ONLY: REALPREC, INTPREC
     INTEGER(INTPREC), INTENT(in) :: dim         ! State dimension
     INTEGER(INTPREC), INTENT(in) :: dim_obs     ! Observation dimension
     REAL(REALPREC), INTENT(inout) :: HP(dim_obs, dim)      ! Matrix HP
     REAL(REALPREC), INTENT(inout) :: HPH(dim_obs, dim_obs) ! Matrix HPH^T
   END SUBROUTINE cb_localize_enkf
END INTERFACE
```

COMMENT:

For an example on using this function see `example_enkf_analysis.F90`. The examples for the analysis steps all use the same initial ensemble, observation error covariance matrix, and observations. the results are identical for EnSRF, ESTKF, and ETKF. The result for the EnKF is different due to sampling errors. The localized filters LESTKF and LETKF yield a different result from the global filters due to the localization (unless the localization radius is set so large that all observations are assimilated in each local domain). A code for checking the consistency of the analysis is provided by `example_enkf_analysis_check`.

4.5.4 `sangoma_ensrf_analysis` – EnSRF Serial Analysis Step (Source File: sangoma_ensrf_analysis.F90)

INTERFACE:

```
SUBROUTINE sangoma_ensrf_analysis(dim, dim_obs, dim_ens, ens, &
     obs, diagR, inflate, cb_H_ensrf, cb_localize_ensrf, flag) &
     BIND(C, name="sangoma_ensrf_analysis_")
```

DESCRIPTION:

Analysis step of ensemble square-root Kalman filter with sequential analysis using scalar Obervations (Whitaker and Hamill, Monthly Weather Review 2002).

This variant is a simplified version of the EnSRF implemented in PDAF. E.g. the parallelization, memory counting and timers have been removed. Also the interface is shortened and automatic arrays are used instead of dynamic allocation.

Covariance localization can be applied in the call-back routine cb_localize_ensrf. The call-back routine cb_H_ensrf is called to apply the serial observation operator.

USES:

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

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim      ! Dimension of model state
INTEGER(INTPREC), INTENT(in)  :: dim_obs  ! Dimension of observation vector
INTEGER(INTPREC), INTENT(in)  :: dim_ens  ! Size of state ensemble
REAL(REALPREC), INTENT(inout) :: ens(dim, dim_ens) ! State ensemble - overwritten
                                          ! In: forecast; Out: analysis ensemble
REAL(REALPREC), INTENT(in)    :: obs(dim_obs)      ! Vector of observations
REAL(REALPREC), INTENT(in)    :: diagR(dim_obs)    ! Diagonal of R
REAL(REALPREC), INTENT(in)    :: inflate  ! Inflation factor
INTEGER(INTPREC), INTENT(inout) :: flag   ! Status flag

! External  call-back subroutines
INTERFACE
   ! Routine applying the observation operator for a single observation
   ! identified by 'iobs' to a state vector
   SUBROUTINE cb_H_ensrf(iobs, dim, dim_obs, state, Hx) BIND(C)
     USE sangoma_base, ONLY: REALPREC, INTPREC
     INTEGER(INTPREC), INTENT(in) :: iobs     ! Observation index
     INTEGER(INTPREC), INTENT(in) :: dim      ! State dimension
     INTEGER(INTPREC), INTENT(in) :: dim_obs  ! Total observation dimension
     REAL(REALPREC), INTENT(in)   :: state(dim)  ! State vector
     REAL(REALPREC), INTENT(out)  :: Hx       ! Observed state vector element
   END SUBROUTINE cb_H_ensrf

   ! Routine applying localization to the matrix HP for a single observation
   ! in the serial observation treatment of the EnSRF.
   SUBROUTINE cb_localize_ensrf(iobs, dim, HP) BIND(C)
     USE sangoma_base, ONLY: REALPREC, INTPREC
     INTEGER(INTPREC), INTENT(in) :: iobs     ! Observation index
     INTEGER(INTPREC), INTENT(in) :: dim      ! State dimension
     REAL(REALPREC), INTENT(inout) :: HP(dim) ! Matrix HP for single observation
   END SUBROUTINE cb_localize_ensrf
END INTERFACE
```

COMMENT:

For an example on using this function see `example_ensrf_analysis.F90`. The examples for the analysis steps all use the same initial ensemble, observation error covariance matrix, and observations. the results are identical for EnSRF, ESTKF, and ETKF. The result for the EnKF is different due to sampling errors. The localized filters LESTKF and LETKF yield a different result from the global filters due to the localization (unless the localization radius is set so large that all observations are assimilated in each local domain). A code for checking the consistency of the analysis is provided by `example_ensrf_analysis_check`.

4.5.5 `sangoma_estkf_analysis` – ESTKF Analysis Step (Source File: sangoma_estkf_analysis.F90)

INTERFACE:

```
SUBROUTINE sangoma_estkf_analysis(dim, dim_obs, dim_ens, ens, &
```

```
     Hens, obs, forget, cb_prodRinvA, flag) &
     BIND(C, name="sangoma_estkf_analysis_")
```

DESCRIPTION:

Analysis step of the ESTKF (Error-Subspace Transform Kalman Filter) with direct transformation of the forecast into the analysis ensemble. This variant does not compute the analysis state, but only the analysis ensemble, whose mean is the analysis state. For details on the ESTKF see L. Nerger et al., Mon. Wea. Rev. 140 (2012) 2335-2345.

This code is a simplified version of the ESTKF implemented in PDAF. E.g. the parallelization, memory counting and timers have been removed. Also the interface is shortened and automatic arrays are used instead of dynamic allocation. The observation operator is applied outside of the analysis routine, which leads to a restriction to linear observation operators. The routine uses a blocked ensemble transformation as in PDAF to avoid the need for a second ensemble array.

This tool uses the libraries BLAS and LAPACK.

USES:

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC
USE sangoma_module_analysis, ONLY: estkf_AOmega, estkf_Omega, estkf_OmegaA
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim          ! Dimension of model state
INTEGER(INTPREC), INTENT(in)  :: dim_obs      ! Dimension of observation vector
INTEGER(INTPREC), INTENT(in)  :: dim_ens      ! Size of ensemble
REAL(REALPREC), INTENT(inout) :: ens(dim, dim_ens) ! State ensemble - overwritten
                                              ! In: forecast; Out: analysis ensemble
REAL(REALPREC), INTENT(inout) :: Hens(dim_obs, dim_ens) ! Observed ensemble
REAL(REALPREC), INTENT(in)    :: obs(dim_obs) ! Vector of observations
REAL(REALPREC), INTENT(in)    :: forget       ! Forgetting factor
INTEGER(INTPREC), INTENT(out) :: flag         ! Status flag

! External call-back subroutine
INTERFACE
   ! Routine computing the product of inverse observation error covariance matrix
   ! with some other matrix, which is a temporary result in the ETKF
   SUBROUTINE cb_prodRinvA(dim_obs, dim_ens, HZ, RiHZ) BIND(C)
     USE sangoma_base, ONLY: REALPREC, INTPREC
     INTEGER(INTPREC), INTENT(in) :: dim_obs              ! Observation dimension
     INTEGER(INTPREC), INTENT(in) :: dim_ens              ! Ensemble size
     REAL(REALPREC), INTENT(in) :: HZ(dim_obs, dim_ens)   ! Input matrix
                                                          !   in observation space
     REAL(REALPREC), INTENT(out):: RiHZ(dim_obs, dim_ens) ! The result of the
                                                          !   multiplication
   END SUBROUTINE cb_prodRinvA
END INTERFACE
```

COMMENT:

For an example on using this function see `example_estkf_analysis.F90`. The examples for the analysis steps all use the same initial ensemble, observation error covariance matrix, and observations. the results are identical for EnSRF, ESTKF, and ETKF. The result for the EnKF is different due to sampling errors. The localized filters LESTKF and LETKF yield a different result from the global filters due to the localization (unless the localization radius is set so large that all observations are assimilated in each local domain). A code for checking the consistency of the analysis is provided by `example_estkf_analysis_check`.

4.5.6 `sangoma_etkf_analysis` – ETKF Analysis Step (Source File: sangoma_etkf_analysis.F90)

INTERFACE:

```
SUBROUTINE sangoma_etkf_analysis(dim, dim_obs, dim_ens, ens, &
     Hens, obs, forget, cb_prodRinvA, flag) &
     BIND(C, name="sangoma_etkf_analysis_")
```

DESCRIPTION:

Analysis step of the ETKF (Ensemble Transform Kalman Filter, C. Bishop et al., Mon. Wea. Rev. 129(2001) 420-436). For efficiency this implementation uses a matrix T analogous to the ESTKF method. The consistent use of the operation of removing the mean from an ensemble matrix in form of a linear transformation (T-matrix) reduces the computational complexity. For details on this variant of the ETKF see Nerger et al., Mon. Wea. Rev. 140 (2012) 2335-2345.

This code is a simplified version of the ETKF implemented in PDAF. E.g. the parallelization, memory counting and timers have been removed. Also the interface is shortened and automatic arrays are used instead of dynamic allocation. The observation operator is applied outside of the analysis routine, which leads to a restriction to linear observation operators. The routine uses a blocked ensemble transformation as in PDAF to avoid the need for a second ensemble array.

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

USES:

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC
USE sangoma_module_analysis, ONLY: etkf_tright, etkf_tleft
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim          ! Dimension of model state
INTEGER(INTPREC), INTENT(in)  :: dim_obs      ! Dimension of observation vector
```

```
INTEGER(INTPREC), INTENT(in)  :: dim_ens      ! Size of ensemble
REAL(REALPREC), INTENT(inout) :: ens(dim, dim_ens) ! State ensemble - overwritten
                                                   ! In: forecast; Out: analysis ensemble
REAL(REALPREC), INTENT(inout) :: Hens(dim_obs, dim_ens) ! Observed ensemble
REAL(REALPREC), INTENT(in)    :: obs(dim_obs) ! Vector of observations
REAL(REALPREC), INTENT(in)    :: forget       ! Forgetting factor
INTEGER(INTPREC), INTENT(out) :: flag         ! Status flag

! External  call-back subroutine
INTERFACE
   ! Routine computing the product of inverse observation error covariance matrix
   ! with some other matrix, which is a temporary result in the ETKF
   SUBROUTINE cb_prodRinvA(dim_obs, dim_ens, HZ, RiHZ) BIND(C)
     USE sangoma_base, ONLY: REALPREC, INTPREC
     INTEGER(INTPREC), INTENT(in) :: dim_obs              ! Observation dimension
     INTEGER(INTPREC), INTENT(in) :: dim_ens              ! Ensemble size
     REAL(REALPREC), INTENT(in) :: HZ(dim_obs, dim_ens)   ! Input matrix
                                                          !   in observation space
     REAL(REALPREC), INTENT(out):: RiHZ(dim_obs, dim_ens) ! The result of the
                                                          !   multiplication
   END SUBROUTINE cb_prodRinvA
END INTERFACE
```

COMMENT:

For an example on using this function see `example_etkf_analysis.F90`. The examples for the analysis steps all use the same initial ensemble, observation error covariance matrix, and observations. the results are identical for EnSRF, ESTKF, and ETKF. The result for the EnKF is different due to sampling errors. The localized filters LESTKF and LETKF yield a different result from the global filters due to the localization (unless the localization radius is set so large that all observations are assimilated in each local domain). A code for checking the consistency of the analysis is provided by `example_etkf_analysis_check`.

4.5.7 `sangoma_lestkf_analysis` – LESTKF Analysis Step (Source File: sangoma_lestkf_analysis.F90)

INTERFACE:

```
SUBROUTINE sangoma_lestkf_analysis(dim, dim_obs, dim_ens, ens, &
     Hens, obs, forget, cb_prodRinvA_local, flag) &
     BIND(C, name="sangoma_lestkf_analysis_")
```

DESCRIPTION:

Analysis step of the LESTKF (Local Error-Subspace Transform Kalman Filter) with direct transformation of the forecast into the analysis ensemble. This variant does not compute the analysis state, but only the analysis ensemble, whose mean is the analysis state. For details on the ESTKF see Nerger et al., Mon. Wea. Rev. 140 (2012) 2335-2345. The localization applied in this

method is 'observation localization' as discussed by Nerger, L. et al., Q. J. Roy. Meteorol. Soc., 138 (2012), 802-812.

This code is a simplified version of the LESTKF implemented in PDAF. E.g. the parallelization, memory counting and timers have been removed. Also the interface is shortened and automatic arrays are used instead of dynamic allocation. The observation operator is applied outside of the analysis routine, which leads to a restriction to linear observation operators. Further, we only rely on the localization weights to perform the localization and do not copy the local observations into a separate array.

The localization is applied in the call-back routine cb_prodRinvA_local.

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

USES:

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC
USE sangoma_module_analysis, ONLY: estkf_AOmega, estkf_Omega, estkf_OmegaA
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim          ! Dimension of model state
INTEGER(INTPREC), INTENT(in)  :: dim_obs      ! Dimension of observation vector
INTEGER(INTPREC), INTENT(in)  :: dim_ens      ! Size of ensemble
REAL(REALPREC), INTENT(inout) :: ens(dim, dim_ens) ! State ensemble - overwritten
                                        ! In: forecast; Out: analysis ensemble
REAL(REALPREC), INTENT(inout) :: Hens(dim_obs, dim_ens) ! Observed ensemble
REAL(REALPREC), INTENT(in)    :: obs(dim_obs) ! Vector of observations
REAL(REALPREC), INTENT(in)    :: forget       ! Forgetting factor
INTEGER(INTPREC), INTENT(out) :: flag         ! Status flag

! External  call-back subroutine
INTERFACE
   ! Routine computing the product of inverse observation error covariance matrix
   ! with some other matrix, which is a temporary result in the ETKF
   SUBROUTINE cb_prodRinvA_local(domain, dim_obs, dim_ens, HZ, RiHZ) BIND(C)
     USE sangoma_base, ONLY: REALPREC, INTPREC
     INTEGER(INTPREC), INTENT(in) :: domain      ! Index of local analysis domain
     INTEGER(INTPREC), INTENT(in) :: dim_obs              ! Observation dimension
     INTEGER(INTPREC), INTENT(in) :: dim_ens              ! Ensemble size
     REAL(REALPREC), INTENT(in) :: HZ(dim_obs, dim_ens)   ! Input matrix
                                                          !   in observation space
     REAL(REALPREC), INTENT(out):: RiHZ(dim_obs, dim_ens) ! The result of the
                                                          !   multiplication
   END SUBROUTINE cb_prodRinvA_local
```

COMMENT:

For an example on using this function see `example_lestkf_analysis.F90`. The localized filters LESTKF and LETKF yield a the same result. However, due to the localization it is different global filters (unless the localization radius is set so large that all observations are assimilated in each local domain). A code for checking the consistency of the analysis is provided by `example_lestkf_analysis_check`.

4.5.8 `sangoma_letkf_analysis` – LETKF Analysis Step (Source File: sangoma_letkf_analysis.F90)

INTERFACE:

```
SUBROUTINE sangoma_letkf_analysis(dim, dim_obs, dim_ens, ens, &
     Hens, obs, forget, cb_prodRinvA_local, flag) &
     BIND(C, name="sangoma_lestkf_analysis_")
```

DESCRIPTION:

Analysis step of the LETKF (Local Ensemble Transform Kalman Filter, B. Hunt et al., Physica D 230(2007) 112-126). For efficiency this implementation uses a matrix T analogous to the ESTKF method. The consistent use of the operation of removing the mean from an ensemble matrix in form of a linear transformation (T-matrix) reduces the computational complexity. For details on this variant of the ETKF see Nerger et al., Mon. Wea. Rev. 140 (2012) 2335-2345. The localization applied in this method is 'observation localization' as discussed by Nerger et al., Q. J. Roy. Meteorol. Soc., 138 (2012), 802-812.

This code is a simplified version of the LETKF implemented in PDAF. E.g. the parallelization, memory counting and timers have been removed. Also the interface is shortened and automatic arrays are used instead of dynamic allocation. The observation operator is applied outside of the analysis routine, which leads to a restriction to linear observation operators. Further, we only rely on the localization weights to perform the localization and do not copy the local observations into a separate array.

The localization is applied in the call-back routine cb_prodRinvA_local.

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

USES:

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC
USE sangoma_module_analysis, ONLY: etkf_tright, etkf_tleft
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim          ! Dimension of model state
INTEGER(INTPREC), INTENT(in)  :: dim_obs      ! Dimension of observation vector
INTEGER(INTPREC), INTENT(in)  :: dim_ens      ! Size of ensemble
REAL(REALPREC), INTENT(inout) :: ens(dim, dim_ens) ! State ensemble - overwritten
                                          ! In: forecast; Out: analysis ensemble
REAL(REALPREC), INTENT(inout) :: Hens(dim_obs, dim_ens) ! Observed ensemble
REAL(REALPREC), INTENT(in)    :: obs(dim_obs) ! Vector of observations
REAL(REALPREC), INTENT(in)    :: forget       ! Forgetting factor
INTEGER(INTPREC), INTENT(out) :: flag         ! Status flag

! External  call-back subroutine
INTERFACE
```

```
! Routine computing the product of inverse observation error covariance matrix
! with some other matrix, which is a temporary result in the ETKF
SUBROUTINE cb_prodRinvA_local(domain, dim_obs, dim_ens, HZ, RiHZ) BIND(C)
  USE sangoma_base, ONLY: REALPREC, INTPREC
  INTEGER(INTPREC), INTENT(in) :: domain      ! Index of local analysis domain
  INTEGER(INTPREC), INTENT(in) :: dim_obs              ! Observation dimension
  INTEGER(INTPREC), INTENT(in) :: dim_ens              ! Ensemble size
  REAL(REALPREC), INTENT(in) :: HZ(dim_obs, dim_ens)   ! Input matrix
                                                       !   in observation space
  REAL(REALPREC), INTENT(out):: RiHZ(dim_obs, dim_ens) ! The result of the
                                                       !   multiplication
END SUBROUTINE cb_prodRinvA_local
```

COMMENT:

For an example on using this function see `example_letkf_analysis.F90`. The localized filters LESTKF and LETKF yield a the same result. However, due to the localization it is different global filters (unless the localization radius is set so large that all observations are assimilated in each local domain). A code for checking the consistency of the analysis is provided by `example_letkf_analysis_check`.

4.5.9 `sangoma_netf_analysis` – NETF Analysis Step (Source File: sangoma_etkf_analysis.F90)

INTERFACE:

```
SUBROUTINE sangoma_netf_analysis(dim, dim_obs, dim_ens, ens, &
     Hens, obs, forget, cb_prodRinvA, flag) &
     BIND(C, name="sangoma_netf_analysis_")
```

DESCRIPTION:

Analysis step of the NETF (Nonlinear Ensemble Transform Filter) following J. Toedter and B. Ahrens, A Second-order Exact Ensemble Square Root Filter for Nonlinear Data Assimlation, Mon. Wea. Rev. 143 (2015) 1347-1367.

This code is a simplified version of the ETKF implemented in PDAF. E.g. the parallelization, memory counting and timers have been removed. Also the interface is shortened and automatic arrays are used instead of dynamic allocation. The observation operator is applied outside of the analysis routine, which leads to a restriction to linear observation operators. The routine uses a blocked ensemble transformation as in PDAF to avoid the need for a second ensemble array.

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

REVISION HISTORY:

```
2014-03 - Paul Kirchgessner - Changed original PDAF-ETKF code to NETF
2015-10 - Lars Nerger - Adaption for Sangoma
```

USES:

```
USE, INTRINSIC :: ISO_C_BINDING
USE sangoma_base, ONLY: REALPREC, INTPREC
USE sangoma_module_analysis, ONLY: etkf_tright, etkf_tleft
```

ARGUMENTS:

```
INTEGER(INTPREC), INTENT(in)  :: dim          ! Dimension of model state
INTEGER(INTPREC), INTENT(in)  :: dim_obs      ! Dimension of observation vector
INTEGER(INTPREC), INTENT(in)  :: dim_ens      ! Size of ensemble
REAL(REALPREC), INTENT(inout) :: ens(dim, dim_ens) ! State ensemble - overwritten
                                              ! In: forecast; Out: analysis ensemble
REAL(REALPREC), INTENT(inout) :: Hens(dim_obs, dim_ens) ! Observed ensemble
REAL(REALPREC), INTENT(in)    :: obs(dim_obs) ! Vector of observations
REAL(REALPREC), INTENT(in)    :: forget       ! Forgetting factor
INTEGER(INTPREC), INTENT(out) :: flag         ! Status flag

! External  call-back subroutine
INTERFACE
   ! Routine computing the product of inverse observation error covariance matrix
   ! with some other matrix, which is a temporary result in the ETKF
   SUBROUTINE cb_prodRinvA(dim_obs, dim_ens, HZ, RiHZ) BIND(C)
     USE sangoma_base, ONLY: REALPREC, INTPREC
     INTEGER(INTPREC), INTENT(in) :: dim_obs              ! Observation dimension
     INTEGER(INTPREC), INTENT(in) :: dim_ens              ! Ensemble size
     REAL(REALPREC), INTENT(in) :: HZ(dim_obs, dim_ens)   ! Input matrix
                                                          !   in observation space
     REAL(REALPREC), INTENT(out):: RiHZ(dim_obs, dim_ens) ! The result of the
                                                          !   multiplication
   END SUBROUTINE cb_prodRinvA
END INTERFACE
```

COMMENT:

For an example on using this function see `example_netf_analysis.F90`. The analysis example is the same as used for the ensemble Kalman filters. However, because NETF is not an ensemble Kalman filter, its analysis ensemble and ensemble mean are distinct from the analysis result of the ensemble Kalman filters.

Chapter 5

Documentation of Matlab-Tools

5.1 Diagnostic Tools

5.1.1 `computeBRIER` — Compute the Brier skill score and its decomposition, and the entropy (Source File: computeBRIER.m)

INTERFACE:

```
[br,brc,brv,unc,pc,s,sunc,pp,g,pr]=computeBRIER(m,nens,ens,ana,xth)
```

DESCRIPTION:

This function computes scores related to one binary event associated with the 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, the Brier score

$$B = E[(p - p'])^2] - E[(p' - pc])^2] + pc(1 - pc)$$

is computed. Further computed are the Brier skill score BS, the partition of B into reliability and resolution, and the entropy S:

$$BS = 1 - B/pc(1 - pc)$$

$$S = -E[p'log(p')]$$

(see details in Deliverable 4.1).

ARGUMENTS:

```
Input:  m                 size of verification set
        nens              size of each ensemble
        ens(m,nens)       ensemble
        ana(m)            analysis or observation (verification)
```

```
          xth(m)              threshold

Output:   br                  Brier global skill score
          brc                 reliability component
          brv                 resolution component
          unc                 uncertainty
          pc                  'climatological' probability
          s                   entropy
          sunc                'climatological' entropy
          pp(nens+1)          predicted probabilities
          g(nens+1)           distribution of pp
          pr(nens+1)          predictable probabilities
```

COMMENT:
For an example on using `computeBRIER` see `example_computeBRIER`.

5.1.2 `computeCRPS` — Compute the CRPS and its decomposition (Source File: computeCRPS.m)

INTERFACE:

```
[CRPS,RELI,RESOL,UNCERT,BB,AA]=computeCRPS(ENS,ANA,missing,NCASES,NENS)
```

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)

ARGUMENTS:

```
Input:    NCASES                Size of verification set
          NENS                  Size of each ensemble
          ENS(NCASES,NENS)      Ensemble of anomalies
          ANA(NCASES)           Analysis of anomaly
          missing(ncases)       0 : the observation is OK
                                1 : the observation should
                                    not be used

Output:   CRPS                  CRPS based on sample set of ANA
```

```
RELI                    Reliability
RESOL                   Resolution
UNCERT                  Uncertainty
BB(1:NENS), AA(1:NENS)  Detailed info on CRPS
```

COMMENT:
For an example on using `computeCRPS` see `example_computeCRPS`.

5.1.3 `computehistogram` — Increment rank histogram (Source File: computehistogram.m)

INTERFACE:

```
[hist,delta]=computehistogram(ncall,dim,dim_ens,element,state,ens,hist)
```

DESCRIPTION:

This function 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 prepared before the first call and filled with zeros externally.

ARGUMENTS:

```
INPUTS:   ncall               number of calls to tool
          dim                 state dimension
          dim_ens             ensemble size
          element             element of vector used for histogram
                              (If element=0, all elements are used)
          state(dim)          state vector
          ens(dim, dim_ens)   state ensemble
IN/OUT:   hist(dim_ens+1)     histogram about the state
OUTPUTS:  delta               deviation measure from flat histogram
```

COMMENT:
For an example on using `computehistogram` see `example_computehistogram`.

5.1.4 `computeRCRV` — Compute the bias and the dispersion of the RCRV (Source File: computeRCRV.m)

INTERFACE:

```
[b,d]=computeRCRV(ens,ana,sig0,missing,m,nens)
```

DESCRIPTION:

Computed is the Random Centered Random Variable. It checks the reliability as the indistinguishability between the observations and the ensemble members.

Method: Computed are the Mean and Standard Deviation of the variable:

$$y = \frac{(o-m)}{\sqrt{(\sigma^2+\sigma_o^2)}}$$

with o: observations, m: ensemble mean, σ: ensemble spread, σ_o: observation error

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)

ARGUMENTS:

```
INPUTS:  m                  Size of verification set
         nens               Size of each ensemble
         ens(m,nens)        Ensemble of anomalies
         ana(m)             Analysis (or observation) of anomaly
         sig0               obs. error
         missing(m)         0 : the observation is OK
                            1 : the observation should not be used

OUTPUTS: b                  mean of y
         d                  standard deviation of y
```

COMMENT:

For an example on using `computeRCRV` see `example_computeRCRV`.

5.1.5 `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:

$$MI = \int (RE * p(y))\, dy \tag{5.1}$$

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.

$$MI = \sum_{i=1}^{M} (RE_i * p(y)_i) * dy. \tag{5.2}$$

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

$$MI = \sum_{i} (RE_i * p(y)_i) \tag{5.3}$$

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 particle 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.

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

INTERFACE:

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

DESCRIPTION:
This script calculates the 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

$$\int p(x|y) \ln \left[\frac{p(x|y)}{p(x)} \right] dx \ . \tag{5.4}$$

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:

$$RE \approx \sum post_w \ln \left(\frac{post\_w}{prior\_w} \right) \tag{5.5}$$

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.

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

INTERFACE:

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

DESCRIPTION:
This function calculates the 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.

5.2 Perturbation Tools

5.2.1 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.

5.3 Transformation Tools

5.3.1 `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.

5.3.2 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.

5.3.3 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.

5.4 Utilities

5.4.1 `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

5.5 Analysis Steps

5.5.1 `sangoma_ensemble_analysis` – Computes the analysis ensemble using various ensemble schemes (Source File: mod_ensemble_analysis.m)

INTERFACE:

```
[Xa,xa] = sangoma_ensemble_analysis(Xf,H,y,R,method,...)
```

DESCRIPTION:

Computes analysis ensemble Xa based on forecast ensemble Xf and observations y using various ensemble scheme.

INPUT ARGUMENTS:

```
Xf: forecast ensemble (n x N)
y: observations (m x 1)
H: operator (m x n) (see also below)
R: observation error covariance  (m x m).
method: Method can be the string 'EnSRF', 'EAKF', 'ETKF', 'ETKF2', 'SEIK',
   'ESTKF' or 'EnKF'.
   'ETKF': use SVD decomposition of invTTt (see Sangoma D3.1)
   'ETKF2': use eigendecomposition decomposition of invTTt (see Hunt et al.,
   2007)
```

OPTIONAL INPUT ARGUMENTS:

```
'debug',debug: set to 1 to enable debugging. Default (0) is no debugging.
'tolerance', tolerance: expected rounding error (default 1e-10) for debugging
   checks. This is not used if debug is 0.
'HXf': if non empty, then it is the product H Xf. In this case, H is not
used (except for EnSRF).
```

OUTPUT ARGUMENTS:

```
Xa: the analysis ensemble (n x N)
xa: the analysis ensemble mean (n x 1)
```

COMMENT:

For an example on using this function see `example_ensemble_analysis`.

5.5.2 `sangoma_local_ensemble_analysis` – Computes the local analysis ensemble using various ensemble schemes (Source File: sangoma_local_ensemble_analysis.m)

INTERFACE:

```
[Xa,xa] = sangoma_local_ensemble_analysis(...
   Xf,H,y,diagR,part,selectObs,method,...)
```

DESCRIPTION:
Computes analysis ensemble Xa based on forecast ensemble Xf using the observation y.

INPUT ARGUMENTS:

```
Xf: forecast ensemble (n x N)
H: observation operator (m x n)
y: observation (m x 1)
diagR: diagonal of the observation error covariance R (m x 1)
part: vector of integer "labels". Every element of the state vector with the
  same number belong to the same subdomain
selectObs: callback routine to select observations with a within a subdomain.
  As input is takes an integer representing the index of the state vector and
  returns a vector of weights (m x 1).
  For example:
     selectObs = @(i) exp(- ((x(i) - xobs(:)).^2 + (y(i) - yobs(:)).^2)/L^2 );
  or
     selectObs = @(i) sangoma_compact_locfun(L,...
         sqrt((x(i) - xobs(:)).^2 + (y(i) - yobs(:)).^2));

  where:
     x and y is the horizontal model grid
     xobs and yobs are the localtion of the observations
     L is a correlation length-scale

method: method is one analysis schemes implemented sangoma_ensemble_analysis
  (except for EnSRF)
```

OPTIONAL INPUT ARGUMENTS:

```
'display', display: if true, then display progress (false is the default)
'minweight', minweight: analysis is performed using observations for which
   weights is larger than minweight. (default 1e-8)
'HXf', HXf: if non empty, then it is the product H Xf. In this case, H is not
   used
```

OUTPUT ARGUMENTS:

```
 Xa: the analysis ensemble (n x N)
 xa: the analysis ensemble mean (n x 1)
```

COMMENT:

`example_sangoma_local_ensemble_analysis.m` shows an example for using this function.

5.5.3 `sangoma_local_EnKF` – Computes the local analysis ensemble using the EnKF (Source File: sangoma_local_EnKF.m)

INTERFACE:

```
[Xa,xa] = sangoma_local_EnKF(Xf,HXf,y,R,rho_PH,rho_HPH,...)
```

DESCRIPTION:

Compute the local stochastic EnKF analysis obtained by covariance localization. It uses the element-wise multiplication of $\mathbf{P}^f\mathbf{H}^T$ and $\mathbf{H}\mathbf{P}^f\mathbf{H}^T$ by a localization function (rho_PH and rho_HPH respectively).

INPUT ARGUMENTS:

```
Xf: forecast ensemble (n x N)
HXf: observed part of the forecast ensemble (m x N)
y: observation (m x 1)
R: observation error covariance R (m x m)
rho_PH: localization function of Pf * H'
rho_HPH: localization function of H * Pf * H'
  For example:
    rho_PH = @(i,j) sangoma_compact_locfun(...
         L,sqrt((x(i) - xobs(j)).^2 + (y(i) - yobs(j)).^2));

    rho_HPH = @(i,j) sangoma_compact_locfun(...
         L,sqrt((xobs(i) - xobs(j)).^2 + (yobs(i) - yobs(j)).^2));

  where:
     x and y is the horizontal model grid
     xobs and yobs are the localtion of the observations
     L is a correlation length-scale
```

OPTIONAL INPUT ARGUMENTS:

```
Yp: ensemble of observation perturbations to be added to y (n x N)
```

OUTPUT ARGUMENTS:

```
Xa: the analysis ensemble (n x N)
xa: the analysis ensemble mean (n x 1)
```

COMMENT:

For an example on using this function see `example_sangoma_local_ EnKF.m`.