TMB: AD computation with Template Model Builder

20130921

Generated by Doxygen 1.8.7

Wed May 28 2014 20:26:01

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

A particular multivariate normal distribution is implemented as a templated C++ class. Let us take the generic zero-mean multivariate normal distribution `MVNORM_t` with covariance matrix `Sigma` as an example. The `_t` symbol attached to the class name reminds us that we are dealing with a class (of a particular C++ type). There are two operations that we can do on objects from the `MVNORM_t` class:

- Declare and initialize in terms of one or more parameters (e.g. `Sigma`)
- Evaluate the negative log-likelihood density at specified point (e.g. a vector `u`)

An example is 17

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Module Documentation

6.1 Getting started

A TMB projects consists of an R file (∗.R) and a C++ file (∗.cpp). The R file does pre- and post processing of data in addition to maximizing the log-likelihood contained in ∗.cpp. See TMB examples for more details. All R functions are documented within the standard help system i R. This tutorial describes how to write the C++ file, and assumes familiarity with C++ and to some extent with R.

The purpose of the C++ program is to evaluate the objective function, i.e. the negative log-likelihood of the model. The program is compiled and called from R, where it can be fed to a function minimizer like nlminb().

The objective function should be of the following C++ type:

```
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
.... Here goes your C++ code .....
}
```

The first line includes the source code for the whole TMB package (and all its dependencies). The objective function is a templated class where <Type> is the data type of both the input values and the return value of the objective function. This allows us to evaluate both the objective function and its derivatives using the same chunk of C++ code (and the AD package CppAD). The technical aspects of this are hidden from the user. There is however one aspect that surprises the new TMB user. When a constant like "1.2" is used in a calculation that affects the return value it must be "cast" to Type:

```
Type nll;            // Define variable that holds the return value (neg. log. lik)
nll = Type(1.2);     // Assign value 1.2; a cast is needed.
```

Obtaining data and parameter values from R

Obviously, we will need to pass both data and parameter values to the objective function. This is done through a set of macros that TMB defines for us. To see which macros are available start typing DATA_ or PARAMETER_ in the Doxygen search field of your browser (you may need to refresh the browser window between each time you make a new search). A simple example if you want to read a vector of numbers (doubles) is the following

```
DATA_INTEGER(n);             // Length of "x"
DATA_VECTOR(x);              // Vector x(0),x(1),...,x(n-1)
```

Note that all vectors and matrices in TMB uses a **zero-based** indexing scheme. It is not necessary to explicitely pass the dimension of "x", but is often convenient. The dimension of x is set on the R side when the C++ program is called, and there are ways of retrieving the dimenson of x inside the C++ program.

An extended C++ language

TMB extends C++ with functionality that is important for formulating likelihood functions. You have three toolboxes available:

- Standard C++ used for infrastructure like loops etc.
- Vector, matrix and array library (see Matrices and arrays)
- Probability distributions (see Probability distributions)

In addition to the variables defined through the `DATA_` or `PARAMETER_` macros there can be "local" variables, for which ordinary C++ scoping rules apply. There must also be a variable that holds the return value (neg. log. likelihood).

```
DATA_VECTOR(x);                 // Vector x(0),x(1),...,x(n-1)
Type tmp = x(1);
Type nll=tmp*tmp;
```

As in ordinary C++ local variable tmp must be assign a value before it can enter into a calculation.

Statistical modelling

TMB can handle complex statistical problems with hierarchical structure (latent random variables) and multiple data sources. Latent random variables must be continuous (discrete distributions are not handled). The `PARAMETER_` macros are used to pass two types of parameters.

- **Parameters:** to be estimated by maximum likelihood. These include fixed effects and variance components in the mixed model litterature. They will also correspond to hyper parameters with non-informative priors in the Bayesian literature.
- **Latent random variables:** to be integrated out of the likelihood using a Laplace approximation.

Which of these are chosen is controlled from R, and is not specified in C++. However, for a latent random variable it is usually necessary to assign a probablity distribution, which is done in C++ file.

The purpose of the C++ program is to calculate the (negative) joint density of data and latent random variables. Each datum and individual latent random gives a contribution to log likelihood, which may be though of as a "distribution allignment" by users familiar with software in the BUGS family.

```
PARAMETER_VECTOR(u);            // Latent random variable
Type nll = Type(0);             // Return value
nll -= dnorm(u(0),0,1)          // Distributional assignment: u(0) ~ N(0,1)
```

The following rules apply:

- Distribution assignments do not need to take place before the latent variable is used in a calculation.
- More complicated distributionional assigments are allowed, say u(0)-u(1) $\sim$ N(0,1), but this requires the user to have a deeper understanding of the probabilistic aspects of the model.
- For latent variables only normal distributions should be used (otherwise the Laplace approximation will perform poorly).
- The library Probability distributions contains many probability distributions, especially multivariate normal distributions. For probability distributions not contained in the library, the user can use raw C++ code. Due to the above rule that latent variables shold be normally distributed this is only relevant for the response distribution.

```
DATA_VECTOR(y);                 // Data vector
Type nll = Type(0);             // Return value
nll -= log(Type(0.5)) - abs(y(0));    // y(0) has a Laplace distribution
```

See TMB examples for more examples

6.2 Matrices and arrays

6.2.1 Relationship to R

In R you can apply both matrix multiplication ("%∗%") and elementwise multiplication ("∗") to objects of type "matrix", i.e. it is the operator that determines the operation. In TMB we instead have two different types of objects, while the multiplication operator "∗" is the same:

- `matrix`: linear algebra
- `array`: elementwise operations; () and [] style indexing.
- `vector`: can be used in linear algebra with `matrix`, but at the same time admits R style element-wise operations.

See matrix_arrays.cpp for examples of use.

6.2.2 Relationship to Eigen

The TMB types `matrix` and `array` inherits from the the Eigen types Matrix and Array. The advanced user of TMB will befinit from familiarity with the `Eigen documentation`.

6.3 Probability distributions

TMB contains several classes of probability distributions. These are organized into C++ name spaces as shown below.

Namespaces

- density

  Namespace to construct multivariate Gaussian distributions via C++ templates
  A particular multivariate normal distribution is implemented as a templated C++ class. Let us take the generic zero-mean multivariate normal distribution `MVNORM_t` *with covariance matrix* `Sigma` *as an example. The* `_t` *symbol attached to the class name reminds us that we are dealing with a class (of a particular C++ type). There are two operations that we can do on objects from the* `MVNORM_t` *class:*

  - *Declare and initialize in terms of one or more parameters (e.g.* `Sigma`*)*
  - *Evaluate the negative log-likelihood density at specified point (e.g. a vector* `u`*)*

  An example is.

6.3.1 Detailed Description

TMB contains several classes of probability distributions. These are organized into C++ name spaces as shown below.

6.4 TMB examples

For a list of all examples please click on the "Examples" tab on the top of the page.

Simple example:

```
// Normal linear mixed model specified through sparse design matrices.
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(x);          // Observations
  DATA_SPARSE_MATRIX(B);   // Random effect design matrix
  DATA_SPARSE_MATRIX(A);   // Fixed effect design matrix
  PARAMETER_VECTOR(u);     // Random effects vector
  PARAMETER_VECTOR(beta);  // Fixed effects vector
  PARAMETER(logsdu);       // Random effect standard deviations
  PARAMETER(logsd0);       // Measurement standard deviation

  // Distribution of random effect (u):
  Type ans=0;
  ans-=dnorm(u, Type(0) /*mean*/, exp(logsdu) /*sd*/, 1 /*log?*/).sum();

  // Distribution of obs given random effects (x|u):
  vector<Type> y=A*beta+B*u;
  ans-=dnorm(x,y,exp(logsd0),1).sum();

  return ans;
}
```

Chapter 7

Namespace Documentation

7.1 density Namespace Reference

Namespace to construct multivariate Gaussian distributions via C++ templates

A particular multivariate normal distribution is implemented as a templated C++ class. Let us take the generic zero-mean multivariate normal distribution `MVNORM_t` with covariance matrix `Sigma` as an example. The `_t` symbol attached to the class name reminds us that we are dealing with a class (of a particular C++ type). There are two operations that we can do on objects from the `MVNORM_t` class:

- Declare and initialize in terms of one or more parameters (e.g. `Sigma`)
- Evaluate the negative log-likelihood density at specified point (e.g. a vector `u`)

An example is.

Classes

- class AR1_t
  Stationary AR1 process.
- class ARk_t
  Stationary AR(k) process.
- class contAR2_t
  Continuous AR(2) process.
- class GMRF_t
  Gaussian Markov Random Field.
- class MVNORM_t
  Multivariate normal distribution with user supplied covariance matrix.
- class N01
  Standardized normal distribution.
- class PROJ_t
  Projection of multivariate gaussian variable.
- class SCALE_t
  Apply scale transformation on a density.
- class SEPARABLE_t
  Separable extension of two densitites.
- class UNSTRUCTURED_CORR_t
  Multivariate normal distribution with unstructered correlation matrix.
- class VECSCALE_t
  Apply a vector scale transformation on a density.

Functions

- template<class scalartype >
  MVNORM_t< scalartype > MVNORM (matrix< scalartype > x)
- template<class scalartype >
  UNSTRUCTURED_CORR_t< scalartype > UNSTRUCTURED_CORR (vector< scalartype > x)
- template<class scalartype , class distribution >
  AR1_t< distribution > AR1 (scalartype phi_, distribution f_)
- template<class scalartype >
  AR1_t< N01< scalartype > > AR1 (scalartype phi_)
- template<class scalartype , class vectortype >
  contAR2_t< scalartype > contAR2 (vectortype grid_, scalartype shape_, scalartype scale_=1)
- template<class scalartype >
  contAR2_t< scalartype > contAR2 (scalartype shape_, scalartype scale_=1)
- template<class scalartype >
  GMRF_t< scalartype > GMRF (Eigen::SparseMatrix< scalartype > Q, int order=1)
- template<class scalartype , class arraytype >
  GMRF_t< scalartype > GMRF (arraytype x, vector< scalartype > delta, int order=1)
- template<class scalartype , class arraytype >
  GMRF_t< scalartype > GMRF (arraytype x, scalartype delta, int order=1)
- template<class scalartype , class distribution >
  SCALE_t< distribution > SCALE (distribution f_, scalartype scale_)
- template<class vectortype , class distribution >
  VECSCALE_t< distribution > VECSCALE (distribution f_, vectortype scale_)
- template<class distribution1 , class distribution2 >
  SEPARABLE_t< distribution1,
  distribution2 > SEPARABLE (distribution1 f_, distribution2 g_)
- template<class distribution >
  PROJ_t< distribution > PROJ (distribution f_, vector< int > i)

7.1.1 Detailed Description

Namespace to construct multivariate Gaussian distributions via C++ templates

A particular multivariate normal distribution is implemented as a templated C++ class. Let us take the generic zero-mean multivariate normal distribution `MVNORM_t` with covariance matrix `Sigma` as an example. The `_t` symbol attached to the class name reminds us that we are dealing with a class (of a particular C++ type). There are two operations that we can do on objects from the `MVNORM_t` class:

- Declare and initialize in terms of one or more parameters (e.g. `Sigma`)
- Evaluate the negative log-likelihood density at specified point (e.g. a vector `u`)

An example is.

```
// Sigma and u are objects that have allready been defined
MVNORM_t<Type> neg_log_density(Sigma);        // Create object from covariance matrix Sigma
Type ans = neg_log_density(u);                // Evaluate neg. log-likelihood
```

Comments:

- The template argument `<Type>` must always be included (as in many other places in TMB) but can be ignored from a user perspective.
- The object vi define here is called `neg_log_density`. You can choose whatever name you like, but `neg_log_density` reminds you that the only thing you will do with it is to evaluate the negative log-likelihood.
- The dimensions of `Sigma` and `u` must match.

New classes of distributions can be built recursively from existing distributions. The behaviour of the different distributions differ in this respect. A very useful example is the Kronecker product (http://en.wikipedia.org/wiki/Kronecker_product) of two multivariate normal distributions (see detailed description of SEPARABLE_t):

```
MVNORM_t<Type> Gauss1(Sigma1);        // First normal distribution (with covariance Sigma1)
MVNORM_t<Type> Gauss2(Sigma2);        // Second normal distribution (with covariance Sigma2)
SEPARABLE_t<MVNORM_t<Type> , MVNORM_t<Type> > Gauss3(Gauss1,Gauss2)
    ;
SEPARABLE_t<MVNORM_t<Type>, MVNORM_t<Type>> Gauss3(Gauss1,Gauss2);
Type ans = Gauss3(u);         // Evaluate neg. log-likelihood
```

where u must be of appropriate dimension.

7.1.2 Function Documentation

7.1.2.1 template<class scalartype , class distribution > AR1_t<distribution> density::AR1 (scalartype *phi_*, distribution *f_*)

Definition at line 298 of file tmbutils.cpp.

7.1.2.2 template<class scalartype > AR1_t<N01<scalartype> > density::AR1 (scalartype *phi_*)

Definition at line 302 of file tmbutils.cpp.

7.1.2.3 template<class scalartype , class vectortype > contAR2_t<scalartype> density::contAR2 (vectortype *grid_*, scalartype *shape_*, scalartype *scale_* = 1)

Definition at line 565 of file tmbutils.cpp.

7.1.2.4 template<class scalartype > contAR2_t<scalartype> density::contAR2 (scalartype *shape_*, scalartype *scale_* = 1)

Definition at line 569 of file tmbutils.cpp.

7.1.2.5 template<class scalartype > GMRF_t<scalartype> density::GMRF (Eigen::SparseMatrix< scalartype > *Q*, int *order* = 1)

Definition at line 696 of file tmbutils.cpp.

7.1.2.6 template<class scalartype , class arraytype > GMRF_t<scalartype> density::GMRF (arraytype *x*, vector< scalartype > *delta*, int *order* = 1)

Definition at line 700 of file tmbutils.cpp.

7.1.2.7 template<class scalartype , class arraytype > GMRF_t<scalartype> density::GMRF (arraytype *x*, scalartype *delta*, int *order* = 1)

Definition at line 704 of file tmbutils.cpp.

7.1.2.8 template<class scalartype > MVNORM_t<scalartype> density::MVNORM (matrix< scalartype > *x*)

Definition at line 109 of file tmbutils.cpp.

7.1.2.9 template<class distribution > PROJ_t<distribution> density::PROJ (distribution *f_*, vector< int > *i*)

Definition at line 1081 of file tmbutils.cpp.

7.1.2.10 template<class scalartype , class distribution > SCALE_t<distribution> density::SCALE (distribution *f_*, scalartype *scale_*)

Definition at line 741 of file tmbutils.cpp.

7.1.2.11 template<class distribution1 , class distribution2 > SEPARABLE_t<distribution1,distribution2> density::SEPARABLE (distribution1 *f_*, distribution2 *g_*)

Definition at line 919 of file tmbutils.cpp.

7.1.2.12 template<class scalartype > UNSTRUCTURED_CORR_t<scalartype> density::UNSTRUCTURED_CORR (vector< scalartype > *x*)

Definition at line 177 of file tmbutils.cpp.

7.1.2.13 template<class vectortype , class distribution > VECSCALE_t<distribution> density::VECSCALE (distribution *f_*, vectortype *scale_*)

Definition at line 780 of file tmbutils.cpp.

7.2 tmbutils Namespace Reference

Classes

- struct array

  Array class used by TMB.
- struct matexp

  Matrix exponential: matrix of arbitrary dimension.
- struct matexp< scalartype, 2 >

  Matrix exponential: 2x2 case which can be handled efficiently.
- struct matrix

  Matrix class used by TMB.
- class order
- class splinefun
- struct vector

  Vector class used by TMB.

Functions

- template<class Type >
  Eigen::SparseMatrix< Type > asSparseMatrix (SEXP M)
- template<class Type >
  Eigen::SparseMatrix< Type > asSparseMatrix (matrix< Type > x)
- template<class Type >
  Eigen::SparseVector< Type > asSparseVector (vector< Type > x)
- template<class Type >
  Eigen::SparseMatrix< Type > kronecker (Eigen::SparseMatrix< Type > x, Eigen::SparseMatrix< Type > y)

- template<class Type >
  matrix< Type > discrLyap (matrix< Type > A_)
- template<class Type >
  matrix< Type > invertSparseMatrix (matrix< Type > A_)
- template<class scalartype , int n1, int n2, int n3, int n4>
  Matrix< scalartype, n1 *n3, n2
  *n4 > kronecker (Matrix< scalartype, n1, n2 > x, Matrix< scalartype, n3, n4 > y)

  Kronecker product of two matrices.
- template<class Type , class T1 , class T2 >
  vector< Type > dnorm (vector< Type > x, T1 mean, T2 sd, int give_log=0)
- template<class Type , class From >
  vector< Type > asVector (From *px, int n)

7.2.1 Function Documentation

7.2.1.1 template<class Type > Eigen::SparseMatrix<Type> tmbutils::asSparseMatrix (SEXP *M*)

Create sparse matrix from R-triplet sparse matrix

Definition at line 8 of file tmbutils.cpp.

7.2.1.2 template<class Type > Eigen::SparseMatrix<Type> tmbutils::asSparseMatrix (matrix< Type > *x*)

Create sparse matrix from dense matrix

Definition at line 26 of file tmbutils.cpp.

7.2.1.3 template<class Type > Eigen::SparseVector<Type> tmbutils::asSparseVector (vector< Type > *x*)

Create sparse vector from dense vector

Definition at line 39 of file tmbutils.cpp.

7.2.1.4 template<class Type , class From > vector<Type> tmbutils::asVector (From * *px,* int *n*)

Definition at line 26 of file tmbutils.cpp.

7.2.1.5 template<class Type > matrix<Type> tmbutils::discrLyap (matrix< Type > *A_*)

Solve discrete Lyapunov equation V=AVA'+I

Definition at line 77 of file tmbutils.cpp.

7.2.1.6 template<class Type , class T1 , class T2 > vector<Type> tmbutils::dnorm (vector< Type > *x,* T1 *mean,* T2 *sd,* int *give_log* = 0)

Definition at line 17 of file tmbutils.cpp.

7.2.1.7 template<class Type > matrix<Type> tmbutils::invertSparseMatrix (matrix< Type > *A_*)

Inverse of PD sparse matrix

Definition at line 102 of file tmbutils.cpp.

7.2.1.8 template<class scalartype , int n1, int n2, int n3, int n4> Matrix<scalartype,n1*n3,n2*n4> tmbutils::kronecker (Matrix< scalartype, n1, n2 > *x*, Matrix< scalartype, n3, n4 > *y*)

Kronecker product of two matrices.

Definition at line 8 of file tmbutils.cpp.

7.2.1.9 template<class Type > Eigen::SparseMatrix<Type> tmbutils::kronecker (Eigen::SparseMatrix< Type > *x*, Eigen::SparseMatrix< Type > *y*)

Kronecker product of two sparse matrices

Definition at line 50 of file tmbutils.cpp.

Chapter 8

Class Documentation

8.1 AR1_t< distribution > Class Template Reference

Stationary AR1 process.

Public Member Functions

- AR1_t ()
- AR1_t (scalartype phi_, distribution f_)
- scalartype operator() (vectortype x)

  Evaluate the negative log density.
- scalartype operator() (arraytype x)
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (typename distribution::scalartype)

Private Attributes

- scalartype phi
- distribution MARGINAL

8.1.1 Detailed Description

template<class distribution>class AR1_t< distribution >

Stationary AR1 process.

Class to evaluate the negative log density of a (multivariate) AR1 process with parameter phi and given marginal distribution.

Parameters

phi	Scalar -1<phi<1

Template Parameters

MARGINAL	The desired (multivariate) marginal distribution.

Let $f(x)$ denote a multivariate Gaussian mean-zero negative log density represented by its covariance matrix Σ. Define recursively the vectors

$$x_0 \sim N(0,\Sigma)$$

$$x_1 = \phi x_0 + \varepsilon_1\,,\ \ \varepsilon_1 \sim N(0,\sigma\Sigma)$$

$$x_i = \phi x_{i-1} + \varepsilon_i\,,\ \ \varepsilon_i \sim N(0,\sigma\Sigma)$$

where $\sigma = \sqrt{1-\phi^2}$. Then $E(x_i) = 0$, $V(x_i) = \Sigma$ and the covariance is $E(x_i x_j') = \phi^{|i-j|}\Sigma$. We refer to this process as a stationary 1st order autoregressive process with multivariate increments with parameter phi and marginal distribution f. Compactly denoted AR1(phi,f).

Note that the construction can be carried out recursively, as "AR1(phi,f)" is itself a distribution that can be used as input to AR1(). See example below:

```
\\ Construct negative log density of standard AR1 process on a line:
Type phi1=0.8;
AR1_t<N01<Type> > f1(phi1);
\\ Can be evaluated on a vector:
vector<Type> x(10);
Type ans=f1(x);
```

Now use f1 as marginal in a new AR1 process with parameter phi2:

```
\\ Construct negative log density of standard AR1 process on a line:
Type phi2=0.5;
AR1_t<AR1_t<N01<Type> > > f2(phi1,f1);
\\ Can be evaluated on a 2-dimensional array:
vector<Type> x(10,20);
Type ans=f2(x);
```

Definition at line 245 of file density.cpp.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 template<class distribution> AR1_t< distribution >::AR1_t ( ) [inline]

Definition at line 251 of file density.cpp.

8.1.2.2 template<class distribution> AR1_t< distribution >::AR1_t (scalartype *phi_*, distribution *f_*) [inline]

Definition at line 252 of file density.cpp.

8.1.3 Member Function Documentation

8.1.3.1 template<class distribution> arraytype AR1_t< distribution >::jacobian (arraytype *x*) [inline]

Definition at line 280 of file density.cpp.

8.1.3.2 template<class distribution> int AR1_t< distribution >::ndim ( ) [inline]

Definition at line 293 of file density.cpp.

8.1.3.3 template<class distribution> scalartype AR1_t< distribution >::operator() (vectortype *x*) [inline]

Evaluate the negative log density.

Definition at line 254 of file density.cpp.

8.1.3.4 template<class distribution> scalartype AR1_t< distribution >::operator() (arraytype *x*) [inline]

Definition at line 267 of file density.cpp.

8.1.3.5 template<class distribution> AR1_t< distribution >::TYPEDEFS (typename distribution::scalartype) [private]

8.1.4 Member Data Documentation

8.1.4.1 template<class distribution> distribution AR1_t< distribution >::MARGINAL [private]

Definition at line 249 of file density.cpp.

Referenced by AR1_t< distribution >::AR1_t(), AR1_t< distribution >::jacobian(), and AR1_t< distribution >::operator()().

8.1.4.2 template<class distribution> scalartype AR1_t< distribution >::phi [private]

Definition at line 248 of file density.cpp.

Referenced by AR1_t< distribution >::AR1_t(), AR1_t< distribution >::jacobian(), and AR1_t< distribution >::operator()().

8.1.4.3 template<class distribution> AR1_t< distribution >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 294 of file density.cpp.

The documentation for this class was generated from the following file:

- density.cpp

8.2 density::AR1_t< distribution > Class Template Reference

Stationary AR1 process.

Public Member Functions

- AR1_t ()
- AR1_t (scalartype phi_, distribution f_)
- scalartype operator() (vectortype x)
  Evaluate the negative log density.
- scalartype operator() (arraytype x)
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (typename distribution::scalartype)

Private Attributes

- scalartype phi
- distribution MARGINAL

8.2.1 Detailed Description

template<class distribution>class density::AR1_t< distribution >

Stationary AR1 process.

Class to evaluate the negative log density of a (multivariate) AR1 process with parameter phi and given marginal distribution.

Parameters

phi	Scalar -1<phi<1

Template Parameters

MARGINAL	The desired (multivariate) marginal distribution.

Let $f(x)$ denote a multivariate Gaussian mean-zero negative log density represented by its covariance matrix Σ. Define recursively the vectors

$$x_0 \sim N(0,\Sigma)$$
$$x_1 = \phi x_0 + \varepsilon_1\,,\ \ \varepsilon_1 \sim N(0,\sigma\Sigma)$$
$$x_i = \phi x_{i-1} + \varepsilon_i\,,\ \ \varepsilon_i \sim N(0,\sigma\Sigma)$$

where $\sigma = \sqrt{1-\phi^2}$. Then $E(x_i) = 0$, $V(x_i) = \Sigma$ and the covariance is $E(x_i x_j') = \phi^{|i-j|}\Sigma$. We refer to this process as a stationary 1st order autoregressive process with multivariate increments with parameter phi and marginal distribution f. Compactly denoted AR1(phi,f).

Note that the construction can be carried out recursively, as "AR1(phi,f)" is itself a distribution that can be used as input to AR1(). See example below:

```
\\ Construct negative log density of standard AR1 process on a line:
Type phi1=0.8;
AR1_t<N01<Type> > f1(phi1);
\\ Can be evaluated on a vector:
vector<Type> x(10);
Type ans=f1(x);
```

Now use f1 as marginal in a new AR1 process with parameter phi2:

```
\\ Construct negative log density of standard AR1 process on a line:
Type phi2=0.5;
AR1_t<AR1_t<N01<Type> > > f2(phi1,f1);
\\ Can be evaluated on a 2-dimensional array:
vector<Type> x(10,20);
Type ans=f2(x);
```

Definition at line 246 of file tmbutils.cpp.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 template<class distribution > density::AR1_t< distribution >::AR1_t ( ) [inline]

Definition at line 252 of file tmbutils.cpp.

8.2.2.2 template<class distribution > density::AR1_t< distribution >::AR1_t (scalartype *phi_*, distribution *f_*) [inline]

Definition at line 253 of file tmbutils.cpp.

8.2.3 Member Function Documentation

8.2.3.1 template<class distribution > arraytype density::AR1_t< distribution >::jacobian (arraytype *x*) [inline]

Definition at line 281 of file tmbutils.cpp.

8.2.3.2 template<class distribution > int density::AR1_t< distribution >::ndim ( ) [inline]

Definition at line 294 of file tmbutils.cpp.

8.2.3.3 template<class distribution > scalartype density::AR1_t< distribution >::operator() (vectortype *x*) [inline]

Evaluate the negative log density.

Definition at line 255 of file tmbutils.cpp.

8.2.3.4 template<class distribution > scalartype density::AR1_t< distribution >::operator() (arraytype *x*) [inline]

Definition at line 268 of file tmbutils.cpp.

8.2.3.5 template<class distribution > density::AR1_t< distribution >::TYPEDEFS (typename distribution::scalartype) [private]

8.2.4 Member Data Documentation

8.2.4.1 template<class distribution > distribution density::AR1_t< distribution >::MARGINAL [private]

Definition at line 250 of file tmbutils.cpp.

8.2.4.2 template<class distribution > scalartype density::AR1_t< distribution >::phi [private]

Definition at line 249 of file tmbutils.cpp.

8.2.4.3 template<class distribution > density::AR1_t< distribution >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 295 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.3 ARk_t< scalartype_ > Class Template Reference

Stationary AR(k) process.

Public Member Functions

- ARk_t ()
- ARk_t (vectortype phi_)
- vectortype cov (int n)

  Covariance extractor. Run Youle-Walker recursions and return a vector of length n representing the auto-covariance function.
- scalartype operator() (vectortype x)

  Evaluate the negative log density.
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (scalartype_)

Private Attributes

- int k
- vectortype phi
- vectortype gamma
- matrixtype V0
- matrixtype Q0
- matrixtype M
- matrixtype I
- scalartype sigma
- scalartype logdetQ0

8.3.1 Detailed Description

template<class scalartype_>class ARk_t< scalartype_ >

Stationary AR(k) process.

Parameters

*phi_*	Vector of length k with parameters.

```
Class to evaluate the negative log density of a stationary
AR(k)-process with parameter vector phi=[phi_1,...,phi_k]:

x[t]=phi_1*x[t-1]+...+phi_k*x[t-k]+eps[t]

where eps[t]~N(0,sigma^2). The parameter sigma^2 is chosen to
obtain V(x[t])=1 so that the class actually specifies a correlation
model.

Examples: ARk(phi)  <-- simple mean zero variance 1 AR(k) process.

Steady state initial distribution is found by (e.g. k=3)

[gamma(1)]     [gamma(0) gamma(1) gamma(2)]      [phi1]
[ ....   ] =   [gamma(1) gamma(0) gamma(1)]  *   [phi2]
[gamma(3)]     [gamma(2) gamma(1) gamma(0)]      [phi3]
```

Definition at line 331 of file density.cpp.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 template<class scalartype_ > ARk_t< scalartype_ >::ARk_t ( ) [inline]

Definition at line 346 of file density.cpp.

8.3.2.2 template<class scalartype_ > ARk_t< scalartype_ >::ARk_t (vectortype *phi_*) [inline]

Definition at line 347 of file density.cpp.

8.3.3 Member Function Documentation

8.3.3.1 template<class scalartype_ > vectortype ARk_t< scalartype_ >::cov (int *n*) [inline]

Covariance extractor. Run Youle-Walker recursions and return a vector of length n representing the auto-covariance function.

Definition at line 388 of file density.cpp.

8.3.3.2 template<class scalartype_ > arraytype ARk_t< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 420 of file density.cpp.

8.3.3.3 template<class scalartype_ > int ARk_t< scalartype_ >::ndim ( ) [inline]

Definition at line 439 of file density.cpp.

8.3.3.4 template<class scalartype_ > scalartype ARk_t< scalartype_ >::operator() (vectortype *x*) [inline]

Evaluate the negative log density.

Definition at line 403 of file density.cpp.

8.3.3.5 template<class scalartype_ > ARk_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.3.4 Member Data Documentation

8.3.4.1 template<class scalartype_ > vectortype ARk_t< scalartype_ >::gamma [private]

Definition at line 336 of file density.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t(), and ARk_t< scalartype_ >::cov().

8.3.4.2 template<class scalartype_ > matrixtype ARk_t< scalartype_ >::I [private]

Definition at line 342 of file density.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t().

8.3.4.3 template<class scalartype_ > int ARk_t< scalartype_ >::k [private]

Definition at line 334 of file density.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t(), ARk_t< scalartype_ >::cov(), ARk_t< scalartype_ >::jacobian(), and ARk_t< scalartype_ >::operator()().

8.3.4.4 template<class scalartype_ > scalartype ARk_t< scalartype_ >::logdetQ0 [private]

Definition at line 344 of file density.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t(), and ARk_t< scalartype_ >::operator()().

8.3.4.5 template<class scalartype_ > matrixtype ARk_t< scalartype_ >::M [private]

Definition at line 341 of file density.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t().

8.3.4.6 template<class scalartype_ > vectortype ARk_t< scalartype_ >::phi [private]

Definition at line 335 of file density.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t(), ARk_t< scalartype_ >::cov(), ARk_t< scalartype_ >::jacobian(), and ARk_t< scalartype_ >::operator()().

8.3.4.7 template<class scalartype_ > matrixtype ARk_t< scalartype_ >::Q0 [private]

Definition at line 339 of file density.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t(), ARk_t< scalartype_ >::jacobian(), and ARk_t< scalartype_ >↩ ::operator()().

8.3.4.8 template<class scalartype_ > scalartype ARk_t< scalartype_ >::sigma [private]

Definition at line 343 of file density.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t(), ARk_t< scalartype_ >::jacobian(), and ARk_t< scalartype_ >↩ ::operator()().

8.3.4.9 template<class scalartype_ > matrixtype ARk_t< scalartype_ >::V0 [private]

Definition at line 338 of file density.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t().

8.3.4.10 template<class scalartype_ > ARk_t< scalartype_ >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 440 of file density.cpp.

The documentation for this class was generated from the following file:

- density.cpp

8.4 density::ARk_t< scalartype_ > Class Template Reference

Stationary AR(k) process.

Public Member Functions

- ARk_t ()
- ARk_t (vectortype phi_)
- vectortype cov (int n)

  Covariance extractor. Run Youle-Walker recursions and return a vector of length n representing the auto-covariance function.
- scalartype operator() (vectortype x)

  Evaluate the negative log density.
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (scalartype_)

Private Attributes

- int k
- vectortype phi
- vectortype gamma
- matrixtype V0
- matrixtype Q0
- matrixtype M
- matrixtype I
- scalartype sigma
- scalartype logdetQ0

8.4.1 Detailed Description

template<class scalartype_>class density::ARk_t< scalartype_ >

Stationary AR(k) process.

Parameters

*phi_*	Vector of length k with parameters.

```
Class to evaluate the negative log density of a stationary
AR(k)-process with parameter vector phi=[phi_1,...,phi_k]:

x[t]=phi_1*x[t-1]+...+phi_k*x[t-k]+eps[t]

where eps[t]~N(0,sigma^2). The parameter sigma^2 is chosen to
obtain V(x[t])=1 so that the class actually specifies a correlation
model.
```

```
Examples: ARk(phi)  <-- simple mean zero variance 1 AR(k) process.

Steady state initial distribution is found by (e.g. k=3)

[gamma(1)]     [gamma(0) gamma(1) gamma(2)]      [phi1]
[ ....   ] =   [gamma(1) gamma(0) gamma(1)]  *   [phi2]
[gamma(3)]     [gamma(2) gamma(1) gamma(0)]      [phi3]
```

Definition at line 332 of file tmbutils.cpp.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 template<class scalartype_ > density::ARk_t< scalartype_ >::ARk_t ( ) [inline]

Definition at line 347 of file tmbutils.cpp.

8.4.2.2 template<class scalartype_ > density::ARk_t< scalartype_ >::ARk_t (vectortype *phi_*) [inline]

Definition at line 348 of file tmbutils.cpp.

8.4.3 Member Function Documentation

8.4.3.1 template<class scalartype_ > vectortype density::ARk_t< scalartype_ >::cov (int *n*) [inline]

Covariance extractor. Run Youle-Walker recursions and return a vector of length n representing the auto-covariance function.

Definition at line 389 of file tmbutils.cpp.

8.4.3.2 template<class scalartype_ > arraytype density::ARk_t< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 421 of file tmbutils.cpp.

8.4.3.3 template<class scalartype_ > int density::ARk_t< scalartype_ >::ndim ( ) [inline]

Definition at line 440 of file tmbutils.cpp.

8.4.3.4 template<class scalartype_ > scalartype density::ARk_t< scalartype_ >::operator() (vectortype *x*) [inline]

Evaluate the negative log density.

Definition at line 404 of file tmbutils.cpp.

8.4.3.5 template<class scalartype_ > density::ARk_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.4.4 Member Data Documentation

8.4.4.1 template<class scalartype_ > vectortype density::ARk_t< scalartype_ >::gamma [private]

Definition at line 337 of file tmbutils.cpp.

8.4.4.2 template<class scalartype_ > matrixtype density::ARk_t< scalartype_ >::I [private]

Definition at line 343 of file tmbutils.cpp.

8.4.4.3 template<class scalartype_ > int density::ARk_t< scalartype_ >::k [private]

Definition at line 335 of file tmbutils.cpp.

8.4.4.4 template<class scalartype_ > scalartype density::ARk_t< scalartype_ >::logdetQ0 [private]

Definition at line 345 of file tmbutils.cpp.

8.4.4.5 template<class scalartype_ > matrixtype density::ARk_t< scalartype_ >::M [private]

Definition at line 342 of file tmbutils.cpp.

8.4.4.6 template<class scalartype_ > vectortype density::ARk_t< scalartype_ >::phi [private]

Definition at line 336 of file tmbutils.cpp.

8.4.4.7 template<class scalartype_ > matrixtype density::ARk_t< scalartype_ >::Q0 [private]

Definition at line 340 of file tmbutils.cpp.

8.4.4.8 template<class scalartype_ > scalartype density::ARk_t< scalartype_ >::sigma [private]

Definition at line 344 of file tmbutils.cpp.

8.4.4.9 template<class scalartype_ > matrixtype density::ARk_t< scalartype_ >::V0 [private]

Definition at line 339 of file tmbutils.cpp.

8.4.4.10 template<class scalartype_ > density::ARk_t< scalartype_ >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 441 of file tmbutils.cpp.

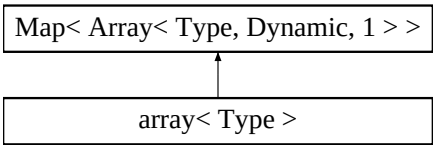
The documentation for this class was generated from the following file:

- tmbutils.cpp

8.5 array< Type > Struct Template Reference

Array class used by TMB.

Inheritance diagram for array< Type >:



Public Types

- typedef Array< Type, Dynamic, 1 > Base
- typedef Map< Base > MapBase

Public Member Functions

- void setdim (vector< int > dim_)
- void initZeroArray (vector< int > dim_)
- vector< int > c (int n1)
- vector< int > c (int n1, int n2)
- vector< int > c (int n1, int n2, int n3)
- vector< int > c (int n1, int n2, int n3, int n4)
- array ()
- array (vector< int > dim_)

  Construct array from dimension vector and fill with zeros.
- array (int n1)
- array (int n1, int n2)
- array (int n1, int n2, int n3)
- array (int n1, int n2, int n3, int n4)
- template<class T >
  array (T &x, vector< int > dim_)
- template<class T >
  array (array< T > &x)
- array (Type *p, vector< int > dim_)
- array & operator= (const array &other)
- void print ()
- int cols ()
- array< Type > col (int i)

  Extract sub-array with write access Index i refers to the outer-most (i.e. final) dimension.
- Type & operator() (int i1)

  Elementwise subsetting 1D array.
- Type & operator() (int i1, int i2)

  Elementwise subsetting 2D array.
- Type & operator() (int i1, int i2, int i3)

  Elementwise subsetting 3D array.
- Type & operator() (int i1, int i2, int i3, int i4)

  Elementwise subsetting 4D array.
- int index (vector< int > tup)
- vector< int > tuple (int i)
- array< Type > perm (vector< int > p)

  Array permutation. Permutes array dimensions corresponding to permutation vector p.
- array< Type > transpose ()

  Array transpose (Special case of array permutation)
- int mod (int i, int n)
- array< Type > rotate (int n)

  Array rotate (Special case of array permutation)

Public Attributes

- vector< int > dim
- vector< int > mult
- Base vectorcopy

8.5.1 Detailed Description

template<class Type>struct array< Type >

Array class used by TMB.

The TMB array class is implemented as an Eigen Array of dynamic length with a dimension attribute. The implementation closely follows the way arrays work in R. Vectorized operations are inherited from the Eigen library.

Examples:

matrix_arrays.cpp.

Definition at line 14 of file array.cpp.

8.5.2 Member Typedef Documentation

8.5.2.1 template<class Type> typedef Array<Type,Dynamic,1> array< Type >::Base

Definition at line 16 of file array.cpp.

8.5.2.2 template<class Type> typedef Map< Base > array< Type >::MapBase

Definition at line 17 of file array.cpp.

8.5.3 Constructor & Destructor Documentation

8.5.3.1 template<class Type> array< Type >::array ( ) [inline]

Definition at line 69 of file array.cpp.

Referenced by array< Type >::col().

8.5.3.2 template<class Type> array< Type >::array (vector< int > *dim_*) [inline]

Construct array from dimension vector and fill with zeros.

Definition at line 72 of file array.cpp.

8.5.3.3 template<class Type> array< Type >::array (int *n1*) [inline]

Definition at line 75 of file array.cpp.

8.5.3.4 template<class Type> array< Type >::array (int *n1*, int *n2*) [inline]

Definition at line 76 of file array.cpp.

8.5.3.5 template<class Type> array< Type >::array (int *n1*, int *n2*, int *n3*) [inline]

Definition at line 77 of file array.cpp.

8.5.3.6 template<class Type> array< Type >::array (int *n1*, int *n2*, int *n3*, int *n4*) [inline]

Definition at line 78 of file array.cpp.

8.5.3.7 template<class Type> template<class T > array< Type >::array (T & *x,* vector< int > *dim_*) [inline]

Definition at line 82 of file array.cpp.

8.5.3.8 template<class Type> template<class T > array< Type >::array (array< T > & *x*) [inline]

Definition at line 90 of file array.cpp.

8.5.3.9 template<class Type> array< Type >::array (Type * *p,* vector< int > *dim_*) [inline]

Definition at line 98 of file array.cpp.

8.5.4 Member Function Documentation

8.5.4.1 template<class Type> vector<int> array< Type >::c (int *n1*) [inline]

Definition at line 46 of file array.cpp.

Referenced by array< Type >::array(), and array< Type >::operator()().

8.5.4.2 template<class Type> vector<int> array< Type >::c (int *n1,* int *n2*) [inline]

Definition at line 51 of file array.cpp.

8.5.4.3 template<class Type> vector<int> array< Type >::c (int *n1,* int *n2,* int *n3*) [inline]

Definition at line 56 of file array.cpp.

8.5.4.4 template<class Type> vector<int> array< Type >::c (int *n1,* int *n2,* int *n3,* int *n4*) [inline]

Definition at line 61 of file array.cpp.

8.5.4.5 template<class Type> array<Type> array< Type >::col (int *i*) [inline]

Extract sub-array with write access Index i refers to the outer-most (i.e. final) dimension.

Definition at line 135 of file array.cpp.

Referenced by contAR2_t< scalartype_ >::operator()().

8.5.4.6 template<class Type> int array< Type >::cols ( ) [inline]

Definition at line 127 of file array.cpp.

Referenced by array< Type >::col().

8.5.4.7 template<class Type> int array< Type >::index (vector< int > *tup*) [inline]

Definition at line 166 of file array.cpp.

Referenced by array< Type >::operator()(), and array< Type >::perm().

8.5.4.8 template<class Type> void array< Type >::initZeroArray (vector< int > *dim_*) [inline]

Definition at line 38 of file array.cpp.

Referenced by array< Type >::array().

8.5.4.9 template<class Type> int array< Type >::mod (int *i*, int *n*) [inline]

Definition at line 201 of file array.cpp.

Referenced by array< Type >::rotate().

8.5.4.10 template<class Type> Type& array< Type >::operator() (int *i1*) [inline]

Elementwise subsetting 1D array.

Definition at line 149 of file array.cpp.

8.5.4.11 template<class Type> Type& array< Type >::operator() (int *i1*, int *i2*) [inline]

Elementwise subsetting 2D array.

Definition at line 153 of file array.cpp.

8.5.4.12 template<class Type> Type& array< Type >::operator() (int *i1*, int *i2*, int *i3*) [inline]

Elementwise subsetting 3D array.

Definition at line 157 of file array.cpp.

8.5.4.13 template<class Type> Type& array< Type >::operator() (int *i1*, int *i2*, int *i3*, int *i4*) [inline]

Elementwise subsetting 4D array.

Definition at line 161 of file array.cpp.

8.5.4.14 template<class Type> array& array< Type >::operator= (const array< Type > & *other*) [inline]

Definition at line 109 of file array.cpp.

8.5.4.15 template<class Type> array<Type> array< Type >::perm (vector< int > *p*) [inline]

Array permutation. Permutes array dimensions corresponding to permutation vector p.

Definition at line 182 of file array.cpp.

Referenced by array< Type >::rotate(), and array< Type >::transpose().

8.5.4.16 template<class Type> void array< Type >::print ( ) [inline]

Definition at line 117 of file array.cpp.

8.5.4.17 template<class Type> array<Type> array< Type >::rotate (int *n*) [inline]

Array rotate (Special case of array permutation)

Rotates array dimension with n steps where n can be any (positive or negative) integer. If e.g. x has dimension [3,4,5,6] then x.rotate(1) has dimension [6,3,4,5].

Definition at line 209 of file array.cpp.

8.5.4.18 template<class Type> void array< Type >::setdim (vector< int > *dim_*) [inline]

Definition at line 30 of file array.cpp.

Referenced by array< Type >::array(), array< Type >::initZeroArray(), and array< Type >::operator=().

8.5.4.19 template<class Type> array<Type> array< Type >::transpose ( ) [inline]

Array transpose (Special case of array permutation)

If e.g. x has dimension [3,4,5,6] then x.transpose() has dimension [6,5,4,3].

Definition at line 195 of file array.cpp.

Referenced by contAR2_t< scalartype_ >::operator()().

8.5.4.20 template<class Type> vector<int> array< Type >::tuple (int *i*) [inline]

Definition at line 169 of file array.cpp.

Referenced by array< Type >::perm().

8.5.5 Member Data Documentation

8.5.5.1 template<class Type> vector<int> array< Type >::dim

Definition at line 19 of file array.cpp.

Referenced by array< Type >::array(), array< Type >::col(), array< Type >::cols(), array< Type >::operator=(), array< Type >::perm(), array< Type >::print(), array< Type >::rotate(), array< Type >::setdim(), array< Type >::transpose(), and array< Type >::tuple().

8.5.5.2 template<class Type> vector<int> array< Type >::mult

Definition at line 26 of file array.cpp.

Referenced by array< Type >::index(), array< Type >::setdim(), and array< Type >::tuple().

8.5.5.3 template<class Type> Base array< Type >::vectorcopy

Definition at line 28 of file array.cpp.

Referenced by array< Type >::array(), and array< Type >::initZeroArray().

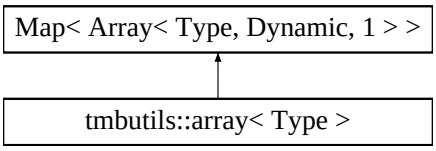
The documentation for this struct was generated from the following file:

- array.cpp

8.6 tmbutils::array< Type > Struct Template Reference

Array class used by TMB.

Inheritance diagram for tmbutils::array< Type >:



Public Types

- typedef Array< Type, Dynamic, 1 > Base
- typedef Map< Base > MapBase

Public Member Functions

- void setdim (vector< int > dim_)
- void initZeroArray (vector< int > dim_)
- vector< int > c (int n1)
- vector< int > c (int n1, int n2)
- vector< int > c (int n1, int n2, int n3)
- vector< int > c (int n1, int n2, int n3, int n4)
- array ()
- array (vector< int > dim_)

  Construct array from dimension vector and fill with zeros.
- array (int n1)
- array (int n1, int n2)
- array (int n1, int n2, int n3)
- array (int n1, int n2, int n3, int n4)
- template<class T >
  array (T &x, vector< int > dim_)
- template<class T >
  array (array< T > &x)
- array (Type *p, vector< int > dim_)
- array & operator= (const array &other)
- void print ()
- int cols ()
- array< Type > col (int i)

  Extract sub-array with write access Index i refers to the outer-most (i.e. final) dimension.
- Type & operator() (int i1)

  Elementwise subsetting 1D array.
- Type & operator() (int i1, int i2)

  Elementwise subsetting 2D array.
- Type & operator() (int i1, int i2, int i3)

  Elementwise subsetting 3D array.
- Type & operator() (int i1, int i2, int i3, int i4)

  Elementwise subsetting 4D array.
- int index (vector< int > tup)
- vector< int > tuple (int i)
- array< Type > perm (vector< int > p)

Array permutation. Permutes array dimensions corresponding to permutation vector p.

- array< Type > transpose ()

  Array transpose (Special case of array permutation)
- int mod (int i, int n)
- array< Type > rotate (int n)

  Array rotate (Special case of array permutation)

Public Attributes

- vector< int > dim
- vector< int > mult
- Base vectorcopy

8.6.1 Detailed Description

template<class Type>struct tmbutils::array< Type >

Array class used by TMB.

The TMB array class is implemented as an Eigen Array of dynamic length with a dimension attribute. The implementation closely follows the way arrays work in R. Vectorized operations are inherited from the Eigen library.

Definition at line 15 of file tmbutils.cpp.

8.6.2 Member Typedef Documentation

8.6.2.1 template<class Type> typedef Array<Type,Dynamic,1> tmbutils::array< Type >::Base

Definition at line 17 of file tmbutils.cpp.

8.6.2.2 template<class Type> typedef Map< Base > tmbutils::array< Type >::MapBase

Definition at line 18 of file tmbutils.cpp.

8.6.3 Constructor & Destructor Documentation

8.6.3.1 template<class Type> tmbutils::array< Type >::array ( ) [inline]

Definition at line 70 of file tmbutils.cpp.

8.6.3.2 template<class Type> tmbutils::array< Type >::array (vector< int > *dim_*) [inline]

Construct array from dimension vector and fill with zeros.

Definition at line 73 of file tmbutils.cpp.

8.6.3.3 template<class Type> tmbutils::array< Type >::array (int *n1*) [inline]

Definition at line 76 of file tmbutils.cpp.

8.6.3.4 template<class Type> tmbutils::array< Type >::array (int *n1*, int *n2*) [inline]

Definition at line 77 of file tmbutils.cpp.

8.6.3.5 template<class Type> tmbutils::array< Type >::array (int *n1*, int *n2*, int *n3*) [inline]

Definition at line 78 of file tmbutils.cpp.

8.6.3.6 template<class Type> tmbutils::array< Type >::array (int *n1*, int *n2*, int *n3*, int *n4*) [inline]

Definition at line 79 of file tmbutils.cpp.

8.6.3.7 template<class Type> template<class T > tmbutils::array< Type >::array (T & *x*, vector< int > *dim_*) [inline]

Definition at line 83 of file tmbutils.cpp.

8.6.3.8 template<class Type> template<class T > tmbutils::array< Type >::array (array< T > & *x*) [inline]

Definition at line 91 of file tmbutils.cpp.

8.6.3.9 template<class Type> tmbutils::array< Type >::array (Type * *p*, vector< int > *dim_*) [inline]

Definition at line 99 of file tmbutils.cpp.

8.6.4 Member Function Documentation

8.6.4.1 template<class Type> vector<int> tmbutils::array< Type >::c (int *n1*) [inline]

Definition at line 47 of file tmbutils.cpp.

8.6.4.2 template<class Type> vector<int> tmbutils::array< Type >::c (int *n1*, int *n2*) [inline]

Definition at line 52 of file tmbutils.cpp.

8.6.4.3 template<class Type> vector<int> tmbutils::array< Type >::c (int *n1*, int *n2*, int *n3*) [inline]

Definition at line 57 of file tmbutils.cpp.

8.6.4.4 template<class Type> vector<int> tmbutils::array< Type >::c (int *n1*, int *n2*, int *n3*, int *n4*) [inline]

Definition at line 62 of file tmbutils.cpp.

8.6.4.5 template<class Type> array<Type> tmbutils::array< Type >::col (int *i*) [inline]

Extract sub-array with write access Index i refers to the outer-most (i.e. final) dimension.

Definition at line 136 of file tmbutils.cpp.

8.6.4.6 template<class Type> int tmbutils::array< Type >::cols ( ) [inline]

Definition at line 128 of file tmbutils.cpp.

8.6.4.7 template<class Type> int tmbutils::array< Type >::index (vector< int > *tup*) [inline]

Definition at line 167 of file tmbutils.cpp.

8.6.4.8 template<class Type> void tmbutils::array< Type >::initZeroArray (vector< int > *dim_*) [inline]

Definition at line 39 of file tmbutils.cpp.

8.6.4.9 template<class Type> int tmbutils::array< Type >::mod (int *i,* int *n*) [inline]

Definition at line 202 of file tmbutils.cpp.

8.6.4.10 template<class Type> Type& tmbutils::array< Type >::operator() (int *i1*) [inline]

Elementwise subsetting 1D array.

Definition at line 150 of file tmbutils.cpp.

8.6.4.11 template<class Type> Type& tmbutils::array< Type >::operator() (int *i1,* int *i2*) [inline]

Elementwise subsetting 2D array.

Definition at line 154 of file tmbutils.cpp.

8.6.4.12 template<class Type> Type& tmbutils::array< Type >::operator() (int *i1,* int *i2,* int *i3*) [inline]

Elementwise subsetting 3D array.

Definition at line 158 of file tmbutils.cpp.

8.6.4.13 template<class Type> Type& tmbutils::array< Type >::operator() (int *i1,* int *i2,* int *i3,* int *i4*) [inline]

Elementwise subsetting 4D array.

Definition at line 162 of file tmbutils.cpp.

8.6.4.14 template<class Type> array& tmbutils::array< Type >::operator= (const array< Type > & *other*) [inline]

Definition at line 110 of file tmbutils.cpp.

8.6.4.15 template<class Type> array<Type> tmbutils::array< Type >::perm (vector< int > *p*) [inline]

Array permutation. Permutes array dimensions corresponding to permutation vector p.

Definition at line 183 of file tmbutils.cpp.

8.6.4.16 template<class Type> void tmbutils::array< Type >::print ( ) [inline]

Definition at line 118 of file tmbutils.cpp.

8.6.4.17 template<class Type> array<Type> tmbutils::array< Type >::rotate (int *n*) [inline]

Array rotate (Special case of array permutation)

Rotates array dimension with n steps where n can be any (positive or negative) integer. If e.g. x has dimension [3,4,5,6] then x.rotate(1) has dimension [6,3,4,5].

Definition at line 210 of file tmbutils.cpp.

8.6.4.18 template<class Type> void tmbutils::array< Type >::setdim (vector< int > *dim_*) [inline]

Definition at line 31 of file tmbutils.cpp.

8.6.4.19 template<class Type> array<Type> tmbutils::array< Type >::transpose ( ) [inline]

Array transpose (Special case of array permutation)

If e.g. x has dimension [3,4,5,6] then x.transpose() has dimension [6,5,4,3].

Definition at line 196 of file tmbutils.cpp.

8.6.4.20 template<class Type> vector<int> tmbutils::array< Type >::tuple (int *i*) [inline]

Definition at line 170 of file tmbutils.cpp.

8.6.5 Member Data Documentation

8.6.5.1 template<class Type> vector<int> tmbutils::array< Type >::dim

Definition at line 20 of file tmbutils.cpp.

Referenced by asSEXP().

8.6.5.2 template<class Type> vector<int> tmbutils::array< Type >::mult

Definition at line 27 of file tmbutils.cpp.

8.6.5.3 template<class Type> Base tmbutils::array< Type >::vectorcopy

Definition at line 29 of file tmbutils.cpp.

The documentation for this struct was generated from the following file:

- tmbutils.cpp

8.7 density::contAR2_t< scalartype_ > Class Template Reference

Continuous AR(2) process.

Public Member Functions

- contAR2_t ()
- contAR2_t (vectortype grid_, scalartype shape_, scalartype scale_=1)
- matrix4x4 expB (scalartype t)

- matrix2x2 V (scalartype t)
- scalartype operator() (vectortype x, vectortype dx)

  Evaluate the negative log density of the process x with nuisance parameters dx.
- scalartype operator() (vectortype x)
- arraytype matmult (matrix2x2 Q, arraytype x)
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Types

- typedef Matrix< scalartype, 2, 2 > matrix2x2
- typedef Matrix< scalartype, 2, 1 > matrix2x1
- typedef Matrix< scalartype, 4, 4 > matrix4x4
- typedef Matrix< scalartype, 4, 1 > matrix4x1

Private Member Functions

- TYPEDEFS (scalartype_)

Private Attributes

- scalartype shape
- scalartype scale
- scalartype c0
- scalartype c1
- vectortype grid
- matrix2x2 A
- matrix2x2 V0
- matrix2x2 I
- matrix4x4 B
- matrix4x4 iB
- matexp< scalartype, 2 > expA
- matrix4x1 vecSigma
- matrix4x1 iBvecSigma
- vector< MVNORM_t< scalartype > > neglogdmvnorm
- vector< matrix2x2 > expAdt

8.7.1 Detailed Description

template<class scalartype_>class density::contAR2_t< scalartype_ >

Continuous AR(2) process.

```
Process with covariance satisfying the 2nd order ode
rho''=c1*rho'-rho on an arbitrary irregular grid.
(shape=c1/2, -1<shape<1). Initial condition rho(0)=1, rho'(0)=0,
rho''(0)=-1.
```

Process is augmented with derivatives in order to obtain exact sparseness of the full precision. That is, if a model is desired on a grid of size n, then additional n extra nuisance parameters must be supplied.

Parameters

*grid_*	Possibly irregular grid of length n
*shape_*	Parameter defining the shape of the correlation function.
*scale_*	Parameter defining the correlation range.

Definition at line 463 of file tmbutils.cpp.

8.7.2 Member Typedef Documentation

8.7.2.1 template<class scalartype_ > typedef Matrix<scalartype,2,1> density::contAR2_t< scalartype_ >::matrix2x1 [private]

Definition at line 467 of file tmbutils.cpp.

8.7.2.2 template<class scalartype_ > typedef Matrix<scalartype,2,2> density::contAR2_t< scalartype_ >::matrix2x2 [private]

Definition at line 466 of file tmbutils.cpp.

8.7.2.3 template<class scalartype_ > typedef Matrix<scalartype,4,1> density::contAR2_t< scalartype_ >::matrix4x1 [private]

Definition at line 469 of file tmbutils.cpp.

8.7.2.4 template<class scalartype_ > typedef Matrix<scalartype,4,4> density::contAR2_t< scalartype_ >::matrix4x4 [private]

Definition at line 468 of file tmbutils.cpp.

8.7.3 Constructor & Destructor Documentation

8.7.3.1 template<class scalartype_ > density::contAR2_t< scalartype_ >::contAR2_t ( ) [inline]

Definition at line 479 of file tmbutils.cpp.

8.7.3.2 template<class scalartype_ > density::contAR2_t< scalartype_ >::contAR2_t (vectortype *grid_*, scalartype *shape_*, scalartype *scale_* = 1) [inline]

Definition at line 480 of file tmbutils.cpp.

8.7.4 Member Function Documentation

8.7.4.1 template<class scalartype_ > matrix4x4 density::contAR2_t< scalartype_ >::expB (scalartype *t*) [inline]

Definition at line 502 of file tmbutils.cpp.

8.7.4.2 template<class scalartype_ > arraytype density::contAR2_t< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 544 of file tmbutils.cpp.

8.7.4.3 template<class scalartype_ > arraytype density::contAR2_t< scalartype_ >::matmult (matrix2x2 *Q*, arraytype *x*) [inline]

Definition at line 538 of file tmbutils.cpp.

8.7.4.4 template<class scalartype_ > int density::contAR2_t< scalartype_ >::ndim ( ) [inline]

Definition at line 561 of file tmbutils.cpp.

8.7.4.5 template<class scalartype_ > scalartype density::contAR2_t< scalartype_ >::operator() (vectortype *x*, vectortype *dx*) [inline]

Evaluate the negative log density of the process x with nuisance parameters dx.

Definition at line 515 of file tmbutils.cpp.

8.7.4.6 template<class scalartype_ > scalartype density::contAR2_t< scalartype_ >::operator() (vectortype *x*) [inline]

Definition at line 531 of file tmbutils.cpp.

8.7.4.7 template<class scalartype_ > density::contAR2_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.7.4.8 template<class scalartype_ > matrix2x2 density::contAR2_t< scalartype_ >::V (scalartype *t*) [inline]

Definition at line 506 of file tmbutils.cpp.

8.7.5 Member Data Documentation

8.7.5.1 template<class scalartype_ > matrix2x2 density::contAR2_t< scalartype_ >::A [private]

Definition at line 472 of file tmbutils.cpp.

8.7.5.2 template<class scalartype_ > matrix4x4 density::contAR2_t< scalartype_ >::B [private]

Definition at line 473 of file tmbutils.cpp.

8.7.5.3 template<class scalartype_ > scalartype density::contAR2_t< scalartype_ >::c0 [private]

Definition at line 470 of file tmbutils.cpp.

8.7.5.4 template<class scalartype_ > scalartype density::contAR2_t< scalartype_ >::c1 [private]

Definition at line 470 of file tmbutils.cpp.

8.7.5.5 template<class scalartype_ > matexp<scalartype,2> density::contAR2_t< scalartype_ >::expA [private]

Definition at line 474 of file tmbutils.cpp.

8.7.5.6 template<class scalartype_ > vector<matrix2x2 > density::contAR2_t< scalartype_ >::expAdt [private]

Definition at line 477 of file tmbutils.cpp.

8.7.5.7 template<class scalartype_ > vectortype density::contAR2_t< scalartype_ >::grid [private]

Definition at line 471 of file tmbutils.cpp.

8.7.5.8 template<class scalartype_ > matrix2x2 density::contAR2_t< scalartype_ >::I [private]

Definition at line 472 of file tmbutils.cpp.

8.7.5.9 template<class scalartype_ > matrix4x4 density::contAR2_t< scalartype_ >::iB [private]

Definition at line 473 of file tmbutils.cpp.

8.7.5.10 template<class scalartype_ > matrix4x1 density::contAR2_t< scalartype_ >::iBvecSigma [private]

Definition at line 475 of file tmbutils.cpp.

8.7.5.11 template<class scalartype_ > vector<MVNORM_t<scalartype> > density::contAR2_t< scalartype_ >::neglogdmvnorm [private]

Definition at line 476 of file tmbutils.cpp.

8.7.5.12 template<class scalartype_ > scalartype density::contAR2_t< scalartype_ >::scale [private]

Definition at line 470 of file tmbutils.cpp.

8.7.5.13 template<class scalartype_ > scalartype density::contAR2_t< scalartype_ >::shape [private]

Definition at line 470 of file tmbutils.cpp.

8.7.5.14 template<class scalartype_ > matrix2x2 density::contAR2_t< scalartype_ >::V0 [private]

Definition at line 472 of file tmbutils.cpp.

8.7.5.15 template<class scalartype_ > density::contAR2_t< scalartype_ >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 562 of file tmbutils.cpp.

8.7.5.16 template<class scalartype_ > matrix4x1 density::contAR2_t< scalartype_ >::vecSigma [private]

Definition at line 475 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.8 contAR2_t< scalartype_ > Class Template Reference

Continuous AR(2) process.

Public Member Functions

- contAR2_t ()
- contAR2_t (vectortype grid_, scalartype shape_, scalartype scale_=1)
- matrix4x4 expB (scalartype t)
- matrix2x2 V (scalartype t)
- scalartype operator() (vectortype x, vectortype dx)

  Evaluate the negative log density of the process x with nuisance parameters dx.
- scalartype operator() (vectortype x)
- arraytype matmult (matrix2x2 Q, arraytype x)
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Types

- typedef Matrix< scalartype, 2, 2 > matrix2x2
- typedef Matrix< scalartype, 2, 1 > matrix2x1
- typedef Matrix< scalartype, 4, 4 > matrix4x4
- typedef Matrix< scalartype, 4, 1 > matrix4x1

Private Member Functions

- TYPEDEFS (scalartype_)

Private Attributes

- scalartype shape
- scalartype scale
- scalartype c0
- scalartype c1
- vectortype grid
- matrix2x2 A
- matrix2x2 V0
- matrix2x2 I
- matrix4x4 B
- matrix4x4 iB
- matexp< scalartype, 2 > expA
- matrix4x1 vecSigma
- matrix4x1 iBvecSigma
- vector< MVNORM_t< scalartype > > neglogdmvnorm
- vector< matrix2x2 > expAdt

8.8.1 Detailed Description

template<class scalartype_>class contAR2_t< scalartype_ >

Continuous AR(2) process.

```
Process with covariance satisfying the 2nd order ode
rho''=c1*rho'-rho on an arbitrary irregular grid.
(shape=c1/2, -1<shape<1). Initial condition rho(0)=1, rho'(0)=0,
rho''(0)=-1.
```

Process is augmented with derivatives in order to obtain exact sparseness of the full precision. That is, if a model is desired on a grid of size n, then additional n extra nuisance parameters must be supplied.

Parameters

*grid_*	Possibly irregular grid of length n
*shape_*	Parameter defining the shape of the correlation function.
*scale_*	Parameter defining the correlation range.

Definition at line 462 of file density.cpp.

8.8.2 Member Typedef Documentation

8.8.2.1 template<class scalartype_> typedef Matrix<scalartype,2,1> contAR2_t< scalartype_ >::matrix2x1 [private]

Definition at line 466 of file density.cpp.

8.8.2.2 template<class scalartype_> typedef Matrix<scalartype,2,2> contAR2_t< scalartype_ >::matrix2x2 [private]

Definition at line 465 of file density.cpp.

8.8.2.3 template<class scalartype_> typedef Matrix<scalartype,4,1> contAR2_t< scalartype_ >::matrix4x1 [private]

Definition at line 468 of file density.cpp.

8.8.2.4 template<class scalartype_> typedef Matrix<scalartype,4,4> contAR2_t< scalartype_ >::matrix4x4 [private]

Definition at line 467 of file density.cpp.

8.8.3 Constructor & Destructor Documentation

8.8.3.1 template<class scalartype_> contAR2_t< scalartype_ >::contAR2_t ( ) [inline]

Definition at line 478 of file density.cpp.

8.8.3.2 template<class scalartype_> contAR2_t< scalartype_ >::contAR2_t (vectortype *grid_*, scalartype *shape_*, scalartype *scale_* = 1) [inline]

Definition at line 479 of file density.cpp.

8.8.4 Member Function Documentation

8.8.4.1 template<class scalartype_> matrix4x4 contAR2_t< scalartype_ >::expB (scalartype *t*) [inline]

Definition at line 501 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::V().

8.8.4.2 template<class scalartype_> arraytype contAR2_t< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 543 of file density.cpp.

8.8.4.3 template<class scalartype_> arraytype contAR2_t< scalartype_ >::matmult (matrix2x2 *Q*, arraytype *x*) [inline]

Definition at line 537 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::jacobian().

8.8.4.4 template<class scalartype_> int contAR2_t< scalartype_ >::ndim ( ) [inline]

Definition at line 560 of file density.cpp.

8.8.4.5 template<class scalartype_> scalartype contAR2_t< scalartype_ >::operator() (vectortype *x*, vectortype *dx*) [inline]

Evaluate the negative log density of the process x with nuisance parameters dx.

Definition at line 514 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::operator()().

8.8.4.6 template<class scalartype_> scalartype contAR2_t< scalartype_ >::operator() (vectortype *x*) [inline]

Definition at line 530 of file density.cpp.

8.8.4.7 template<class scalartype_> contAR2_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.8.4.8 template<class scalartype_> matrix2x2 contAR2_t< scalartype_ >::V (scalartype *t*) [inline]

Definition at line 505 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5 Member Data Documentation

8.8.5.1 template<class scalartype_> matrix2x2 contAR2_t< scalartype_ >::A [private]

Definition at line 471 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5.2 template<class scalartype_> matrix4x4 contAR2_t< scalartype_ >::B [private]

Definition at line 472 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5.3 template<class scalartype_> scalartype contAR2_t< scalartype_ >::c0 [private]

Definition at line 469 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5.4 template<class scalartype_> scalartype contAR2_t< scalartype_ >::c1 [private]

Definition at line 469 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5.5 template<class scalartype_> matexp<scalartype,2> contAR2_t< scalartype_ >::expA [private]

Definition at line 473 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t(), and contAR2_t< scalartype_ >::expB().

8.8.5.6 template<class scalartype_> vector<matrix2x2 > contAR2_t< scalartype_ >::expAdt [private]

Definition at line 476 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t(), contAR2_t< scalartype_ >::jacobian(), and contAR2_t< scalartype_ >::operator()().

8.8.5.7 template<class scalartype_> vectortype contAR2_t< scalartype_ >::grid [private]

Definition at line 470 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t(), contAR2_t< scalartype_ >::jacobian(), and contAR2_t< scalartype_ >::operator()().

8.8.5.8 template<class scalartype_> matrix2x2 contAR2_t< scalartype_ >::I [private]

Definition at line 471 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5.9 template<class scalartype_> matrix4x4 contAR2_t< scalartype_ >::iB [private]

Definition at line 472 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5.10 template<class scalartype_> matrix4x1 contAR2_t< scalartype_ >::iBvecSigma [private]

Definition at line 474 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t(), and contAR2_t< scalartype_ >::V().

8.8.5.11 template<class scalartype_> vector<MVNORM_t<scalartype> > contAR2_t< scalartype_ >::neglogdmvnorm [private]

Definition at line 475 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t(), contAR2_t< scalartype_ >::jacobian(), and contAR2_t< scalartype_ >::operator()().

8.8.5.12 template<class scalartype_> scalartype contAR2_t< scalartype_ >::scale [private]

Definition at line 469 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5.13 template<class scalartype_> scalartype contAR2_t< scalartype_ >::shape [private]

Definition at line 469 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5.14 template<class scalartype_> matrix2x2 contAR2_t< scalartype_ >::V0 [private]

Definition at line 471 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

8.8.5.15 template<class scalartype_> contAR2_t< scalartype_ >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 561 of file density.cpp.

8.8.5.16 template<class scalartype_> matrix4x1 contAR2_t< scalartype_ >::vecSigma [private]

Definition at line 474 of file density.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t().

The documentation for this class was generated from the following file:

- density.cpp

8.9 density::GMRF_t< scalartype_ > Class Template Reference

Gaussian Markov Random Field.

Public Member Functions

- GMRF_t ()
- GMRF_t (Eigen::SparseMatrix< scalartype > Q_, int order_=1)
- GMRF_t (arraytype x, vectortype delta, int order_=1)
- void setQ (Eigen::SparseMatrix< scalartype > Q_, int order=1)
- scalartype Quadform (vectortype x)
- scalartype operator() (vectortype x)
- arraytype jacobian (arraytype x)
- int ndim ()
- vectortype variance ()

Private Member Functions

- TYPEDEFS (scalartype_)
- int sqdist (vectortype x, vectortype x_)

Private Attributes

- Eigen::SparseMatrix< scalartype > Q
- scalartype logdetQ

8.9.1 Detailed Description

template<class scalartype_>class density::GMRF_t< scalartype_ >

Gaussian Markov Random Field.

```
Class to evaluate the negative log density of a mean zero multivariate
normal distribution with a sparse precision matrix. Let Q denote the
precision matrix. Then the density is proportional to
|Q|^.5*exp(-.5*x'*Q*x)

Three constructors are available:

1. General case
===============
The user supplies the precision matrix Q of class Eigen::SparseMatrix<Type>

2. Special case: GMRF on d-dimensional lattice.
===============================================
The user supplies a d-dim lattice for which Q is automatically
constructed like this:
First order Gaussian Markov Random Field on (subset of) d-dim grid.
Grid is specified through the first array argument to constructor,
with individual nodes determined by the outdermost dimension
e.g. x= 1 1 2 2
       1 2 1 2
corresponding to a 2x2 lattice with 4 nodes and d=2.

Example of precision in 2D:

   -1
-1 4+c -1
   -1

The precision Q is convolved with it self "order" times. This way
more smoothness can be obtained. The quadratic form contribution
is .5*x'*Q^order*x

3. Vector of deltas
===================
The parameter "delta" describes the (inverse) correlation. It is
allowed to specify a vector of deltas so that different spatial
regions can have different spatial correlation.

NOTE: The variance in the model depends on delta. In other words:
The model may be thought of as an arbitrary scaled correlation
model and is thus not really meaningful without an additional scale
parameter (see SCALE_t and VECSCALE_t classes).
```

Definition at line 621 of file tmbutils.cpp.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 template<class scalartype_ > density::GMRF_t< scalartype_ >::GMRF_t ( ) [inline]

Definition at line 636 of file tmbutils.cpp.

8.9.2.2 template<class scalartype_ > density::GMRF_t< scalartype_ >::GMRF_t (Eigen::SparseMatrix< scalartype > *Q_*, int *order_* = 1) [inline]

Definition at line 637 of file tmbutils.cpp.

8.9.2.3 template<class scalartype_ > density::GMRF_t< scalartype_ >::GMRF_t (arraytype *x*, vectortype *delta*, int *order_* = 1) [inline]

Definition at line 640 of file tmbutils.cpp.

8.9.3 Member Function Documentation

8.9.3.1 template<class scalartype_ > arraytype density::GMRF_t< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 678 of file tmbutils.cpp.

8.9.3.2 template<class scalartype_ > int density::GMRF_t< scalartype_ >::ndim ( ) [inline]

Definition at line 686 of file tmbutils.cpp.

8.9.3.3 template<class scalartype_ > scalartype density::GMRF_t< scalartype_ >::operator() (vectortype *x*) [inline]

Definition at line 674 of file tmbutils.cpp.

8.9.3.4 template<class scalartype_ > scalartype density::GMRF_t< scalartype_ >::Quadform (vectortype *x*) [inline]

Definition at line 671 of file tmbutils.cpp.

8.9.3.5 template<class scalartype_ > void density::GMRF_t< scalartype_ >::setQ (Eigen::SparseMatrix< scalartype > *Q_*, int *order* = 1) [inline]

Definition at line 659 of file tmbutils.cpp.

8.9.3.6 template<class scalartype_ > int density::GMRF_t< scalartype_ >::sqdist (vectortype *x*, vectortype *x_*) [inline], [private]

Definition at line 626 of file tmbutils.cpp.

8.9.3.7 template<class scalartype_ > density::GMRF_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.9.3.8 template<class scalartype_ > vectortype density::GMRF_t< scalartype_ >::variance ( ) [inline]

Definition at line 687 of file tmbutils.cpp.

8.9.4 Member Data Documentation

8.9.4.1 template<class scalartype_ > scalartype density::GMRF_t< scalartype_ >::logdetQ [private]

Definition at line 625 of file tmbutils.cpp.

8.9.4.2 template<class scalartype_ > Eigen::SparseMatrix<scalartype> density::GMRF_t< scalartype_ >::Q [private]

Definition at line 624 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.10 GMRF_t< scalartype_ > Class Template Reference

Gaussian Markov Random Field.

Public Member Functions

- GMRF_t ()
- GMRF_t (Eigen::SparseMatrix< scalartype > Q_, int order_=1)
- GMRF_t (arraytype x, vectortype delta, int order_=1)
- void setQ (Eigen::SparseMatrix< scalartype > Q_, int order=1)
- scalartype Quadform (vectortype x)
- scalartype operator() (vectortype x)
- arraytype jacobian (arraytype x)
- int ndim ()
- vectortype variance ()

Private Member Functions

- TYPEDEFS (scalartype_)
- int sqdist (vectortype x, vectortype x_)

Private Attributes

- Eigen::SparseMatrix< scalartype > Q
- scalartype logdetQ

8.10.1 Detailed Description

template<class scalartype_>class GMRF_t< scalartype_ >

Gaussian Markov Random Field.

```
Class to evaluate the negative log density of a mean zero multivariate
normal distribution with a sparse precision matrix. Let Q denote the
precision matrix. Then the density is proportional to
|Q|^.5*exp(-.5*x'*Q*x)

Three constructors are available:
```

```
1. General case
===============
The user supplies the precision matrix Q of class Eigen::SparseMatrix<Type>

2. Special case: GMRF on d-dimensional lattice.
================================================
The user supplies a d-dim lattice for which Q is automatically
constructed like this:
First order Gaussian Markov Random Field on (subset of) d-dim grid.
Grid is specified through the first array argument to constructor,
with individual nodes determined by the outdermost dimension
e.g. x= 1 1 2 2
       1 2 1 2
corresponding to a 2x2 lattice with 4 nodes and d=2.

Example of precision in 2D:

   -1
-1 4+c -1
   -1

The precision Q is convolved with it self "order" times. This way
more smoothness can be obtained. The quadratic form contribution
is .5*x'*Q^order*x

3. Vector of deltas
===================
The parameter "delta" describes the (inverse) correlation. It is
allowed to specify a vector of deltas so that different spatial
regions can have different spatial correlation.

NOTE: The variance in the model depends on delta. In other words:
The model may be thought of as an arbitrary scaled correlation
model and is thus not really meaningful without an additional scale
parameter (see SCALE_t and VECSCALE_t classes).
```

Definition at line 620 of file density.cpp.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 template<class scalartype_> GMRF_t< scalartype_ >::GMRF_t ( ) [inline]

Definition at line 635 of file density.cpp.

8.10.2.2 template<class scalartype_> GMRF_t< scalartype_ >::GMRF_t (Eigen::SparseMatrix< scalartype > *Q_*, int *order_* = 1) [inline]

Definition at line 636 of file density.cpp.

8.10.2.3 template<class scalartype_> GMRF_t< scalartype_ >::GMRF_t (arraytype *x*, vectortype *delta*, int *order_* = 1) [inline]

Definition at line 639 of file density.cpp.

8.10.3 Member Function Documentation

8.10.3.1 template<class scalartype_> arraytype GMRF_t< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 677 of file density.cpp.

8.10.3.2 template<class scalartype_> int GMRF_t< scalartype_ >::ndim ( ) [inline]

Definition at line 685 of file density.cpp.

8.10.3.3 template<class scalartype_> scalartype GMRF_t< scalartype_ >::operator() (vectortype *x*) [inline]

Definition at line 673 of file density.cpp.

8.10.3.4 template<class scalartype_> scalartype GMRF_t< scalartype_ >::Quadform (vectortype *x*) [inline]

Definition at line 670 of file density.cpp.

Referenced by GMRF_t< scalartype_ >::operator()().

8.10.3.5 template<class scalartype_> void GMRF_t< scalartype_ >::setQ (Eigen::SparseMatrix< scalartype > *Q_*, int *order* = 1) [inline]

Definition at line 658 of file density.cpp.

Referenced by GMRF_t< scalartype_ >::GMRF_t().

8.10.3.6 template<class scalartype_> int GMRF_t< scalartype_ >::sqdist (vectortype *x*, vectortype *x_*) [inline], [private]

Definition at line 625 of file density.cpp.

Referenced by GMRF_t< scalartype_ >::GMRF_t().

8.10.3.7 template<class scalartype_> GMRF_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.10.3.8 template<class scalartype_> vectortype GMRF_t< scalartype_ >::variance ( ) [inline]

Definition at line 686 of file density.cpp.

8.10.4 Member Data Documentation

8.10.4.1 template<class scalartype_> scalartype GMRF_t< scalartype_ >::logdetQ [private]

Definition at line 624 of file density.cpp.

Referenced by GMRF_t< scalartype_ >::operator()(), and GMRF_t< scalartype_ >::setQ().

8.10.4.2 template<class scalartype_> Eigen::SparseMatrix<scalartype> GMRF_t< scalartype_ >::Q [private]

Definition at line 623 of file density.cpp.

Referenced by GMRF_t< scalartype_ >::jacobian(), GMRF_t< scalartype_ >::Quadform(), GMRF_t< scalartype_ >::setQ(), and GMRF_t< scalartype_ >::variance().

The documentation for this class was generated from the following file:

- density.cpp

8.11 isDouble< Type > Struct Template Reference

`#include <tmb_core.hpp>`

Public Types

- enum { value =false }

8.11.1 Detailed Description

template<class Type>struct isDouble< Type >

Definition at line 94 of file tmb_core.hpp.

8.11.2 Member Enumeration Documentation

8.11.2.1 template<class Type > anonymous enum

Enumerator

value

Definition at line 95 of file tmb_core.hpp.

The documentation for this struct was generated from the following file:

- tmb_core.hpp

8.12 isDouble< double > Struct Template Reference

`#include <tmb_core.hpp>`

Public Types

- enum { value =true }

8.12.1 Detailed Description

template<>struct isDouble< double >

Definition at line 98 of file tmb_core.hpp.

8.12.2 Member Enumeration Documentation

8.12.2.1 anonymous enum

Enumerator

value

Definition at line 99 of file tmb_core.hpp.

The documentation for this struct was generated from the following file:

- tmb_core.hpp

8.13 tmbutils::matexp< scalartype, dim > Struct Template Reference

Matrix exponential: matrix of arbitrary dimension.

Public Types

- typedef Matrix< scalartype, dim, dim > matrix
- typedef Matrix< std::complex < scalartype >,dim, dim > cmatrix
- typedef Matrix< std::complex < scalartype >,dim, 1 > cvector

Public Member Functions

- matexp ()
- matexp (matrix A_)
- matrix operator() (scalartype t)

Public Attributes

- cmatrix V
- cmatrix iV
- cvector lambda
- EigenSolver< matrix > eigensolver

8.13.1 Detailed Description

template<class scalartype, int dim>struct tmbutils::matexp< scalartype, dim >

Matrix exponential: matrix of arbitrary dimension.

Definition at line 9 of file tmbutils.cpp.

8.13.2 Member Typedef Documentation

8.13.2.1 template<class scalartype , int dim> typedef Matrix<std::complex<scalartype> ,dim,dim> tmbutils::matexp< scalartype, dim >::cmatrix

Definition at line 11 of file tmbutils.cpp.

8.13.2.2 template<class scalartype , int dim> typedef Matrix<std::complex<scalartype> ,dim,1> tmbutils::matexp< scalartype, dim >::cvector

Definition at line 12 of file tmbutils.cpp.

8.13.2.3 template<class scalartype , int dim> typedef Matrix<scalartype,dim,dim> tmbutils::matexp< scalartype, dim >::matrix

Definition at line 10 of file tmbutils.cpp.

8.13.3 Constructor & Destructor Documentation

8.13.3.1 template<class scalartype , int dim> tmbutils::matexp< scalartype, dim >::matexp ( ) [inline]

Definition at line 17 of file tmbutils.cpp.

8.13.3.2 template<class scalartype , int dim> tmbutils::matexp< scalartype, dim >::matexp (matrix *A_*) [inline]

Definition at line 18 of file tmbutils.cpp.

8.13.4 Member Function Documentation

8.13.4.1 template<class scalartype , int dim> matrix tmbutils::matexp< scalartype, dim >::operator() (scalartype *t*) [inline]

Definition at line 24 of file tmbutils.cpp.

8.13.5 Member Data Documentation

8.13.5.1 template<class scalartype , int dim> EigenSolver< matrix > tmbutils::matexp< scalartype, dim >::eigensolver

Definition at line 16 of file tmbutils.cpp.

8.13.5.2 template<class scalartype , int dim> cmatrix tmbutils::matexp< scalartype, dim >::iV

Definition at line 14 of file tmbutils.cpp.

8.13.5.3 template<class scalartype , int dim> cvector tmbutils::matexp< scalartype, dim >::lambda

Definition at line 15 of file tmbutils.cpp.

8.13.5.4 template<class scalartype , int dim> cmatrix tmbutils::matexp< scalartype, dim >::V

Definition at line 13 of file tmbutils.cpp.

The documentation for this struct was generated from the following file:

- tmbutils.cpp

8.14 matexp< scalartype, dim > Struct Template Reference

Matrix exponential: matrix of arbitrary dimension.

Public Types

- typedef Matrix< scalartype, dim, dim > matrix
- typedef Matrix< std::complex < scalartype >,dim, dim > cmatrix
- typedef Matrix< std::complex < scalartype >,dim, 1 > cvector

Public Member Functions

- matexp ()
- matexp (matrix A_)
- matrix operator() (scalartype t)

Public Attributes

- cmatrix V
- cmatrix iV
- cvector lambda
- EigenSolver< matrix > eigensolver

8.14.1 Detailed Description

template<class scalartype, int dim>struct matexp< scalartype, dim >

Matrix exponential: matrix of arbitrary dimension.

Definition at line 8 of file matexp.cpp.

8.14.2 Member Typedef Documentation

8.14.2.1 template<class scalartype, int dim> typedef Matrix<std::complex<scalartype> ,dim,dim> matexp< scalartype, dim >::cmatrix

Definition at line 10 of file matexp.cpp.

8.14.2.2 template<class scalartype, int dim> typedef Matrix<std::complex<scalartype> ,dim,1> matexp< scalartype, dim >::cvector

Definition at line 11 of file matexp.cpp.

8.14.2.3 template<class scalartype, int dim> typedef Matrix<scalartype,dim,dim> matexp< scalartype, dim >::matrix

Definition at line 9 of file matexp.cpp.

8.14.3 Constructor & Destructor Documentation

8.14.3.1 template<class scalartype, int dim> matexp< scalartype, dim >::matexp ( ) [inline]

Definition at line 16 of file matexp.cpp.

8.14.3.2 template<class scalartype, int dim> matexp< scalartype, dim >::matexp (matrix *A_*) [inline]

Definition at line 17 of file matexp.cpp.

8.14.4 Member Function Documentation

8.14.4.1 template<class scalartype, int dim> matrix matexp< scalartype, dim >::operator() (scalartype *t*) [inline]

Definition at line 23 of file matexp.cpp.

8.14.5 Member Data Documentation

8.14.5.1 template<class scalartype, int dim> EigenSolver< matrix > matexp< scalartype, dim >::eigensolver

Definition at line 15 of file matexp.cpp.

Referenced by matexp< scalartype, dim >::matexp().

8.14.5.2 template<class scalartype, int dim> cmatrix matexp< scalartype, dim >::iV

Definition at line 13 of file matexp.cpp.

Referenced by matexp< scalartype, dim >::matexp(), matexp< scalartype, 2 >::matexp(), matexp< scalartype, dim >::operator()(), and matexp< scalartype, 2 >::operator()().

8.14.5.3 template<class scalartype, int dim> cvector matexp< scalartype, dim >::lambda

Definition at line 14 of file matexp.cpp.

Referenced by matexp< scalartype, dim >::matexp(), matexp< scalartype, 2 >::matexp(), matexp< scalartype, dim >::operator()(), and matexp< scalartype, 2 >::operator()().

8.14.5.4 template<class scalartype, int dim> cmatrix matexp< scalartype, dim >::V

Definition at line 12 of file matexp.cpp.

Referenced by matexp< scalartype, dim >::matexp(), matexp< scalartype, 2 >::matexp(), matexp< scalartype, dim >::operator()(), and matexp< scalartype, 2 >::operator()().

The documentation for this struct was generated from the following file:

- matexp.cpp

8.15 tmbutils::matexp< scalartype, 2 > Struct Template Reference

Matrix exponential: 2x2 case which can be handled efficiently.

Public Types

- typedef std::complex< scalartype > complex
- typedef Matrix< scalartype, 2, 2 > matrix
- typedef Matrix< complex,2, 2 > cmatrix
- typedef Matrix< complex,2, 1 > cvector

Public Member Functions

- matexp ()
- matexp (matrix A_)
- matrix operator() (scalartype t)

Public Attributes

- cmatrix V
- cmatrix iV
- cvector lambda

8.15.1 Detailed Description

template<class scalartype>struct tmbutils::matexp< scalartype, 2 >

Matrix exponential: 2x2 case which can be handled efficiently.

Definition at line 42 of file tmbutils.cpp.

8.15.2 Member Typedef Documentation

8.15.2.1 template<class scalartype > typedef Matrix<complex ,2,2> tmbutils::matexp< scalartype, 2 >::cmatrix

Definition at line 45 of file tmbutils.cpp.

8.15.2.2 template<class scalartype > typedef std::complex<scalartype> tmbutils::matexp< scalartype, 2 >::complex

Definition at line 43 of file tmbutils.cpp.

8.15.2.3 template<class scalartype > typedef Matrix<complex ,2,1> tmbutils::matexp< scalartype, 2 >::cvector

Definition at line 46 of file tmbutils.cpp.

8.15.2.4 template<class scalartype > typedef Matrix<scalartype,2,2> tmbutils::matexp< scalartype, 2 >::matrix

Definition at line 44 of file tmbutils.cpp.

8.15.3 Constructor & Destructor Documentation

8.15.3.1 template<class scalartype > tmbutils::matexp< scalartype, 2 >::matexp ( ) [inline]

Definition at line 50 of file tmbutils.cpp.

8.15.3.2 template<class scalartype > tmbutils::matexp< scalartype, 2 >::matexp (matrix *A_*) [inline]

Definition at line 51 of file tmbutils.cpp.

8.15.4 Member Function Documentation

8.15.4.1 template<class scalartype > matrix tmbutils::matexp< scalartype, 2 >::operator() (scalartype *t*) [inline]

Definition at line 60 of file tmbutils.cpp.

8.15.5 Member Data Documentation

8.15.5.1 template<class scalartype > cmatrix tmbutils::matexp< scalartype, 2 >::iV

Definition at line 48 of file tmbutils.cpp.

8.15.5.2 template<class scalartype > cvector tmbutils::matexp< scalartype, 2 >::lambda

Definition at line 49 of file tmbutils.cpp.

8.15.5.3 template<class scalartype > cmatrix tmbutils::matexp< scalartype, 2 >::V

Definition at line 47 of file tmbutils.cpp.

The documentation for this struct was generated from the following file:

- tmbutils.cpp

8.16 matexp< scalartype, 2 > Struct Template Reference

Matrix exponential: 2x2 case which can be handled efficiently.

Public Types

- typedef std::complex< scalartype > complex
- typedef Matrix< scalartype, 2, 2 > matrix
- typedef Matrix< complex,2, 2 > cmatrix
- typedef Matrix< complex,2, 1 > cvector

Public Member Functions

- matexp ()
- matexp (matrix A_)
- matrix operator() (scalartype t)

Public Attributes

- cmatrix V
- cmatrix iV
- cvector lambda

8.16.1 Detailed Description

template<class scalartype>struct matexp< scalartype, 2 >

Matrix exponential: 2x2 case which can be handled efficiently.

Definition at line 41 of file matexp.cpp.

8.16.2 Member Typedef Documentation

8.16.2.1 template<class scalartype > typedef Matrix<complex ,2,2> matexp< scalartype, 2 >::cmatrix

Definition at line 44 of file matexp.cpp.

8.16.2.2 template<class scalartype > typedef std::complex<scalartype> matexp< scalartype, 2 >::complex

Definition at line 42 of file matexp.cpp.

8.16.2.3 template<class scalartype > typedef Matrix<complex ,2,1> matexp< scalartype, 2 >::cvector

Definition at line 45 of file matexp.cpp.

8.16.2.4 template<class scalartype > typedef Matrix<scalartype,2,2> matexp< scalartype, 2 >::matrix

Definition at line 43 of file matexp.cpp.

8.16.3 Constructor & Destructor Documentation

8.16.3.1 template<class scalartype > matexp< scalartype, 2 >::matexp ( ) [inline]

Definition at line 49 of file matexp.cpp.

8.16.3.2 template<class scalartype > matexp< scalartype, 2 >::matexp (matrix *A_*) [inline]

Definition at line 50 of file matexp.cpp.

8.16.4 Member Function Documentation

8.16.4.1 template<class scalartype > matrix matexp< scalartype, 2 >::operator() (scalartype *t*) [inline]

Definition at line 59 of file matexp.cpp.

8.16.5 Member Data Documentation

8.16.5.1 template<class scalartype > cmatrix matexp< scalartype, 2 >::iV

Definition at line 47 of file matexp.cpp.

8.16.5.2 template<class scalartype > cvector matexp< scalartype, 2 >::lambda

Definition at line 48 of file matexp.cpp.

8.16.5.3 template<class scalartype > cmatrix matexp< scalartype, 2 >::V

Definition at line 46 of file matexp.cpp.

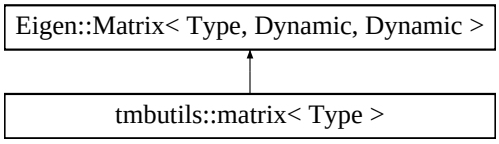
The documentation for this struct was generated from the following file:

- matexp.cpp

8.17 tmbutils::matrix< Type > Struct Template Reference

Matrix class used by TMB.

Inheritance diagram for tmbutils::matrix< Type >:



Public Types

- typedef Matrix< Type, Dynamic, Dynamic > Base

Public Member Functions

- matrix (void)
- template<class T1 > matrix (T1 x)
- template<class T1 , class T2 > matrix (T1 x, T2 y)
- template<class T1 > matrix & operator= (const T1 &other)
- vector< Type > vec ()

8.17.1 Detailed Description

template<class Type>struct tmbutils::matrix< Type >

Matrix class used by TMB.

The TMB matrix class is implemented as an Eigen Matrix of dynamic dimension. In particular, linear algebra methods are inherited from the Eigen library.

Definition at line 82 of file tmbutils.cpp.

8.17.2 Member Typedef Documentation

8.17.2.1 template<class Type > typedef Matrix<Type,Dynamic,Dynamic> tmbutils::matrix< Type >::Base

Definition at line 84 of file tmbutils.cpp.

8.17.3 Constructor & Destructor Documentation

8.17.3.1 template<class Type > tmbutils::matrix< Type >::matrix (void) [inline]

Definition at line 85 of file tmbutils.cpp.

8.17.3.2 template<class Type > template<class T1 > tmbutils::matrix< Type >::matrix (T1 *x*) [inline]

Definition at line 87 of file tmbutils.cpp.

8.17.3.3 template<class Type > template<class T1 , class T2 > tmbutils::matrix< Type >::matrix (T1 *x*, T2 *y*) [inline]

Definition at line 89 of file tmbutils.cpp.

8.17.4 Member Function Documentation

8.17.4.1 template<class Type > template<class T1 > matrix& tmbutils::matrix< Type >::operator= (const T1 & *other*) [inline]

Definition at line 92 of file tmbutils.cpp.

8.17.4.2 template<class Type > vector<Type> tmbutils::matrix< Type >::vec ( ) [inline]

The vec operator stacks the matrix columns into a single vector.

Definition at line 101 of file tmbutils.cpp.

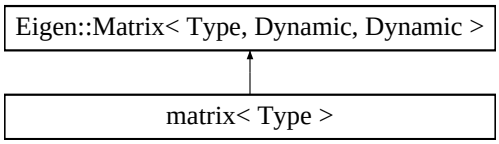
The documentation for this struct was generated from the following file:

- tmbutils.cpp

8.18 matrix< Type > Struct Template Reference

Matrix class used by TMB.

Inheritance diagram for matrix< Type >:



Public Types

- typedef Matrix< Type, Dynamic, Dynamic > Base

Public Member Functions

- matrix (void)
- template<class T1 > matrix (T1 x)
- template<class T1 , class T2 > matrix (T1 x, T2 y)
- template<class T1 > matrix & operator= (const T1 &other)
- vector< Type > vec ()

8.18.1 Detailed Description

template<class Type>struct matrix< Type >

Matrix class used by TMB.

The TMB matrix class is implemented as an Eigen Matrix of dynamic dimension. In particular, linear algebra methods are inherited from the Eigen library.

Examples:

matrix_arrays.cpp, nmix.cpp, rw.cpp, sdv_multi.cpp, and spatial.cpp.

Definition at line 81 of file vector.cpp.

8.18.2 Member Typedef Documentation

8.18.2.1 template<class Type> typedef Matrix<Type,Dynamic,Dynamic> matrix< Type >::Base

Definition at line 83 of file vector.cpp.

8.18.3 Constructor & Destructor Documentation

8.18.3.1 template<class Type> matrix< Type >::matrix (void) [inline]

Definition at line 84 of file vector.cpp.

8.18.3.2 template<class Type> template<class T1 > matrix< Type >::matrix (T1 *x*) [inline]

Definition at line 86 of file vector.cpp.

8.18.3.3 template<class Type> template<class T1 , class T2 > matrix< Type >::matrix (T1 *x*, T2 *y*) [inline]

Definition at line 88 of file vector.cpp.

8.18.4 Member Function Documentation

8.18.4.1 template<class Type> template<class T1 > matrix& matrix< Type >::operator= (const T1 & *other*) [inline]

Definition at line 91 of file vector.cpp.

8.18.4.2 template<class Type> vector<Type> matrix< Type >::vec ( ) [inline]

The vec operator stacks the matrix columns into a single vector.

Definition at line 100 of file vector.cpp.

Referenced by discrLyap(), and report_stack< Type >::push().

The documentation for this struct was generated from the following file:

- vector.cpp

8.19 memory_manager_struct Struct Reference

Controls the life span of objects created in the C++ template (jointly R/C++)

`#include <tmb_core.hpp>`

Public Member Functions

- void RegisterCFinalizer (SEXP list)

  Register "list" in memory_manager_struct.
- void CallCFinalizer (SEXP x)

  Revmoves "x" from memory_manager_struct.
- void clear ()
- memory_manager_struct ()

Public Attributes

- int counter

  Number of objects alive that "memory_manager_struct" has allocated.
- std::map< SEXP_t, SEXP_t > alive

8.19.1 Detailed Description

Controls the life span of objects created in the C++ template (jointly R/C++)

Definition at line 20 of file tmb_core.hpp.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 memory_manager_struct::memory_manager_struct ( ) [inline]

Definition at line 42 of file tmb_core.hpp.

8.19.3 Member Function Documentation

8.19.3.1 void memory_manager_struct::CallCFinalizer (SEXP *x*) [inline]

Revmoves "x" from memory_manager_struct.

Definition at line 30 of file tmb_core.hpp.

Referenced by finalize(), finalizeADFun(), finalizeDoubleFun(), and finalizeparallelADFun().

8.19.3.2 void memory_manager_struct::clear ( ) [inline]

Definition at line 34 of file tmb_core.hpp.

8.19.3.3 void memory_manager_struct::RegisterCFinalizer (SEXP *list*) [inline]

Register "list" in memory_manager_struct.

Definition at line 24 of file tmb_core.hpp.

Referenced by ptrList().

8.19.4 Member Data Documentation

8.19.4.1 std::map<SEXP_t,SEXP_t> memory_manager_struct::alive

Definition at line 22 of file tmb_core.hpp.

Referenced by CallCFinalizer(), clear(), and RegisterCFinalizer().

8.19.4.2 int memory_manager_struct::counter

Number of objects alive that "memory_manager_struct" has allocated.

Definition at line 21 of file tmb_core.hpp.

Referenced by CallCFinalizer(), memory_manager_struct(), and RegisterCFinalizer().

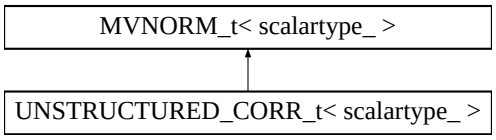
The documentation for this struct was generated from the following file:

- tmb_core.hpp

8.20 MVNORM_t< scalartype_ > Class Template Reference

Multivariate normal distribution with user supplied covariance matrix.

Inheritance diagram for MVNORM_t< scalartype_ >:



Public Member Functions

- MVNORM_t ()
- MVNORM_t (matrixtype Sigma_)
- matrixtype cov ()

  Covariance extractor.
- matrixtype chol (const matrixtype &a)
- vectortype lsolve (matrixtype &l, vectortype y)
- arraytype lltsolve (matrixtype &l, arraytype y)
- void setSigma (matrixtype Sigma_)
- scalartype Quadform (vectortype x)
- scalartype operator() (vectortype x)

  Evaluate the negative log density.
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (scalartype_)

Private Attributes

- scalartype logdetQ
- matrixtype L
- matrixtype Sigma

8.20.1 Detailed Description

template<class scalartype_>class MVNORM_t< scalartype_ >

Multivariate normal distribution with user supplied covariance matrix.

Class to evaluate the negative log density of a mean zero multivariate Gaussian variable with general covariance matrix Sigma. Intended for small dense covariance matrices.

Definition at line 22 of file density.cpp.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 template<class scalartype_ > MVNORM_t< scalartype_ >::MVNORM_t ( ) [inline]

Definition at line 29 of file density.cpp.

8.20.2.2 template<class scalartype_ > MVNORM_t< scalartype_ >::MVNORM_t (matrixtype *Sigma_*) [inline]

Definition at line 30 of file density.cpp.

8.20.3 Member Function Documentation

8.20.3.1 template<class scalartype_ > matrixtype MVNORM_t< scalartype_ >::chol (const matrixtype & *a*) [inline]

Definition at line 38 of file density.cpp.

Referenced by MVNORM_t< scalartype_ >::setSigma().

8.20.3.2 template<class scalartype_ > matrixtype MVNORM_t< scalartype_ >::cov ( ) [inline]

Covariance extractor.

Definition at line 35 of file density.cpp.

8.20.3.3 template<class scalartype_ > arraytype MVNORM_t< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 101 of file density.cpp.

8.20.3.4 template<class scalartype_ > arraytype MVNORM_t< scalartype_ >::lltsolve (matrixtype & *l*, arraytype *y*) [inline]

Definition at line 66 of file density.cpp.

Referenced by MVNORM_t< scalartype_ >::jacobian().

8.20.3.5 template<class scalartype_ > vectortype MVNORM_t< scalartype_ >::lsolve (matrixtype & *l*, vectortype *y*) [inline]

Definition at line 53 of file density.cpp.

Referenced by MVNORM_t< scalartype_ >::Quadform().

8.20.3.6 template<class scalartype_ > int MVNORM_t< scalartype_ >::ndim ( ) [inline]

Definition at line 104 of file density.cpp.

8.20.3.7 template<class scalartype_ > scalartype MVNORM_t< scalartype_ >::operator() (vectortype *x*) [inline]

Evaluate the negative log density.

Definition at line 98 of file density.cpp.

8.20.3.8 template<class scalartype_ > scalartype MVNORM_t< scalartype_ >::Quadform (vectortype *x*) [inline]

Definition at line 93 of file density.cpp.

Referenced by MVNORM_t< scalartype_ >::operator()().

8.20.3.9 template<class scalartype_ > void MVNORM_t< scalartype_ >::setSigma (matrixtype *Sigma_*) [inline]

Definition at line 87 of file density.cpp.

Referenced by MVNORM_t< scalartype_ >::MVNORM_t(), and UNSTRUCTURED_CORR_t< scalartype_ >::↩ UNSTRUCTURED_CORR_t().

8.20.3.10 template<class scalartype_ > MVNORM_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.20.4 Member Data Documentation

8.20.4.1 template<class scalartype_ > matrixtype MVNORM_t< scalartype_ >::L [private]

Lower cholesky of *covariance*

Definition at line 26 of file density.cpp.

Referenced by MVNORM_t< scalartype_ >::jacobian(), MVNORM_t< scalartype_ >::Quadform(), MVNORM_t< scalartype_ >::setSigma(), and UNSTRUCTURED_CORR_t< scalartype_ >::UNSTRUCTURED_CORR_t().

8.20.4.2 template<class scalartype_ > scalartype MVNORM_t< scalartype_ >::logdetQ [private]

Definition at line 24 of file density.cpp.

Referenced by MVNORM_t< scalartype_ >::operator()(), and MVNORM_t< scalartype_ >::setSigma().

8.20.4.3 template<class scalartype_ > matrixtype MVNORM_t< scalartype_ >::Sigma [private]

Definition at line 27 of file density.cpp.

Referenced by MVNORM_t< scalartype_ >::cov(), MVNORM_t< scalartype_ >::setSigma(), and UNSTRUCTU↩ RED_CORR_t< scalartype_ >::UNSTRUCTURED_CORR_t().

8.20.4.4 template<class scalartype_ > MVNORM_t< scalartype_ >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 105 of file density.cpp.

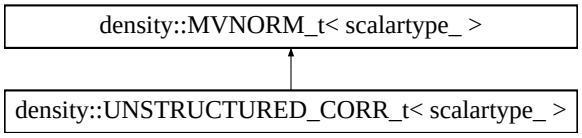
The documentation for this class was generated from the following file:

- density.cpp

8.21 density::MVNORM_t< scalartype_ > Class Template Reference

Multivariate normal distribution with user supplied covariance matrix.

Inheritance diagram for density::MVNORM_t< scalartype_ >:



Public Member Functions

- MVNORM_t ()
- MVNORM_t (matrixtype Sigma_)
- matrixtype cov ()

  Covariance extractor.
- matrixtype chol (const matrixtype &a)
- vectortype lsolve (matrixtype &l, vectortype y)
- arraytype lltsolve (matrixtype &l, arraytype y)
- void setSigma (matrixtype Sigma_)
- scalartype Quadform (vectortype x)
- scalartype operator() (vectortype x)

  Evaluate the negative log density.
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (scalartype_)

Private Attributes

- scalartype logdetQ
- matrixtype L
- matrixtype Sigma

8.21.1 Detailed Description

template<class scalartype_>class density::MVNORM_t< scalartype_ >

Multivariate normal distribution with user supplied covariance matrix.

Class to evaluate the negative log density of a mean zero multivariate Gaussian variable with general covariance matrix Sigma. Intended for small dense covariance matrices.

Examples:

rw.cpp, sdv_multi.cpp, and spatial.cpp.

Definition at line 23 of file tmbutils.cpp.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 template<class scalartype_> density::MVNORM_t< scalartype_ >::MVNORM_t ( ) [inline]

Definition at line 30 of file tmbutils.cpp.

8.21.2.2 template<class scalartype_> density::MVNORM_t< scalartype_ >::MVNORM_t (matrixtype *Sigma_*) [inline]

Definition at line 31 of file tmbutils.cpp.

8.21.3 Member Function Documentation

8.21.3.1 template<class scalartype_> matrixtype density::MVNORM_t< scalartype_ >::chol (const matrixtype & *a*) [inline]

Definition at line 39 of file tmbutils.cpp.

8.21.3.2 template<class scalartype_> matrixtype density::MVNORM_t< scalartype_ >::cov ( ) [inline]

Covariance extractor.

Definition at line 36 of file tmbutils.cpp.

8.21.3.3 template<class scalartype_> arraytype density::MVNORM_t< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 102 of file tmbutils.cpp.

Referenced by PROJ_t< distribution >::jacobian().

8.21.3.4 template<class scalartype_> arraytype density::MVNORM_t< scalartype_ >::lltsolve (matrixtype & *l,* arraytype *y*) [inline]

Definition at line 67 of file tmbutils.cpp.

8.21.3.5 template<class scalartype_> vectortype density::MVNORM_t< scalartype_ >::lsolve (matrixtype & *l,* vectortype *y*) [inline]

Definition at line 54 of file tmbutils.cpp.

8.21.3.6 template<class scalartype_> int density::MVNORM_t< scalartype_ >::ndim ( ) [inline]

Definition at line 105 of file tmbutils.cpp.

8.21.3.7 template<class scalartype_> scalartype density::MVNORM_t< scalartype_ >::operator() (vectortype *x*) [inline]

Evaluate the negative log density.

Definition at line 99 of file tmbutils.cpp.

8.21.3.8 template<class scalartype_> scalartype density::MVNORM_t< scalartype_ >::Quadform (vectortype *x*) [inline]

Definition at line 94 of file tmbutils.cpp.

8.21.3.9 template<class scalartype_> void density::MVNORM_t< scalartype_ >::setSigma (matrixtype *Sigma_*) [inline]

Definition at line 88 of file tmbutils.cpp.

8.21.3.10 template<class scalartype_> density::MVNORM_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.21.4 Member Data Documentation

8.21.4.1 template<class scalartype_> matrixtype density::MVNORM_t< scalartype_ >::L [private]

Lower cholesky of *covariance*

Definition at line 27 of file tmbutils.cpp.

8.21.4.2 template<class scalartype_> scalartype density::MVNORM_t< scalartype_ >::logdetQ [private]

Definition at line 25 of file tmbutils.cpp.

8.21.4.3 template<class scalartype_> matrixtype density::MVNORM_t< scalartype_ >::Sigma [private]

Definition at line 28 of file tmbutils.cpp.

8.21.4.4 template<class scalartype_> density::MVNORM_t< scalartype_ >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 106 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.22 N01< scalartype_ > Class Template Reference

Standardized normal distribution.

Public Member Functions

- scalartype operator() (array< scalartype > x)

  Evaluate the negative log density.
- scalartype operator() (scalartype x)
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (scalartype_)

8.22.1 Detailed Description

template<class scalartype_>class N01< scalartype_ >

Standardized normal distribution.
Class to evaluate the negative log density of a (multivariate) standard normal distribution.

```
Examples: N01()
```

Definition at line 189 of file density.cpp.

8.22.2 Member Function Documentation

8.22.2.1 template<class scalartype_ > arraytype N01< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 199 of file density.cpp.

8.22.2.2 template<class scalartype_ > int N01< scalartype_ >::ndim ( ) [inline]

Definition at line 200 of file density.cpp.

8.22.2.3 template<class scalartype_ > scalartype N01< scalartype_ >::operator() (array< scalartype > *x*) [inline]

Evaluate the negative log density.

Definition at line 193 of file density.cpp.

8.22.2.4 template<class scalartype_ > scalartype N01< scalartype_ >::operator() (scalartype *x*) [inline]

Definition at line 196 of file density.cpp.

8.22.2.5 template<class scalartype_ > N01< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.22.3 Member Data Documentation

8.22.3.1 template<class scalartype_ > N01< scalartype_ >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 201 of file density.cpp.

The documentation for this class was generated from the following file:

- density.cpp

8.23 density::N01< scalartype_ > Class Template Reference

Standardized normal distribution.

Public Member Functions

- scalartype operator() (array< scalartype > x)
  Evaluate the negative log density.
- scalartype operator() (scalartype x)
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (scalartype_)

8.23.1 Detailed Description

template<class scalartype_>class density::N01< scalartype_ >

Standardized normal distribution.
Class to evaluate the negative log density of a (multivariate) standard normal distribution.

```
Examples: N01()
```

Definition at line 190 of file tmbutils.cpp.

8.23.2 Member Function Documentation

8.23.2.1 template<class scalartype_ > arraytype density::N01< scalartype_ >::jacobian (arraytype *x*) [inline]

Definition at line 200 of file tmbutils.cpp.

8.23.2.2 template<class scalartype_ > int density::N01< scalartype_ >::ndim ( ) [inline]

Definition at line 201 of file tmbutils.cpp.

8.23.2.3 template<class scalartype_ > scalartype density::N01< scalartype_ >::operator() (array< scalartype > *x*) [inline]

Evaluate the negative log density.

Definition at line 194 of file tmbutils.cpp.

8.23.2.4 template<class scalartype_ > scalartype density::N01< scalartype_ >::operator() (scalartype *x*) [inline]

Definition at line 197 of file tmbutils.cpp.

8.23.2.5 template<class scalartype_ > density::N01< scalartype_ >::TYPEDEFS (scalartype_) [private]

8.23.3 Member Data Documentation

8.23.3.1 template<class scalartype_ > density::N01< scalartype_ >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 202 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.24 objective_function< Type > Class Template Reference

Type definition of user-provided objective function (i.e. neg. log. like)

`#include <tmb_core.hpp>`

Public Member Functions

- void pushParname (const char *x)

  Called once for each occurance of PARAMETER_.
- bool parallel_region ()
- int count_parallel_regions ()
- void set_parallel_region (int i)
- objective_function (SEXP data_, SEXP parameters_, SEXP report_)

  Constructor which among other things gives a value to "theta".
- SEXP defaultpar ()

  Extract theta vector from objetive function object.
- SEXP parNames ()

  Extract parnames vector from objetive function object.
- double value (double x)
- double value (AD< double > x)
- double value (AD< AD< double > > x)
- double value (AD< AD< AD< double > > > x)
- int nparms (SEXP obj)

  Find the length of theta, i.e. in application obj=parameters.
- void fill (vector< Type > &x, const char *nam)
- void fill (matrix< Type > &x, const char *nam)
- template<class ArrayType >
  void fill (ArrayType &x, const char *nam)

- template<class ArrayType >
  void fillmap (ArrayType &x, const char *nam)
- SEXP getShape (const char *nam, RObjectTester expectedtype=NULL)
- template<class ArrayType >
  ArrayType fillShape (ArrayType x, const char *nam)
- void fill (Type &x, char const *nam)
- Type operator() ()

Public Attributes

- SEXP data
- SEXP parameters
- SEXP report
- int index
- vector< Type > theta

  Consists of unlist(parameters_)
- vector< const char * > thetanames

  In R notation: names(theta). Contains repeated values.
- report_stack< Type > reportvector

  Used by "ADREPORT".
- bool reversefill
- vector< const char * > parnames

  One name for each PARAMETER_ in user template.
- bool parallel_ignore_statements
- int current_parallel_region
- int selected_parallel_region
- int max_parallel_regions

8.24.1 Detailed Description

template<class Type>class objective_function< Type >

Type definition of user-provided objective function (i.e. neg. log. like)

Definition at line 295 of file tmb_core.hpp.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 template<class Type> objective_function< Type >::objective_function (SEXP *data_*, SEXP *parameters_*, SEXP *report_*) [inline]

Constructor which among other things gives a value to "theta".

Definition at line 372 of file tmb_core.hpp.

8.24.3 Member Function Documentation

8.24.3.1 template<class Type> int objective_function< Type >::count_parallel_regions ( ) [inline]

Definition at line 346 of file tmb_core.hpp.

Referenced by MakeADFunObject(), and MakeADGradObject().

8.24.3.2 template<class Type> SEXP objective_function< Type >::defaultpar ( ) [inline]

Extract theta vector from objetive function object.

Definition at line 397 of file tmb_core.hpp.

Referenced by MakeADFunObject(), MakeADGradObject(), and MakeADHessObject().

8.24.3.3 template<class Type> void objective_function< Type >::fill (vector< Type > & *x,* const char * *nam*) [inline]

Definition at line 447 of file tmb_core.hpp.

Referenced by objective_function< Type >::fillShape().

8.24.3.4 template<class Type> void objective_function< Type >::fill (matrix< Type > & *x,* const char * *nam*) [inline]

Definition at line 455 of file tmb_core.hpp.

8.24.3.5 template<class Type> template<class ArrayType > void objective_function< Type >::fill (ArrayType & *x,* const char * *nam*) [inline]

Definition at line 466 of file tmb_core.hpp.

8.24.3.6 template<class Type> void objective_function< Type >::fill (Type & *x,* char const * *nam*) [inline]

Definition at line 510 of file tmb_core.hpp.

8.24.3.7 template<class Type> template<class ArrayType > void objective_function< Type >::fillmap (ArrayType & *x,* const char * *nam*) [inline]

Definition at line 477 of file tmb_core.hpp.

Referenced by objective_function< Type >::fillShape().

8.24.3.8 template<class Type> template<class ArrayType > ArrayType objective_function< Type >::fillShape (ArrayType *x,* const char * *nam*) [inline]

Definition at line 502 of file tmb_core.hpp.

8.24.3.9 template<class Type> SEXP objective_function< Type >::getShape (const char * *nam,* RObjectTester *expectedtype* = NULL) [inline]

Definition at line 492 of file tmb_core.hpp.

8.24.3.10 template<class Type> int objective_function< Type >::nparms (SEXP *obj*) [inline]

Find the length of theta, i.e. in application obj=parameters.

Definition at line 435 of file tmb_core.hpp.

Referenced by objective_function< Type >::objective_function().

8.24.3.11 template<class Type> Type objective_function< Type >::operator() ( )

Examples:

ar1xar1.cpp, atomic.cpp, linreg.cpp, matrix_arrays.cpp, nmix.cpp, orange_big.cpp, randomregression.cpp, rw.cpp, sdv_multi.cpp, simple.cpp, socatt.cpp, spatial.cpp, and sumtest.cpp.

Referenced by objective_function< Type >::count_parallel_regions().

8.24.3.12 template<class Type> bool objective_function< Type >::parallel_region ( ) [inline]

Definition at line 337 of file tmb_core.hpp.

8.24.3.13 template<class Type> SEXP objective_function< Type >::parNames ( ) [inline]

Extract parnames vector from objetive function object.

Definition at line 415 of file tmb_core.hpp.

Referenced by getParameterOrder().

8.24.3.14 template<class Type> void objective_function< Type >::pushParname (const char * *x*) [inline]

Called once for each occurance of PARAMETER_.

Definition at line 311 of file tmb_core.hpp.

Referenced by objective_function< Type >::fill(), and objective_function< Type >::fillmap().

8.24.3.15 template<class Type> void objective_function< Type >::set_parallel_region (int *i*) [inline]

Definition at line 355 of file tmb_core.hpp.

Referenced by MakeADFunObject(), MakeADGradObject(), and MakeADHessObject2().

8.24.3.16 template<class Type> double objective_function< Type >::value (double *x*) [inline]

Definition at line 429 of file tmb_core.hpp.

Referenced by objective_function< Type >::defaultpar().

8.24.3.17 template<class Type> double objective_function< Type >::value (AD< double > *x*) [inline]

Definition at line 430 of file tmb_core.hpp.

8.24.3.18 template<class Type> double objective_function< Type >::value (AD< AD< double > > *x*) [inline]

Definition at line 431 of file tmb_core.hpp.

8.24.3.19 template<class Type> double objective_function< Type >::value (AD< AD< AD< double > > > *x*) [inline]

Definition at line 432 of file tmb_core.hpp.

8.24.4 Member Data Documentation

8.24.4.1 template<class Type> int objective_function< Type >::current_parallel_region

Definition at line 332 of file tmb_core.hpp.

Referenced by objective_function< Type >::count_parallel_regions(), objective_function< Type >::objective_function(), objective_function< Type >::parallel_region(), and objective_function< Type >::set_parallel_region().

8.24.4.2 template<class Type> SEXP objective_function< Type >::data

Definition at line 299 of file tmb_core.hpp.

Referenced by objective_function< Type >::objective_function().

8.24.4.3 template<class Type> int objective_function< Type >::index

Definition at line 303 of file tmb_core.hpp.

Referenced by EvalDoubleFunObject(), objective_function< Type >::fill(), objective_function< Type >::fillmap(), and objective_function< Type >::objective_function().

8.24.4.4 template<class Type> int objective_function< Type >::max_parallel_regions

Definition at line 334 of file tmb_core.hpp.

Referenced by objective_function< Type >::count_parallel_regions(), objective_function< Type >::objective_function(), and objective_function< Type >::parallel_region().

8.24.4.5 template<class Type> bool objective_function< Type >::parallel_ignore_statements

Definition at line 331 of file tmb_core.hpp.

Referenced by objective_function< Type >::count_parallel_regions(), objective_function< Type >::parallel_region(), and objective_function< Type >::set_parallel_region().

8.24.4.6 template<class Type> SEXP objective_function< Type >::parameters

Definition at line 300 of file tmb_core.hpp.

Referenced by objective_function< Type >::fillmap(), objective_function< Type >::fillShape(), objective_function< Type >::getShape(), and objective_function< Type >::objective_function().

8.24.4.7 template<class Type> vector<const char*> objective_function< Type >::parnames

One name for each PARAMETER_ in user template.

Definition at line 308 of file tmb_core.hpp.

Referenced by EvalDoubleFunObject(), objective_function< Type >::parNames(), and objective_function< Type >::pushParname().

8.24.4.8 template<class Type> SEXP objective_function< Type >::report

Definition at line 301 of file tmb_core.hpp.

Referenced by objective_function< Type >::objective_function().

8.24.4.9 template<class Type> report_stack<Type> objective_function< Type >::reportvector

Used by "ADREPORT".

Definition at line 306 of file tmb_core.hpp.

Referenced by EvalDoubleFunObject(), and MakeADFunObject().

8.24.4.10 template<class Type> bool objective_function< Type >::reversefill

Definition at line 307 of file tmb_core.hpp.

Referenced by objective_function< Type >::fill(), objective_function< Type >::fillmap(), and objective_function< Type >::objective_function().

8.24.4.11 template<class Type> int objective_function< Type >::selected_parallel_region

Definition at line 333 of file tmb_core.hpp.

Referenced by objective_function< Type >::count_parallel_regions(), objective_function< Type >::objective_function(), objective_function< Type >::parallel_region(), and objective_function< Type >::set_parallel_region().

8.24.4.12 template<class Type> vector<Type> objective_function< Type >::theta

Consists of unlist(parameters_)

Definition at line 304 of file tmb_core.hpp.

Referenced by objective_function< Type >::defaultpar(), EvalDoubleFunObject(), objective_function< Type >::fill(), objective_function< Type >::fillmap(), MakeADFunObject(), MakeADGradObject(), MakeADHessObject(), MakeADHessObject2(), and objective_function< Type >::objective_function().

8.24.4.13 template<class Type> vector<const char*> objective_function< Type >::thetanames

In R notation: names(theta). Contains repeated values.

Definition at line 305 of file tmb_core.hpp.

Referenced by objective_function< Type >::defaultpar(), objective_function< Type >::fill(), objective_function< Type >::fillmap(), and objective_function< Type >::objective_function().

The documentation for this class was generated from the following file:

- tmb_core.hpp

8.25 tmbutils::order< Type > Class Template Reference

Public Member Functions

- order (vector< Type > x)
- vector< Type > operator() (vector< Type > x)
- array< Type > operator() (array< Type > x)

Public Attributes

- vector< Type > iperm
- matrix< Type > P
- int n

8.25.1 Detailed Description

template<class Type>class tmbutils::order< Type >

Definition at line 13 of file tmbutils.cpp.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 template<class Type > tmbutils::order< Type >::order (vector< Type > *x*) [inline]

Definition at line 18 of file tmbutils.cpp.

8.25.3 Member Function Documentation

8.25.3.1 template<class Type > vector<Type> tmbutils::order< Type >::operator() (vector< Type > *x*) [inline]

Definition at line 40 of file tmbutils.cpp.

8.25.3.2 template<class Type > array<Type> tmbutils::order< Type >::operator() (array< Type > *x*) [inline]

Definition at line 50 of file tmbutils.cpp.

8.25.4 Member Data Documentation

8.25.4.1 template<class Type > vector<Type> tmbutils::order< Type >::iperm

Definition at line 15 of file tmbutils.cpp.

8.25.4.2 template<class Type > int tmbutils::order< Type >::n

Definition at line 17 of file tmbutils.cpp.

8.25.4.3 template<class Type > matrix<Type> tmbutils::order< Type >::P

Definition at line 16 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.26 order< Type > Class Template Reference

Public Member Functions

- order (vector< Type > x)
- vector< Type > operator() (vector< Type > x)
- array< Type > operator() (array< Type > x)

Public Attributes

- vector< Type > iperm
- matrix< Type > P
- int n

8.26.1 Detailed Description

template<class Type>class order< Type >

Definition at line 12 of file order.cpp.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 template<class Type> order< Type >::order (vector< Type > *x*) [inline]

Definition at line 17 of file order.cpp.

8.26.3 Member Function Documentation

8.26.3.1 template<class Type> vector<Type> order< Type >::operator() (vector< Type > *x*) [inline]

Definition at line 39 of file order.cpp.

8.26.3.2 template<class Type> array<Type> order< Type >::operator() (array< Type > *x*) [inline]

Definition at line 49 of file order.cpp.

8.26.4 Member Data Documentation

8.26.4.1 template<class Type> vector<Type> order< Type >::iperm

Definition at line 14 of file order.cpp.

Referenced by order< Type >::order().

8.26.4.2 template<class Type> int order< Type >::n

Definition at line 16 of file order.cpp.

Referenced by order< Type >::operator()(), and order< Type >::order().

8.26.4.3 template<class Type> matrix<Type> order< Type >::P

Definition at line 15 of file order.cpp.

Referenced by order< Type >::operator()(), and order< Type >::order().

The documentation for this class was generated from the following file:

- order.cpp

8.27 parallel_accumulator< Type > Struct Template Reference

`#include <tmb_core.hpp>`

Public Member Functions

- parallel_accumulator (objective_function< Type > *obj_)
- void operator+= (Type x)
- void operator-= (Type x)
- operator Type ()

Public Attributes

- Type result
- objective_function< Type > * obj

8.27.1 Detailed Description

template<class Type>struct parallel_accumulator< Type >

Definition at line 526 of file tmb_core.hpp.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 template<class Type > parallel_accumulator< Type >::parallel_accumulator (objective_function< Type > * *obj_*) [inline]

Definition at line 529 of file tmb_core.hpp.

8.27.3 Member Function Documentation

8.27.3.1 template<class Type > parallel_accumulator< Type >::operator Type ( ) [inline]

Definition at line 543 of file tmb_core.hpp.

8.27.3.2 template<class Type > void parallel_accumulator< Type >::operator+= (Type *x*) [inline]

Definition at line 537 of file tmb_core.hpp.

8.27.3.3 template<class Type > void parallel_accumulator< Type >::operator-= (Type *x*) [inline]

Definition at line 540 of file tmb_core.hpp.

8.27.4 Member Data Documentation

8.27.4.1 template<class Type > objective_function<Type>* parallel_accumulator< Type >::obj

Definition at line 528 of file tmb_core.hpp.

Referenced by parallel_accumulator< Type >::operator+=(), parallel_accumulator< Type >::operator-=(), and parallel_accumulator< Type >::parallel_accumulator().

8.27.4.2 template<class Type > Type parallel_accumulator< Type >::result

Definition at line 527 of file tmb_core.hpp.

Referenced by parallel_accumulator< Type >::operator Type(), parallel_accumulator< Type >::operator+=(), parallel_accumulator< Type >::operator-=(), and parallel_accumulator< Type >::parallel_accumulator().

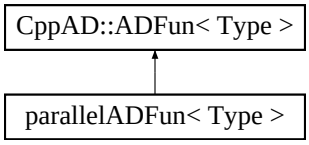
The documentation for this struct was generated from the following file:

- tmb_core.hpp

8.28 parallelADFun< Type > Struct Template Reference

`#include <start_parallel.hpp>`

Inheritance diagram for parallelADFun< Type >:



Public Types

- typedef ADFun< Type > Base

Public Member Functions

- parallelADFun (vector< Base * > vecpf_)
- parallelADFun (vector< sphess * > H)
- ~parallelADFun ()
- sphess_t< parallelADFun< double > > convert ()
- template<typename VectorBase >
  VectorBase subset (const VectorBase &x, size_t tapeid, int p=1)
- template<typename VectorBase >
  void addinsert (VectorBase &x, const VectorBase &y, size_t tapeid, int p=1)
- size_t Domain ()
- size_t Range ()
- template<typename VectorBase >
  VectorBase Forward (size_t p, const VectorBase &x, std::ostream &s=std::cout)
- template<typename VectorBase >
  VectorBase Reverse (size_t p, const VectorBase &v)
- template<typename VectorBase >
  VectorBase Jacobian (const VectorBase &x)
- template<typename VectorBase >
  VectorBase Hessian (const VectorBase &x, size_t rangecomponent)
- void optimize ()

Public Attributes

- int ntapes
- vector< Base * > vecpf
- vector< vector< size_t > > vecind

- size_t domain
- size_t range
- vector< sphess * > H_
- vector< int > veci
- vector< int > vecj

8.28.1 Detailed Description

template<class Type>struct parallelADFun< Type >

Definition at line 52 of file start_parallel.hpp.

8.28.2 Member Typedef Documentation

8.28.2.1 template<class Type> typedef ADFun<Type> parallelADFun< Type >::Base

Definition at line 53 of file start_parallel.hpp.

8.28.3 Constructor & Destructor Documentation

8.28.3.1 template<class Type> parallelADFun< Type >::parallelADFun (vector< Base * > *vecpf_*) [inline]

Definition at line 74 of file start_parallel.hpp.

8.28.3.2 template<class Type> parallelADFun< Type >::parallelADFun (vector< sphess * > *H*) [inline]

Definition at line 95 of file start_parallel.hpp.

8.28.3.3 template<class Type> parallelADFun< Type >::~parallelADFun ( ) [inline]

Definition at line 141 of file start_parallel.hpp.

8.28.4 Member Function Documentation

8.28.4.1 template<class Type> template<typename VectorBase > void parallelADFun< Type >::addinsert (VectorBase & *x*, const VectorBase & *y*, size_t *tapeid*, int *p* = 1) [inline]

Definition at line 164 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::Forward(), parallelADFun< Type >::Hessian(), and parallelADFun< Type >::Jacobian().

8.28.4.2 template<class Type> sphess_t<parallelADFun<double> > parallelADFun< Type >::convert ( ) [inline]

Definition at line 148 of file start_parallel.hpp.

8.28.4.3 template<class Type> size_t parallelADFun< Type >::Domain ( ) [inline]

Definition at line 171 of file start_parallel.hpp.

8.28.4.4 template<class Type> template<typename VectorBase > VectorBase parallelADFun< Type >::Forward (size_t *p,* const VectorBase & *x,* std::ostream & *s* = std::cout) [inline]

Definition at line 180 of file start_parallel.hpp.

8.28.4.5 template<class Type> template<typename VectorBase > VectorBase parallelADFun< Type >::Hessian (const VectorBase & *x,* size_t *rangecomponent*) [inline]

Definition at line 220 of file start_parallel.hpp.

8.28.4.6 template<class Type> template<typename VectorBase > VectorBase parallelADFun< Type >::Jacobian (const VectorBase & *x*) [inline]

Definition at line 208 of file start_parallel.hpp.

8.28.4.7 template<class Type> void parallelADFun< Type >::optimize ( ) [inline]

Definition at line 232 of file start_parallel.hpp.

Referenced by optimizeADFunObject().

8.28.4.8 template<class Type> size_t parallelADFun< Type >::Range ( ) [inline]

Definition at line 172 of file start_parallel.hpp.

8.28.4.9 template<class Type> template<typename VectorBase > VectorBase parallelADFun< Type >::Reverse (size_t *p,* const VectorBase & *v*) [inline]

Definition at line 196 of file start_parallel.hpp.

8.28.4.10 template<class Type> template<typename VectorBase > VectorBase parallelADFun< Type >::subset (const VectorBase & *x,* size_t *tapeid,* int *p* = 1) [inline]

Definition at line 154 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::Reverse().

8.28.5 Member Data Documentation

8.28.5.1 template<class Type> size_t parallelADFun< Type >::domain

Definition at line 63 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::Domain(), parallelADFun< Type >::Hessian(), parallelADFun< Type >::Jacobian(), parallelADFun< Type >::parallelADFun(), and parallelADFun< Type >::Reverse().

8.28.5.2 template<class Type> vector<sphess* > parallelADFun< Type >::H_

Definition at line 66 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::parallelADFun().

8.28.5.3 template<class Type> int parallelADFun< Type >::ntapes

Definition at line 60 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::Forward(), parallelADFun< Type >::Hessian(), parallelADFun< Type >::Jacobian(), parallelADFun< Type >::optimize(), parallelADFun< Type >::parallelADFun(), and parallelADFun< Type >::Reverse().

8.28.5.4 template<class Type> size_t parallelADFun< Type >::range

Definition at line 64 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::Forward(), parallelADFun< Type >::Jacobian(), parallelADFun< Type >::parallelADFun(), and parallelADFun< Type >::Range().

8.28.5.5 template<class Type> vector<int> parallelADFun< Type >::veci

Definition at line 68 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::convert(), and parallelADFun< Type >::parallelADFun().

8.28.5.6 template<class Type> vector<vector<size_t> > parallelADFun< Type >::vecind

Definition at line 62 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::addinsert(), parallelADFun< Type >::parallelADFun(), and parallelADFun< Type >::subset().

8.28.5.7 template<class Type> vector<int> parallelADFun< Type >::vecj

Definition at line 69 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::convert(), and parallelADFun< Type >::parallelADFun().

8.28.5.8 template<class Type> vector<Base*> parallelADFun< Type >::vecpf

Definition at line 61 of file start_parallel.hpp.

Referenced by parallelADFun< Type >::Forward(), parallelADFun< Type >::Hessian(), parallelADFun< Type >::Jacobian(), parallelADFun< Type >::optimize(), parallelADFun< Type >::parallelADFun(), parallelADFun< Type >::Reverse(), and parallelADFun< Type >::~parallelADFun().

The documentation for this struct was generated from the following file:

- start_parallel.hpp

8.29 piecewice< Type > Class Template Reference

`#include <Vectorize.hpp>`

Public Member Functions

- piecewice (const vector< Type > &x_, const vector< Type > &y, bool leftcontinuous_=true)
- Type operator() (const Type &t)

Public Attributes

- bool leftcontinuous

Private Attributes

- Type y0
- vector< Type > x
- vector< Type > dy
- int n

8.29.1 Detailed Description

template<class Type>class piecewice< Type >

Definition at line 168 of file Vectorize.hpp.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 template<class Type > piecewice< Type >::piecewice (const vector< Type > & *x_*, const vector< Type > & *y*, bool *leftcontinuous_* = true) [inline]

Definition at line 175 of file Vectorize.hpp.

8.29.3 Member Function Documentation

8.29.3.1 template<class Type > Type piecewice< Type >::operator() (const Type & *t*) [inline]

Definition at line 184 of file Vectorize.hpp.

8.29.4 Member Data Documentation

8.29.4.1 template<class Type > vector<Type> piecewice< Type >::dy [private]

Definition at line 171 of file Vectorize.hpp.

Referenced by piecewice< Type >::operator()(), and piecewice< Type >::piecewice().

8.29.4.2 template<class Type > bool piecewice< Type >::leftcontinuous

Definition at line 174 of file Vectorize.hpp.

Referenced by piecewice< Type >::operator()().

8.29.4.3 template<class Type > int piecewice< Type >::n [private]

Definition at line 172 of file Vectorize.hpp.

Referenced by piecewice< Type >::operator()(), and piecewice< Type >::piecewice().

8.29.4.4 template<class Type > vector<Type> piecewice< Type >::x [private]

Definition at line 171 of file Vectorize.hpp.

Referenced by piecewice< Type >::operator()(), and piecewice< Type >::piecewice().

8.29.4.5 template<class Type > Type piecewice< Type >::y0 [private]

Definition at line 170 of file Vectorize.hpp.

Referenced by piecewice< Type >::operator()(), and piecewice< Type >::piecewice().

The documentation for this class was generated from the following file:

- Vectorize.hpp

8.30 PROJ_t< distribution > Class Template Reference

Projection of multivariate gaussian variable.

Public Member Functions

- PROJ_t ()
- PROJ_t (distribution f_, vector< int > proj_)
- void initialize (int n_)
- vectortype projB (vectortype x)
- vectortype setZeroB (vectortype x)
- scalartype operator() (vectortype x)
- arraytype projB (arraytype x)
- arraytype setZeroB (arraytype x)
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- vector< int > proj
- vector< int > cproj
- int n
- int nA
- int nB
- matrixtype Q
- MVNORM_t< scalartype > dmvnorm
- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (typename distribution::scalartype)

Private Attributes

- distribution f
- bool initialized

8.30.1 Detailed Description

template<class distribution>class PROJ_t< distribution >

Projection of multivariate gaussian variable.

Preserves sparseness if possible. Generally it is not.

```
Given a gaussian density f:R^n -> R.
Given an integer vector "proj" with elements in 1,...,n.
Construct the mariginal density of "x[proj]".

Details:
--------
Let x=[x_A]
      [x_B]
with precision
    Q=[Q_AA  Q_AB]
      [Q_BA  Q_BB]
and assume that proj=A.
The marginal density is (with notation 0:=0*x_B )
p_A(x_A)=p(x_A,x_B)/p(x_B|x_A)=p(x_A,0)/p(0|x_A)
Now see that
1. p(x_A,0) is easy because full precision is sparse.
2. p(0|x_A) is N(-Q_BB^-1 * Q_BA x_A,  Q_BB^-1) so
   p(0|x_A) = |Q_BB|^.5 * exp(-.5*x_A Q_AB * Q_BB^-1 * Q_BA x_A)

   Trick to evaluate this with what we have available:
   Note 1: Q_BA x_A = [0 I_BB] * full_jacobian([ x_A  ]
                                               [ 0    ] )


           Call this quantity "y_B" we have
           p(0|x_A) = |Q_BB|^.5 * exp(-.5*y_B' * Q_BB^-1 * y_B)

   Note 2: Consider now a density with _covariance_ Q_BB
           phi(y)=|Q_BB|^-.5 * exp(-.5*y' * Q_BB^-1 * y)
           Then
           phi(y)/phi(0)^2=|Q_BB|^.5 * exp(-.5*y' * Q_BB^-1 * y)
           which is actually the desired expression of p(0|x_A).

Summary:
-------
Negative log-density of A-marginal is
-log p(x_A,0) + log phi(y) - 2*log(phi(0))
= f(x_A,0) - dmvnorm(y_B) + 2*dmvnorm(0)
```

Definition at line 968 of file density.cpp.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 template<class distribution> PROJ_t< distribution >::PROJ_t ( ) [inline]

Definition at line 979 of file density.cpp.

8.30.2.2 template<class distribution> PROJ_t< distribution >::PROJ_t (distribution *f_*, vector< int > *proj_*) [inline]

Definition at line 980 of file density.cpp.

8.30.3 Member Function Documentation

8.30.3.1 template<class distribution> void PROJ_t< distribution >::initialize (int *n_*) [inline]

Definition at line 985 of file density.cpp.

Referenced by PROJ_t< distribution >::jacobian(), and PROJ_t< distribution >::operator()().

8.30.3.2 template<class distribution> arraytype PROJ_t< distribution >::jacobian (arraytype *x*) [inline]

Definition at line 1054 of file density.cpp.

8.30.3.3 template<class distribution> int PROJ_t< distribution >::ndim ( ) [inline]

Definition at line 1075 of file density.cpp.

8.30.3.4 template<class distribution> scalartype PROJ_t< distribution >::operator() (vectortype *x*) [inline]

Definition at line 1031 of file density.cpp.

8.30.3.5 template<class distribution> vectortype PROJ_t< distribution >::projB (vectortype *x*) [inline]

Definition at line 1022 of file density.cpp.

Referenced by PROJ_t< distribution >::jacobian(), and PROJ_t< distribution >::operator()().

8.30.3.6 template<class distribution> arraytype PROJ_t< distribution >::projB (arraytype *x*) [inline]

Definition at line 1042 of file density.cpp.

8.30.3.7 template<class distribution> vectortype PROJ_t< distribution >::setZeroB (vectortype *x*) [inline]

Definition at line 1027 of file density.cpp.

Referenced by PROJ_t< distribution >::jacobian(), and PROJ_t< distribution >::operator()().

8.30.3.8 template<class distribution> arraytype PROJ_t< distribution >::setZeroB (arraytype *x*) [inline]

Definition at line 1050 of file density.cpp.

8.30.3.9 template<class distribution> PROJ_t< distribution >::TYPEDEFS (typename distribution::scalartype) [private]

8.30.4 Member Data Documentation

8.30.4.1 template<class distribution> vector<int> PROJ_t< distribution >::cproj

Definition at line 975 of file density.cpp.

Referenced by PROJ_t< distribution >::initialize(), PROJ_t< distribution >::jacobian(), PROJ_t< distribution >::projB(), and PROJ_t< distribution >::setZeroB().

8.30.4.2 template<class distribution> MVNORM_t<scalartype> PROJ_t< distribution >::dmvnorm

Definition at line 978 of file density.cpp.

Referenced by PROJ_t< distribution >::initialize(), PROJ_t< distribution >::jacobian(), and PROJ_t< distribution >::operator()().

8.30.4.3 template<class distribution> distribution PROJ_t< distribution >::f [private]

Definition at line 971 of file density.cpp.

Referenced by PROJ_t< distribution >::initialize(), PROJ_t< distribution >::jacobian(), PROJ_t< distribution >::ndim(), PROJ_t< distribution >::operator()(), and PROJ_t< distribution >::PROJ_t().

8.30.4.4 template<class distribution> bool PROJ_t< distribution >::initialized [private]

Definition at line 972 of file density.cpp.

Referenced by PROJ_t< distribution >::initialize(), and PROJ_t< distribution >::PROJ_t().

8.30.4.5 template<class distribution> int PROJ_t< distribution >::n

Definition at line 976 of file density.cpp.

Referenced by PROJ_t< distribution >::initialize(), and PROJ_t< distribution >::projB().

8.30.4.6 template<class distribution> int PROJ_t< distribution >::nA

Definition at line 976 of file density.cpp.

Referenced by PROJ_t< distribution >::initialize().

8.30.4.7 template<class distribution> int PROJ_t< distribution >::nB

Definition at line 976 of file density.cpp.

Referenced by PROJ_t< distribution >::initialize(), PROJ_t< distribution >::jacobian(), PROJ_t< distribution >::projB(), and PROJ_t< distribution >::setZeroB().

8.30.4.8 template<class distribution> vector<int> PROJ_t< distribution >::proj

Definition at line 974 of file density.cpp.

Referenced by PROJ_t< distribution >::initialize(), and PROJ_t< distribution >::PROJ_t().

8.30.4.9 template<class distribution> matrixtype PROJ_t< distribution >::Q

Definition at line 977 of file density.cpp.

Referenced by PROJ_t< distribution >::initialize().

8.30.4.10 template<class distribution> PROJ_t< distribution >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 1076 of file density.cpp.

The documentation for this class was generated from the following file:

- density.cpp

8.31 density::PROJ_t< distribution > Class Template Reference

Projection of multivariate gaussian variable.

Public Member Functions

- PROJ_t ()
- PROJ_t (distribution f_, vector< int > proj_)
- void initialize (int n_)
- vectortype projB (vectortype x)
- vectortype setZeroB (vectortype x)
- scalartype operator() (vectortype x)
- arraytype projB (arraytype x)
- arraytype setZeroB (arraytype x)
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- vector< int > proj
- vector< int > cproj
- int n
- int nA
- int nB
- matrixtype Q
- MVNORM_t< scalartype > dmvnorm
- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (typename distribution::scalartype)

Private Attributes

- distribution f
- bool initialized

8.31.1 Detailed Description

template<class distribution>class density::PROJ_t< distribution >

Projection of multivariate gaussian variable.

Preserves sparseness if possible. Generally it is not.

```
Given a gaussian density f:R^n -> R.
Given an integer vector "proj" with elements in 1,...,n.
Construct the mariginal density of "x[proj]".

Details:
--------
```

```
Let x=[x_A]
      [x_B]
with precision
    Q=[Q_AA  Q_AB]
      [Q_BA  Q_BB]
and assume that proj=A.
The marginal density is (with notation 0:=0*x_B )
p_A(x_A)=p(x_A,x_B)/p(x_B|x_A)=p(x_A,0)/p(0|x_A)
Now see that
1. p(x_A,0) is easy because full precision is sparse.
2. p(0|x_A) is N(-Q_BB^-1 * Q_BA x_A,  Q_BB^-1) so
   p(0|x_A) = |Q_BB|^.5 * exp(-.5*x_A Q_AB * Q_BB^-1 * Q_BA x_A)

   Trick to evaluate this with what we have available:
   Note 1: Q_BA x_A = [0 I_BB] * full_jacobian([ x_A  ]
                                               [ 0    ] )


           Call this quantity "y_B" we have
           p(0|x_A) = |Q_BB|^.5 * exp(-.5*y_B' * Q_BB^-1 * y_B)

   Note 2: Consider now a density with _covariance_ Q_BB
           phi(y)=|Q_BB|^-.5 * exp(-.5*y' * Q_BB^-1 * y)
           Then
           phi(y)/phi(0)^2=|Q_BB|^.5 * exp(-.5*y' * Q_BB^-1 * y)
           which is actually the desired expression of p(0|x_A).

Summary:
-------
Negative log-density of A-marginal is
-log p(x_A,0) + log phi(y) - 2*log(phi(0))
= f(x_A,0) - dmvnorm(y_B) + 2*dmvnorm(0)
```

Definition at line 969 of file tmbutils.cpp.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 template<class distribution > density::PROJ_t< distribution >::PROJ_t ( ) [inline]

Definition at line 980 of file tmbutils.cpp.

8.31.2.2 template<class distribution > density::PROJ_t< distribution >::PROJ_t (distribution *f_*, vector< int > *proj_*) [inline]

Definition at line 981 of file tmbutils.cpp.

8.31.3 Member Function Documentation

8.31.3.1 template<class distribution > void density::PROJ_t< distribution >::initialize (int *n_*) [inline]

Definition at line 986 of file tmbutils.cpp.

8.31.3.2 template<class distribution > arraytype density::PROJ_t< distribution >::jacobian (arraytype *x*) [inline]

Definition at line 1055 of file tmbutils.cpp.

8.31.3.3 template<class distribution > int density::PROJ_t< distribution >::ndim ( ) [inline]

Definition at line 1076 of file tmbutils.cpp.

8.31.3.4 template<class distribution > scalartype density::PROJ_t< distribution >::operator() (vectortype *x*) [inline]

Definition at line 1032 of file tmbutils.cpp.

8.31.3.5 template<class distribution > vectortype density::PROJ_t< distribution >::projB (vectortype *x*) [inline]

Definition at line 1023 of file tmbutils.cpp.

8.31.3.6 template<class distribution > arraytype density::PROJ_t< distribution >::projB (arraytype *x*) [inline]

Definition at line 1043 of file tmbutils.cpp.

8.31.3.7 template<class distribution > vectortype density::PROJ_t< distribution >::setZeroB (vectortype *x*) [inline]

Definition at line 1028 of file tmbutils.cpp.

8.31.3.8 template<class distribution > arraytype density::PROJ_t< distribution >::setZeroB (arraytype *x*) [inline]

Definition at line 1051 of file tmbutils.cpp.

8.31.3.9 template<class distribution > density::PROJ_t< distribution >::TYPEDEFS (typename distribution::scalartype) [private]

8.31.4 Member Data Documentation

8.31.4.1 template<class distribution > vector<int> density::PROJ_t< distribution >::cproj

Definition at line 976 of file tmbutils.cpp.

8.31.4.2 template<class distribution > MVNORM_t<scalartype> density::PROJ_t< distribution >::dmvnorm

Definition at line 979 of file tmbutils.cpp.

8.31.4.3 template<class distribution > distribution density::PROJ_t< distribution >::f [private]

Definition at line 972 of file tmbutils.cpp.

8.31.4.4 template<class distribution > bool density::PROJ_t< distribution >::initialized [private]

Definition at line 973 of file tmbutils.cpp.

8.31.4.5 template<class distribution > int density::PROJ_t< distribution >::n

Definition at line 977 of file tmbutils.cpp.

8.31.4.6 template<class distribution > int density::PROJ_t< distribution >::nA

Definition at line 977 of file tmbutils.cpp.

8.31.4.7 template<class distribution > int density::PROJ_t< distribution >::nB

Definition at line 977 of file tmbutils.cpp.

8.31.4.8 template<class distribution > vector<int> density::PROJ_t< distribution >::proj

Definition at line 975 of file tmbutils.cpp.

8.31.4.9 template<class distribution > matrixtype density::PROJ_t< distribution >::Q

Definition at line 978 of file tmbutils.cpp.

8.31.4.10 template<class distribution > density::PROJ_t< distribution >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 1077 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.32 report_stack< Type > Struct Template Reference

Used by ADREPORT.

`#include <tmb_core.hpp>`

Public Member Functions

- void clear ()
- void increase (int n, const char *name)
- void push (Type x, const char *name)
- template<class VectorType >
  void push (VectorType x, const char *name)
- void push (matrix< Type > x, const char *name)
- operator vector< Type > ()
- SEXP reportnames ()
- size_t size ()

Public Attributes

- vector< const char * > names
- vector< int > namelength
- vector< Type > result

8.32.1 Detailed Description

template<class Type>struct report_stack< Type >

Used by ADREPORT.

Definition at line 236 of file tmb_core.hpp.

8.32.2 Member Function Documentation

8.32.2.1 template<class Type > void report_stack< Type >::clear ( ) [inline]

Definition at line 240 of file tmb_core.hpp.

8.32.2.2 template<class Type > void report_stack< Type >::increase (int *n*, const char * *name*) [inline]

Definition at line 246 of file tmb_core.hpp.

Referenced by report_stack< Type >::push().

8.32.2.3 template<class Type > report_stack< Type >::operator vector< Type > ( ) [inline]

Definition at line 271 of file tmb_core.hpp.

8.32.2.4 template<class Type > void report_stack< Type >::push (Type *x*, const char * *name*) [inline]

Definition at line 254 of file tmb_core.hpp.

Referenced by report_stack< Type >::push().

8.32.2.5 template<class Type > template<class VectorType > void report_stack< Type >::push (VectorType *x*, const char * *name*) [inline]

Definition at line 260 of file tmb_core.hpp.

8.32.2.6 template<class Type > void report_stack< Type >::push (matrix< Type > *x*, const char * *name*) [inline]

Definition at line 267 of file tmb_core.hpp.

8.32.2.7 template<class Type > SEXP report_stack< Type >::reportnames ( ) [inline]

Definition at line 275 of file tmb_core.hpp.

8.32.2.8 template<class Type > size_t report_stack< Type >::size ( ) [inline]

Definition at line 290 of file tmb_core.hpp.

8.32.3 Member Data Documentation

8.32.3.1 template<class Type > vector<int> report_stack< Type >::namelength

Definition at line 238 of file tmb_core.hpp.

Referenced by report_stack< Type >::clear(), report_stack< Type >::increase(), and report_stack< Type >::reportnames().

8.32.3.2 template<class Type > vector<const char*> report_stack< Type >::names

Definition at line 237 of file tmb_core.hpp.

Referenced by report_stack< Type >::clear(), report_stack< Type >::increase(), and report_stack< Type >::reportnames().

8.32.3.3 template<class Type > vector<Type> report_stack< Type >::result

Definition at line 239 of file tmb_core.hpp.

Referenced by report_stack< Type >::clear(), report_stack< Type >::increase(), report_stack< Type >::operator vector< Type >(), report_stack< Type >::push(), report_stack< Type >::reportnames(), and report_stack< Type >::size().

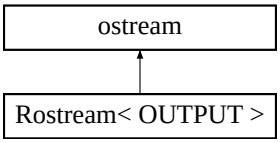
The documentation for this struct was generated from the following file:

- tmb_core.hpp

8.33 Rostream< OUTPUT > Class Template Reference

`#include <Rstream.hpp>`

Inheritance diagram for Rostream< OUTPUT >:



Public Member Functions

- Rostream ()
- ~Rostream ()

Private Types

- typedef Rstreambuf< OUTPUT > Buffer

Private Attributes

- Buffer * buf

8.33.1 Detailed Description

template<bool OUTPUT>class Rostream< OUTPUT >

Definition at line 24 of file Rstream.hpp.

8.33.2 Member Typedef Documentation

8.33.2.1 template<bool OUTPUT> typedef Rstreambuf<OUTPUT> Rostream< OUTPUT >::Buffer [private]

Definition at line 25 of file Rstream.hpp.

8.33.3 Constructor & Destructor Documentation

8.33.3.1 template<bool OUTPUT> Rostream< OUTPUT >::Rostream ( ) [inline]

Definition at line 28 of file Rstream.hpp.

8.33.3.2 template<bool OUTPUT> Rostream< OUTPUT >::~Rostream ( ) [inline]

Definition at line 32 of file Rstream.hpp.

8.33.4 Member Data Documentation

8.33.4.1 template<bool OUTPUT> Buffer* Rostream< OUTPUT >::buf [private]

Definition at line 26 of file Rstream.hpp.

Referenced by Rostream< OUTPUT >::~Rostream().

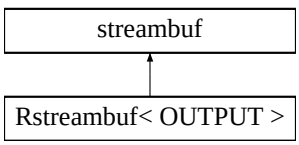
The documentation for this class was generated from the following file:

- Rstream.hpp

8.34 Rstreambuf< OUTPUT > Class Template Reference

`#include <Rstream.hpp>`

Inheritance diagram for Rstreambuf< OUTPUT >:



Public Member Functions

- Rstreambuf ()

Protected Member Functions

- virtual std::streamsize xsputn (const char *s, std::streamsize n)
- virtual int overflow (int c=EOF)
- virtual int sync ()
- template<>
  std::streamsize xsputn (const char *s, std::streamsize num)
- template<>
  std::streamsize xsputn (const char *s, std::streamsize num)
- template<>
  int overflow (int c)
- template<>
  int overflow (int c)
- template<>
  int sync ()
- template<>
  int sync ()

8.34.1 Detailed Description

template<bool OUTPUT>class Rstreambuf< OUTPUT >

Definition at line 13 of file Rstream.hpp.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 template<bool OUTPUT> Rstreambuf< OUTPUT >::Rstreambuf ( ) [inline]

Definition at line 15 of file Rstream.hpp.

8.34.3 Member Function Documentation

8.34.3.1 template<bool OUTPUT> virtual int Rstreambuf< OUTPUT >::overflow (int *c* = EOF) [protected], [virtual]

8.34.3.2 template<> int Rstreambuf< true >::overflow (int *c*) [inline], [protected]

Definition at line 48 of file Rstream.hpp.

8.34.3.3 template<> int Rstreambuf< false >::overflow (int *c*) [inline], [protected]

Definition at line 52 of file Rstream.hpp.

8.34.3.4 template<bool OUTPUT> virtual int Rstreambuf< OUTPUT >::sync ( ) [protected], [virtual]

8.34.3.5 template<> int Rstreambuf< true >::sync ( ) [inline], [protected]

Definition at line 56 of file Rstream.hpp.

8.34.3.6 template<> int Rstreambuf< false >::sync ( ) [inline], [protected]

Definition at line 60 of file Rstream.hpp.

8.34.3.7 template<bool OUTPUT> virtual std::streamsize Rstreambuf< OUTPUT >::xsputn (const char * *s*, std::streamsize *n*) [protected], [virtual]

8.34.3.8 template<> std::streamsize Rstreambuf< true >::xsputn (const char * *s*, std::streamsize *num*) [inline], [protected]

Definition at line 40 of file Rstream.hpp.

8.34.3.9 template<> std::streamsize Rstreambuf< false >::xsputn (const char * *s*, std::streamsize *num*) [inline], [protected]

Definition at line 44 of file Rstream.hpp.

The documentation for this class was generated from the following file:

- Rstream.hpp

8.35 density::SCALE_t< distribution > Class Template Reference

Apply scale transformation on a density.

Public Member Functions

- SCALE_t ()
- SCALE_t (distribution f_, scalartype scale_)
- scalartype operator() (arraytype x)

  Evaluate the negative log density.
- arraytype jacobian (arraytype x)
- int ndim ()
- vectortype variance ()

Private Member Functions

- TYPEDEFS (typename distribution::scalartype)

Private Attributes

- distribution f
- scalartype scale

8.35.1 Detailed Description

template<class distribution>class density::SCALE_t< distribution >

Apply scale transformation on a density.

Assume x has density f. Construct the density of y=scale*x where scale is a scalar.

Parameters

*f_*	distribution
*scale_*	scalar

Definition at line 718 of file tmbutils.cpp.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 template<class distribution > density::SCALE_t< distribution >::SCALE_t ( ) [inline]

Definition at line 724 of file tmbutils.cpp.

8.35.2.2 template<class distribution > density::SCALE_t< distribution >::SCALE_t (distribution *f_*, scalartype *scale_*) [inline]

Definition at line 725 of file tmbutils.cpp.

8.35.3 Member Function Documentation

8.35.3.1 template<class distribution > arraytype density::SCALE_t< distribution >::jacobian (arraytype *x*) [inline]

Definition at line 732 of file tmbutils.cpp.

8.35.3.2 template<class distribution > int density::SCALE_t< distribution >::ndim ( ) [inline]

Definition at line 735 of file tmbutils.cpp.

8.35.3.3 template<class distribution > scalartype density::SCALE_t< distribution >::operator() (arraytype *x*) [inline]

Evaluate the negative log density.

Definition at line 727 of file tmbutils.cpp.

8.35.3.4 template<class distribution > density::SCALE_t< distribution >::TYPEDEFS (typename distribution::scalartype) [private]

8.35.3.5 template<class distribution > vectortype density::SCALE_t< distribution >::variance ( ) [inline]

Definition at line 736 of file tmbutils.cpp.

8.35.4 Member Data Documentation

8.35.4.1 template<class distribution > distribution density::SCALE_t< distribution >::f [private]

Definition at line 721 of file tmbutils.cpp.

8.35.4.2 template<class distribution > scalartype density::SCALE_t< distribution >::scale [private]

Definition at line 722 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.36 SCALE_t< distribution > Class Template Reference

Apply scale transformation on a density.

Public Member Functions

- SCALE_t ()
- SCALE_t (distribution f_, scalartype scale_)
- scalartype operator() (arraytype x)

  Evaluate the negative log density.
- arraytype jacobian (arraytype x)
- int ndim ()
- vectortype variance ()

Private Member Functions

- TYPEDEFS (typename distribution::scalartype)

Private Attributes

- distribution f
- scalartype scale

8.36.1 Detailed Description

template<class distribution>class SCALE_t< distribution >

Apply scale transformation on a density.

Assume x has density f. Construct the density of y=scale*x where scale is a scalar.

Parameters

*f_*	distribution
*scale_*	scalar

Definition at line 717 of file density.cpp.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 template<class distribution> SCALE_t< distribution >::SCALE_t ( ) [inline]

Definition at line 723 of file density.cpp.

8.36.2.2 template<class distribution> SCALE_t< distribution >::SCALE_t (distribution *f_*, scalartype *scale_*) [inline]

Definition at line 724 of file density.cpp.

8.36.3 Member Function Documentation

8.36.3.1 template<class distribution> arraytype SCALE_t< distribution >::jacobian (arraytype *x*) [inline]

Definition at line 731 of file density.cpp.

8.36.3.2 template<class distribution> int SCALE_t< distribution >::ndim ( ) [inline]

Definition at line 734 of file density.cpp.

8.36.3.3 template<class distribution> scalartype SCALE_t< distribution >::operator() (arraytype *x*) [inline]

Evaluate the negative log density.

Definition at line 726 of file density.cpp.

8.36.3.4 template<class distribution> SCALE_t< distribution >::TYPEDEFS (typename distribution::scalartype) [private]

8.36.3.5 template<class distribution> vectortype SCALE_t< distribution >::variance ( ) [inline]

Definition at line 735 of file density.cpp.

8.36.4 Member Data Documentation

8.36.4.1 template<class distribution> distribution SCALE_t< distribution >::f [private]

Definition at line 720 of file density.cpp.

Referenced by SCALE_t< distribution >::jacobian(), SCALE_t< distribution >::ndim(), SCALE_t< distribution >::operator()(), SCALE_t< distribution >::SCALE_t(), and SCALE_t< distribution >::variance().

8.36.4.2 template<class distribution> scalartype SCALE_t< distribution >::scale [private]

Definition at line 721 of file density.cpp.

Referenced by SCALE_t< distribution >::jacobian(), SCALE_t< distribution >::operator()(), SCALE_t< distribution >::SCALE_t(), and SCALE_t< distribution >::variance().

The documentation for this class was generated from the following file:

- density.cpp

8.37 SEPARABLE_t< distribution1, distribution2 > Class Template Reference

Separable extension of two densitites.

Public Member Functions

- SEPARABLE_t ()
- SEPARABLE_t (distribution1 f_, distribution2 g_)
- arraytype jacobian (arraytype x)
- arraytype zeroVector (vector< int > d, int n)

- scalartype operator() (arraytype x)
- int ndim ()
- scalartype operator() (arraytype x, int i)

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (typename distribution1::scalartype)

Private Attributes

- distribution1 f
- distribution2 g

8.37.1 Detailed Description

template<class distribution1, class distribution2>class SEPARABLE_t< distribution1, distribution2 >

Separable extension of two densitites.

Take two densities f and g, and construct the density of their separable extension, defined as the multivariate Gaussian distribution with covariance matrix equal to the kronecker product between the covariance matrices of the two distributions. Note that f acts on the outermost array dimension and g acts on the fastest running array dimension.

```
More precisely: evaluate density
h(x)=|S/(2*pi)|^.5*exp(-.5*x'*S*x)
where S=kronecker(Q,R)=Q%x%R assuming we have access to densities
f(x)=|Q/(2*pi)|^.5*exp(-.5*x'*Q*x)
g(x)=|R/(2*pi)|^.5*exp(-.5*x'*R*x)
(Note: R corresponds to fastest running array dimension in Q%x%R ...)
Let nq=nrow(Q) and nr=nrow(R),
using rules of the kronecker product we have that
* Quadratic form = .5*x'*S*x = .5*x'*(Q%x%I)*(I%x%R)*x
* Normalizing constant =
|S/(2*pi)|^.5 =
|(Q/sqrt(2*pi))%x%(R/sqrt(2*pi))|^.5 =
|(Q/sqrt(2*pi))|^(nr*.5) |(R/sqrt(2*pi))|^(nq*.5) =
... something that can be expressed through the normalizing
constants f(0) and g(0) ...
f(0)^nr * g(0)^nq * sqrt(2*pi)^(nq*nr)
```

Example:

```
// Separable extension of two AR1 processes
Type phi1=0.8;
AR1_t<N01<Type> > f(phi1);
Type phi2=0.8;
AR1_t<N01<Type> > g(phi2);
SEPARABLE_t<AR1_t<N01<Type> > , AR1_t<N01<Type> > > h(
      f,g);
// Can be evaluated on an array:
array<Type> x(10,20);
Type ans=h(x);
```

Definition at line 828 of file density.cpp.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 template<class distribution1, class distribution2> SEPARABLE_t< distribution1, distribution2 >::SEPARABLE_t ( ) [inline]

Definition at line 834 of file density.cpp.

8.37.2.2 template<class distribution1, class distribution2> SEPARABLE_t< distribution1, distribution2 >::SEPARABLE_t (distribution1 *f_*, distribution2 *g_*) [inline]

Definition at line 835 of file density.cpp.

8.37.3 Member Function Documentation

8.37.3.1 template<class distribution1, class distribution2> arraytype SEPARABLE_t< distribution1, distribution2 >::jacobian (arraytype *x*) [inline]

Definition at line 843 of file density.cpp.

Referenced by SEPARABLE_t< distribution1, distribution2 >::operator()().

8.37.3.2 template<class distribution1, class distribution2> int SEPARABLE_t< distribution1, distribution2 >::ndim ( ) [inline]

Definition at line 880 of file density.cpp.

Referenced by SEPARABLE_t< distribution1, distribution2 >::operator()().

8.37.3.3 template<class distribution1, class distribution2> scalartype SEPARABLE_t< distribution1, distribution2 >::operator() (arraytype *x*) [inline]

Definition at line 861 of file density.cpp.

8.37.3.4 template<class distribution1, class distribution2> scalartype SEPARABLE_t< distribution1, distribution2 >::operator() (arraytype *x*, int *i*) [inline]

Definition at line 892 of file density.cpp.

8.37.3.5 template<class distribution1, class distribution2> SEPARABLE_t< distribution1, distribution2 >::TYPEDEFS (typename distribution1::scalartype) [private]

8.37.3.6 template<class distribution1, class distribution2> arraytype SEPARABLE_t< distribution1, distribution2 >::zeroVector (vector< int > *d*, int *n*) [inline]

Definition at line 852 of file density.cpp.

Referenced by SEPARABLE_t< distribution1, distribution2 >::operator()().

8.37.4 Member Data Documentation

8.37.4.1 template<class distribution1, class distribution2> distribution1 SEPARABLE_t< distribution1, distribution2 >::f [private]

Definition at line 831 of file density.cpp.

Referenced by SEPARABLE_t< distribution1, distribution2 >::jacobian(), SEPARABLE_t< distribution1, distribution2 >::ndim(), SEPARABLE_t< distribution1, distribution2 >::operator()(), and SEPARABLE_t< distribution1, distribution2 >::SEPARABLE_t().

8.37.4.2 template<class distribution1, class distribution2> distribution2 SEPARABLE_t< distribution1, distribution2 >::g [private]

Definition at line 832 of file density.cpp.

Referenced by SEPARABLE_t< distribution1, distribution2 >::jacobian(), SEPARABLE_t< distribution1, distribution2 >::ndim(), SEPARABLE_t< distribution1, distribution2 >::operator()(), and SEPARABLE_t< distribution1, distribution2 >::SEPARABLE_t().

8.37.4.3 template<class distribution1, class distribution2> SEPARABLE_t< distribution1, distribution2 >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 881 of file density.cpp.

The documentation for this class was generated from the following file:

- density.cpp

8.38 density::SEPARABLE_t< distribution1, distribution2 > Class Template Reference

Separable extension of two densitites.

Public Member Functions

- SEPARABLE_t ()
- SEPARABLE_t (distribution1 f_, distribution2 g_)
- arraytype jacobian (arraytype x)
- arraytype zeroVector (vector< int > d, int n)
- scalartype operator() (arraytype x)
- int ndim ()
- scalartype operator() (arraytype x, int i)

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (typename distribution1::scalartype)

Private Attributes

- distribution1 f
- distribution2 g

8.38.1 Detailed Description

template<class distribution1, class distribution2>class density::SEPARABLE_t< distribution1, distribution2 >

Separable extension of two densitites.

Take two densities f and g, and construct the density of their separable extension, defined as the multivariate Gaussian distribution with covariance matrix equal to the kronecker product between the covariance matrices of the two distributions. Note that f acts on the outermost array dimension and g acts on the fastest running array dimension.

```
More precisely: evaluate density
h(x)=|S/(2*pi)|^.5*exp(-.5*x'*S*x)
where S=kronecker(Q,R)=Q%x%R assuming we have access to densities
f(x)=|Q/(2*pi)|^.5*exp(-.5*x'*Q*x)
g(x)=|R/(2*pi)|^.5*exp(-.5*x'*R*x)
(Note: R corresponds to fastest running array dimension in Q%x%R ...)
Let nq=nrow(Q) and nr=nrow(R),
using rules of the kronecker product we have that
* Quadratic form = .5*x'*S*x = .5*x'*(Q%x%I)*(I%x%R)*x
* Normalizing constant =
|S/(2*pi)|^.5 =
|(Q/sqrt(2*pi))%x%(R/sqrt(2*pi))|^.5 =
|(Q/sqrt(2*pi))|^(nr*.5) |(R/sqrt(2*pi))|^(nq*.5) =
... something that can be expressed through the normalizing
constants f(0) and g(0) ...
f(0)^nr * g(0)^nq * sqrt(2*pi)^(nq*nr)
```

Example:

```
// Separable extension of two AR1 processes
Type phi1=0.8;
AR1_t<N01<Type> > f(phi1);
Type phi2=0.8;
AR1_t<N01<Type> > g(phi2);
SEPARABLE_t<AR1_t<N01<Type> > , AR1_t<N01<Type> > > h(
      f,g);
// Can be evaluated on an array:
array<Type> x(10,20);
Type ans=h(x);
```

Definition at line 829 of file tmbutils.cpp.

8.38.2 Constructor & Destructor Documentation

8.38.2.1 template<class distribution1 , class distribution2 > density::SEPARABLE_t< distribution1, distribution2 >::SEPARABLE_t ( ) [inline]

Definition at line 835 of file tmbutils.cpp.

8.38.2.2 template<class distribution1 , class distribution2 > density::SEPARABLE_t< distribution1, distribution2 >::SEPARABLE_t (distribution1 *f_*, distribution2 *g_*) [inline]

Definition at line 836 of file tmbutils.cpp.

8.38.3 Member Function Documentation

8.38.3.1 template<class distribution1 , class distribution2 > arraytype density::SEPARABLE_t< distribution1, distribution2 >::jacobian (arraytype *x*) [inline]

Definition at line 844 of file tmbutils.cpp.

8.38.3.2 template<class distribution1 , class distribution2 > int density::SEPARABLE_t< distribution1, distribution2 >::ndim ( ) [inline]

Definition at line 881 of file tmbutils.cpp.

8.38.3.3 template<class distribution1 , class distribution2 > scalartype density::SEPARABLE_t< distribution1, distribution2 >::operator() (arraytype *x*) [inline]

Definition at line 862 of file tmbutils.cpp.

8.38.3.4 template<class distribution1 , class distribution2 > scalartype density::SEPARABLE_t< distribution1, distribution2 >::operator() (arraytype *x*, int *i*) [inline]

Definition at line 893 of file tmbutils.cpp.

8.38.3.5 template<class distribution1 , class distribution2 > density::SEPARABLE_t< distribution1, distribution2 >::TYPEDEFS (typename distribution1::scalartype) [private]

8.38.3.6 template<class distribution1 , class distribution2 > arraytype density::SEPARABLE_t< distribution1, distribution2 >::zeroVector (vector< int > *d*, int *n*) [inline]

Definition at line 853 of file tmbutils.cpp.

8.38.4 Member Data Documentation

8.38.4.1 template<class distribution1 , class distribution2 > distribution1 density::SEPARABLE_t< distribution1, distribution2 >::f [private]

Definition at line 832 of file tmbutils.cpp.

8.38.4.2 template<class distribution1 , class distribution2 > distribution2 density::SEPARABLE_t< distribution1, distribution2 >::g [private]

Definition at line 833 of file tmbutils.cpp.

8.38.4.3 template<class distribution1 , class distribution2 > density::SEPARABLE_t< distribution1, distribution2 >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 882 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.39 SEXP_t Struct Reference

TMB: SEXP type.

`#include <tmb_core.hpp>`

Public Member Functions

- SEXP_t (SEXP x)
  SEXP_t: assignment.
- SEXP_t ()
  SEXP_t: default constructor.
- operator SEXP ()
  SEXP_t:

Public Attributes

- SEXP value
  SEXP_t: data entry.

8.39.1 Detailed Description

TMB: SEXP type.

Definition at line 12 of file tmb_core.hpp.

8.39.2 Constructor & Destructor Documentation

8.39.2.1 SEXP_t::SEXP_t (SEXP *x*) [inline]

SEXP_t: assignment.

Definition at line 14 of file tmb_core.hpp.

8.39.2.2 SEXP_t::SEXP_t ( ) [inline]

SEXP_t: default constructor.

Definition at line 15 of file tmb_core.hpp.

8.39.3 Member Function Documentation

8.39.3.1 SEXP_t::operator SEXP ( ) [inline]

SEXP_t:

Definition at line 16 of file tmb_core.hpp.

8.39.4 Member Data Documentation

8.39.4.1 SEXP SEXP_t::value

SEXP_t: data entry.

Definition at line 13 of file tmb_core.hpp.

Referenced by operator SEXP(), operator<(), and SEXP_t().

The documentation for this struct was generated from the following file:

- tmb_core.hpp

8.40 sphess_t< ADFunType > Struct Template Reference

`#include <start_parallel.hpp>`

Public Member Functions

- sphess_t (ADFunType *pf_, vector< int > i_, vector< int > j_)

Public Attributes

- ADFunType * pf
- vector< int > i
- vector< int > j

8.40.1 Detailed Description

template<class ADFunType>struct sphess_t< ADFunType >

Definition at line 32 of file start_parallel.hpp.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 template<class ADFunType> sphess_t< ADFunType >::sphess_t (ADFunType * *pf_*, vector< int > *i_*, vector< int > *j_*) [inline]

Definition at line 33 of file start_parallel.hpp.

8.40.3 Member Data Documentation

8.40.3.1 template<class ADFunType> vector<int> sphess_t< ADFunType >::i

Definition at line 35 of file start_parallel.hpp.

Referenced by asSEXP(), and sphess_t< ADFunType >::sphess_t().

8.40.3.2 template<class ADFunType> vector<int> sphess_t< ADFunType >::j

Definition at line 36 of file start_parallel.hpp.

Referenced by asSEXP(), and sphess_t< ADFunType >::sphess_t().

8.40.3.3 template<class ADFunType> ADFunType* sphess_t< ADFunType >::pf

Definition at line 34 of file start_parallel.hpp.

Referenced by asSEXP(), and sphess_t< ADFunType >::sphess_t().

The documentation for this struct was generated from the following file:

- start_parallel.hpp

8.41 tmbutils::splinefun< Type > Class Template Reference

Public Member Functions

- splinefun ()
- splinefun (const splinefun &fun)
- splinefun (const vector< Type > &x_, const vector< Type > &y_, int method_=3)
- void erase_data ()
- ~splinefun ()
- void construct (const vector< Type > &x_, const vector< Type > &y_, int method_=3)
- Type operator() (const Type &x_)
- void natural_spline (int n, Type *x, Type *y, Type *b, Type *c, Type *d)
- void fmm_spline (int n, Type *x, Type *y, Type *b, Type *c, Type *d)
- void periodic_spline (int n, Type *x, Type *y, Type *b, Type *c, Type *d, Type *e)
- void spline_coef (int *method, int *n, Type *x, Type *y, Type *b, Type *c, Type *d, Type *e)
- void spline_eval (int *method, int *nu, Type *u, Type *v, int *n, Type *x, Type *y, Type *b, Type *c, Type *d)

Private Attributes

- int method [1]
- int n [1]
- Type * x
- Type * y
- Type * b
- Type * c
- Type * d
- Type * e

8.41.1 Detailed Description

template<class Type>class tmbutils::splinefun< Type >

Definition at line 54 of file tmbutils.cpp.

8.41.2 Constructor & Destructor Documentation

8.41.2.1 template<class Type > tmbutils::splinefun< Type >::splinefun ( ) [inline]

Definition at line 72 of file tmbutils.cpp.

8.41.2.2 template<class Type > tmbutils::splinefun< Type >::splinefun (const splinefun< Type > & *fun*) [inline]

Definition at line 77 of file tmbutils.cpp.

8.41.2.3 template<class Type > tmbutils::splinefun< Type >::splinefun (const vector< Type > & *x_*, const vector< Type > & *y_*, int *method_* = 3) [inline]

Definition at line 96 of file tmbutils.cpp.

8.41.2.4 template<class Type > tmbutils::splinefun< Type >::~splinefun ( ) [inline]

Definition at line 109 of file tmbutils.cpp.

8.41.3 Member Function Documentation

8.41.3.1 template<class Type > void tmbutils::splinefun< Type >::construct (const vector< Type > & *x_*, const vector< Type > & *y_*, int *method_* = 3) [inline]

Definition at line 113 of file tmbutils.cpp.

8.41.3.2 template<class Type > void tmbutils::splinefun< Type >::erase_data ( ) [inline]

Definition at line 101 of file tmbutils.cpp.

8.41.3.3 template<class Type > void tmbutils::splinefun< Type >::fmm_spline (int *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*) [inline]

Definition at line 239 of file tmbutils.cpp.

8.41.3.4 template<class Type > void tmbutils::splinefun< Type >::natural_spline (int *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*) [inline]

Definition at line 161 of file tmbutils.cpp.

8.41.3.5 template<class Type > Type tmbutils::splinefun< Type >::operator() (const Type & *x_*) [inline]

Definition at line 133 of file tmbutils.cpp.

8.41.3.6 template<class Type > void tmbutils::splinefun< Type >::periodic_spline (int *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*, Type * *e*) [inline]

Definition at line 328 of file tmbutils.cpp.

8.41.3.7 template<class Type > void tmbutils::splinefun< Type >::spline_coef (int * *method*, int * *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*, Type * *e*) [inline]

Definition at line 443 of file tmbutils.cpp.

8.41.3.8 template<class Type > void tmbutils::splinefun< Type >::spline_eval (int * *method*, int * *nu*, Type * *u*, Type * *v*, int * *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*) [inline]

Definition at line 458 of file tmbutils.cpp.

8.41.4 Member Data Documentation

8.41.4.1 template<class Type > Type* tmbutils::splinefun< Type >::b [private]

Definition at line 62 of file tmbutils.cpp.

8.41.4.2 template<class Type > Type* tmbutils::splinefun< Type >::c [private]

Definition at line 63 of file tmbutils.cpp.

8.41.4.3 template<class Type > Type* tmbutils::splinefun< Type >::d [private]

Definition at line 64 of file tmbutils.cpp.

8.41.4.4 template<class Type > Type* tmbutils::splinefun< Type >::e [private]

Definition at line 65 of file tmbutils.cpp.

8.41.4.5 template<class Type > int tmbutils::splinefun< Type >::method[1] [private]

Definition at line 58 of file tmbutils.cpp.

8.41.4.6 template<class Type > int tmbutils::splinefun< Type >::n[1] [private]

Definition at line 59 of file tmbutils.cpp.

8.41.4.7 template<class Type > Type* tmbutils::splinefun< Type >::x [private]

Definition at line 60 of file tmbutils.cpp.

8.41.4.8 template<class Type > Type* tmbutils::splinefun< Type >::y [private]

Definition at line 61 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.42 splinefun< Type > Class Template Reference

Public Member Functions

- splinefun ()
- splinefun (const splinefun &fun)
- splinefun (const vector< Type > &x_, const vector< Type > &y_, int method_=3)
- void erase_data ()
- ~splinefun ()
- void construct (const vector< Type > &x_, const vector< Type > &y_, int method_=3)
- Type operator() (const Type &x_)
- void natural_spline (int n, Type *x, Type *y, Type *b, Type *c, Type *d)
- void fmm_spline (int n, Type *x, Type *y, Type *b, Type *c, Type *d)
- void periodic_spline (int n, Type *x, Type *y, Type *b, Type *c, Type *d, Type *e)
- void spline_coef (int *method, int *n, Type *x, Type *y, Type *b, Type *c, Type *d, Type *e)
- void spline_eval (int *method, int *nu, Type *u, Type *v, int *n, Type *x, Type *y, Type *b, Type *c, Type *d)

Private Attributes

- int method [1]
- int n [1]
- Type * x
- Type * y
- Type * b
- Type * c
- Type * d
- Type * e

8.42.1 Detailed Description

template<class Type>class splinefun< Type >

Definition at line 53 of file splines.cpp.

8.42.2 Constructor & Destructor Documentation

8.42.2.1 template<class Type > splinefun< Type >::splinefun ( ) [inline]

Definition at line 71 of file splines.cpp.

8.42.2.2 template<class Type > splinefun< Type >::splinefun (const splinefun< Type > & *fun*) [inline]

Definition at line 76 of file splines.cpp.

8.42.2.3 template<class Type > splinefun< Type >::splinefun (const vector< Type > & *x_*, const vector< Type > & *y_*, int *method_* = 3) [inline]

Definition at line 95 of file splines.cpp.

8.42.2.4 template<class Type > splinefun< Type >::~splinefun ( ) [inline]

Definition at line 108 of file splines.cpp.

8.42.3 Member Function Documentation

8.42.3.1 template<class Type > void splinefun< Type >::construct (const vector< Type > & *x_*, const vector< Type > & *y_*, int *method_* = 3) [inline]

Definition at line 112 of file splines.cpp.

Referenced by splinefun< Type >::splinefun().

8.42.3.2 template<class Type > void splinefun< Type >::erase_data ( ) [inline]

Definition at line 100 of file splines.cpp.

Referenced by splinefun< Type >::construct(), and splinefun< Type >::~splinefun().

8.42.3.3 template<class Type > void splinefun< Type >::fmm_spline (int *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*) [inline]

Definition at line 238 of file splines.cpp.

Referenced by splinefun< Type >::spline_coef().

8.42.3.4 template<class Type > void splinefun< Type >::natural_spline (int *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*) [inline]

Definition at line 160 of file splines.cpp.

Referenced by splinefun< Type >::spline_coef().

8.42.3.5 template<class Type > Type splinefun< Type >::operator() (const Type & *x_*) [inline]

Definition at line 132 of file splines.cpp.

8.42.3.6 template<class Type > void splinefun< Type >::periodic_spline (int *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*, Type * *e*) [inline]

Definition at line 327 of file splines.cpp.

Referenced by splinefun< Type >::spline_coef().

8.42.3.7 template<class Type > void splinefun< Type >::spline_coef (int * *method*, int * *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*, Type * *e*) [inline]

Definition at line 442 of file splines.cpp.

Referenced by splinefun< Type >::construct().

8.42.3.8 template<class Type > void splinefun< Type >::spline_eval (int * *method*, int * *nu*, Type * *u*, Type * *v*, int * *n*, Type * *x*, Type * *y*, Type * *b*, Type * *c*, Type * *d*) [inline]

Definition at line 457 of file splines.cpp.

Referenced by splinefun< Type >::operator()().

8.42.4 Member Data Documentation

8.42.4.1 template<class Type > Type* splinefun< Type >::b [private]

Definition at line 61 of file splines.cpp.

Referenced by splinefun< Type >::construct(), splinefun< Type >::operator()(), and splinefun< Type >::splinefun().

8.42.4.2 template<class Type > Type* splinefun< Type >::c [private]

Definition at line 62 of file splines.cpp.

Referenced by splinefun< Type >::construct(), splinefun< Type >::operator()(), and splinefun< Type >::splinefun().

8.42.4.3 template<class Type > Type* splinefun< Type >::d [private]

Definition at line 63 of file splines.cpp.

Referenced by splinefun< Type >::construct(), splinefun< Type >::operator()(), and splinefun< Type >::splinefun().

8.42.4.4 template<class Type > Type* splinefun< Type >::e [private]

Definition at line 64 of file splines.cpp.

Referenced by splinefun< Type >::construct(), and splinefun< Type >::splinefun().

8.42.4.5 template<class Type > int splinefun< Type >::method[1] [private]

Definition at line 57 of file splines.cpp.

Referenced by splinefun< Type >::construct(), splinefun< Type >::operator()(), and splinefun< Type >::splinefun().

8.42.4.6 template<class Type > int splinefun< Type >::n[1] [private]

Definition at line 58 of file splines.cpp.

Referenced by splinefun< Type >::construct(), splinefun< Type >::fmm_spline(), splinefun< Type >::natural_spline(), splinefun< Type >::operator()(), splinefun< Type >::periodic_spline(), splinefun< Type >::spline_eval(), and splinefun< Type >::splinefun().

8.42.4.7 template<class Type > Type* splinefun< Type >::x [private]

Definition at line 59 of file splines.cpp.

Referenced by splinefun< Type >::construct(), splinefun< Type >::operator()(), and splinefun< Type >::splinefun().

8.42.4.8 template<class Type > Type* splinefun< Type >::y [private]

Definition at line 60 of file splines.cpp.

Referenced by splinefun< Type >::construct(), splinefun< Type >::operator()(), and splinefun< Type >::splinefun().

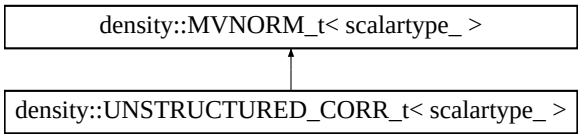
The documentation for this class was generated from the following file:

- splines.cpp

8.43 density::UNSTRUCTURED_CORR_t< scalartype_ > Class Template Reference

Multivariate normal distribution with unstructered correlation matrix.

Inheritance diagram for density::UNSTRUCTURED_CORR_t< scalartype_ >:



Private Member Functions

- TYPEDEFS (scalartype_)
- UNSTRUCTURED_CORR_t ()
- UNSTRUCTURED_CORR_t (vectortype x)

Additional Inherited Members

8.43.1 Detailed Description

template<class scalartype_>class density::UNSTRUCTURED_CORR_t< scalartype_ >

Multivariate normal distribution with unstructered correlation matrix.

Class to evaluate the negative log density of a multivariate Gaussian variable with unstructured symmetric positive definite correlation matrix.

The unstructured correlation matrix is parameterized via a lower triangular matrix with unit diagonal i.e. (n∗n-n)/2 parameters to describe an n dimensional correlation matrix.

For instance in the case n=4 the correlation matrix is given by

$$\Sigma = D^{-\frac{1}{2}} L L' D^{-\frac{1}{2}}$$

where

$$L = \begin{pmatrix} 1 & & & \\ x_0 & 1 & & \\ x_1 & x_3 & 1 & \\ x_2 & x_4 & x_5 & 1 \end{pmatrix}$$

and

$$D = diag(LL')$$

Example:

```
// Construct density object of dimension 4
vector<Type> Lx(6);
UNSTRUCTURED_CORR_t<Type> neg_log_density(Lx);
// Evaluate density
vector<Type> x(4);
Type ans=neg_log_density(x);
```

Remarks

The correlation matrix is available through member "Sigma".

Definition at line 150 of file tmbutils.cpp.

8.43.2 Constructor & Destructor Documentation

8.43.2.1 template<class scalartype_ > density::UNSTRUCTURED_CORR_t< scalartype_ >::UNSTRUCTURED_CORR_t ( ) [inline], [private]

Definition at line 152 of file tmbutils.cpp.

8.43.2.2 template<class scalartype_ > density::UNSTRUCTURED_CORR_t< scalartype_ >::UNSTRUCTURED_CORR_t (vectortype *x*) [inline], [private]

Definition at line 153 of file tmbutils.cpp.

8.43.3 Member Function Documentation

8.43.3.1 template<class scalartype_ > density::UNSTRUCTURED_CORR_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

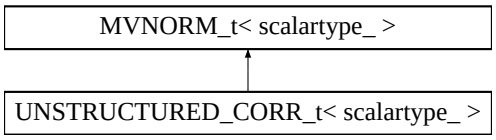
The documentation for this class was generated from the following file:

- tmbutils.cpp

8.44 UNSTRUCTURED_CORR_t< scalartype_ > Class Template Reference

Multivariate normal distribution with unstructered correlation matrix.

Inheritance diagram for UNSTRUCTURED_CORR_t< scalartype_ >:



Private Member Functions

- TYPEDEFS (scalartype_)
- UNSTRUCTURED_CORR_t ()
- UNSTRUCTURED_CORR_t (vectortype x)

Additional Inherited Members

8.44.1 Detailed Description

template<class scalartype_>class UNSTRUCTURED_CORR_t< scalartype_ >

Multivariate normal distribution with unstructered correlation matrix.

Class to evaluate the negative log density of a multivariate Gaussian variable with unstructured symmetric positive definite correlation matrix.

The unstructured correlation matrix is parameterized via a lower triangular matrix with unit diagonal i.e. (n∗n-n)/2 parameters to describe an n dimensional correlation matrix.

For instance in the case n=4 the correlation matrix is given by

$$\Sigma = D^{-\frac{1}{2}} L L' D^{-\frac{1}{2}}$$

where

$$L = \begin{pmatrix} 1 & & & \\ x_0 & 1 & & \\ x_1 & x_3 & 1 & \\ x_2 & x_4 & x_5 & 1 \end{pmatrix}$$

and

$$D = diag(LL')$$

Example:

```
// Construct density object of dimension 4
vector<Type> Lx(6);
UNSTRUCTURED_CORR_t<Type> neg_log_density(Lx);
// Evaluate density
vector<Type> x(4);
Type ans=neg_log_density(x);
```

Remarks

The correlation matrix is available through member "Sigma".

Definition at line 149 of file density.cpp.

8.44.2 Constructor & Destructor Documentation

8.44.2.1 template<class scalartype_> UNSTRUCTURED_CORR_t< scalartype_ >::UNSTRUCTURED_CORR_t ( ) [inline], [private]

Definition at line 151 of file density.cpp.

8.44.2.2 template<class scalartype_> UNSTRUCTURED_CORR_t< scalartype_ >::UNSTRUCTURED_CORR_t (vectortype *x*) [inline], [private]

Definition at line 152 of file density.cpp.

8.44.3 Member Function Documentation

8.44.3.1 template<class scalartype_> UNSTRUCTURED_CORR_t< scalartype_ >::TYPEDEFS (scalartype_) [private]

The documentation for this class was generated from the following file:

- density.cpp

8.45 density::VECSCALE_t< distribution > Class Template Reference

Apply a vector scale transformation on a density.

Public Member Functions

- VECSCALE_t ()
- VECSCALE_t (distribution f_, vectortype scale_)
- scalartype operator() (arraytype x)

  Evaluate the negative log density.
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (typename distribution::scalartype)

Private Attributes

- distribution f
- vectortype scale

8.45.1 Detailed Description

template<class distribution>class density::VECSCALE_t< distribution >

Apply a vector scale transformation on a density.

Assume x has density f. Construct the density of y=scale∗x where scale is a vector.

Parameters

*f_*	distribution
*scale_*	vector

Definition at line 753 of file tmbutils.cpp.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 template<class distribution > density::VECSCALE_t< distribution >::VECSCALE_t ( ) [inline]

Definition at line 759 of file tmbutils.cpp.

8.45.2.2 template<class distribution > density::VECSCALE_t< distribution >::VECSCALE_t (distribution *f_*, vectortype *scale_*) [inline]

Definition at line 760 of file tmbutils.cpp.

8.45.3 Member Function Documentation

8.45.3.1 template<class distribution > arraytype density::VECSCALE_t< distribution >::jacobian (arraytype *x*) [inline]

Definition at line 768 of file tmbutils.cpp.

8.45.3.2 template<class distribution > int density::VECSCALE_t< distribution >::ndim ( ) [inline]

Definition at line 776 of file tmbutils.cpp.

8.45.3.3 template<class distribution > scalartype density::VECSCALE_t< distribution >::operator() (arraytype *x*) [inline]

Evaluate the negative log density.

Definition at line 762 of file tmbutils.cpp.

8.45.3.4 template<class distribution > density::VECSCALE_t< distribution >::TYPEDEFS (typename distribution::scalartype) [private]

8.45.4 Member Data Documentation

8.45.4.1 template<class distribution > distribution density::VECSCALE_t< distribution >::f [private]

Definition at line 756 of file tmbutils.cpp.

8.45.4.2 template<class distribution > vectortype density::VECSCALE_t< distribution >::scale [private]

Definition at line 757 of file tmbutils.cpp.

8.45.4.3 template<class distribution > density::VECSCALE_t< distribution >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 777 of file tmbutils.cpp.

The documentation for this class was generated from the following file:

- tmbutils.cpp

8.46 VECSCALE_t< distribution > Class Template Reference

Apply a vector scale transformation on a density.

Public Member Functions

- VECSCALE_t ()
- VECSCALE_t (distribution f_, vectortype scale_)
- scalartype operator() (arraytype x)

  Evaluate the negative log density.
- arraytype jacobian (arraytype x)
- int ndim ()

Public Attributes

- VARIANCE_NOT_YET_IMPLEMENTED

Private Member Functions

- TYPEDEFS (typename distribution::scalartype)

Private Attributes

- distribution f
- vectortype scale

8.46.1 Detailed Description

template<class distribution>class VECSCALE_t< distribution >

Apply a vector scale transformation on a density.

Assume x has density f. Construct the density of y=scale*x where scale is a vector.

Parameters

*f_*	distribution
*scale_*	vector

Definition at line 752 of file density.cpp.

8.46.2 Constructor & Destructor Documentation

8.46.2.1 template<class distribution> VECSCALE_t< distribution >::VECSCALE_t ( ) [inline]

Definition at line 758 of file density.cpp.

8.46.2.2 template<class distribution> VECSCALE_t< distribution >::VECSCALE_t (distribution *f_*, vectortype *scale_*) [inline]

Definition at line 759 of file density.cpp.

8.46.3 Member Function Documentation

8.46.3.1 template<class distribution> arraytype VECSCALE_t< distribution >::jacobian (arraytype *x*) [inline]

Definition at line 767 of file density.cpp.

8.46.3.2 template<class distribution> int VECSCALE_t< distribution >::ndim ( ) [inline]

Definition at line 775 of file density.cpp.

8.46.3.3 template<class distribution> scalartype VECSCALE_t< distribution >::operator() (arraytype *x*) [inline]

Evaluate the negative log density.

Definition at line 761 of file density.cpp.

8.46.3.4 template<class distribution> VECSCALE_t< distribution >::TYPEDEFS (typename distribution::scalartype) [private]

8.46.4 Member Data Documentation

8.46.4.1 template<class distribution> distribution VECSCALE_t< distribution >::f [private]

Definition at line 755 of file density.cpp.

Referenced by VECSCALE_t< distribution >::jacobian(), VECSCALE_t< distribution >::ndim(), VECSCALE_t< distribution >::operator()(), and VECSCALE_t< distribution >::VECSCALE_t().

8.46.4.2 template<class distribution> vectortype VECSCALE_t< distribution >::scale [private]

Definition at line 756 of file density.cpp.

Referenced by VECSCALE_t< distribution >::jacobian(), VECSCALE_t< distribution >::operator()(), and VECSCALE_t< distribution >::VECSCALE_t().

8.46.4.3 template<class distribution> VECSCALE_t< distribution >::VARIANCE_NOT_YET_IMPLEMENTED

Definition at line 776 of file density.cpp.

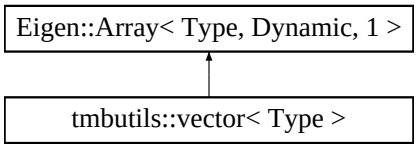
The documentation for this class was generated from the following file:

- density.cpp

8.47 tmbutils::vector< Type > Struct Template Reference

Vector class used by TMB.

Inheritance diagram for tmbutils::vector< Type >:



Public Types

- typedef Type value_type
- typedef Array< Type, Dynamic, 1 > Base

Public Member Functions

- vector (void)
- template<class T1 >
  vector (T1 x)
- template<class T1 , class T2 >
  vector (T1 x, T2 y)
- template<class T1 >
  vector & operator= (const T1 &other)
- vector< Type > operator() (vector< int > ind)
- template<template< class > class Vector, class T >
  operator Vector< T > ()
- template<template< class > class Vector, class T >
  vector (Vector< T > x)

8.47.1 Detailed Description

template<class Type>struct tmbutils::vector< Type >

Vector class used by TMB.

The TMB vector class is implemented as an Eigen Array of dynamic length. In particular, vectorized operations are inherited from the Eigen library.

Examples:

orange_big.cpp.

Definition at line 15 of file tmbutils.cpp.

8.47.2 Member Typedef Documentation

8.47.2.1 template<class Type> typedef Array<Type,Dynamic,1> tmbutils::vector< Type >::Base

Definition at line 18 of file tmbutils.cpp.

8.47.2.2 template<class Type> typedef Type tmbutils::vector< Type >::value_type

Definition at line 17 of file tmbutils.cpp.

8.47.3 Constructor & Destructor Documentation

8.47.3.1 template<class Type> tmbutils::vector< Type >::vector (void) [inline]

Definition at line 19 of file tmbutils.cpp.

8.47.3.2 template<class Type> template<class T1 > tmbutils::vector< Type >::vector (T1 *x*) [inline]

Definition at line 22 of file tmbutils.cpp.

8.47.3.3 template<class Type> template<class T1 , class T2 > tmbutils::vector< Type >::vector (T1 *x*, T2 *y*) [inline]

Definition at line 25 of file tmbutils.cpp.

8.47.3.4 template<class Type> template<template< class > class Vector, class T > tmbutils::vector< Type >::vector (Vector< T > *x*) [inline]

Definition at line 69 of file tmbutils.cpp.

8.47.4 Member Function Documentation

8.47.4.1 template<class Type> template<template< class > class Vector, class T > tmbutils::vector< Type >::operator Vector< T > ( ) [inline]

Definition at line 56 of file tmbutils.cpp.

8.47.4.2 template<class Type> vector<Type> tmbutils::vector< Type >::operator() (vector< int > *ind*) [inline]

Definition at line 43 of file tmbutils.cpp.

8.47.4.3 template<class Type> template<class T1 > vector& tmbutils::vector< Type >::operator= (const T1 & *other*) [inline]

Definition at line 29 of file tmbutils.cpp.

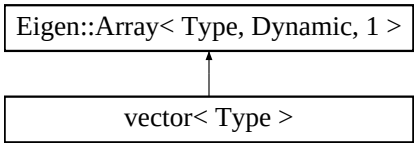
The documentation for this struct was generated from the following file:

- tmbutils.cpp

8.48 vector< Type > Struct Template Reference

Vector class used by TMB.

Inheritance diagram for vector< Type >:



Public Types

- typedef Type value_type
- typedef Array< Type, Dynamic, 1 > Base

Public Member Functions

- vector (void)
- template<class T1 >
  vector (T1 x)
- template<class T1 , class T2 >
  vector (T1 x, T2 y)
- template<class T1 >
  vector & operator= (const T1 &other)
- vector< Type > operator() (vector< int > ind)
- template<template< class > class Vector, class T >
  operator Vector< T > ()
- template<template< class > class Vector, class T >
  vector (Vector< T > x)

8.48.1 Detailed Description

template<class Type>struct vector< Type >

Vector class used by TMB.

The TMB vector class is implemented as an Eigen Array of dynamic length. In particular, vectorized operations are inherited from the Eigen library.

Examples:

atomic.cpp, matrix_arrays.cpp, nmix.cpp, rw.cpp, sdv_multi.cpp, simple.cpp, socatt.cpp, and spatial.cpp.

Definition at line 14 of file vector.cpp.

8.48.2 Member Typedef Documentation

8.48.2.1 template<class Type> typedef Array<Type,Dynamic,1> vector< Type >::Base

Definition at line 17 of file vector.cpp.

8.48.2.2 template<class Type> typedef Type vector< Type >::value_type

Definition at line 16 of file vector.cpp.

8.48.3 Constructor & Destructor Documentation

8.48.3.1 template<class Type> vector< Type >::vector (void) [inline]

Definition at line 18 of file vector.cpp.

8.48.3.2 template<class Type> template<class T1 > vector< Type >::vector (T1 *x*) [inline]

Definition at line 21 of file vector.cpp.

8.48.3.3 template<class Type> template<class T1 , class T2 > vector< Type >::vector (T1 *x*, T2 *y*) [inline]

Definition at line 24 of file vector.cpp.

8.48.3.4 template<class Type> template<template< class > class Vector, class T > vector< Type >::vector (Vector< T > *x*) [inline]

Definition at line 68 of file vector.cpp.

8.48.4 Member Function Documentation

8.48.4.1 template<class Type> template<template< class > class Vector, class T > vector< Type >::operator Vector< T >( ) [inline]

Definition at line 55 of file vector.cpp.

8.48.4.2 template<class Type> vector<Type> vector< Type >::operator() (vector< int > *ind*) [inline]

Definition at line 42 of file vector.cpp.

8.48.4.3 template<class Type> template<class T1 > vector& vector< Type >::operator= (const T1 & *other*) [inline]

Definition at line 28 of file vector.cpp.

The documentation for this struct was generated from the following file:

- vector.cpp

Chapter 9

File Documentation

9.1 array.cpp File Reference

Classes

- struct array< Type >

  Array class used by TMB.

Macros

- #define INHERIT(OP)
- #define INHERIT(OP)

9.1.1 Macro Definition Documentation

9.1.1.1 #define INHERIT(*OP*)

Value:

```
template <class T>                                      \
  array<Type> OP(T y){return array(MapBase::OP(y),dim);}
```

Definition at line 216 of file tmbutils.cpp.

9.1.1.2 #define INHERIT(*OP*)

Value:

```
template <class T>                                      \
  array<Type> OP(T y){return array(MapBase::OP(y),dim);}
```

9.2 asMatrix.hpp File Reference

9.3 atomic_macro.hpp File Reference

```
#include "cppad/cppad.hpp"
```

Macros

- #define NTHREADS 1
- #define THREADNUM 0
- #define ATOMIC_TEMPLATE(function_name, Base, ns_name, code)
- #define PRINT_INFO(ns_name)
- #define ATOMIC_TEMPLATE_LAZY(Base, ns_name, nsFor, nsForRev)
- #define ATOMIC_FRONTEND(function_name, Base, ns_name)
- #define AD1 CppAD::AD<double>
- #define AD2 CppAD::AD<CppAD::AD<double> >
- #define ATOMIC_TRIGGER(function_name, ns_name, ns2, ns3)
- #define FORREW(ns)
- #define ATOMIC_FUNCTION(function_name, code)
- #define REGISTER_ATOMIC(function_name)

Functions

- template<class T >
  void printVec (T x)

9.3.1 Macro Definition Documentation

9.3.1.1 #define AD1 CppAD::AD<double>

Definition at line 378 of file atomic_macro.hpp.

9.3.1.2 #define AD2 CppAD::AD<CppAD::AD<double> >

Definition at line 379 of file atomic_macro.hpp.

9.3.1.3 #define ATOMIC_FRONTEND(*function_name, Base, ns_name*)

Value:

```
CppAD::vector<CppAD::AD<Base > > function_name  \
  (CppAD::vector<CppAD::AD<Base > > x){                 \
  CppAD::vector<CppAD::AD<Base > > y(ns_name::m);       \
  ns_name::fun(0,x,y);                                  \
  return y;                                             \
}                                                       \
tmbutils::vector<CppAD::AD<Base > > function_name               \
  (tmbutils::vector<CppAD::AD<Base > > x){                      \
    return tmbutils::vector<CppAD::AD<Base > >(function_name(  \
    CppAD::vector<CppAD::AD<Base > >(x)));              \
}
```

Definition at line 362 of file atomic_macro.hpp.

9.3.1.4 #define ATOMIC_FUNCTION(*function_name, code*)

Value:

```
ATOMIC_TEMPLATE(function_name,double,atomic,code)
      \
ATOMIC_TEMPLATE(function_name,CppAD::AD<double>,atomic2,code)
      \
ATOMIC_TEMPLATE(function_name,CppAD::AD<CppAD::AD<double> >,atomic3,code)
      \
ATOMIC_TEMPLATE(myf,double,atomic4,FORREW(atomic2))
```

```
      \
ATOMIC_TEMPLATE(myf,AD1,atomic5,FORREW(atomic3))
      \
ATOMIC_TEMPLATE(myf,double,atomic6,FORREW(atomic5))
      \
ATOMIC_TEMPLATE_LAZY(AD1,lazy2,atomic,atomic4)
      \
ATOMIC_TEMPLATE_LAZY(AD1,lazy5,atomic4,atomic6)
      \
ATOMIC_TEMPLATE_LAZY(AD2,lazy3,lazy2,lazy5)
      \
ATOMIC_FRONTEND(function_name,double ,atomic)
      \
ATOMIC_FRONTEND(function_name,CppAD::AD<double>,lazy2)
      \
ATOMIC_FRONTEND(function_name,CppAD::AD<CppAD::AD<double> >,lazy3)
      \
ATOMIC_TRIGGER(function_name,atomic,atomic2,atomic3)
```

Definition at line 437 of file atomic_macro.hpp.

9.3.1.5 #define ATOMIC_TEMPLATE(*function_name, Base, ns_name, code*)

Definition at line 109 of file atomic_macro.hpp.

9.3.1.6 #define ATOMIC_TEMPLATE_LAZY(*Base, ns_name, nsFor, nsForRev*)

Definition at line 237 of file atomic_macro.hpp.

9.3.1.7 #define ATOMIC_TRIGGER(*function_name, ns_name, ns2, ns3*)

Value:

```
CppAD::vector<double> function_name(CppAD::vector<double> x){           \
  CppAD::vector<double> y=ns_name::function_name(x);                     \
  if(ns_name::pf==NULL){                                                 \
    int n=x.size();                                                      \
    int m=y.size();                                                      \
    ns_name::generate_tape(n,m);                                         \
    ns2::generate_tape(n,m);                                             \
    ns3::generate_tape(n,m);                                             \
    atomic4::generate_tape(m+n,n);                                       \
    atomic5::generate_tape(m+n,n);                                       \
    atomic6::generate_tape(m+n+n,m+n);                                   \
    lazy2::generate_tape(n,m);                                           \
    lazy3::generate_tape(n,m);                                           \
    lazy5::generate_tape(m+n,n);                                         \
  }                                                                      \
  return y;                                                              \
}                                                                        \
tmbutils::vector<double > function_name                                  \
(tmbutils::vector<double > x){                                          \
  return tmbutils::vector<double >(function_name(CppAD::vector<double >(x))); \
}
```

Definition at line 380 of file atomic_macro.hpp.

9.3.1.8 #define FORREW(*ns*)

Value:

```
template<class T>T myf(T x){                  \
  int n=ns::n; int m=ns::m;                   \
  T tmp1(n);                                  \
  T tmp2(m);                                  \
  for(size_t i=0;i<n;i++)tmp1[i]=x[i];        \
  for(size_t i=n;i<n+m;i++)tmp2[i-n]=x[i];    \
  ns::vpf[0]->Forward(0,tmp1);                \
  return ns::vpf[0]->Reverse(1,tmp2);         \
}
```

Definition at line 426 of file atomic_macro.hpp.

9.3.1.9 #define NTHREADS 1

Definition at line 105 of file atomic_macro.hpp.

9.3.1.10 #define PRINT_INFO(*ns_name*)

Definition at line 231 of file atomic_macro.hpp.

9.3.1.11 #define REGISTER_ATOMIC(*function_name*)

Value:

```
ATOMIC_FUNCTION(function_name,                                          \
template<class Type>                                                    \
CppAD::vector<Type> function_name(CppAD::vector<Type> x){               \
  return ::function_name(tmbutils::vector<Type>(x));                            \
})
```

Examples:

atomic.cpp.

Definition at line 463 of file atomic_macro.hpp.

9.3.1.12 #define THREADNUM 0

Definition at line 106 of file atomic_macro.hpp.

9.3.2 Function Documentation

9.3.2.1 template<class T > void printVec (T *x*)

Definition at line 19 of file atomic_macro.hpp.

9.4 config.hpp File Reference

Classes

- struct **config_struct**

Macros

- #define SET(name, value) set(#name,name,value);

Functions

- SEXP TMBconfig (SEXP envir, SEXP cmd)

9.4.1 Macro Definition Documentation

9.4.1.1 #define SET(*name, value*) set(#name,name,value);

Definition at line 38 of file config.hpp.

9.4.2 Function Documentation

9.4.2.1 SEXP TMBconfig (SEXP *envir*, SEXP *cmd*)

Definition at line 58 of file config.hpp.

9.5 convenience.hpp File Reference

Templates to get convenient R-like syntax.

Functions

- template<class Type >
  vector< vector< Type > > split (vector< Type > x, vector< int > fac)
  Similar to R's split function: split(x,fac) devides x into groups defined by fac .
- template<template< class > class Vector, class Type >
  Type sum (Vector< Type > x)
- template<class Type >
  vector< Type > operator* (matrix< Type > A, vector< Type > x)
- template<class Type >
  vector< Type > operator* (Eigen::SparseMatrix< Type > A, vector< Type > x)

9.5.1 Detailed Description

Templates to get convenient R-like syntax.

Definition in file convenience.hpp.

9.5.2 Function Documentation

9.5.2.1 template<class Type > vector<Type> operator* (matrix< Type > *A*, vector< Type > *x*)

Matrix * vector

Simplifies syntax in that .matrix() can be avoided. Recall: TMB type vector is of Eigen type Array.

Definition at line 36 of file convenience.hpp.

9.5.2.2 template<class Type > vector<Type> operator* (Eigen::SparseMatrix< Type > *A*, vector< Type > *x*)

SparseMatrix * vector

Definition at line 40 of file convenience.hpp.

9.5.2.3 template<class Type > vector<vector<Type> > split (vector< Type > *x*, vector< int > *fac*)

Similar to R's split function: split(x,fac) devides x into groups defined by fac .

Returns a "vector of vectors".

Examples:

nmix.cpp.

Definition at line 10 of file convenience.hpp.

9.5.2.4 template<template< class > class Vector, class Type > Type sum (Vector< Type > *x*)

Sum of vector, matrix or array

Examples:

linreg.cpp, matrix_arrays.cpp, nmix.cpp, rw.cpp, sdv_multi.cpp, and socatt.cpp.

Definition at line 29 of file convenience.hpp.

Referenced by ARk_t< scalartype_ >::ARk_t(), VECSCALE_t< distribution >::operator()(), and GMRF_t< scalartype_ >::setQ().

9.6 convert.hpp File Reference

Convert vector/matrix-Types to double SEXP types.

Functions

- double asDouble (int x)
- double asDouble (double x)
- double asDouble (AD< double > x)
- double asDouble (AD< AD< double > > x)
- double asDouble (AD< AD< AD< double > > > x)
- template<class Type >
  SEXP asSEXP (const matrix< Type > &a)

  Convert TMB matrix, vector, scalar or int to R style.
- template<class Type >
  SEXP asSEXP (const vector< Type > &a)
- template<class Type >
  SEXP asSEXP (const Type &a)
- SEXP asSEXP (const int &a)
- template<class Type >
  SEXP asSEXP (const AD< Type > &a)
- template<template< class > class Vector, class Type >
  SEXP asSEXP (const Vector< Type > &a)
- template<class Type >
  vector< Type > asVector (SEXP x)

  Construct c++-vector from SEXP object.
- template<class Type >
  matrix< Type > asMatrix (const vector< Type > &x, int nr, int nc)

  Vector <-> Matrix conversion (for row-major matrices)
- template<class Type >
  matrix< Type > asMatrix (SEXP x)

  Construct c++-matrix from SEXP object.
- template<class Type >
  vector< Type > asVector (matrix< Type > x)
- template<class Type >
  SEXP asSEXP (const tmbutils::array< Type > &a)

9.6.1 Detailed Description

Convert vector/matrix-Types to double SEXP types.

Definition in file convert.hpp.

9.6.2 Function Documentation

9.6.2.1 double asDouble (int *x*)

Definition at line 5 of file convert.hpp.

Referenced by asSEXP(), and printVec().

9.6.2.2 double asDouble (double *x*)

Definition at line 6 of file convert.hpp.

9.6.2.3 double asDouble (AD< double > *x*)

Definition at line 7 of file convert.hpp.

9.6.2.4 double asDouble (AD< AD< double > > *x*)

Definition at line 8 of file convert.hpp.

9.6.2.5 double asDouble (AD< AD< AD< double > > > *x*)

Definition at line 9 of file convert.hpp.

9.6.2.6 template<class Type > matrix<Type> asMatrix (const vector< Type > & *x,* int *nr,* int *nc*)

Vector <-> Matrix conversion (for row-major matrices)

Definition at line 90 of file convert.hpp.

Referenced by EvalADFunObjectTemplate(), and HessianSparsityPattern().

9.6.2.7 template<class Type > matrix<Type> asMatrix (SEXP *x*)

Construct c++-matrix from SEXP object.

Definition at line 103 of file convert.hpp.

9.6.2.8 template<class Type > SEXP asSEXP (const matrix< Type > & *a*)

Convert TMB matrix, vector, scalar or int to R style.

Definition at line 13 of file convert.hpp.

Referenced by asSEXP().

9.6.2.9 template<class Type > SEXP asSEXP (const vector< Type > & *a*)

Definition at line 33 of file convert.hpp.

9.6.2.10 template<class Type > SEXP asSEXP (const Type & *a*)

Definition at line 44 of file convert.hpp.

9.6.2.11 SEXP asSEXP (const int & *a*)

Definition at line 52 of file convert.hpp.

9.6.2.12 template<class Type > SEXP asSEXP (const AD< Type > & *a*)

Definition at line 62 of file convert.hpp.

9.6.2.13 template<template< class > class Vector, class Type > SEXP asSEXP (const Vector< Type > & *a*)

Definition at line 66 of file convert.hpp.

9.6.2.14 template<class Type > SEXP asSEXP (const tmbutils::array< Type > & *a*)

Definition at line 129 of file convert.hpp.

9.6.2.15 template<class Type > vector<Type> asVector (SEXP *x*)

Construct c++-vector from SEXP object.

Definition at line 79 of file convert.hpp.

9.6.2.16 template<class Type > vector<Type> asVector (matrix< Type > *x*)

Definition at line 117 of file convert.hpp.

9.7 density.cpp File Reference

Classes to construct multivariate Gaussian density objects.

Classes

- class MVNORM_t< scalartype_ >
  Multivariate normal distribution with user supplied covariance matrix.
- class UNSTRUCTURED_CORR_t< scalartype_ >
  Multivariate normal distribution with unstructered correlation matrix.
- class N01< scalartype_ >
  Standardized normal distribution.
- class AR1_t< distribution >
  Stationary AR1 process.
- class ARk_t< scalartype_ >
  Stationary AR(k) process.
- class contAR2_t< scalartype_ >
  Continuous AR(2) process.
- class GMRF_t< scalartype_ >
  Gaussian Markov Random Field.
- class SCALE_t< distribution >
  Apply scale transformation on a density.
- class VECSCALE_t< distribution >
  Apply a vector scale transformation on a density.

- class SEPARABLE_t< distribution1, distribution2 >

  Separable extension of two densitites.
- class PROJ_t< distribution >

  Projection of multivariate gaussian variable.

Macros

- #define TYPEDEFS(scalartype_)
- #define VARIANCE_NOT_YET_IMPLEMENTED vectortype variance(){};
- #define TYPEDEFS(scalartype_)
- #define VARIANCE_NOT_YET_IMPLEMENTED vectortype variance(){};

Functions

- template<class scalartype >
  MVNORM_t< scalartype > MVNORM (matrix< scalartype > x)
- template<class scalartype >
  UNSTRUCTURED_CORR_t< scalartype > UNSTRUCTURED_CORR (vector< scalartype > x)
- template<class scalartype , class distribution >
  AR1_t< distribution > AR1 (scalartype phi_, distribution f_)
- template<class scalartype >
  AR1_t< N01< scalartype > > AR1 (scalartype phi_)
- template<class scalartype , class vectortype >
  contAR2_t< scalartype > contAR2 (vectortype grid_, scalartype shape_, scalartype scale_=1)
- template<class scalartype >
  contAR2_t< scalartype > contAR2 (scalartype shape_, scalartype scale_=1)
- template<class scalartype >
  GMRF_t< scalartype > GMRF (Eigen::SparseMatrix< scalartype > Q, int order=1)
- template<class scalartype , class arraytype >
  GMRF_t< scalartype > GMRF (arraytype x, vector< scalartype > delta, int order=1)
- template<class scalartype , class arraytype >
  GMRF_t< scalartype > GMRF (arraytype x, scalartype delta, int order=1)
- template<class scalartype , class distribution >
  SCALE_t< distribution > SCALE (distribution f_, scalartype scale_)
- template<class vectortype , class distribution >
  VECSCALE_t< distribution > VECSCALE (distribution f_, vectortype scale_)
- template<class distribution1 , class distribution2 >
  SEPARABLE_t< distribution1,
  distribution2 > SEPARABLE (distribution1 f_, distribution2 g_)
- template<class distribution >
  PROJ_t< distribution > PROJ (distribution f_, vector< int > i)

9.7.1 Detailed Description

Classes to construct multivariate Gaussian density objects.

Definition in file density.cpp.

9.7.2 Macro Definition Documentation

9.7.2.1 #define TYPEDEFS(*scalartype_*)

Value:

```
public:                                         \
typedef scalartype_ scalartype;                 \
typedef vector<scalartype> vectortype;                  \
typedef matrix<scalartype> matrixtype;                  \
typedef array<scalartype> arraytype;
```

Definition at line 7 of file tmbutils.cpp.

9.7.2.2 #define TYPEDEFS(*scalartype_*)

Value:

```
public:                                         \
typedef scalartype_ scalartype;                 \
typedef vector<scalartype> vectortype;                  \
typedef matrix<scalartype> matrixtype;                  \
typedef array<scalartype> arraytype;
```

9.7.2.3 #define VARIANCE_NOT_YET_IMPLEMENTED vectortype variance(){};

9.7.2.4 #define VARIANCE_NOT_YET_IMPLEMENTED vectortype variance(){};

Definition at line 14 of file tmbutils.cpp.

9.7.3 Function Documentation

9.7.3.1 template<class scalartype , class distribution > AR1_t<distribution> AR1 (scalartype *phi_*, distribution *f_*)

Examples:

ar1xar1.cpp.

Definition at line 297 of file density.cpp.

9.7.3.2 template<class scalartype > AR1_t<N01<scalartype> > AR1 (scalartype *phi_*)

Definition at line 301 of file density.cpp.

9.7.3.3 template<class scalartype , class vectortype > contAR2_t<scalartype> contAR2 (vectortype *grid_*, scalartype *shape_*, scalartype *scale_* = 1)

Definition at line 564 of file density.cpp.

9.7.3.4 template<class scalartype > contAR2_t<scalartype> contAR2 (scalartype *shape_*, scalartype *scale_* = 1)

Definition at line 568 of file density.cpp.

9.7.3.5 template<class scalartype > GMRF_t<scalartype> GMRF (Eigen::SparseMatrix< scalartype > *Q*, int *order* = 1)

Definition at line 695 of file density.cpp.

9.7.3.6 template<class scalartype , class arraytype > GMRF_t<scalartype> GMRF (arraytype *x*, vector< scalartype > *delta*, int *order* = 1)

Definition at line 699 of file density.cpp.

9.7.3.7 template<class scalartype , class arraytype > GMRF_t<scalartype> GMRF (arraytype *x*, scalartype *delta*, int *order* = 1)

Definition at line 703 of file density.cpp.

9.7.3.8 template<class scalartype > MVNORM_t<scalartype> MVNORM (matrix< scalartype > *x*)

Definition at line 108 of file density.cpp.

9.7.3.9 template<class distribution > PROJ_t<distribution> PROJ (distribution *f_*, vector< int > *i*)

Definition at line 1080 of file density.cpp.

9.7.3.10 template<class scalartype , class distribution > SCALE_t<distribution> SCALE (distribution *f_*, scalartype *scale_*)

Definition at line 740 of file density.cpp.

9.7.3.11 template<class distribution1 , class distribution2 > SEPARABLE_t<distribution1,distribution2> SEPARABLE (distribution1 *f_*, distribution2 *g_*)

Examples:

ar1xar1.cpp.

Definition at line 918 of file density.cpp.

9.7.3.12 template<class scalartype > UNSTRUCTURED_CORR_t<scalartype> UNSTRUCTURED_CORR (vector< scalartype > *x*)

Definition at line 176 of file density.cpp.

9.7.3.13 template<class vectortype , class distribution > VECSCALE_t<distribution> VECSCALE (distribution *f_*, vectortype *scale_*)

Definition at line 779 of file density.cpp.

9.8 dnorm.hpp File Reference

Functions

- template<class Type >
  Type dnorm (Type x, Type mean, Type sd, int give_log=0)

9.8.1 Function Documentation

9.8.1.1 template<class Type > Type dnorm (Type *x*, Type *mean*, Type *sd*, int *give_log* = 0)

Examples:

atomic.cpp, linreg.cpp, randomregression.cpp, rw.cpp, sdv_multi.cpp, simple.cpp, and socatt.cpp.

Definition at line 3 of file dnorm.hpp.

Referenced by ARk_t< scalartype_ >::operator()().

9.9 kronecker.cpp File Reference

Kronecker product of two matrices.

Functions

- template<class scalartype , int n1, int n2, int n3, int n4>
  Matrix< scalartype, n1 *n3, n2
  *n4 > kronecker (Matrix< scalartype, n1, n2 > x, Matrix< scalartype, n3, n4 > y)
  Kronecker product of two matrices.

9.9.1 Detailed Description

Kronecker product of two matrices.

Definition in file kronecker.cpp.

9.9.2 Function Documentation

9.9.2.1 template<class scalartype , int n1, int n2, int n3, int n4> Matrix<scalartype,n1*n3,n2*n4> kronecker (Matrix< scalartype, n1, n2 > *x*, Matrix< scalartype, n3, n4 > *y*)

Kronecker product of two matrices.

Definition at line 7 of file kronecker.cpp.

Referenced by contAR2_t< scalartype_ >::contAR2_t(), and contAR2_t< scalartype_ >::expB().

9.10 lgamma.hpp File Reference

Gamma function and gamma probability densities.

Functions

- template<class Type >
  Type lgamma (const Type &y)
- template<class Type >
  Type dnbinom (const Type &x, const Type &mu0, const Type &var0, int give_log=0)
  Negative binomial probability function.
- template<class Type >
  Type dpois (const Type &x, const Type &lambda, int give_log=0)
  Poisson probability function.
- template<class Type >
  Type dgamma (Type y, Type shape, Type scale, int give_log=0)
  Density of X where X~gamma distributed.
- template<class Type >
  Type dlgamma (Type y, Type shape, Type scale, int give_log=0)
  Density of log(X) where X~gamma distributed.

9.10.1 Detailed Description

Gamma function and gamma probability densities.

Definition in file lgamma.hpp.

9.10.2 Function Documentation

9.10.2.1 template<class Type > Type dgamma (Type *y,* Type *shape,* Type *scale,* int *give_log* = 0)

Density of X where X∼gamma distributed.

Definition at line 63 of file lgamma.hpp.

9.10.2.2 template<class Type > Type dlgamma (Type *y,* Type *shape,* Type *scale,* int *give_log* = 0) [inline]

Density of log(X) where X∼gamma distributed.

Definition at line 71 of file lgamma.hpp.

9.10.2.3 template<class Type > Type dnbinom (const Type & *x,* const Type & *mu0,* const Type & *var0,* int *give_log* = 0) [inline]

Negative binomial probability function.

Considering the return value we need to make sure that: (1) n>0 (2) 0<p (3) p<1 This is obtained by adding small constants appropriate places.

Definition at line 41 of file lgamma.hpp.

9.10.2.4 template<class Type > Type dpois (const Type & *x,* const Type & *lambda,* int *give_log* = 0) [inline]

Poisson probability function.

Definition at line 55 of file lgamma.hpp.

9.10.2.5 template<class Type > Type lgamma (const Type & *y*) [inline]

Examples:

nmix.cpp.

Definition at line 5 of file lgamma.hpp.

Referenced by dgamma(), dlgamma(), dnbinom(), and dpois().

9.11 mainpage.txt File Reference

9.12 matexp.cpp File Reference

Matrix exponential.

Classes

- struct matexp< scalartype, dim >

Matrix exponential: matrix of arbitrary dimension.

- struct matexp< scalartype, 2 >

  Matrix exponential: 2x2 case which can be handled efficiently.

9.12.1 Detailed Description

Matrix exponential.

Definition in file matexp.cpp.

9.13 order.cpp File Reference

"Differentiable" sorting of a vector.

Classes

- class order< Type >

9.13.1 Detailed Description

"Differentiable" sorting of a vector.

Example:

```
order<Type> perm(x);
vector<Type> xsort=perm(x);
```

Definition in file order.cpp.

9.14 Rstream.hpp File Reference

```
#include <iostream>
#include <cstdio>
#include <streambuf>
```

Classes

- class Rstreambuf< OUTPUT >
- class Rostream< OUTPUT >

Variables

- Rostream< true > Rcout
- Rostream< false > Rcerr

9.14.1 Variable Documentation

9.14.1.1 Rostream<false> Rcerr

Definition at line 65 of file Rstream.hpp.

9.14.1.2 Rostream<true> Rcout

Definition at line 64 of file Rstream.hpp.

9.15 splines.cpp File Reference

Classes

- class splinefun< Type >

Macros

- #define A b
- #define B d
- #define C c
- #define L b
- #define M d
- #define E e
- #define Y c
- #define D c
- #define X c
- #define A b
- #define B d
- #define C c
- #define L b
- #define M d
- #define E e
- #define Y c
- #define D c
- #define X c

9.15.1 Macro Definition Documentation

9.15.1.1 #define A b

Examples:

simple.cpp.

Referenced by discrLyap(), invertSparseMatrix(), and splinefun< Type >::periodic_spline().

9.15.1.2 #define A b

9.15.1.3 #define B d

Examples:

simple.cpp.

Referenced by splinefun< Type >::periodic_spline().

9.15.1.4 #define B d

9.15.1.5 #define C c

Referenced by discrLyap(), splinefun< Type >::periodic_spline(), and GMRF_t< scalartype_ >::variance().

9.15.1.6 #define C c

9.15.1.7 #define D c

Referenced by matexp< scalartype, 2 >::matexp(), splinefun< Type >::periodic_spline(), and GMRF_t< scalartype_ >::setQ().

9.15.1.8 #define D c

9.15.1.9 #define E e

Referenced by splinefun< Type >::periodic_spline().

9.15.1.10 #define E e

9.15.1.11 #define L b

Examples:

sdv_multi.cpp.

Referenced by ARk_t< scalartype_ >::ARk_t(), and splinefun< Type >::periodic_spline().

9.15.1.12 #define L b

9.15.1.13 #define M d

9.15.1.14 #define M d

Examples:

orange_big.cpp.

Referenced by splinefun< Type >::periodic_spline().

9.15.1.15 #define X c

9.15.1.16 #define X c

Examples:

socatt.cpp, and spatial.cpp.

Referenced by splinefun< Type >::periodic_spline().

9.15.1.17 #define Y c

9.15.1.18 #define Y c

Examples:

linreg.cpp.

Referenced by splinefun< Type >::periodic_spline().

9.16 spmat.cpp File Reference

Extends Eigen::SparseMatrix class.

Functions

- template<class Type >
  Eigen::SparseMatrix< Type > asSparseMatrix (SEXP M)
- template<class Type >
  Eigen::SparseMatrix< Type > asSparseMatrix (matrix< Type > x)
- template<class Type >
  Eigen::SparseVector< Type > asSparseVector (vector< Type > x)
- template<class Type >
  Eigen::SparseMatrix< Type > kronecker (Eigen::SparseMatrix< Type > x, Eigen::SparseMatrix< Type > y)
- template<class Type >
  matrix< Type > discrLyap (matrix< Type > A_)
- template<class Type >
  matrix< Type > invertSparseMatrix (matrix< Type > A_)

9.16.1 Detailed Description

Extends Eigen::SparseMatrix class.

Definition in file spmat.cpp.

9.16.2 Function Documentation

9.16.2.1 template<class Type > Eigen::SparseMatrix<Type> asSparseMatrix (SEXP *M*)

Create sparse matrix from R-triplet sparse matrix

Definition at line 7 of file spmat.cpp.

Referenced by discrLyap(), and invertSparseMatrix().

9.16.2.2 template<class Type > Eigen::SparseMatrix<Type> asSparseMatrix (matrix< Type > *x*)

Create sparse matrix from dense matrix

Definition at line 25 of file spmat.cpp.

9.16.2.3 template<class Type > Eigen::SparseVector<Type> asSparseVector (vector< Type > *x*)

Create sparse vector from dense vector

Definition at line 38 of file spmat.cpp.

9.16.2.4 template<class Type > matrix<Type> discrLyap (matrix< Type > A_)

Solve discrete Lyapunov equation V=AVA'+I

Definition at line 76 of file spmat.cpp.

9.16.2.5 template<class Type > matrix<Type> invertSparseMatrix (matrix< Type > A_)

Inverse of PD sparse matrix

Definition at line 101 of file spmat.cpp.

Referenced by GMRF_t< scalartype_ >::variance().

9.16.2.6 template<class Type > Eigen::SparseMatrix<Type> kronecker (Eigen::SparseMatrix< Type > x, Eigen::SparseMatrix< Type > y)

Kronecker product of two sparse matrices

Definition at line 49 of file spmat.cpp.

Referenced by discrLyap().

9.17 start_parallel.hpp File Reference

Classes

- struct sphess_t< ADFunType >
- struct parallelADFun< Type >

Typedefs

- typedef sphess_t< ADFun< double > > sphess

  sphess_t<ADFun<double> > sphess

9.17.1 Typedef Documentation

9.17.1.1 typedef sphess_t<ADFun<double> > sphess

sphess_t<ADFun<double> > sphess

Definition at line 40 of file start_parallel.hpp.

9.18 TMB.hpp File Reference

Includes and sets all stuff needed to compile the user defined objective function.

```
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <R.h>
#include <Rinternals.h>
#include "Rstream.hpp"
#include "cppad/cppad.hpp"
#include "tmbutils/tmbutils.cpp"
#include "convert.hpp"
#include "config.hpp"
#include "dnorm.hpp"
#include "lgamma.hpp"
#include "Vectorize.hpp"
#include "start_parallel.hpp"
#include "tmb_core.hpp"
#include "convenience.hpp"
```

Macros

- #define TMB_DEBUG 0
- #define TMB_PRINT(x) std::cout << #x << ": " << x << "\n"; std::cout.flush();
- #define NDEBUG 1
- #define NDEBUG 1

Functions

- void eigen_Rprintf (const char *x)

9.18.1 Detailed Description

Includes and sets all stuff needed to compile the user defined objective function.

Definition in file TMB.hpp.

9.18.2 Macro Definition Documentation

9.18.2.1 #define NDEBUG 1

Definition at line 32 of file TMB.hpp.

9.18.2.2 #define NDEBUG 1

Definition at line 32 of file TMB.hpp.

9.18.2.3 #define TMB_DEBUG 0

Definition at line 5 of file TMB.hpp.

9.18.2.4 #define TMB_PRINT(*x*) std::cout << #x << ": " << x << "\n"; std::cout.flush();

Definition at line 6 of file TMB.hpp.

Referenced by parallelADFun< Type >::parallelADFun().

9.18.3 Function Documentation

9.18.3.1 void eigen_Rprintf (const char * x)

Definition at line 29 of file TMB.hpp.

9.19 tmb_core.hpp File Reference

Interfaces to R and CppAD.

Classes

- struct SEXP_t

  TMB: SEXP type.
- struct memory_manager_struct

  Controls the life span of objects created in the C++ template (jointly R/C++)
- struct isDouble< Type >
- struct isDouble< double >
- struct report_stack< Type >

  Used by ADREPORT.
- class objective_function< Type >

  Type definition of user-provided objective function (i.e. neg. log. like)
- struct parallel_accumulator< Type >

Macros

- #define PARAMETER_MATRIX(name) tmbutils::matrix<Type> name(objective_function::fillShape(asMatrix<Type>(objective_function::getShape(#name,&isMatrix)),#name));

  Get parameter matrix from R and declare it as matrix<Type>
- #define PARAMETER_VECTOR(name) vector<Type> name(objective_function::fillShape(asVector<Type>(objective_function::getShape(#name,&isNumeric)),#name));

  Get parameter vector from R and declare it as vector<Type>
- #define PARAMETER(name) Type name(objective_function::fillShape(asVector<Type>(objective_function::getShape(#name,&isNumericScalar)),#name)[0]);

  Get parameter scalar from R and declare it as Type.
- #define DATA_VECTOR(name)

  Get data vector from R and declare it as vector<Type>
- #define DATA_MATRIX(name)

  Get data matrix from R and declare it as matrix<Type>
- #define DATA_SCALAR(name)

  Get data scalar from R and declare it as Type.
- #define DATA_INTEGER(name)

  Get data scalar from R and declare it as int.
- #define DATA_FACTOR(name)

  Get data vector of type "factor" from R and declare it as a zero-based integer vector.
- #define DATA_IVECTOR(name)

  Get data vector of type "integer" from R. (DATA_INTEGER is for a scalar integer)
- #define NLEVELS(name) LENGTH(getAttrib(getListElement(this->data,#name),install("levels")))

  Get the number of levels of a data factor from R.
- #define DATA_SPARSE_MATRIX(name)

Get sparse matrix from R and declare it as Eigen::SparseMatrix<Type>

- #define REPORT(name)

  Report scalar, vector or array back to R without derivative information.
- #define ADREPORT(name) objective_function::reportvector.push(name,#name);

  Report scalar, vector or array back to R with derivative information.
- #define PARALLEL_REGION if(this->parallel_region())
- #define DATA_ARRAY(name)

  Get data array from R and declare it as array<Type>
- #define PARAMETER_ARRAY(name) tmbutils::array<Type> name(objective_function::fillShape(tmbutils::asArray<Type>(objective_function::getShape(#name,&isArray)),#name));

  Get parameter array from R and declare it as array<Type>
- #define DATA_IMATRIX(name)

  Get data matrix from R and declare it as matrix<int>
- #define DATA_IARRAY(name)

  Get data array from R and declare it as array<int>
- #define KEEP_COL(col) (keepcol[col])
- #define KEEP_ROW(row, col) (KEEP_COL(row)&(row>=col))

Typedefs

- typedef Rboolean(* RObjectTester)(SEXP)

Functions

- bool operator< (SEXP_t x, SEXP_t y)
- SEXP ptrList (SEXP x)

  Convert x to TMB-format for R/C++ communication.
- void RObjectTestExpectedType (SEXP x, RObjectTester expectedtype, const char *nam)
- Rboolean isValidSparseMatrix (SEXP x)
- Rboolean isNumericScalar (SEXP x)
- template<class Type >
  matrix< int > HessianSparsityPattern (ADFun< Type > *pf)

  Get the hessian sparsity pattern of ADFun object pointer.
- SEXP getListElement (SEXP list, const char *str, RObjectTester expectedtype=NULL)

  Get list element named "str", or return NULL.
- void Independent (vector< double > x)

  Do nothing if we are trying to tape non AD-types.
- template<class ADFunType >
  SEXP EvalADFunObjectTemplate (SEXP f, SEXP theta, SEXP control)

  Evaluates an ADFun object from R.
- template<class ADFunType >
  void finalize (SEXP x)

  Garbage collect an ADFun or parallelADFun object pointer.
- ADFun< double > * MakeADFunObject (SEXP data, SEXP parameters, SEXP report, SEXP control, int parallel_region=-1, SEXP &info=R_NilValue)

  Construct ADFun object.
- void finalizeADFun (SEXP x)

  Garbage collect an ADFun object pointer.
- void finalizeparallelADFun (SEXP x)
- SEXP MakeADFunObject (SEXP data, SEXP parameters, SEXP report, SEXP control)

  Construct ADFun object.

- SEXP InfoADFunObject (SEXP f)
- SEXP optimizeADFunObject (SEXP f)

  Call tape optimization function in CppAD.
- SEXP getTag (SEXP f)

  Get tag of external pointer.
- SEXP EvalADFunObject (SEXP f, SEXP theta, SEXP control)
- void finalizeDoubleFun (SEXP x)
- SEXP MakeDoubleFunObject (SEXP data, SEXP parameters, SEXP report)
- SEXP EvalDoubleFunObject (SEXP f, SEXP theta, SEXP control)
- SEXP getParameterOrder (SEXP data, SEXP parameters, SEXP report)

  Gets parameter order by running the user template.
- ADFun< double > * MakeADGradObject (SEXP data, SEXP parameters, SEXP report, int parallel_region=-1)
- SEXP MakeADGradObject (SEXP data, SEXP parameters, SEXP report)

  Tape the gradient using nested AD types.
- SEXP MakeADHessObject (SEXP data, SEXP parameters, SEXP report, SEXP hessianrows, SEXP hessiancols)

  Tape the hessian[cbind(i,j)] using nested AD types.
- sphess MakeADHessObject2 (SEXP data, SEXP parameters, SEXP report, SEXP skip, int parallel_region=-1)

  Tape the hessian[cbind(i,j)] using nested AD types.
- template<class ADFunType >
  SEXP asSEXP (const sphess_t< ADFunType > &H, const char *tag)

  Convert sparse matrix H to SEXP format that can be returned to R.
- SEXP MakeADHessObject2 (SEXP data, SEXP parameters, SEXP report, SEXP skip)
- SEXP dummy_getParameterOrder (SEXP data, SEXP parameters, SEXP report)

Variables

- struct memory_manager_struct memory_manager
- bool _openmp =false

9.19.1 Detailed Description

Interfaces to R and CppAD.

Definition in file tmb_core.hpp.

9.19.2 Macro Definition Documentation

9.19.2.1 #define ADREPORT(*name*) objective_function::reportvector.push(name,#name);

Report scalar, vector or array back to R with derivative information.

Examples:

linreg.cpp, orange_big.cpp, and rw.cpp.

Definition at line 180 of file tmb_core.hpp.

9.19.2.2 #define DATA_ARRAY(*name*)

Value:

```
tmbutils::array<Type> name(tmbutils::asArray<Type>(        \
        getListElement(objective_function::data,#name,&isArray)));
```

Get data array from R and declare it as array<Type>

Examples:

rw.cpp, and sdv_multi.cpp.

Definition at line 183 of file tmb_core.hpp.

9.19.2.3 #define DATA_FACTOR(*name*)

Value:

```
vector<int> name(asVector<int>(      \
        getListElement(objective_function::data,#name,&isNumeric)));
```

Get data vector of type "factor" from R and declare it as a zero-based integer vector.

The following example (R code) shows what you have on the R side and what is being received by the C++ template:

```
> x=factor(letters[4:10])
> x
[1] d e f g h i j
Levels: d e f g h i j

# The zero-based integer vector that the C++ template sees
> unclass(x) - 1
[1] 0 1 2 3 4 5 6
```

Examples:

nmix.cpp, orange_big.cpp, randomregression.cpp, and socatt.cpp.

Definition at line 165 of file tmb_core.hpp.

9.19.2.4 #define DATA_IARRAY(*name*)

Value:

```
tmbutils::array<int> name(tmbutils::asArray<int>(   \
        getListElement(objective_function::data,#name,&isArray)));
```

Get data array from R and declare it as array<int>

Definition at line 191 of file tmb_core.hpp.

9.19.2.5 #define DATA_IMATRIX(*name*)

Value:

```
matrix<int> name(asMatrix<int>(      \
        getListElement(objective_function::data,#name,&isMatrix)));
```

Get data matrix from R and declare it as matrix<int>

Definition at line 188 of file tmb_core.hpp.

9.19.2.6 #define DATA_INTEGER(*name*)

Value:

```
int name(CppAD::Integer(asVector<Type>( \
        getListElement(objective_function::data,#name,&
      isNumericScalar))[0]));
```

Get data scalar from R and declare it as int.

Examples:

matrix_arrays.cpp, nmix.cpp, orange_big.cpp, sdv_multi.cpp, socatt.cpp, and spatial.cpp.

Definition at line 148 of file tmb_core.hpp.

9.19.2.7 #define DATA_IVECTOR(*name*)

Value:

```
vector<int> name(asVector<int>(      \
        getListElement(objective_function::data,#name,&isNumeric)));
```

Get data vector of type "integer" from R. (DATA_INTEGER is for a scalar integer)

Definition at line 168 of file tmb_core.hpp.

9.19.2.8 #define DATA_MATRIX(*name*)

Value:

```
matrix<Type> name(asMatrix<Type>(   \
        getListElement(objective_function::data,#name,&isMatrix)));
```

Get data matrix from R and declare it as matrix<Type>

Examples:

nmix.cpp, socatt.cpp, and spatial.cpp.

Definition at line 142 of file tmb_core.hpp.

9.19.2.9 #define DATA_SCALAR(*name*)

Value:

```
Type name(asVector<Type>(               \
        getListElement(objective_function::data,#name,&
      isNumericScalar))[0]);
```

Get data scalar from R and declare it as Type.

Definition at line 145 of file tmb_core.hpp.

9.19.2.10 #define DATA_SPARSE_MATRIX(*name*)

Value:

```
Eigen::SparseMatrix<Type> name(tmbutils::asSparseMatrix<Type>( \
        getListElement(objective_function::data,#name,&
      isValidSparseMatrix)));
```

Get sparse matrix from R and declare it as Eigen::SparseMatrix<Type>

Examples:

simple.cpp.

Definition at line 173 of file tmb_core.hpp.

9.19.2.11 #define DATA_VECTOR(*name*)

Value:

```
vector<Type> name(asVector<Type>(    \
        getListElement(objective_function::data,#name,&isNumeric)));
```

Get data vector from R and declare it as vector<Type>

Examples:

ar1xar1.cpp, linreg.cpp, nmix.cpp, orange_big.cpp, randomregression.cpp, simple.cpp, and spatial.cpp.

Definition at line 139 of file tmb_core.hpp.

9.19.2.12 #define KEEP_COL(*col*) (keepcol[col])

Referenced by MakeADHessObject2().

9.19.2.13 #define KEEP_ROW(*row, col*) (KEEP_COL(row)&(row>=col))

Referenced by MakeADHessObject2().

9.19.2.14 #define NLEVELS(*name*) LENGTH(getAttrib(getListElement(this->data,#name),install("levels")))

Get the number of levels of a data factor from R.

Definition at line 171 of file tmb_core.hpp.

9.19.2.15 #define PARALLEL_REGION if(this->parallel_region())

Definition at line 181 of file tmb_core.hpp.

9.19.2.16 #define PARAMETER(*name*) Type name(objective_function::fillShape(asVector<Type>(objective_↩ function::getShape(#name,&isNumericScalar)),#name)[0]);

Get parameter scalar from R and declare it as Type.

Examples:

ar1xar1.cpp, linreg.cpp, matrix_arrays.cpp, nmix.cpp, orange_big.cpp, randomregression.cpp, rw.cpp, simple.cpp, socatt.cpp, and spatial.cpp.

Definition at line 137 of file tmb_core.hpp.

9.19.2.17 #define PARAMETER_ARRAY(*name*) tmbutils::array<Type> name(objective_function::fillShape(tmbutils::asArray<Type>(objective_function::getShape(#name,&isArray)),#name));

Get parameter array from R and declare it as array<Type>

Examples:

ar1xar1.cpp, atomic.cpp, rw.cpp, and sdv_multi.cpp.

Definition at line 186 of file tmb_core.hpp.

9.19.2.18 #define PARAMETER_MATRIX(*name*) tmbutils::matrix<Type> name(objective_function::fillShape(asMatrix<Type>(objective_function::getShape(#name,&isMatrix)),#name));

Get parameter matrix from R and declare it as matrix<Type>

Definition at line 133 of file tmb_core.hpp.

9.19.2.19 #define PARAMETER_VECTOR(*name*) vector<Type> name(objective_function::fillShape(asVector<Type>(objective_function::getShape(#name,&isNumeric)),#name));

Get parameter vector from R and declare it as vector<Type>

Examples:

nmix.cpp, orange_big.cpp, randomregression.cpp, rw.cpp, sdv_multi.cpp, simple.cpp, socatt.cpp, spatial.cpp, and sumtest.cpp.

Definition at line 135 of file tmb_core.hpp.

9.19.2.20 #define REPORT(*name*)

Value:

```
if(isDouble<Type>::value && this->current_parallel_region<0){          \
                        defineVar(install(#name),asSEXP(name),
      objective_function::report);}
```

Report scalar, vector or array back to R without derivative information.

Definition at line 177 of file tmb_core.hpp.

9.19.3 Typedef Documentation

9.19.3.1 typedef Rboolean(* RObjectTester)(SEXP)

Definition at line 107 of file tmb_core.hpp.

9.19.4 Function Documentation

9.19.4.1 template<class ADFunType > SEXP asSEXP (const sphess_t< ADFunType > & *H,* const char * *tag*)

Convert sparse matrix H to SEXP format that can be returned to R.

Definition at line 1196 of file tmb_core.hpp.

Referenced by EvalADFunObjectTemplate(), EvalDoubleFunObject(), InfoADFunObject(), and MakeADHessObject2().

9.19.4.2 SEXP dummy_getParameterOrder (SEXP *data,* SEXP *parameters,* SEXP *report*)

Definition at line 1283 of file tmb_core.hpp.

9.19.4.3 SEXP EvalADFunObject (SEXP *f,* SEXP *theta,* SEXP *control*)

Examples:

called_from_R.cpp.

Definition at line 821 of file tmb_core.hpp.

9.19.4.4 template<class ADFunType > SEXP EvalADFunObjectTemplate (SEXP *f,* SEXP *theta,* SEXP *control*)

Evaluates an ADFun object from R.

Template argument can be "ADFun" or an object extending "ADFun" such as "parallelADFun".

Parameters

f	R external pointer to ADFunType
theta	R vector of parameters
control	R list controlling what to be returned

It is assumed that $f : R^n \to R^m$ where n and m are found from f. The list "control" can contain the following components:

order: mandatory integer 0,1,2, or 3 with order of derivatives to be calculated.
hessiancols, hessianrows: Optional one-based integer vectors of the same length. Used only in the case where order=2 to extract specific entries of hessian.
sparsitypattern: Integer flag. Return sparsity pattern instead of numerical values?
rangeweight: Optional R vector of doubles of length m. If supplied, a 1st order reverse mode sweep is performed in this range direction.
rangecomponent: Optional one-based integer (scalar) between 1 and m. Used to select a given component of the vector f(x). dumpstack: Integer flag. If non zero the entire operation stack is dumped as text output during 0-order forward sweep.

Possible output depends on "order".

order==0: Calculate f(x) output as vector of length m.
order==1: If "rangeweight" is supplied, calculate the gradient of the function x -> inner_prod(f(x),w) from R^n->R. Otherwise, calculate the full jacobian (of dimension m*n).
order==2: If nothing further is specified, calculate the full hessian of the function x->f(x)[rangecomponent] from R^n->R

All other usage is considered deprecated/experimental and may be removed in the future.

Definition at line 586 of file tmb_core.hpp.

Referenced by EvalADFunObject().

9.19.4.5 SEXP EvalDoubleFunObject (SEXP *f,* SEXP *theta,* SEXP *control*)

Examples:

called_from_R.cpp.

Definition at line 867 of file tmb_core.hpp.

9.19.4.6 template<class ADFunType > void finalize (SEXP *x*)

Garbage collect an ADFun or parallelADFun object pointer.

Definition at line 670 of file tmb_core.hpp.

9.19.4.7 void finalizeADFun (SEXP *x*)

Garbage collect an ADFun object pointer.

Definition at line 710 of file tmb_core.hpp.

Referenced by MakeADFunObject(), MakeADGradObject(), and MakeADHessObject().

9.19.4.8 void finalizeDoubleFun (SEXP *x*)

Definition at line 839 of file tmb_core.hpp.

Referenced by MakeDoubleFunObject().

9.19.4.9 void finalizeparallelADFun (SEXP *x*)

Definition at line 716 of file tmb_core.hpp.

Referenced by MakeADFunObject(), and MakeADGradObject().

9.19.4.10 SEXP getListElement (SEXP *list,* const char * *str,* RObjectTester *expectedtype* = NULL)

Get list element named "str", or return NULL.

Definition at line 213 of file tmb_core.hpp.

Referenced by EvalADFunObjectTemplate(), objective_function< Type >::fillmap(), objective_function< Type >::fillShape(), objective_function< Type >::getShape(), and MakeADFunObject().

9.19.4.11 SEXP getParameterOrder (SEXP *data,* SEXP *parameters,* SEXP *report*)

Gets parameter order by running the user template.

We spend a function evaluation on getting the parameter order (!)

Examples:

called_from_R.cpp.

Definition at line 893 of file tmb_core.hpp.

Referenced by dummy_getParameterOrder().

9.19.4.12 SEXP getTag (SEXP *f*)

Get tag of external pointer.

Definition at line 817 of file tmb_core.hpp.

9.19.4.13 template<class Type > matrix<int> HessianSparsityPattern (ADFun< Type > * *pf*)

Get the hessian sparsity pattern of ADFun object pointer.

Definition at line 197 of file tmb_core.hpp.

Referenced by EvalADFunObjectTemplate().

9.19.4.14 void Independent (vector< double > *x*)

Do nothing if we are trying to tape non AD-types.

Definition at line 232 of file tmb_core.hpp.

Referenced by MakeADFunObject(), MakeADGradObject(), MakeADHessObject(), and MakeADHessObject2().

9.19.4.15 SEXP InfoADFunObject (SEXP *f*)

Examples:

called_from_R.cpp.

Definition at line 779 of file tmb_core.hpp.

9.19.4.16 Rboolean isNumericScalar (SEXP *x*)

Definition at line 122 of file tmb_core.hpp.

9.19.4.17 Rboolean isValidSparseMatrix (SEXP *x*)

Definition at line 118 of file tmb_core.hpp.

9.19.4.18 ADFun<double>* MakeADFunObject (SEXP *data*, SEXP *parameters*, SEXP *report*, SEXP *control*, int *parallel_region* = -1, SEXP & *info* = R_NilValue)

Construct ADFun object.

Examples:

called_from_R.cpp.

Definition at line 679 of file tmb_core.hpp.

Referenced by MakeADFunObject().

9.19.4.19 SEXP MakeADFunObject (SEXP *data*, SEXP *parameters*, SEXP *report*, SEXP *control*)

Construct ADFun object.

Definition at line 724 of file tmb_core.hpp.

9.19.4.20 ADFun< double >* MakeADGradObject (SEXP *data*, SEXP *parameters*, SEXP *report*, int *parallel_region* = -1)

Examples:

called_from_R.cpp.

Definition at line 909 of file tmb_core.hpp.

Referenced by MakeADGradObject().

9.19.4.21 SEXP MakeADGradObject (SEXP *data*, SEXP *parameters*, SEXP *report*)

Tape the gradient using nested AD types.

Definition at line 932 of file tmb_core.hpp.

9.19.4.22 SEXP MakeADHessObject (SEXP *data*, SEXP *parameters*, SEXP *report*, SEXP *hessianrows*, SEXP *hessiancols*)

Tape the hessian[cbind(i,j)] using nested AD types.

Definition at line 985 of file tmb_core.hpp.

9.19.4.23 sphess MakeADHessObject2 (SEXP *data*, SEXP *parameters*, SEXP *report*, SEXP *skip*, int *parallel_region* = -1)

Tape the hessian[cbind(i,j)] using nested AD types.

skip: integer vector of columns to skip from the hessian (will not change dimension

- only treat h[:,skip] and h[skip,:] as zero). Negative subscripts are not allowed.

Examples:

called_from_R.cpp.

Definition at line 1054 of file tmb_core.hpp.

Referenced by MakeADHessObject2().

9.19.4.24 SEXP MakeADHessObject2 (SEXP *data*, SEXP *parameters*, SEXP *report*, SEXP *skip*)

Definition at line 1245 of file tmb_core.hpp.

9.19.4.25 SEXP MakeDoubleFunObject (SEXP *data*, SEXP *parameters*, SEXP *report*)

Examples:

called_from_R.cpp.

Definition at line 846 of file tmb_core.hpp.

9.19.4.26 bool operator< (SEXP_t *x*, SEXP_t *y*)

Definition at line 18 of file tmb_core.hpp.

9.19.4.27 SEXP optimizeADFunObject (SEXP *f*)

Call tape optimization function in CppAD.

Examples:

called_from_R.cpp.

Definition at line 800 of file tmb_core.hpp.

9.19.4.28 SEXP ptrList (SEXP *x*)

Convert x to TMB-format for R/C++ communication.

All external pointers returned from TMB should be placed in a list container of length one. Additional information should be set as attributes to the pointer. The memory_manager_struct above knows how to look up the list container given the external pointer. By setting the list element to NULL the memory_manager can trigger the garbage collector (and thereby the finalizers) when the library is unloaded.

Definition at line 57 of file tmb_core.hpp.

Referenced by asSEXP(), MakeADFunObject(), MakeADGradObject(), MakeADHessObject(), and MakeDouble↩FunObject().

9.19.4.29 void RObjectTestExpectedType (SEXP *x,* RObjectTester *expectedtype,* const char * *nam*)

Definition at line 108 of file tmb_core.hpp.

Referenced by getListElement(), and objective_function< Type >::getShape().

9.19.5 Variable Documentation

9.19.5.1 bool _openmp =false

Definition at line 89 of file tmb_core.hpp.

9.19.5.2 struct memory_manager_struct memory_manager

Referenced by finalize(), finalizeADFun(), finalizeDoubleFun(), finalizeparallelADFun(), and ptrList().

9.20 tmbutils.cpp File Reference

Namespace of utility functions for TMB.

```
#include "vector.cpp"
#include "array.cpp"
#include "spmat.cpp"
#include "kronecker.cpp"
#include "matexp.cpp"
#include "splines.cpp"
#include "order.cpp"
#include "density.cpp"
```

Classes

- struct tmbutils::vector< Type >

  Vector class used by TMB.
- struct tmbutils::matrix< Type >

  Matrix class used by TMB.
- struct tmbutils::array< Type >

  Array class used by TMB.
- struct tmbutils::matexp< scalartype, dim >

  Matrix exponential: matrix of arbitrary dimension.
- struct tmbutils::matexp< scalartype, 2 >

  Matrix exponential: 2x2 case which can be handled efficiently.
- class tmbutils::splinefun< Type >
- class tmbutils::order< Type >
- class density::MVNORM_t< scalartype_ >

  Multivariate normal distribution with user supplied covariance matrix.
- class density::UNSTRUCTURED_CORR_t< scalartype_ >

  Multivariate normal distribution with unstructered correlation matrix.
- class density::N01< scalartype_ >

  Standardized normal distribution.
- class density::AR1_t< distribution >

  Stationary AR1 process.
- class density::ARk_t< scalartype_ >

  Stationary AR(k) process.
- class density::contAR2_t< scalartype_ >

  Continuous AR(2) process.
- class density::GMRF_t< scalartype_ >

  Gaussian Markov Random Field.
- class density::SCALE_t< distribution >

  Apply scale transformation on a density.
- class density::VECSCALE_t< distribution >

  Apply a vector scale transformation on a density.
- class density::SEPARABLE_t< distribution1, distribution2 >

  Separable extension of two densitites.
- class density::PROJ_t< distribution >

  Projection of multivariate gaussian variable.

Namespaces

- tmbutils
- density

  Namespace to construct multivariate Gaussian distributions via C++ templates
  A particular multivariate normal distribution is implemented as a templated C++ class. Let us take the generic zero-mean multivariate normal distribution `MVNORM_t` with covariance matrix `Sigma` as an example. The `_t` symbol attached to the class name reminds us that we are dealing with a class (of a particular C++ type). There are two operations that we can do on objects from the `MVNORM_t` class:
  - *Declare and initialize in terms of one or more parameters (e.g. `Sigma`)*
  - *Evaluate the negative log-likelihood density at specified point (e.g. a vector `u`)*

  An example is.

Functions

- template<class Type >
  Eigen::SparseMatrix< Type > tmbutils::asSparseMatrix (SEXP M)
- template<class Type >
  Eigen::SparseMatrix< Type > tmbutils::asSparseMatrix (matrix< Type > x)
- template<class Type >
  Eigen::SparseVector< Type > tmbutils::asSparseVector (vector< Type > x)
- template<class Type >
  Eigen::SparseMatrix< Type > tmbutils::kronecker (Eigen::SparseMatrix< Type > x, Eigen::SparseMatrix< Type > y)
- template<class Type >
  matrix< Type > tmbutils::discrLyap (matrix< Type > A_)
- template<class Type >
  matrix< Type > tmbutils::invertSparseMatrix (matrix< Type > A_)
- template<class scalartype , int n1, int n2, int n3, int n4>
  Matrix< scalartype, n1 *n3, n2
  *n4 > tmbutils::kronecker (Matrix< scalartype, n1, n2 > x, Matrix< scalartype, n3, n4 > y)

  Kronecker product of two matrices.
- template<class Type , class T1 , class T2 >
  vector< Type > tmbutils::dnorm (vector< Type > x, T1 mean, T2 sd, int give_log=0)
- template<class Type , class From >
  vector< Type > tmbutils::asVector (From *px, int n)
- template<class scalartype >
  MVNORM_t< scalartype > density::MVNORM (matrix< scalartype > x)
- template<class scalartype >
  UNSTRUCTURED_CORR_t< scalartype > density::UNSTRUCTURED_CORR (vector< scalartype > x)
- template<class scalartype , class distribution >
  AR1_t< distribution > density::AR1 (scalartype phi_, distribution f_)
- template<class scalartype >
  AR1_t< N01< scalartype > > density::AR1 (scalartype phi_)
- template<class scalartype , class vectortype >
  contAR2_t< scalartype > density::contAR2 (vectortype grid_, scalartype shape_, scalartype scale_=1)
- template<class scalartype >
  contAR2_t< scalartype > density::contAR2 (scalartype shape_, scalartype scale_=1)
- template<class scalartype >
  GMRF_t< scalartype > density::GMRF (Eigen::SparseMatrix< scalartype > Q, int order=1)
- template<class scalartype , class arraytype >
  GMRF_t< scalartype > density::GMRF (arraytype x, vector< scalartype > delta, int order=1)
- template<class scalartype , class arraytype >
  GMRF_t< scalartype > density::GMRF (arraytype x, scalartype delta, int order=1)
- template<class scalartype , class distribution >
  SCALE_t< distribution > density::SCALE (distribution f_, scalartype scale_)
- template<class vectortype , class distribution >
  VECSCALE_t< distribution > density::VECSCALE (distribution f_, vectortype scale_)
- template<class distribution1 , class distribution2 >
  SEPARABLE_t< distribution1,
  distribution2 > density::SEPARABLE (distribution1 f_, distribution2 g_)
- template<class distribution >
  PROJ_t< distribution > density::PROJ (distribution f_, vector< int > i)

9.20.1 Detailed Description

Namespace of utility functions for TMB.

Definition in file tmbutils.cpp.

9.21 vector.cpp File Reference

Defines TMB vectors.

Classes

- struct vector< Type >
  Vector class used by TMB.
- struct matrix< Type >
  Matrix class used by TMB.

9.21.1 Detailed Description

Defines TMB vectors.

Array templates using inheritance.

Definition in file vector.cpp.

9.22 Vectorize.hpp File Reference

Classes

- class piecewice< Type >

Macros

- #define VECTORIZE1(FUN)
- #define VECTORIZE32(FUN)
- #define VECTORIZE31(FUN)
- #define VECTORIZE21(FUN)
- #define VECTORIZE2(FUN)

Functions

- VECTORIZE1 (abs)
- VECTORIZE1 (acos)
- VECTORIZE1 (asin)
- VECTORIZE1 (atan)
- VECTORIZE1 (cos)
- VECTORIZE1 (erf)
- VECTORIZE1 (exp)
- VECTORIZE1 (lgamma)
- VECTORIZE1 (log)
- VECTORIZE1 (log10)
- VECTORIZE1 (sin)
- VECTORIZE1 (sqrt)
- VECTORIZE2 (pow)
- VECTORIZE31 (dnorm)
- VECTORIZE31 (dnbinom)
- VECTORIZE31 (dgamma)
- VECTORIZE31 (dlgamma)

- VECTORIZE21 (dpois)
- double max (const vector< double > &x)
- template<class Type >
  Type max (const vector< Type > &x)
- double min (const vector< double > &x)
- template<class Type >
  Type min (const vector< Type > &x)
- template<class Type >
  vector< Type > select_row (const matrix< Type > &x, Type index)
- template<class Type >
  Type select_elt (const vector< Type > &x, Type i)

9.22.1 Macro Definition Documentation

9.22.1.1 #define VECTORIZE1(*FUN*)

Value:

```
template <class Type>                           \
vector<Type> FUN(const vector<Type> &x)             \
{                                               \
  vector<Type> res(x.size());                   \
  for(int i=0;i<x.size();i++)res[i]=FUN(x[i]);  \
  return res;                                   \
}                                               \
template <class Type>                           \
matrix<Type> FUN(const matrix<Type> &x)             \
{                                               \
  matrix<Type> res(x.rows(),x.cols());  \
  for(int i=0;i<x.rows();i++)           \
    for(int j=0;j<x.cols();j++)         \
      res(i,j)=FUN(x(i,j));                     \
  return res;                                   \
}
```

Definition at line 3 of file Vectorize.hpp.

9.22.1.2 #define VECTORIZE2(*FUN*)

Value:

```
template <class T>                                      \
vector<T> FUN(const vector<T> &x1, const T &x2){        \
  int n=x1.size();                                      \
  vector<T> res(n);                                     \
  for(int i=0;i<n;i++)res[i]=FUN(x1[i],x2);             \
  return res;                                           \
}
```

Definition at line 50 of file Vectorize.hpp.

9.22.1.3 #define VECTORIZE21(*FUN*)

Value:

```
template <class T>                                                      \
vector<T> FUN(const vector<T> &x1, const vector<T> &x2, int x4){        \
  int n1=x1.size();int n2=x2.size();                                    \
  int n=int(fmax(n1,n2));                                               \
  vector<T> res(n);                                                     \
  for(int i=0;i<n;i++)res[i]=FUN(x1[i%n1],x2[i%n2],x4);                 \
  return res;                                                           \
}
```

Definition at line 40 of file Vectorize.hpp.

9.22.1.4 #define VECTORIZE31(*FUN*)

Value:

```
template <class T>                                                          \
vector<T> FUN(const vector<T> &x1, const vector<T> &x2,                           \
              const vector<T> &x3, int x4){                                \
  int n1=x1.size();int n2=x2.size();int n3=x3.size();                       \
  int n=int(fmax(fmax(n1,n2),n3));                                          \
  vector<T> res(n);                                                         \
  for(int i=0;i<n;i++)res[i]=FUN(x1[i%n1],x2[i%n2],x3[i%n3],x4);            \
  return res;                                                               \
}
```

Definition at line 30 of file Vectorize.hpp.

9.22.1.5 #define VECTORIZE32(*FUN*)

Value:

```
template <class T,class T4,class T5>                                        \
vector<T> FUN(const vector<T> &x1, const vector<T> &x2,                    \
              const vector<T> &x3, T4 x4, T5 x5){                          \
  int n1=x1.size();int n2=x2.size();int n3=x3.size();                       \
  int n=fmax(fmax(n1,n2),n3);                                               \
  vector<T> res(n);                                                         \
  for(int i=0;i<n;i++)res[i]=FUN(x1[i%n1],x2[i%n2],x3[i%n3],x4,x5);         \
  return res;                                                               \
}
```

Definition at line 20 of file Vectorize.hpp.

9.22.2 Function Documentation

9.22.2.1 double max (const vector< double > & *x*)

Definition at line 82 of file Vectorize.hpp.

9.22.2.2 template<class Type > Type max (const vector< Type > & *x*)

Definition at line 92 of file Vectorize.hpp.

9.22.2.3 double min (const vector< double > & *x*)

Definition at line 100 of file Vectorize.hpp.

9.22.2.4 template<class Type > Type min (const vector< Type > & *x*)

Definition at line 110 of file Vectorize.hpp.

9.22.2.5 template<class Type > Type select_elt (const vector< Type > & *x*, Type *i*)

Definition at line 154 of file Vectorize.hpp.

9.22.2.6 template<class Type > vector<Type> select_row (const matrix< Type > & *x*, Type *index*)

Definition at line 139 of file Vectorize.hpp.

9.22.2.7 VECTORIZE1 (abs)

9.22.2.8 VECTORIZE1 (acos)

9.22.2.9 VECTORIZE1 (asin)

9.22.2.10 VECTORIZE1 (atan)

9.22.2.11 VECTORIZE1 (cos)

9.22.2.12 VECTORIZE1 (erf)

9.22.2.13 VECTORIZE1 (exp)

9.22.2.14 VECTORIZE1 (lgamma)

9.22.2.15 VECTORIZE1 (log)

9.22.2.16 VECTORIZE1 (log10)

9.22.2.17 VECTORIZE1 (sin)

9.22.2.18 VECTORIZE1 (sqrt)

9.22.2.19 VECTORIZE2 (pow)

9.22.2.20 VECTORIZE21 (dpois)

9.22.2.21 VECTORIZE31 (dnorm)

9.22.2.22 VECTORIZE31 (dnbinom)

9.22.2.23 VECTORIZE31 (dgamma)

9.22.2.24 VECTORIZE31 (dlgamma)

Chapter 10

Example Documentation

10.1 ar1xar1.cpp

```
// Separable covariance on lattice with AR1 structure in each direction.
#include <TMB.hpp>

/* Parameter transform */
template <class Type>
Type f(Type x){return Type(2)/(Type(1) + exp(-Type(2) * x)) - Type(1);}

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(N)
  PARAMETER_ARRAY(eta);
  PARAMETER(transf_phi1); /* fastest running dim */
  PARAMETER(transf_phi2); /* slowest running dim */
  Type phi1=f(transf_phi1);
  Type phi2=f(transf_phi2);

  using namespace density;
  Type res=0;
  // phi1 fastest running
  // res+=AR1(phi2,AR1(phi1))(eta);
  // Equivalent:
  res+=SEPARABLE(AR1(phi2),AR1(phi1))(eta);

  // logdpois = N log lam - lam
  for(int i=0;i<N.size();i++)res-=N[i]*eta[i]-exp(eta[i]);

  return res;

}
```

10.2 atomic.cpp

```
// Demonstrate user specified atomic functions.
#include <TMB.hpp>
#include <atomic_macro.hpp>

template<class Type>
vector<Type> dowork(vector<Type> x){
  int n=400;
  vector<Type> y(n);
  for(int i=0;i<n;i++)y[i]=dnorm(Type(i)/Type(n),x[0],x[1],0);
  Type s=0;
  for(int i=0;i<n;i++)s+=y[i];
  for(int i=0;i<n;i++)y[i]/=s;
  return y;
}
REGISTER_ATOMIC(dowork);


template<class Type>
Type objective_function<Type>::operator() ()
{
  PARAMETER_ARRAY(x);
  int m=x.cols();
```

```
  Type res=0;
  int n=400;
  vector<Type> tmp(n);
  tmp.setZero();
  for(int i=0;i<m;i++)tmp+=dowork(vector<Type>(x.col(i)));
  res=tmp.sum();
  return res;
}
```

10.3 called_from_R.cpp

```
// Collection of .Call()'s in "TMB.R". The purpose of this file is to show which C++ functions are called
      from R.
// Each function call is proceeded by the corresponding line from TMB.R
// Warning: this file may be outdated.
#include <TMB.hpp>

//  .Call("destructive_CHM_update",L,H,as.double(t),PACKAGE="Matrix")
destructive_CHM_update();

//    parNameOrder <- .Call("getParameterOrder",data,parameters,new.env(),PACKAGE=DLL)
getParameterOrder();

//      ADFun <<- .Call("MakeADFunObject",data,parameters,reportenv,
MakeADFunObject();

//      Fun <<- .Call("MakeDoubleFunObject",data,parameters,reportenv,PACKAGE=DLL)
MakeDoubleFunObject();

//
       .Call("EvalDoubleFunObject",Fun$ptr,unlist(parameters),control=list(order=as.integer(0)),PACKAGE=DLL)
EvalDoubleFunObject();

//      ADGrad <<- .Call("MakeADGradObject",data,parameters,reportenv,PACKAGE=DLL)
MakeADGradObject();

//      res <- .Call("EvalADFunObject",ADFun$ptr,theta,
//        .Call("EvalADFunObject",e$ADHess$ptr,theta,
//  ev <- function(par=obj$env$par).Call("EvalADFunObject", ADHess$ptr, par,
EvalADFunObject();

//      res <- .Call("EvalDoubleFunObject",Fun$ptr,theta,
EvalDoubleFunObject();

//      solveSubset <- function(L).Call("tmb_invQ",L,PACKAGE="TMB")
tmb_invQ();

//      solveSubset2 <- function(L).Call("tmb_invQ_tril_halfdiag",L,PACKAGE="TMB")
tmb_invQ_tril_halfdiag();

//        m <- .Call("match_pattern",A,B,PACKAGE="TMB") ## Same length as A@x with pointers to B@x
match_pattern();

//      ##.Call("destructive_CHM_update",L,hessian,as.double(0),PACKAGE="Matrix")
destructive_CHM_update();

//  .Call("omp_num_threads",n,PACKAGE="TMB")
omp_num_threads();

//    unlist(.Call("InfoADFunObject",get(name,env),PACKAGE=obj$env$DLL))
InfoADFunObject();

//    unlist(.Call("optimizeADFunObject",get(name,env)$ptr,PACKAGE=obj$env$DLL))
//    .Call("optimizeADFunObject",ADHess$ptr,PACKAGE=obj$env$DLL)
optimizeADFunObject();

//  ADHess <- .Call("MakeADHessObject2", obj$env$data, obj$env$parameters,
MakeADHessObject2();

//        return( .Call("setxslot",Hrandom,ev(par),PACKAGE="TMB") )
setxslot();

//  ok <- .Call("have_tmb_symbolic",PACKAGE="TMB")
have_tmb_symbolic();

//  L <- .Call("tmb_symbolic",h,PACKAGE="TMB")
tmb_symbolic();
```

10.4 linreg.cpp

```
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(Y);
  DATA_VECTOR(x);
  PARAMETER(a);
  PARAMETER(b);
  PARAMETER(logSigma);
  ADREPORT(exp(2*logSigma));
  Type nll=-sum(dnorm(Y,a+b*x,exp(logSigma),true));
  return nll;
}
```

10.5 matrix_arrays.cpp

```
// Shows use of vector, matrix and array operations.
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  // Data objects
  DATA_INTEGER(i);          // Scalar integer

  // Parameter objects
  PARAMETER(p)

  // Objects of double type so that they can be printed
  vector<double> v1(2);
  v1 << 9,11;
  std::cout << "v1 = \n"        << v1                          << "\n";
  matrix<double> m1(2,2);
  m1 << 1,2,3,4;
  std::cout << "m1 = \n"        << m1                          << "\n";
  array<double> a1(2,2);
  a1 << 11,12,13,14;
  std::cout << "a1 = \n"        << a1                          << "\n";

  // Obtaining dimensions of objects
  vector<int> d1 = a1.dim;
  std::cout << "Dimension of array a1= \n"      << d1                          << "\n";
  vector<int> d2(2);
  d2(0) = m1.rows();
  d2(1) = m1.cols();
  std::cout << "Dimension of matrix m1= \n"     << d2                          << "\n";
  int d3 = v1.size();
  std::cout << "Dimension of vector v1= "       << d3                          << "\n";

  // Matrix multiplication versus elementwise operations
  std::cout << "Element-wise product of arrays = \n"    << a1*a1                                  << "\n";
  std::cout << "Element-wise division of arrays = \n"   << a1/a1                                  << "\n";
  std::cout << "Element-wise product of matrices = \n"  << m1.array()*m1.array()                  << "\n";
  std::cout << "matrix product = \n"                    << m1*m1                                  << "\n";
  std::cout << "matrix-vector product = \n"             << m1*v1                                  << "\n";
  std::cout << "Element-wise product of vectors = \n"   << v1*v1                                  << "\n";
  std::cout << "Element-wise division of vectors = \n"  << v1/v1                                  << "\n";
  std::cout << "Inner product of vectors = "            << (v1*v1).sum()                       << "\n";

  // Indexing objects
  std::cout << "Element (1,1) of matrix m1= "           << m1(1,1)                                << "\n";
  std::cout << "2nd row of matrix m1= \n"               << m1.row(1)                              << "\n";
  std::cout << "2nd col of matrix m1= \n"               << m1.col(1)                              << "\n";
  std::cout << "Element (1,1) of array a1= "            << a1(1,1)                                << "\n";
  std::cout << "2nd row of array a1= \n"                << a1.row(1)                              << "\n";
  std::cout << "2nd col of array a1= \n"                << a1.col(1)                              << "\n";
  std::cout << "2nd element of vector v1= "             << v1(1)                                  << "\n";

  // Subsetting matrices and vectors
  std::cout << "First element of v1= "                  << v1.head(1)                             << "\n";
  std::cout << "Last element of v1= "                   << v1.tail(1)                             << "\n";
  std::cout << "Block of m1 consisting of m1(1,1)= "    << m1.block(0,0,1,1)                      << "\n";



  // Generic matrix operations that we must ensure compiles
  m1.transpose();
  m1.diagonal();
  m1.asDiagonal();
```

```
  Type ans;
  return ans;

}
```

10.6 nmix.cpp

```
// nmix example from https://groups.nceas.ucsb.edu/non-linear-modeling/projects/nmix
#include <TMB.hpp>

template<class Type>
Type nll_group(int i, Type p0,Type p1,Type log_lambda, Type log_sigma,
               vector<Type> u,matrix<Type> y,
      vector<Type> N, vector<Type> x,
               matrix<Type> IDind){
  using CppAD::Integer;
  int R=y.rows();
  int T=y.cols();
  int S=N.size();
  Type sigma = exp(log_sigma);
  Type lambda = exp(log_lambda);
  Type e=1e-12;
  Type nll=0;
  int nIDi=u.size();
  vector<Type> logf(S);
  vector<Type> logg(S);
  vector<Type> fg(S);
  vector<Type> tmp=Type(-1.0)*(p0 + p1*x(i) + sigma*u);
  vector<Type> p = Type(1.0)/(Type(1.0)+exp(tmp));
  for(int k=0;k<S;k++){
    logf(k) = log_lambda*N(k) - (lambda + lgamma(N(k)+1));
  }
  Type tmp1,tmp2,tmp3;
  for(int k=0;k<S;k++) {
    logg(k) = 0;
    for(int j=0;j<T;j++) {
      if(N(k)>=y(i,j)){
        tmp1=lgamma(N(k)+1) - lgamma(y(i,j)+1) - lgamma(N(k)-y(i,j)+1);
        tmp2=log(p(Integer(IDind(i,j)))+e)*y(i,j);
        tmp3=log(Type(1.0) + e - p(Integer(IDind(i,j))))*( N(k)-y(i,j));
        logg(k)+=tmp1+tmp2+tmp3;
      }
      else {
        logg(k) = -1000;
      }
    }
  }
  fg=exp(logf+logg);
  nll -= log(e + sum(fg));
  return nll;
}

template<class Type>
Type objective_function<Type>::operator() ()
{
  /* data section */
  DATA_INTEGER(R);              // Number of sites
  DATA_INTEGER(T);              // Number of occasions
  DATA_INTEGER(S);              // Number of possible values of N
  DATA_INTEGER(nG);             // Number of groups
  DATA_VECTOR(N);               // Possible values of N
  DATA_VECTOR(nID);             // Number of observers present at a site
  DATA_FACTOR(ID);              // IDs of observer present at each site
  DATA_MATRIX(IDind);           // Group ID RT matrix
  DATA_FACTOR(IDfac);           // To split ID (instead of ragged matrix)
  DATA_VECTOR(x);               // Site-specific covariate
  DATA_MATRIX(y);               // Count data with R rows and T columns
  /* Parameter section */
  PARAMETER(log_lambda);
  PARAMETER(p0);
  PARAMETER(p1);
  PARAMETER(log_sigma);         // log of random effect SD
  PARAMETER_VECTOR(u);          // Length nG
  vector<vector<int> > idspl=split(ID,IDfac);
  /* Procedure section */
  Type nll=0;
  nll+=Type(.5)*(u*u).sum();
  for(int i=0;i<R;i++){
    nll+=nll_group(i, p0,p1,log_lambda,log_sigma,u(idspl(i)),y,N,x,IDind);
```

```
  }
  return nll;
}
```

10.7 orange_big.cpp

```
// Scaled up version of the Orange Tree example (50,000 latent random variables)
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_INTEGER(n);
  DATA_VECTOR(y);
  DATA_VECTOR(t);
  DATA_INTEGER(M);
  DATA_FACTOR(ngroup);
  DATA_INTEGER(multiply);
  PARAMETER_VECTOR(beta);
  PARAMETER(log_sigma);
  PARAMETER(log_sigma_u);
  PARAMETER_VECTOR(u);

  Type sigma=exp(log_sigma);
  Type sigma_u=exp(log_sigma_u);

  ADREPORT(sigma);
  ADREPORT(sigma_u);

  using namespace density;
  int i,j,k,ii;

  Type g=0;

  for(k=0;k< multiply;k++)
  {
    ii = 0;
    for(i=0;i< M;i++)
    {
      // Random effects contribution
      Type u1 = u[i+k*M];
      g -= -(log_sigma_u);
      g -= -.5*pow(u1/sigma_u,2);

      vector<Type> a(3);
      a[0] = 192.0 + beta[0] + u1;
      a[1] = 726.0 + beta[1];
      a[2] = 356.0 + beta[2];

      Type tmp;
      Type f;

      for(j=0;j<ngroup(i);j++)
      {
        f = a[0]/(1+exp(-(t[ii]-a[1])/a[2]));
        tmp = (y[ii] - f)/sigma;
        g -= -log_sigma - 0.5*tmp*tmp;
        ii++;
      }
    }
  }

  return g;
}
```

10.8 randomregression.cpp

```
// Regression model with random slope and intercept.
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_FACTOR(group);
  DATA_VECTOR(x);
  DATA_VECTOR(t);
  PARAMETER_VECTOR(a);      // Slope for given individual
  PARAMETER_VECTOR(b);      // Intercept for given individual
  PARAMETER_VECTOR(mu);     // length 2 - one for slope, one for intercept
```

```
  PARAMETER_VECTOR(sigma); // length 2 - one for slope, one for intercept
  PARAMETER(sigma0);
  // sigma=abs(sigma);
  // sigma0=abs(sigma0);

  int nobs=x.size();
  int ngroups=a.size();
  Type res(0.0);
  int j;

  /* Prior: slope~N(mu0,sd0), intercept~N(mu1,sd1) */
  for(int j=0;j<ngroups;j++){
    res-=dnorm(a[j],mu[0],sigma[0],1);
    res-=dnorm(b[j],mu[1],sigma[1],1);
  }

  /* Observations: x|a,b ~ N(a*t+b,sigma0) */
  for(int i=0;i<nobs;i++){
    j=group[i];
    res-=dnorm(x[i],a[j]*t[i]+b[j],sigma0,1);
  }
  return res;
}
```

10.9 rw.cpp

```
// Random walk with multivariate correlated increments and measurement noise.
#include <TMB.hpp>

/* Parameter transform */
template <class Type>
Type f(Type x){return Type(2)/(Type(1) + exp(-Type(2) * x)) - Type(1);}

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_ARRAY(obs); /* timeSteps x stateDim */
  PARAMETER_ARRAY(u); /* State */
  PARAMETER(transf_rho);
  PARAMETER_VECTOR(logsds);
  PARAMETER_VECTOR(logsdObs);
  int stateDim=obs.dim[0];
  int timeSteps=obs.dim[1];
  Type rho=f(transf_rho);
  vector<Type> sds=exp(logsds);
  vector<Type> sdObs=exp(logsdObs);
  matrix<Type> cov(stateDim,stateDim);
  for(int i=0;i<stateDim;i++)
    for(int j=0;j<stateDim;j++)
      cov(i,j)=pow(rho,Type(abs(i-j)))*sds[i]*sds[j];
  using namespace density;
  MVNORM_t<Type> neg_log_density(cov);
  Type ans=0;
  ans-=dnorm(vector<Type>(u.col(0)),Type(0),Type(1),1).sum();
  for(int i=1;i<timeSteps;i++){
    ans+=neg_log_density(u.col(i)-u.col(i-1));
    ans-=dnorm(vector<Type>(obs.col(i)),vector<Type>(u.col(i)),sdObs,1).sum();
  }
  ADREPORT(rho*exp(u.col(1)));
  return ans;
}
```

10.10 rw_sparse.cpp

10.11 sdv_multi.cpp

```
// Multivatiate SV model from Skaug and Yu 2013, Comp. Stat & data Analysis (to appear)
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
```

```
  DATA_INTEGER(n);
  DATA_INTEGER(p);
  DATA_ARRAY(y);
  PARAMETER_VECTOR(phi);
  PARAMETER_VECTOR(log_sigma);
  PARAMETER_VECTOR(mu_x);
  PARAMETER_VECTOR(off_diag_x);
  PARAMETER_ARRAY(h);

  int i,j;
  Type g=0;

  vector<Type> sigma=exp(log_sigma);

  // Likelihood contribution: stationary distribution for initial state
  vector<Type> tmp(p);
  for(j=0;j<p;j++)
    tmp(j) = sigma(j)/sqrt(Type(1.0)-phi(j)*phi(j));
  g -= sum(dnorm(vector<Type>(h.col(0)),Type(0),tmp,1));

  // Likelihood contribution: State transitions
  for(i=1;i<n;i++)
  {
    vector<Type> tmp2(p);
    for(j=0;j<p;j++)
      tmp2(j) = phi(j)*h(j,i-1);
    g -= sum(dnorm(vector<Type>(h.col(i)),tmp2,sigma,1));
  }

  // Cholesky factor of Sigma
  matrix<Type> L(p,p);
  L.setIdentity();

  int k=0;
  for(i=1;i<p;i++)
  {
    Type Norm2=L(i,i);
    for(j=0;j<=i-1;j++)
    {
      L(i,j) = off_diag_x(k++);
      Norm2 += L(i,j)*L(i,j);
    }
    for(j=0;j<=i;j++)
      L(i,j) /= sqrt(Norm2);
  }

  matrix<Type> Sigma = L * L.transpose();

  using namespace density;

  // Likelihood contribution: observations
  for(i=0;i<n;i++)
  {
    vector<Type> sigma_y = exp(Type(0.5)*(mu_x + vector<Type>(h.col(i))));

    // Scale up correlation matrix
    matrix<Type> Sigma_y(p,p);
    for(int i2=0;i2<p;i2++)
      for(j=0;j<p;j++)
        Sigma_y(i2,j) = Sigma(i2,j)*sigma_y(i2)*sigma_y(j);

    MVNORM_t<Type> neg_log_density(Sigma_y);
    g += neg_log_density(vector<Type>(y.col(i)));
  }

  return g;
}
```

10.12 simple.cpp

```
// Normal linear mixed model specified through sparse design matrices.
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(x);          // Observations
  DATA_SPARSE_MATRIX(B);   // Random effect design matrix
  DATA_SPARSE_MATRIX(A);   // Fixed effect design matrix
  PARAMETER_VECTOR(u);     // Random effects vector
  PARAMETER_VECTOR(beta); // Fixed effects vector
  PARAMETER(logsdu);       // Random effect standard deviations
  PARAMETER(logsd0);       // Measurement standard deviation
```

```
  // Distribution of random effect (u):
  Type ans=0;
  ans-=dnorm(u, Type(0) /*mean*/, exp(logsdu) /*sd*/, 1 /*log?*/).sum();

  // Distribution of obs given random effects (x|u):
  vector<Type> y=A*beta+B*u;
  ans-=dnorm(x,y,exp(logsd0),1).sum();

  return ans;
}
```

10.13 socatt.cpp

```
// socatt from ADMB example collection.
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{

  DATA_FACTOR(y); //categorical response vector
  DATA_INTEGER(S); //number of response categories
  DATA_MATRIX(X); // Fixed effects design matrix
  DATA_FACTOR(group);
  PARAMETER_VECTOR(b); // Fixed effects
  PARAMETER(logsigma);
  PARAMETER_VECTOR(tmpk); // kappa ( category thresholds)
  PARAMETER_VECTOR(u);    // Random effects
  Type sigma = exp(logsigma);
  vector<Type> alpha = tmpk;
  for(int s=1;s<tmpk.size();s++)
    alpha(s) = alpha(s-1) + exp(tmpk(s));
  Type ans=0;
  ans -= sum(dnorm(u,Type(0),Type(1),true));
  vector<Type> eta = X*b;
  for(int i=0; i<y.size(); i++){
    eta(i) += sigma*u(group(i));
    Type P;
    if(y(i)==(S-1)) P = 1.0; else P = Type(1)/(Type(1)+exp(-(alpha(y(i))-eta(i))));
    if(y(i)>0) P -= Type(1)/(Type(1)+exp(-(alpha(y(i)-1)-eta(i))));
    ans -= log(1.e-20+P);
  }

  return ans;
}
```

10.14 spatial.cpp

```
// Spatial poisson GLMM on a grid, with exponentially decaying correlation function
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_INTEGER(n);
  DATA_VECTOR(y);
  DATA_MATRIX(X)
  DATA_MATRIX(dd)
  PARAMETER_VECTOR(b);
  PARAMETER(a);
  PARAMETER(log_sigma);
  PARAMETER_VECTOR(u);
  Type sigma2=exp(2.0*log_sigma);

  using namespace density;
  int i,j;
  Type res=0;

  vector<Type> eta(n);
  eta = X*b + exp(log_sigma)*u;

  //
  matrix<Type> cov(n,n);
  for (i=0;i<n;i++)
  {
    cov(i,i)=Type(1);
    for ( j=0;j<i;j++)
```

```
    {
      cov(i,j)=exp(-a*dd(i,j));                  // Exponentially decaying correlation
      cov(j,i)=cov(i,j);
    }
  }

  MVNORM_t<Type> neg_log_density(cov);
  res+=neg_log_density(u);

  // logdpois = N log lam - lam
  for(i=0;i<n;i++) res -= y[i]*eta[i]-exp(eta[i]);

  return res;

}
```

10.15 sumtest.cpp

```
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() ()
{
  PARAMETER_VECTOR(x);
  Type res=0;
  for(int i=0;i<x.size();i++)res+=x[i];
  res=res*res;
  return res;
}
```