BFilt

1.0

Generated by Doxygen 1.5.7

Wed Feb 25 14:57:28 2009

Contents

Chapter 1

Main Page

1.1 Description

BFilt is a multi-platform and open-source C++ bayesian filtering library. It contains useful and classical algorithms in state estimation of hidden markov models. So you can easily construct discrete-disrete (DD) and continuous-discrete (CD) models (linear or nonlinear) for filtering (Kalman, EKF, UKF, particle filters, ...) and simulation methods. Indeed, markovian model simulators can be used for particle filters. Libraries such as BFL and Bayes++ consider only discrete-discrete filtering. With BFilt, you can easily construct your own CD or DD models for filtering. For CD models stochastic discretization methods (Euler, Runge Kutta, Local linearization, Heun) are implemented in simulation and filtering.

1.2 Dependances

LAPACK and CPPLAPACK libraries are used for linear algebra operations. For best performances it is recommended to compile yourself the LAPPACK libraries with ATLAS. The Gnu Scientific Library (GSL) achieves random drawing in simulators. These open-source and multi-platform libraries must be installed before install BFilt.

1.3 Installation

Go to the bin directory

```
cd BFilt/bin
```

Run Cmake (>2.6)

```
cmake ../src
```

Compile Bfilt

```
make
```

Install BFilt in /usr/local/lib or /usr/local/inlcude

```
make install
```

If you want to change the default install directory you can type

```
ccmake
```

and change CMAKE_INSTALL_PREFIX

1.4 Auteur

Author:

paul <paul.frogerais@univ-rennes1.fr>

Date:

Fri Sep 12 18:34:36 2008

Chapter 2

An AR process

This is an example on the following auto regressive (AR) process :

$$X_k = 0.8X_{k-1} + Wk$$

where, $X_k \in \mathcal{R}$, $W_k \sim \mathcal{N}(0, 0.1)$

This state is then observed by the output $Y_k \in \mathcal{R}$:

$$Y_k = X_k + V_k$$

where $V_k \sim \mathcal{N}(0, 1)$

2.1 Define the AR model

First the ar process must be define as a sister class of Gaussian_Linear_Model.

```
#ifndef __AR_PROCESS
#define __AR_PROCESS

#include <bfilt/gaussian_model.h>

class AR_Process : public Gaussian_Linear_Model
{
public :
      AR_Process(void);
};


#endif
```

The Gaussian Linear Model are implemented in the following form :

$$X_k = FX_{k-1} + f + GW_k$$

$$Y_k = HX_{k-1} + h + V_k$$

The constructor of AR_Process is then :

```
#include "ar_process.h"

AR_Process::AR_Process(void)
{
      // State Equation
      F.resize(1,1);
      F(0,0) = 0.8;

      f.resize(1);
      f(0) = 0.;

      G.resize(1,1);
      G.identity();

      Qw.resize(1);
      Qw(0,0)= 0.1;

      // Observation noise
      H.resize(1,1);
```

```
    H(0,0) = 1;

    h.resize(1);
    h(0) = 0.;

    Qv.resize(1);
    Qv(0,0)=1;

    // Init state
    X0.resize(1);
    X0(0) = 10.;

    R0.resize(1);
    R0.zero();
}
```

2.2 The main program

In the main program, the model will be first simulted with a specific simulator for gaussian model (G_-Simulator). Then the simulated output sequence $y_{0:N}$ is given to the input of a discrete-discrete kalman filter (DD_Filter) to estimate the state $\hat{X}_{0:k}$. First, all this objects are declared :

```
int main(int argc, char **argv)
{
    AR_Process model;           // The AR model

    G_Simulator sim(&model);      // The simulator

    DD_Kalman  filter(&model);   // The Kalman filter
```

Then 100 samples are simulated :

```
    sim.Simulate(100);
```

The kalman filter is apply on the output sequence :

You can save the simulated sequences :

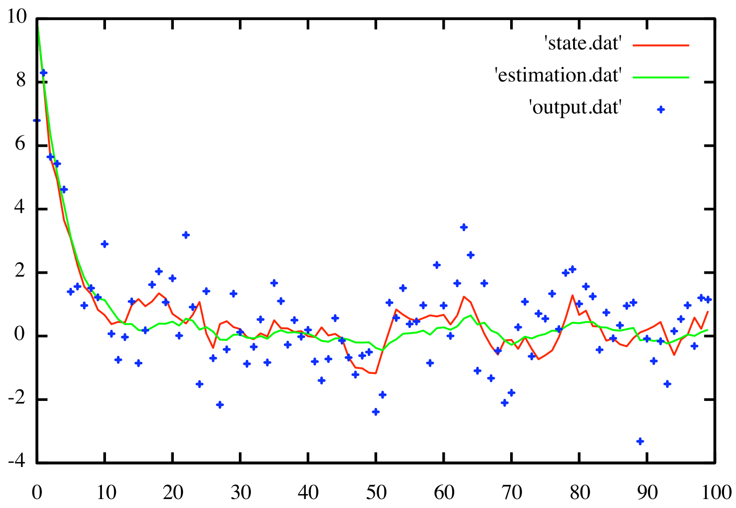
```
    sim.Save_Y("output.dat");
    sim.Save_X("state.dat");
```

and the estimated state :

```
    filter.Save_X("estimation.dat");
```

After compileing and execution, with Gnuplot you can plot :

```
  plot 'state.dat' w l, 'estimation.dat' w l, 'output.dat'
```



To obtain the following graph :

2.3 The CMakeList.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.6)

PROJECT(Van_Der_Pol)
ADD_DEFINITIONS(" -O3")

# GSL
SET(BFILT_LIB bfilt)

# Include et Link Directories

IF(APPLE)
  MESSAGE("-- Apple Configuration")
  INCLUDE_DIRECTORIES(
    /sw/include/
    )
ENDIF(APPLE)

# Executables and "stand-alone " librairies
ADD_EXECUTABLE (Van_Der_Pol
  van_der_pol.cpp
  example_3.cpp
  )

# Linkage
TARGET_LINK_LIBRARIES(Van_Der_Pol
  ${BFILT_LIB}
  )
```

Chapter 3

An Ornstien-Uhlenbeck process

This example illustrate how to use BFilt for continuous-discrete filtering. Here the state is described by the following linear stochastic differential equation :

$$d\begin{pmatrix} x \\ \dot{x} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -w_0^2 & -\gamma \end{pmatrix}\begin{pmatrix} x \\ \dot{x} \end{pmatrix} dt + \begin{pmatrix} 0 \\ b \end{pmatrix} dt + \begin{pmatrix} 0 \\ g \end{pmatrix} dW(t)$$

Where W(t) is a Wiener process,

$w_0^2 = 16, \gamma = 2, b = 8, g = 2$ and the initials conditions $X_0 = (0, 0)$ and $R_0 = diag[0, 3]$.

The state $X(t) = (x, \dot{x})(t)$ is then observed by the output $Y_k \in \mathcal{R}$:

$$Y_k = x(t_k) + V_k$$

at discrete time t_k. The sampling period $T_s = t_{k-1} - t_k = 0.2s$ and $V_k \sim \mathcal{N}(0, 0.001)$. In fact only the position is observed. First this model must be define as a sister class of linear time invariant continuous discrete models (Linear_CD_Model).

```
#ifndef __ORNSTEIN_UHL__
#define __ORNSTEIN_UHL__

#include <bfilt/gaussian_model.h>

class Ornstein_Uhlenbeck_Model : public Linear_CD_Model
{
public :
      double gamma;
      double w;
      double b;
      double g;
      Ornstein_Uhlenbeck_Model(void);


};

#endif
```

The Linear_CD_Model are implemented in the following form :

$$dX(t) = AX(t)dt + Bdt + CdW(t)$$

$$Y_k = HX(t_k) + h + V_k$$

The constructor of Ornstein_Uhlenbeck_Model is then :

```
#include "ornstein_uhlenbeck.h"

Ornstein_Uhlenbeck_Model::Ornstein_Uhlenbeck_Model(void)
{
      // parameters
      w = 4.;
      gamma = 2.;
      b = 8.;
      g = 2.;

      // Matrices of the state equation
      A.resize(2,2);
      A(0,0) = 0.;
      A(0,1) = 1.;
      A(1,0) = - (w*w);
      A(1,1) = -gamma;
```

```
    B.resize(2);
    B(0) = 0;
    B(1) = b;

    C.resize(2,1);
    C(0,0)=0.;
    C(1,0)=g;

    // Matrices of the Observation equation
    H.resize(1,2);
    H(0,0) = 1.;
    H(0,1) = 0.;

    h.resize(2);
    h.zero();

    Qw.resize(1);
    Qw.identity();
    Qw*=0.01;

    Qv.resize(1);
    Qv(0,0) = 0.001;

    // Sampling period
    Ts = 0.2;

    // Initial conditions
    R0.resize(2);
    R0.zero();
    R0(0,0)=0.;
    R0(1,1)=3.;

    X0.resize(2);
    X0.zero();

}
```

3.1 The main program

In the main program, the model will be first simulted with a specific simulator for Linear_CD_Model (LTI_CD_Simulator). The simulated output sequence $y_{0:N}$ is given to the input of the continuous-discrete kalman filter (CD_Filter) to estimate the state trajectory $\hat{X}_{0:k}$. First, all this objects are declared :

```
int main(int argc, char **argv)
{
    Ornstein_Uhlenbeck_Model model; // The model

    LTI_CD_Simulator sim(&model);   // The simulator

    CD_Kalman  filter(&model);      // The Kalman filter
```

Then 10 second are simulated :

```
    sim.Simulate(10.);
```

The kalman filter is apply on the output sequence :

You can save the simulated sequences :

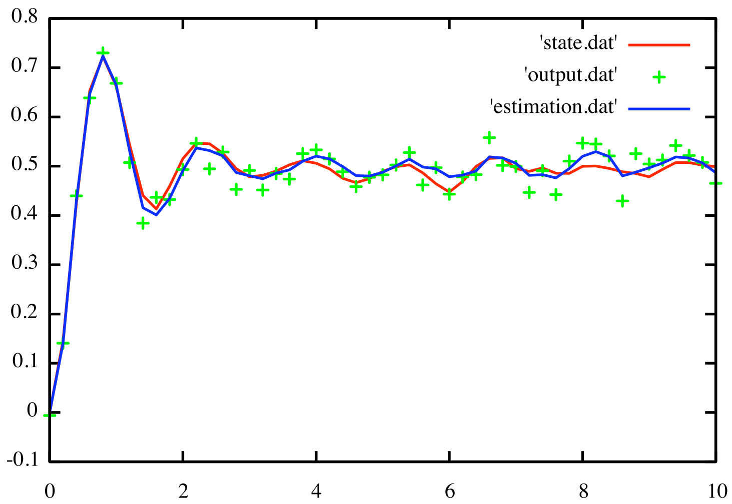
```
sim.Save_Y("output.dat");
sim.Save_X("state.dat");
```

and the estimated state :

```
filter.Save_X("estimation.dat");
```

After compileing and execution, with Gnuplot you can plot :

```
plot 'state.dat' w l, 'estimation.dat' w l, 'output.dat'
```



To obtain the following graph :

3.2 The CMakeList.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.6)

PROJECT(Van_Der_Pol)
ADD_DEFINITIONS(" -O3")

# GSL
SET(BFILT_LIB bfilt)

# Include et Link Directories

IF(APPLE)
  MESSAGE("-- Apple Configuration")
  INCLUDE_DIRECTORIES(
    /sw/include/
    )
ENDIF(APPLE)

# Executables and "stand-alone " librairies
ADD_EXECUTABLE (Van_Der_Pol
  van_der_pol.cpp
  example_3.cpp
  )
```

```
# Linkage
TARGET_LINK_LIBRARIES(Van_Der_Pol
  ${BFILT_LIB}
  )
```

Chapter 4

Van Der Pol oscillator

This example on the Van der Pol oscillator shows how to use BFilt for non-linear continuous-discrete model. Here the van_der_pol class :

van_der_pol.h

```
#ifndef __VAN_DER_POL
#define __VAN_DER_POL


#include <bfilt/gaussian_model.h>


class Van_Der_Pol : public Continuous_Discrete_Model
{
public :
      double lambda;

      Van_Der_Pol(void);
      dcovector Drift_Function(const dcovector & X);
      dgematrix J_Drift_Function(const dcovector & X);
      dcovector Observation_Function(const dcovector& X);
      dgematrix J_Observation_Function(const dcovector & X);
      dgematrix Diffusion_Function(void);
};


#endif
```

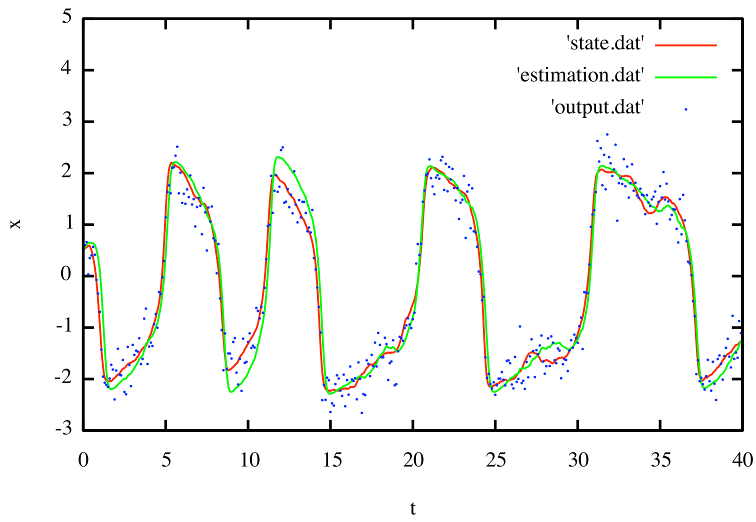
van_der_pol.cpp

```
#include "van_der_pol.h"

Van_Der_Pol::Van_Der_Pol(void)
{
      lambda = 3.;

      Qw.resize(1);
      Qw(0,0)= 1.;
      Qv.resize(1);
      Qv(0,0)=0.1;

      X0.resize(2);
      X0(0) = 0.5;
      X0(1) = 0.5;
      R0.resize(2);
      R0.zero();
      R0(0,0)=0.;
      R0(1,1)=.1;
      Ts=.1;

}


dcovector Van_Der_Pol::Drift_Function(const dcovector & X)
{
      dcovector dX(X.l);

      dX(0) = X(1);
      dX(1) = lambda * (1. - X(0) * X(0)) * X(1) - X(0);

      return dX;
}

dgematrix Van_Der_Pol::J_Drift_Function(const dcovector & X)
{
      dgematrix F(X.l,X.l);
```

```
	F(0,0) = 0.;
	F(0,1) = 1.;
	F(1,0) = -2. * lambda * X(0) * X(1);
	F(1,1) = - lambda * X(0) * X(0);

	return F;
}
dcovector Van_Der_Pol::Observation_Function(const dcovector& X)
{
	dcovector Y(1);
	Y(0) = X(0);
	return Y;
}

dgematrix Van_Der_Pol::J_Observation_Function(const dcovector & X)
{
	dgematrix H(1,2);
	H(0,0) = 0.;
	H(0,1) = 1.;
	return H;
}
dgematrix Van_Der_Pol::Diffusion_Function(void)
{
	dgematrix G(2,1);
	G(0,0) = 0.;
	G(1,0) = 1.;

	return G;
}
```

The main program :

```
#include <bfilt/simulator.h>
#include <bfilt/extended_kalman_filter.h>
#include "van_der_pol.h"



int main(int argc, char **argv)
{
	Van_Der_Pol model;              // The model

	CD_Simulator sim(&model);   // The simulator

	CD_Extended_Kalman_Filter  filter(&model,THGL);      // The filter

	// Simulation 40 seconds
	sim.Simulate(40.);


	// Filtering from the simulated output sim.Y
	filter.Filtering(sim.Y);

	// Output Files for simulation
	sim.Save_Y("output.dat");
	sim.Save_X("state.dat");

	// Output File for filtering
	filter.Save_X("estimation.dat");

	return 0;
}
```

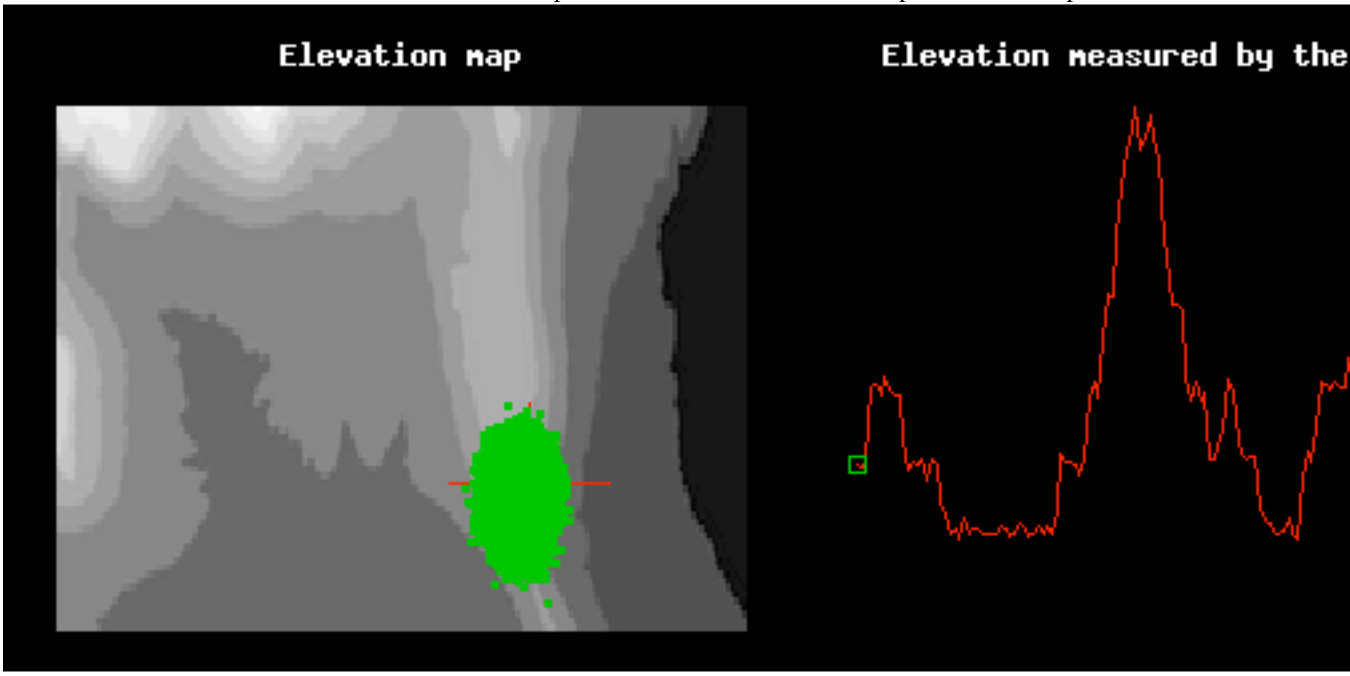


Results can be plotted (here with gnuplot):

Chapter 5

Terrain navigation

This example illustrate performances of particle filter to highly non-linear filter. The promblem here involves a plane whose the trajectory is a brownian motion. This aircraft measure the elevation. The measure of this elevation and an elevation map are then used to estimate the position of the plane.



plane.h

```
#ifndef __PLANE
#define __PLANE

#include <bfilt/gaussian_model.h>

class Plane : public Gaussian_Nonlinear_Model
{
      vector<double> Map;
      double xmin;
      double xmax;
      double ymin;
      double ymax;

      double sigv;
      double sigc;
public :
      Plane(const char *filename);

      dcovector State_Function(const dcovector &X, const dcovector &W);
      dcovector Observation_Function(const dcovector & X);
};


#endif
```

plane.cpp

```
#include "plane.h"

Plane::Plane(const char * filename)
{
```

```
        int x,y,z;
        ifstream file(filename);

        if(file)
                {
                                file>>xmax;
                                file>>ymin;
                                file>>z;
                                Map.push_back(z);

                        while(!file.eof())
                                {
                                file>>x;
                                file>>y;
                                file>>z;
                                Map.push_back(z);

                                if(x>xmax)
                                        xmax=x;
                                if(x<xmin)
                                        xmin=x;
                                if(y>ymax)
                                        ymax=y;
                                if(y<ymin)
                                        ymin=y;

                                }
                        file.close();
                }
        else
                {
                        cout<<"Plane :: error file"<<endl;
                }

        Qw.resize(2);
        Qw.identity();
        Qv.resize(1);
        Qv.identity();
        Qv*=5.;
        R0.resize(4);
        R0.zero();
        R0(0,0)=10.;
        R0(1,1)=10.;
        R0(2,2) = .001;
        R0(3,3) = 0.0001;
        X0.resize(4);
        X0(0)=120.;
        X0(1)=20.;
        X0(2)=1.5;
        X0(3)=2.35;


        sigv = .001;
        sigc = 0.03;

}

dcovector Plane::State_Function(const dcovector &X, const dcovector &W)
{
        dcovector U(4);

        U(0) = X(0) + X(2) * cos(X(3));
        U(1) = X(1) + X(2) * sin(X(3));
        U(2) = X(2) + sigv * W(0);
        U(3) = X(3) + sigc * W(1);
        if (U(0)>xmax)
                {
```

```
                  U(0)=xmax;
                  U(3)=3.14-U(3);
            }
      if (U(1)>ymax)
            {
                  U(1)=ymax;
                  U(3)=-U(3);
            }
      if (U(0)<xmin)
            {
                  U(0)=xmin;
                  U(3)=3.14-U(3);
            }
      if (U(1)<ymin)
            {
                  U(1)=ymin;
                  U(3)=-U(3);
            }

      return U;
}
dcovector Plane::Observation_Function(const dcovector & X)
{
      int x = (int)(X(0));
      int y = (int)(X(1));
      int j= (xmax+1)*y + x;
      dcovector Y(1);
      Y(0)=Map[j];
      return Y;
}
```

The main program :

```
#include <bfilt/simulator.h>
#include <bfilt/sisr_filter.h>
#include "plane.h"

// This example illustrate performances of particle filter to highly non-linear
// filter. The promblem here involves a plane whose the trajectory is a brownian
// motion. This aircraft measure the elevation. The measure of this elevation and
// an elevation map are then used to estimate the position of the plane.
int main(int argc, char **argv)
{
      int k;
      int i;
      int j;
      vector<Weighted_Sample> cloud;

      ofstream file_c("../data/cloud.dat"); // To save the cloud
      ofstream file_s("../data/state.dat"); // To save the state

      Plane       plane("../data/map_2.dat");       // The plane model
      G_Simulator sim(&plane);                    // To simulate a this model

      Bootstrap_Filter filter(100000,&plane);       // A bootstrap filter to estimate the position

      sim.Simulate(150);                      // simulation of 250 samples

      sim.Save_Y("../data/output.dat");     // save the output

      // Here Init() and Update methods are used
      // for filtering because we want to get the cloud
      // at each step and save it in cloud.dat

      filter.Init();                       // Initialization of the boostrap filter
```

```
      for (k=0; k<sim.Y.size(); k++)
            {

                  cloud=filter.CloudGet();              // The current cloud is return

                  for(i=0; i<cloud.size(); i++)         // and here it is saved
                        {
                              for(j=0; j<cloud[i].Value.l; j++)
                                    file_c<<cloud[i].Value(j)<<" ";
                              file_c<<endl;
                        }

                  for (j=0; j<sim.X[k].l; j++)       // The state is also saved
                        file_s<<sim.X[k](j)<<" ";
                  file_s<<endl<<endl<<endl;
                  file_c<<endl<<endl;

                  if( filter.Update(sim.Y[k]))              // The filter is then update with a new observ
                        filter.Init();
            }
      file_c.close();
      file_s.close();

      return 0;
}
```

Chapter 6

Class Index

6.1 Class Hierarchy

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

Chapter 7

Class Index

7.1 Class List

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

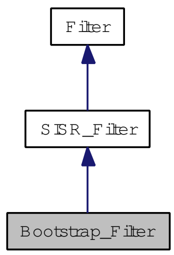
Chapter 8

Class Documentation

8.1 Bootstrap_Filter Class Reference

`#include <sisr_filter.h>`

Inheritance diagram for Bootstrap_Filter:



Public Member Functions

- Bootstrap_Filter (void)
- ∼Bootstrap_Filter (void)
- Bootstrap_Filter (const int &Ns, Simulator ∗s)
- Bootstrap_Filter (const int &Ns, Gaussian_Nonlinear_Model ∗m)

Private Attributes

- Simulator ∗ sim

8.1.1 Constructor & Destructor Documentation

8.1.1.1 Bootstrap_Filter::Bootstrap_Filter (void)

8.1.1.2 Bootstrap_Filter::∼Bootstrap_Filter (void)

8.1.1.3 Bootstrap_Filter::Bootstrap_Filter (const int & *Ns*, Simulator ∗ *s*)

8.1.1.4 Bootstrap_Filter::Bootstrap_Filter (const int & *Ns*, Gaussian_Nonlinear_Model ∗ *m*)

8.1.2 Member Data Documentation

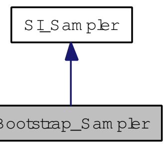
8.1.2.1 Simulator∗ Bootstrap_Filter::sim `[private]`

8.2 Bootstrap_Sampler Class Reference

This sampler use the transition as importance density.

```
#include <sisr_filter.h>
```

Inheritance diagram for Bootstrap_Sampler:



Public Member Functions

- Bootstrap_Sampler (void)
- Bootstrap_Sampler (Simulator ∗m)
- vector< Weighted_Sample > DrawInitCloud (const int &NbSample)

    draw a set of possible init state

- vector< Weighted_Sample > Draw (const dcovector &Y_k, const vector< Weighted_Sample > &X_km1)

    Draw a set of samples from the importance density Xk given Y0:k X0:k-1.

- long double Weight (vector< Weighted_Sample > &cloud, const dcovector &Y_k, const vector< Weighted_Sample > &X_k)

    Modify the weights of cloud for the weighting step in the sisr.

8.2.1 Detailed Description

This sampler use the transition as importance density.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 Bootstrap_Sampler::Bootstrap_Sampler (void)

8.2.2.2 Bootstrap_Sampler::Bootstrap_Sampler (Simulator ∗ *m*)

8.2.3 Member Function Documentation

8.2.3.1 vector<Weighted_Sample > Bootstrap_Sampler::Draw (const dcovector & *Y_k*, const vector< Weighted_Sample > & *X_km1*) [virtual]

Draw a set of samples from the importance density Xk given Y0:k X0:k-1.

Parameters:

Y_k The observation from 0 to k

X_km1 The cloud from 0 to km1

Returns:

A cloud representing the importance density q(Xk|Y0:k,X0:k-1)

Implements SI_Sampler.

8.2.3.2 vector< Weighted_Sample > Bootstrap_Sampler::DrawInitCloud (const int & *NbSample*) [virtual]

draw a set of possible init state

Parameters:

NbSample Number of sample

Returns:

A set of weighted samples

Implements SI_Sampler.

8.2.3.3 long double Bootstrap_Sampler::Weight (vector< Weighted_Sample > & *cloud*, const dcovector & *Y_k*, const vector< Weighted_Sample > & *X_km1*) [virtual]

Modify the weights of cloud for the weighting step in the sisr.

Parameters:

cloud The curent coud at k

Y_k The observation at k

X_km1 the cloud from at km1

Returns:

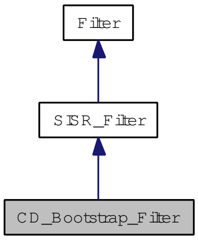
The sum of the weights

Implements SI_Sampler.

8.3 CD_Bootstrap_Filter Class Reference

`#include <sisr_filter.h>`

Inheritance diagram for CD_Bootstrap_Filter:



Public Member Functions

- CD_Bootstrap_Filter (void)
- ~CD_Bootstrap_Filter (void)
- CD_Bootstrap_Filter (const int &Ns, CD_Simulator *s)
- CD_Bootstrap_Filter (const int &Ns, Continuous_Discrete_Model *m)
- CD_Bootstrap_Filter (const int &Ns, Linear_CD_Model *m)
- virtual int Save_X (const char *filename)

Private Attributes

- CD_Simulator * sim

8.3.1 Constructor & Destructor Documentation

8.3.1.1 CD_Bootstrap_Filter::CD_Bootstrap_Filter (void)

8.3.1.2 CD_Bootstrap_Filter::~CD_Bootstrap_Filter (void)

8.3.1.3 CD_Bootstrap_Filter::CD_Bootstrap_Filter (const int & *Ns*, CD_Simulator * *s*)

8.3.1.4 CD_Bootstrap_Filter::CD_Bootstrap_Filter (const int & *Ns*, Continuous_Discrete_Model * *m*)

8.3.1.5 CD_Bootstrap_Filter::CD_Bootstrap_Filter (const int & *Ns*, Linear_CD_Model * *m*)

8.3.2 Member Function Documentation

8.3.2.1 virtual int CD_Bootstrap_Filter::Save_X (const char * *filename*) `[virtual]`

Save the estimation $\{\hat{X}_{k|k}, k = 0, ...N\}$

Parameters:

filename

Returns:

0 if everything is ok

Reimplemented from Filter.

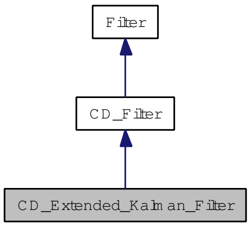
8.3.3 Member Data Documentation

8.3.3.1 CD_Simulator* CD_Bootstrap_Filter::sim [private]

8.4 CD_Extended_Kalman_Filter Class Reference

`#include <extended_kalman_filter.h>`

Inheritance diagram for CD_Extended_Kalman_Filter:



Public Member Functions

- CD_Extended_Kalman_Filter (void)
- CD_Extended_Kalman_Filter (Continuous_Discrete_Model ∗m, const int &sh=RK4)

Public Attributes

- int Scheme

Protected Member Functions

- int _update (const dcovector &Y)
- void _thgl__prediction (dcovector &M, dgematrix &P)
- void _euler_prediction (dcovector &M, dgematrix &P)
- void _rk4___prediction (dcovector &M, dgematrix &P)
- void _heun__prediction (dcovector &M, dgematrix &P)
- void _rk4___prediction_FM (dcovector &M, dgematrix &P)

8.4.1 Constructor & Destructor Documentation

8.4.1.1 CD_Extended_Kalman_Filter::CD_Extended_Kalman_Filter (void)

8.4.1.2 CD_Extended_Kalman_Filter::CD_Extended_Kalman_Filter (Continuous_Discrete_Model ∗ *m*, const int & *sh* = `RK4`)

8.4.2 Member Function Documentation

8.4.2.1 void CD_Extended_Kalman_Filter::_euler_prediction (dcovector & *M*, dgematrix & *P*) `[protected]`

8.4.2.2 void CD_Extended_Kalman_Filter::_heun__prediction (dcovector & *M*, dgematrix & *P*) `[protected]`

8.4.2.3 void CD_Extended_Kalman_Filter::_rk4___prediction (dcovector & *M*, dgematrix & *P*) `[protected]`

8.4.2.4 void CD_Extended_Kalman_Filter::_rk4___prediction_FM (dcovector & *M*, dgematrix & *P*) `[protected]`

8.4.2.5 void CD_Extended_Kalman_Filter::_thgl__prediction (dcovector & *M*, dgematrix & *P*) `[protected]`

8.4.2.6 int CD_Extended_Kalman_Filter::_update (const dcovector & *Y*) `[protected, virtual]`

Specific update for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

Implements Filter.

8.4.3 Member Data Documentation

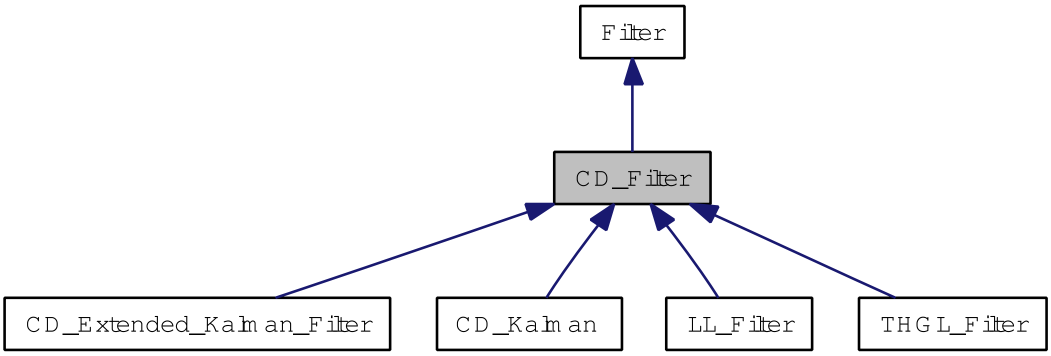
8.4.3.1 int CD_Extended_Kalman_Filter::Scheme

8.5 CD_Filter Class Reference

Abstract class of continuous-discrete filters.

`#include <filter.h>`

Inheritance diagram for CD_Filter:



Public Member Functions

- CD_Filter (void)

  A constructor.

- CD_Filter (Continuous_Discrete_Model ∗m)
- int Save_X (const char ∗filename)
- dcovector Expected_Get (void)

Public Attributes

- dcovector M

  The current mean $\hat{X}_{k|k} = E[X_k|Y_{0:k}]$.

- dgematrix R

  The current covariance $\hat{P}_{k|k} = E[(X_k - \hat{X}_{k|k})(X_k - \hat{X}_{k|k})]$.

- dcovector Xp

  The prediction $\hat{X}_{k-1|k} = E[X_{k-1}|Y_{0:k}]$.

- dgematrix Rp

  The prediction covariance $\hat{P}_{k-1|k} = E[(X_k - \hat{X}_{k-1|k})(X_k - \hat{X}_{k-1|k})]$.

Protected Member Functions

- int _init (void)

8.5.1 Detailed Description

Abstract class of continuous-discrete filters.

For continusous-discrete models (Continuous_Discrete_Model), these filters approximate the probability density of the state transition $p_{X(t_k)|X(t_{k-1})}$ and the probability of the observation $p_{Y_k|X(t_k)}$ by gaussian densities. The approximation is exact in the case of linear continous-discrete models (Linear_CD_-Model) and lead to the continuous-discrete Kalman Filter (CD_Kalman). For other non-linear models (Continuous_Discrete_Model) Local linearization filter (LL_Filter) or continous-discrete Filter EKF (CD_-Extended_Kalman_Filter) can be used.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 CD_Filter::CD_Filter (void)

A constructor.

8.5.2.2 CD_Filter::CD_Filter (Continuous_Discrete_Model $*$ *m*)

A constructor

Parameters:

m A discrete-discrete gaussian non-linear model

8.5.3 Member Function Documentation

8.5.3.1 int CD_Filter::_init (void) `[protected, virtual]`

Specific init for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

Implements Filter.

8.5.3.2 dcovector CD_Filter::Expected_Get (void) `[virtual]`

Get the current estimation $\hat{X}_{k|k}$

Returns:

$\hat{X}_{k|k}$

Implements Filter.

8.5.3.3 int CD_Filter::Save_X (const char * *filename*) [virtual]

Save the estimation $\{\hat{X}_{k|k}, k = 0, ...N\}$

Parameters:

filename

Returns:

0 if everything is ok

Reimplemented from Filter.

8.5.4 Member Data Documentation

8.5.4.1 dcovector CD_Filter::M

The current mean $\hat{X}_{k|k} = E[X_k|Y_{0:k}]$.

8.5.4.2 dgematrix CD_Filter::R

The current covariance $\hat{P}_{k|k} = E[(X_k - \hat{X}_{k|k})(X_k - \hat{X}_{k|k})]$.

8.5.4.3 dgematrix CD_Filter::Rp

The prediction covariance $\hat{P}_{k-1|k} = E[(X_k - \hat{X}_{k-1|k})(X_k - \hat{X}_{k-1|k})]$.

8.5.4.4 dcovector CD_Filter::Xp

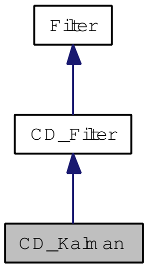
The prediction $\hat{X}_{k-1|k} = E[X_{k-1}|Y_{0:k}]$.

8.6 CD_Kalman Class Reference

The continuous-discrete kalman filter.

`#include <filter.h>`

Inheritance diagram for CD_Kalman:



Public Member Functions

- CD_Kalman (void)
- CD_Kalman (Linear_CD_Model *m)

Protected Member Functions

- int _update (const dcovector &Y)

8.6.1 Detailed Description

The continuous-discrete kalman filter.

Give an exact solution of $\hat{X}_{k|k}$ and $\hat{P}_{k|k}$ for continuous-discrete linear models (Linear_CD_Model).

8.6.2 Constructor & Destructor Documentation

8.6.2.1 CD_Kalman::CD_Kalman (void)

8.6.2.2 CD_Kalman::CD_Kalman (Linear_CD_Model * *m*)

A constructor

Parameters:

m The continuous discrete model

8.6.3 Member Function Documentation

8.6.3.1 int CD_Kalman::_update (const dcovector & *Y*) `[protected, virtual]`

Specific update for each filter

Parameters:

Y The observed sample

Returns:

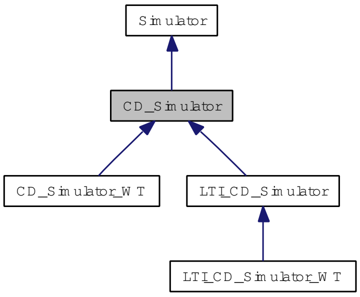
0 if no problem

Implements Filter.

8.7 CD_Simulator Class Reference

`#include <simulator.h>`

Inheritance diagram for CD_Simulator:



Public Member Functions

- CD_Simulator (void)
- CD_Simulator (Continuous_Discrete_Model ∗cd_m, const int &scheme=SRK4, const int &apha=10)
- virtual dcovector Draw_Init (void)

  Draw a sample from p(X0).

- dcovector Draw_Transition (const dcovector &Xkm1)

  Draw a sample from the transition densisty p(Xk|Xk-1).

- dcovector Draw_Observation (const dcovector &Xk)

  Calculate the value of the density of probability of Y given X : p(Y|X).

- long double Observation_Density (const dcovector &Y, const dcovector &X)

  calculate the value of the density of probability of Y given X : p(Y|X)

- int Save_X (const char ∗filename)

  Save the simulated state trajectory in filename.

- int Save_Y (const char ∗filename)

  Save the simulated observation trajectory in filename.

- int Simulate (const double &T)
- void Set_Alpha (const int &al)

Public Attributes

- double Dy
- double Dx

Protected Member Functions

- virtual dcovector draw_state (const dcovector &X)
- void _update (void)

Protected Attributes

- int scheme
- int _a

8.7.1 Constructor & Destructor Documentation

8.7.1.1 CD_Simulator::CD_Simulator (void)

8.7.1.2 CD_Simulator::CD_Simulator (Continuous_Discrete_Model * *cd_m*, const int & *scheme* = `SRK4`, const int & *apha* = `10`)

8.7.2 Member Function Documentation

8.7.2.1 void CD_Simulator::_update (void) `[protected, virtual]`

Reimplemented from Simulator.

8.7.2.2 virtual dcovector CD_Simulator::Draw_Init (void) `[virtual]`

Draw a sample from p(X0).

Returns:

A sample from p(X0)

Implements Simulator.

Reimplemented in CD_Simulator_WT, and LTI_CD_Simulator_WT.

8.7.2.3 dcovector CD_Simulator::Draw_Observation (const dcovector & *Xk*) `[virtual]`

Calculate the value of the density of probability of Y given X : p(Y|X).

Parameters:

Xk The state at k

Returns:

The simulated observation

Implements Simulator.

8.7.2.4 virtual dcovector CD_Simulator::draw_state (const dcovector & *X*) `[protected, virtual]`

Reimplemented in LTI_CD_Simulator.

8.7.2.5 dcovector CD_Simulator::Draw_Transition (const dcovector & *Xkm1*) [virtual]

Draw a sample from the transition densisty p(Xk|Xk-1).

Parameters:

Xkm1 X(k-1) the preceding state

Returns:

Implements Simulator.

8.7.2.6 long double CD_Simulator::Observation_Density (const dcovector & *Y*, const dcovector & *X*) [virtual]

calculate the value of the density of probability of Y given X : p(Y|X)

Parameters:

Y The osbervation

X The state

Returns:

The value of the density

Implements Simulator.

8.7.2.7 int CD_Simulator::Save_X (const char * *filename*) [virtual]

Save the simulated state trajectory in filename.

Parameters:

filename The file

Returns:

0 if it's ok

Reimplemented from Simulator.

8.7.2.8 int CD_Simulator::Save_Y (const char * *filename*) [virtual]

Save the simulated observation trajectory in filename.

Parameters:

filename The file

Returns:

0 if it's ok

Reimplemented from Simulator.

8.7.2.9 void CD_Simulator::Set_Alpha (const int & *al*)

8.7.2.10 int CD_Simulator::Simulate (const double & *T*)

8.7.3 Member Data Documentation

8.7.3.1 int CD_Simulator::_a `[protected]`

8.7.3.2 double CD_Simulator::Dx

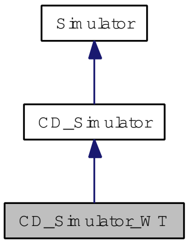
8.7.3.3 double CD_Simulator::Dy

8.7.3.4 int CD_Simulator::scheme `[protected]`

8.8 CD_Simulator_WT Class Reference

`#include <simulator.h>`

Inheritance diagram for CD_Simulator_WT:



Public Member Functions

- CD_Simulator_WT (void)
- CD_Simulator_WT (Continuous_Discrete_Model ∗cd_m, const int &scheme, const int &apha, const double &tb, const double &t)
- dcovector Draw_Init (void)

    Draw a sample from p(X0).

Private Attributes

- vector< dcovector > Xt
- double TB
- double T

8.8.1 Constructor & Destructor Documentation

8.8.1.1 CD_Simulator_WT::CD_Simulator_WT (void)

8.8.1.2 CD_Simulator_WT::CD_Simulator_WT (Continuous_Discrete_Model ∗ *cd_m*, const int & *scheme*, const int & *apha*, const double & *tb*, const double & *t*)

8.8.2 Member Function Documentation

8.8.2.1 dcovector CD_Simulator_WT::Draw_Init (void) `[virtual]`

Draw a sample from p(X0).

Returns:

A sample from p(X0)

Reimplemented from CD_Simulator.

8.8.3 Member Data Documentation

8.8.3.1 double CD_Simulator_WT::T [private]

8.8.3.2 double CD_Simulator_WT::TB [private]

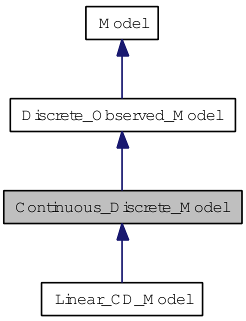
8.8.3.3 vector<dcovector> CD_Simulator_WT::Xt [private]

8.9 Continuous_Discrete_Model Class Reference

Continuous Discrete Model: The continuous state : $dX(t) = F(X)dt + G()*d\beta$ The discrete Observation $Yk = H(X(tk)) + Vk$.

`#include <gaussian_model.h>`

Inheritance diagram for Continuous_Discrete_Model:



Public Member Functions

- Continuous_Discrete_Model (void)
- virtual ~Continuous_Discrete_Model (void)
- virtual dcovector Drift_Function (const dcovector &X)=0

  *the drift function of dX(t) = F(X)dt + G(X)*dW*

- virtual dgematrix J_Drift_Function (const dcovector &X)

  the jacobian of the drift function evaluate at X

- virtual dgematrix Diffusion_Function (void)=0

  the diffusion function

- virtual void Init (void)

Public Attributes

- double Ts

  The sampling periode Ts=tk - tk-1.

8.9.1 Detailed Description

Continuous Discrete Model: The continuous state : $dX(t) = F(X)dt + G()*d\beta$ The discrete Observation $Yk = H(X(tk)) + Vk$.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 Continuous_Discrete_Model::Continuous_Discrete_Model (void)

The constructor

8.9.2.2 virtual Continuous_Discrete_Model::∼Continuous_Discrete_Model (void) `[virtual]`

The destructor

8.9.3 Member Function Documentation

8.9.3.1 virtual dgematrix Continuous_Discrete_Model::Diffusion_Function (void) `[pure virtual]`

the diffusion function

Parameters:

X the state

Returns:

G(X).

Implemented in Linear_CD_Model.

8.9.3.2 virtual dcovector Continuous_Discrete_Model::Drift_Function (const dcovector & *X*) `[pure virtual]`

the drift function of dX(t) = F(X)dt + G(X)∗dW

Parameters:

X the state.

Returns:

f(X)

Implemented in Linear_CD_Model.

8.9.3.3 virtual void Continuous_Discrete_Model::Init (void) `[inline, virtual]`

Initialized CD model

Reimplemented in Linear_CD_Model.

8.9.3.4 virtual dgematrix Continuous_Discrete_Model::J_Drift_Function (const dcovector & *X*) `[virtual]`

the jacobian of the drift function evaluate at X

Parameters:

X

Returns:

the jacobian matrix

Reimplemented in Linear_CD_Model.

8.9.4 Member Data Documentation

8.9.4.1 double Continuous_Discrete_Model::Ts

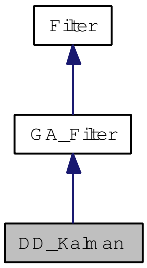
The sampling periode Ts=tk - tk-1.

8.10 DD_Kalman Class Reference

The discrete-discrete kalman filter.

`#include <filter.h>`

Inheritance diagram for DD_Kalman:



Public Member Functions

- DD_Kalman (void)
- DD_Kalman (Gaussian_Linear_Model ∗m)

Protected Member Functions

- int _update (const dcovector &Y)

8.10.1 Detailed Description

The discrete-discrete kalman filter.

Give an exact solution of $\hat{X}_{k|k}$ and $\hat{P}_{k|k}$ for discrete-discrete linear models (Gaussian_Linear_Model).

8.10.2 Constructor & Destructor Documentation

8.10.2.1 DD_Kalman::DD_Kalman (void)

8.10.2.2 DD_Kalman::DD_Kalman (Gaussian_Linear_Model ∗ *m*)

A constructor

Parameters:

m The discrete-discrete model

8.10.3 Member Function Documentation

8.10.3.1 int DD_Kalman::_update (const dcovector & *Y*) `[protected, virtual]`

Specific update for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

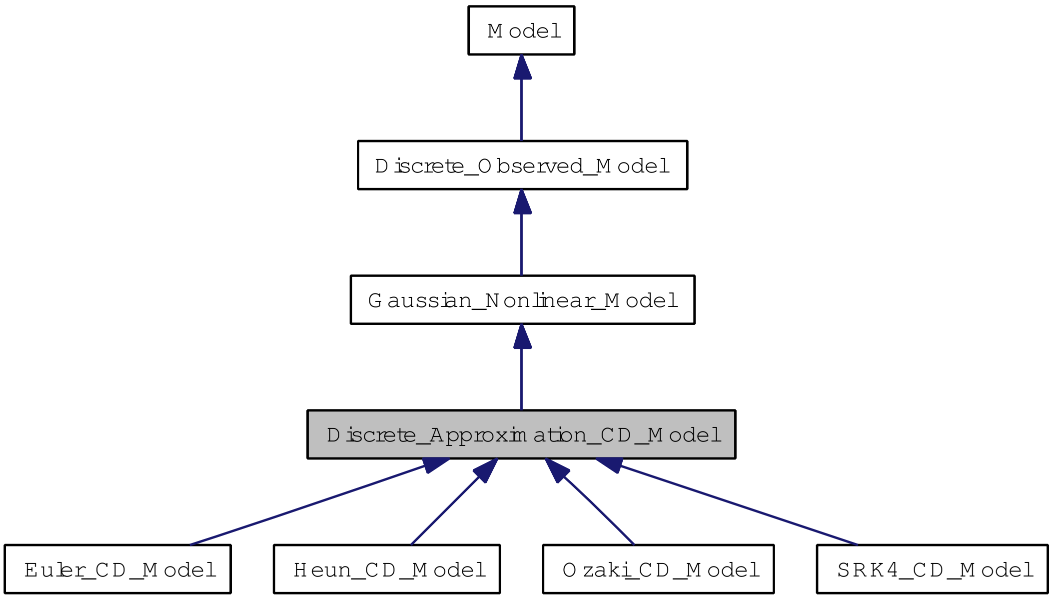
Implements Filter.

8.11 Discrete_Approximation_CD_Model Class Reference

the continuous state equation is discretly approximate by X(tk) = f'(X(tk-1),Wk)

`#include <gaussian_model.h>`

Inheritance diagram for Discrete_Approximation_CD_Model:



Public Member Functions

- Discrete_Approximation_CD_Model (void)
- virtual ∼Discrete_Approximation_CD_Model (void)
- Discrete_Approximation_CD_Model (Continuous_Discrete_Model ∗m)
- Discrete_Approximation_CD_Model (Continuous_Discrete_Model ∗m, const int &a)

  the constructor

- dcovector State_Function (const dcovector &X, const dcovector &W)

  The state Xk=F(Xk-1,Wk).

- dgematrix Jx_State_Function (const dcovector &X, const dcovector &W)

  the X jacobian of the State function evaluate at X,W

- void Get_Linear_Parameters (const dcovector &X, const dcovector &W, dgematrix &F, dgematrix &G, dcovector &Xp)

  computed linearized parameter for EKF in X,W

- virtual void Get_Linear_Scheme (const dcovector &X, const dcovector &W, dgematrix &F, dgematrix &J, dcovector &Xp)=0

  Get the Linearized parameters Scheme in X,W.

- dgematrix Jw_State_Function (const dcovector &X, const dcovector &W)

the W jacobian of the State function evaluate at X,W

- dcovector Observation_Function (const dcovector &X)

  The observation Yk=H(Xk) + Vk.

- virtual dgematrix J_Observation_Function (const dcovector &X)

  the jacobian of the observation function evaluate at X

- virtual dcovector Scheme (const dcovector &X, const dcovector &W)=0
- virtual dgematrix Jx_Scheme (const dcovector &X, const dcovector &W)=0
- virtual dgematrix Jw_Scheme (const dcovector &X, const dcovector &W)=0
- virtual void Init (void)

  Init The model if needed.

- void Set_Alpha (const int &a)
- int Get_Alpha (void)

Protected Attributes

- Continuous_Discrete_Model ∗ cd_model

  the continuous discrete model

- int alpha

  the resolution of the discrete step Td = Ts ∗ a (Ts = sample duration of discrete observation)

8.11.1 Detailed Description

the continuous state equation is discretly approximate by X(tk) = f'(X(tk-1),Wk)

8.11.2 Constructor & Destructor Documentation

8.11.2.1 Discrete_Approximation_CD_Model::Discrete_Approximation_CD_Model (void)

8.11.2.2 virtual Discrete_Approximation_CD_Model::∼Discrete_Approximation_CD_Model (void) [virtual]

8.11.2.3 Discrete_Approximation_CD_Model::Discrete_Approximation_CD_Model (Continuous_Discrete_Model ∗ *m*)

8.11.2.4 Discrete_Approximation_CD_Model::Discrete_Approximation_CD_Model (Continuous_Discrete_Model ∗ *m*, const int & *a*)

the constructor

Parameters:

m the CD model

a the resolution of the discrete step Td = Ts ∗ a (Ts = sample duration of discrete observation)

8.11.3 Member Function Documentation

8.11.3.1 int Discrete_Approximation_CD_Model::Get_Alpha (void)

8.11.3.2 void Discrete_Approximation_CD_Model::Get_Linear_Parameters (const dcovector & *X*, const dcovector & *W*, dgematrix & *F*, dgematrix & *G*, dcovector & *Xp*) `[virtual]`

computed linearized parameter for EKF in X,W

Parameters:

X The state value

W The noise value

F The jacobian of f(X,W) in X

G The jacobian in f(X,W) in W

Xp The prediction Xp = f(X,W)

Reimplemented from Gaussian_Nonlinear_Model.

8.11.3.3 virtual void Discrete_Approximation_CD_Model::Get_Linear_Scheme (const dcovector & *X*, const dcovector & *W*, dgematrix & *F*, dgematrix & *J*, dcovector & *Xp*) `[pure virtual]`

Get the Linearized parameters Scheme in X,W.

Parameters:

X The state value

W The noise value

F The jacobian of f(X,W) in X

G The jacobian in f(X,W) in W

Xp The prediction Xp = f(X,W)

Implemented in Euler_CD_Model, SRK4_CD_Model, Heun_CD_Model, and Ozaki_CD_Model.

8.11.3.4 virtual void Discrete_Approximation_CD_Model::Init (void) `[virtual]`

Init The model if needed.

Reimplemented from Gaussian_Nonlinear_Model.

8.11.3.5 virtual dgematrix Discrete_Approximation_CD_Model::J_Observation_Function (const dcovector & *X*) `[virtual]`

the jacobian of the observation function evaluate at X

Parameters:

X

Returns:

The jacobian matrix

Reimplemented from Discrete_Observed_Model.

8.11.3.6 virtual dgematrix Discrete_Approximation_CD_Model::Jw_Scheme (const dcovector & *X*, const dcovector & *W*) [pure virtual]

Implemented in Euler_CD_Model, SRK4_CD_Model, Heun_CD_Model, and Ozaki_CD_Model.

8.11.3.7 dgematrix Discrete_Approximation_CD_Model::Jw_State_Function (const dcovector & *X*, const dcovector & *W*) [virtual]

the W jacobian of the State function evaluate at X,W

Parameters:

X evaluate at X

W evalate at W

Returns:

The jacobian matrix

Reimplemented from Gaussian_Nonlinear_Model.

8.11.3.8 virtual dgematrix Discrete_Approximation_CD_Model::Jx_Scheme (const dcovector & *X*, const dcovector & *W*) [pure virtual]

Implemented in Euler_CD_Model, SRK4_CD_Model, Heun_CD_Model, and Ozaki_CD_Model.

8.11.3.9 dgematrix Discrete_Approximation_CD_Model::Jx_State_Function (const dcovector & *X*, const dcovector & *W*) [virtual]

the X jacobian of the State function evaluate at X,W

Parameters:

X evaluate at X

W evalate at W

Returns:

The jacobian matrix

Reimplemented from Gaussian_Nonlinear_Model.

8.11.3.10 dcovector Discrete_Approximation_CD_Model::Observation_Function (const dcovector & *X*) [virtual]

The observation Yk=H(Xk) + Vk.

Parameters:

X The state at k

Returns:

The observation at k

Implements Discrete_Observed_Model.

8.11.3.11 virtual dcovector Discrete_Approximation_CD_Model::Scheme (const dcovector & *X*, const dcovector & *W*) [pure virtual]

Implemented in Euler_CD_Model, SRK4_CD_Model, Heun_CD_Model, and Ozaki_CD_Model.

8.11.3.12 void Discrete_Approximation_CD_Model::Set_Alpha (const int & *a*)

8.11.3.13 dcovector Discrete_Approximation_CD_Model::State_Function (const dcovector & *X*, const dcovector & *W*) [virtual]

The state Xk=F(Xk-1,Wk).

Parameters:

X The state at k-1

W The Noise

Returns:

The state at k

Implements Gaussian_Nonlinear_Model.

8.11.4 Member Data Documentation

8.11.4.1 int Discrete_Approximation_CD_Model::alpha [protected]

the resolution of the discrete step Td = Ts $*$ a (Ts = sample duration of discrete observation)

8.11.4.2 Continuous_Discrete_Model$*$ Discrete_Approximation_CD_Model::cd_model [protected]

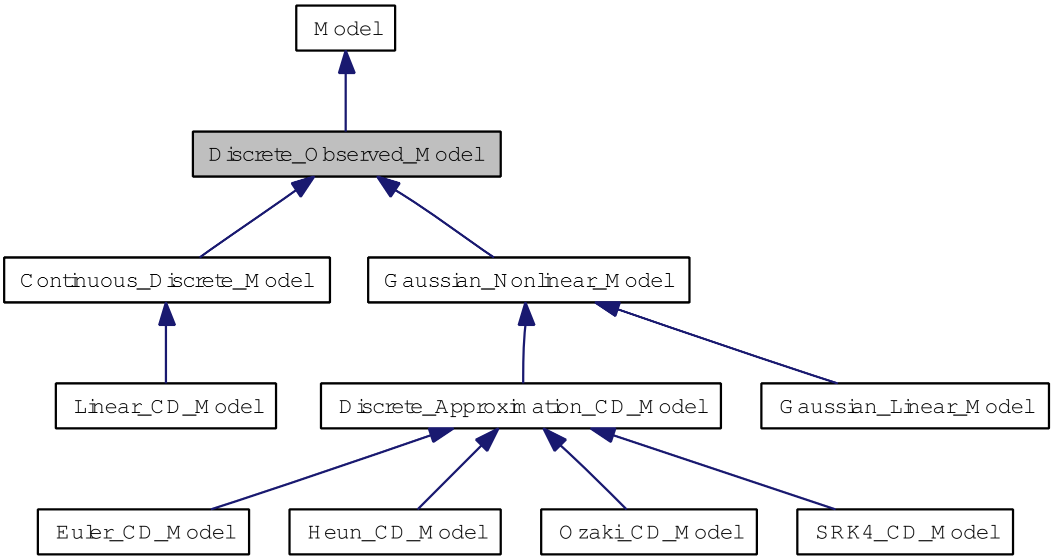
the continuous discrete model

8.12 Discrete_Observed_Model Class Reference

Class of discretely observed model.

`#include <gaussian_model.h>`

Inheritance diagram for Discrete_Observed_Model:



Public Member Functions

- Discrete_Observed_Model (void)

  The Constructor.

- virtual ~Discrete_Observed_Model (void)

  The Destructor.

- virtual dcovector Observation_Function (const dcovector &X)=0

  The observation Yk=H(Xk) + Vk.

- virtual dgematrix J_Observation_Function (const dcovector &X)

  the jacobian of the observation function evaluate at X

- virtual void Get_Init_Parameters (dcovector &mean, dsymatrix &Cov)

  Return the first an second moment of the initial law p(X0).

Public Attributes

- dsymatrix Qw

  The covariance matrix of state noise.

- dsymatrix Qv

The covariance matrix of observation noise.

Protected Attributes

- dsymatrix R0

  The covariance matrix of p(X0).

- dcovector X0

  The mean of p(X0).

8.12.1 Detailed Description

Class of discretely observed model.

The output Y_k is a discrete form of the hidden state. The init state is gaussian $\sim \mathcal{N}(X0, R0)$. The state and observation noises Wk, V_k are zero-mean gaussians processes. Their respective covariances are Q_w and Q_v .

8.12.2 Constructor & Destructor Documentation

8.12.2.1 Discrete_Observed_Model::Discrete_Observed_Model (void)

The Constructor.

8.12.2.2 virtual Discrete_Observed_Model::~Discrete_Observed_Model (void) `[virtual]`

The Destructor.

Returns:

8.12.3 Member Function Documentation

8.12.3.1 virtual void Discrete_Observed_Model::Get_Init_Parameters (dcovector & *mean*, dsymatrix & *Cov*) `[virtual]`

Return the first an second moment of the initial law p(X0).

Parameters:

mean The mean X0

Cov The Covariance R0

8.12.3.2 virtual dgematrix Discrete_Observed_Model::J_Observation_Function (const dcovector & *X*) [virtual]

the jacobian of the observation function evaluate at X

Parameters:

X

Returns:

The jacobian matrix

Reimplemented in Gaussian_Linear_Model, Linear_CD_Model, and Discrete_Approximation_CD_-Model.

8.12.3.3 virtual dcovector Discrete_Observed_Model::Observation_Function (const dcovector & *X*) [pure virtual]

The observation Yk=H(Xk) + Vk.

Parameters:

X The state at k

Returns:

The observation at k

Implemented in Gaussian_Linear_Model, Linear_CD_Model, and Discrete_Approximation_CD_Model.

8.12.4 Member Data Documentation

8.12.4.1 dsymatrix Discrete_Observed_Model::Qv

The covariance matrix of observation noise.

8.12.4.2 dsymatrix Discrete_Observed_Model::Qw

The covariance matrix of state noise.

8.12.4.3 dsymatrix Discrete_Observed_Model::R0 [protected]

The covariance matrix of p(X0).

8.12.4.4 dcovector Discrete_Observed_Model::X0 [protected]

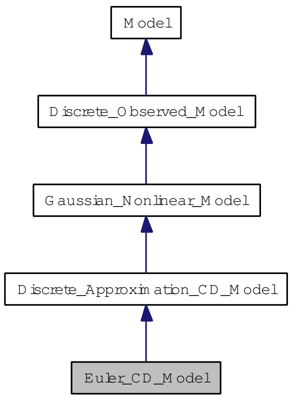
The mean of p(X0).

8.13 Euler_CD_Model Class Reference

continuous discret model: the state SDE is discretly approximate by an Euler method

```
#include <gaussian_model.h>
```

Inheritance diagram for Euler_CD_Model:



Public Member Functions

- Euler_CD_Model (void)
- Euler_CD_Model (Continuous_Discrete_Model *m, const int &a)
- dcovector Scheme (const dcovector &X, const dcovector &W)
- void Get_Linear_Scheme (const dcovector &X, const dcovector &W, dgematrix &F, dgematrix &J, dcovector &Xp)

  Get the Linearized parameters Scheme in X,W.

- dgematrix Jx_Scheme (const dcovector &X, const dcovector &W)
- dgematrix Jw_Scheme (const dcovector &X, const dcovector &W)

8.13.1 Detailed Description

continuous discret model: the state SDE is discretly approximate by an Euler method

8.13.2 Constructor & Destructor Documentation

8.13.2.1 Euler_CD_Model::Euler_CD_Model (void)

8.13.2.2 Euler_CD_Model::Euler_CD_Model (Continuous_Discrete_Model * *m*, const int & *a*)

8.13.3 Member Function Documentation

8.13.3.1 void Euler_CD_Model::Get_Linear_Scheme (const dcovector & *X*, const dcovector & *W*, dgematrix & *F*, dgematrix & *J*, dcovector & *Xp*) [virtual]

Get the Linearized parameters Scheme in X,W.

Parameters:

- ***X*** The state value
- ***W*** The noise value
- ***F*** The jacobian of f(X,W) in X
- ***G*** The jacobian in f(X,W) in W
- ***Xp*** The prediction Xp = f(X,W)

Implements Discrete_Approximation_CD_Model.

8.13.3.2 dgematrix Euler_CD_Model::Jw_Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

8.13.3.3 dgematrix Euler_CD_Model::Jx_Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

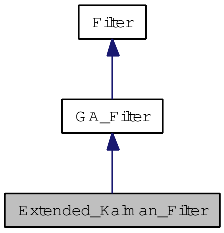
8.13.3.4 dcovector Euler_CD_Model::Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

8.14 Extended_Kalman_Filter Class Reference

```
#include <extended_kalman_filter.h>
```

Inheritance diagram for Extended_Kalman_Filter:



Public Member Functions

- Extended_Kalman_Filter (void)
- Extended_Kalman_Filter (Gaussian_Nonlinear_Model ∗m)

Protected Member Functions

- int _update (const dcovector &Y)

8.14.1 Constructor & Destructor Documentation

8.14.1.1 Extended_Kalman_Filter::Extended_Kalman_Filter (void)

8.14.1.2 Extended_Kalman_Filter::Extended_Kalman_Filter (Gaussian_Nonlinear_Model ∗ *m*)

8.14.2 Member Function Documentation

8.14.2.1 int Extended_Kalman_Filter::_update (const dcovector & *Y*) `[protected, virtual]`

Specific update for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

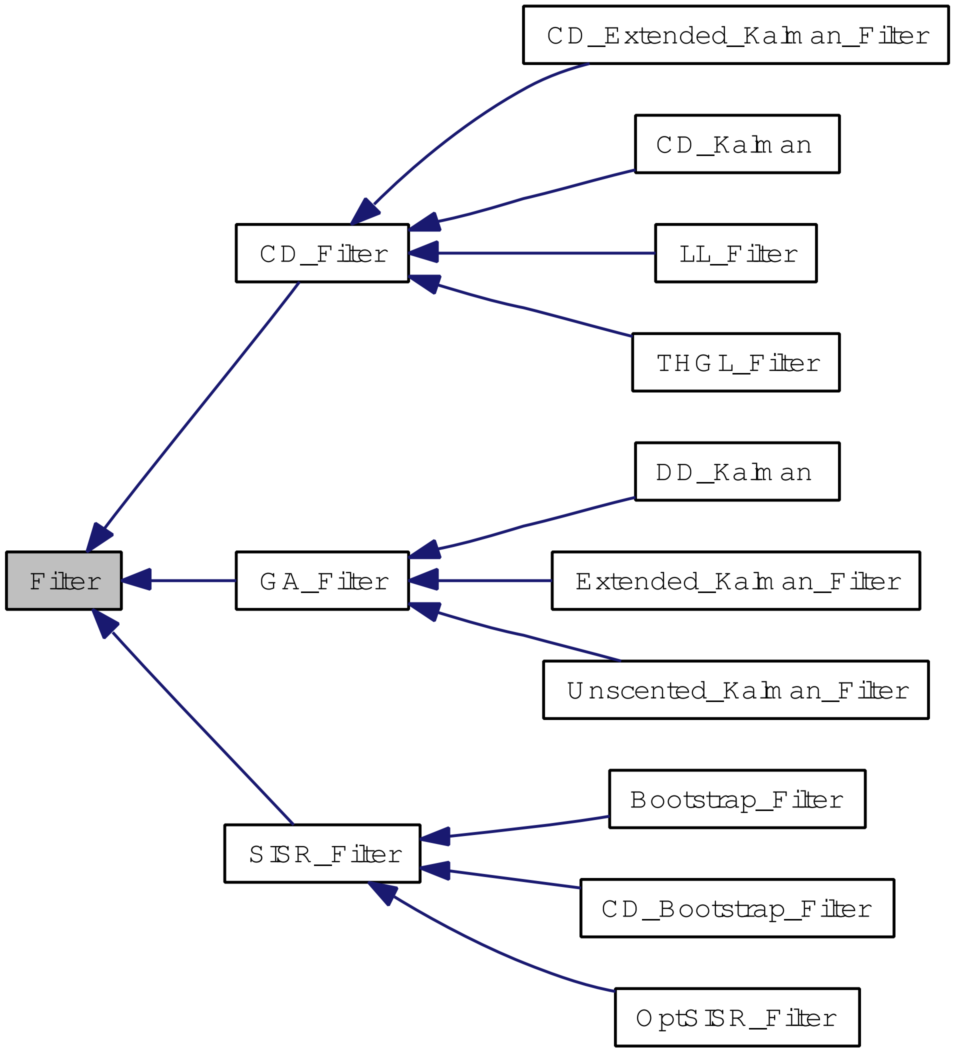
Implements Filter.

8.15 Filter Class Reference

Abstract class of all filters.

`#include <filter.h>`

Inheritance diagram for Filter:



Public Member Functions

- Filter (void)
- virtual ~Filter (void)
- int Update (const dcovector &Y)
- int Filtering (const vector< dcovector > &Y)
- virtual dcovector Expected_Get (void)=0
- int Init (void)
- double Likelihood_Get (void)
- virtual int Save_X (const char *filename)

Public Attributes

- Model $*$ model

  The hidden markov model.

- vector< dcovector > X

  $\{\hat{X}_{k|k}, k = 0, ...N\}$

Protected Member Functions

- virtual int _update (const dcovector &Y)=0
- virtual int _init (void)=0

Protected Attributes

- double Likelihood

  The likelihood $p_{Y_{0:N}}(y_{0:N})$.

8.15.1 Detailed Description

Abstract class of all filters.

Filters calculate recursively an estimation $\hat{X}_{k|k}$ of the STATE X_k of a hidden markov model (Model) given observations $Y_{0:k}$.

They compute also recursively the likelihood $p_{Y_{0:N}}(y_{0:N})$.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 Filter::Filter (void)

A constructor

8.15.2.2 virtual Filter::∼Filter (void) `[virtual]`

The destructor

8.15.3 Member Function Documentation

8.15.3.1 virtual int Filter::_init (void) `[protected, pure virtual]`

Specific init for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

Implemented in GA_Filter, CD_Filter, SISR_Filter, and Unscented_Kalman_Filter.

8.15.3.2 virtual int Filter::_update (const dcovector & *Y*) [protected, pure virtual]

Specific update for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

Implemented in Extended_Kalman_Filter, CD_Extended_Kalman_Filter, CD_Kalman, DD_Kalman, LL_Filter, SISR_Filter, THGL_Filter, and Unscented_Kalman_Filter.

8.15.3.3 virtual dcovector Filter::Expected_Get (void) [pure virtual]

Evaluate the current estimation of the state

Returns:

$\hat{X}_{k|k}$

Implemented in GA_Filter, CD_Filter, and SISR_Filter.

8.15.3.4 int Filter::Filtering (const vector< dcovector > & *Y*)

Perform a trajectory state estimation given a sequence $y_{0:N}$

Parameters:

Y The sequence

Returns:

0 if everything is ok

8.15.3.5 int Filter::Init (void)

To init the filter at k=0

8.15.3.6 double Filter::Likelihood_Get (void)

Return the current likelihood $p_{Y_{0:k}}(y_{0:k})$

Returns:

$p_{Y_{0:N}}(y_{0:N})$

8.15.3.7 virtual int Filter::Save_X (const char * *filename*) `[virtual]`

Save the estimation $\{\hat{X}_{k|k}, k = 0, ...N\}$

Parameters:

filename

Returns:

0 if everything is ok

Reimplemented in CD_Filter, and CD_Bootstrap_Filter.

8.15.3.8 int Filter::Update (const dcovector & *Y*)

Perform an estimation step with a new observation

Parameters:

Y The new observed sample

Returns:

0 if everything is ok

8.15.4 Member Data Documentation

8.15.4.1 double Filter::Likelihood `[protected]`

The likelihood $p_{Y_{0:N}}(y_{0:N})$.

8.15.4.2 Model* Filter::model

The hidden markov model.

8.15.4.3 vector<dcovector> Filter::X

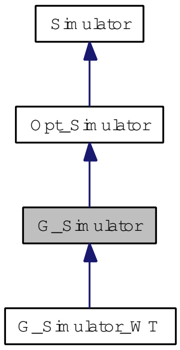
$\{\hat{X}_{k|k}, k = 0, ...N\}$

The estimated trajectory of the state

8.16 G_Simulator Class Reference

`#include <simulator.h>`

Inheritance diagram for G_Simulator:



Public Member Functions

- G_Simulator (void)
- G_Simulator (Gaussian_Nonlinear_Model ∗m)
- dcovector Draw_Init (void)

  Draw a sample from p(X0).

- dcovector Draw_Transition (const dcovector &Xkm1)

  Draw a sample from the transition densisty p(Xk|Xk-1).

- dcovector Draw_Observation (const dcovector &Xk)

  Calculate the value of the density of probability of Y given X : p(Y|X).

- long double Observation_Density (const dcovector &Y, const dcovector &X)

  calculate the value of the density of probability of Y given X : p(Y|X)

- dcovector Draw_Optimal (const dcovector &Yk, const dcovector &Xkm1)

  Draw a sample from the optimal densisty p(Xk|Yk,Xk-1).

- long double Obs_Optimal_Density (const dcovector &Yk, const dcovector &Xkm1)

  calculate the value of the density of probability of Yk given Xk-1 : p(Yk|Xk-1)

8.16.1 Constructor & Destructor Documentation

8.16.1.1 G_Simulator::G_Simulator (void)

8.16.1.2 G_Simulator::G_Simulator (Gaussian_Nonlinear_Model * *m*)

8.16.2 Member Function Documentation

8.16.2.1 dcovector G_Simulator::Draw_Init (void) `[virtual]`

Draw a sample from p(X0).

Returns:

A sample from p(X0)

Implements Simulator.

Reimplemented in G_Simulator_WT.

8.16.2.2 dcovector G_Simulator::Draw_Observation (const dcovector & *Xk*) `[virtual]`

Calculate the value of the density of probability of Y given X : p(Y|X).

Parameters:

Xk The state at k

Returns:

The simulated observation

Implements Simulator.

8.16.2.3 dcovector G_Simulator::Draw_Optimal (const dcovector & *Yk*, const dcovector & *Xkm1*) `[virtual]`

Draw a sample from the optimal densisty p(Xk|Yk,Xk-1).

Parameters:

Yk The obseration at k

Xkm1 X(k-1) the state value at k-1

Returns:

A sample from the optimal importance density

Implements Opt_Simulator.

8.16.2.4 dcovector G_Simulator::Draw_Transition (const dcovector & *Xkm1*) `[virtual]`

Draw a sample from the transition densisty p(Xk|Xk-1).

Parameters:

Xkm1 X(k-1) the preceding state

Returns:

Implements Simulator.

8.16.2.5 long double G_Simulator::Obs_Optimal_Density (const dcovector & *Yk*, const dcovector & *Xkm1*) `[virtual]`

calculate the value of the density of probability of Yk given Xk-1 : p(Yk|Xk-1)

Parameters:

Yk the osbervation at k

Xkm1 The state at k-1

Returns:

The value of the density p(Yk|Xk-1)

Implements Opt_Simulator.

8.16.2.6 long double G_Simulator::Observation_Density (const dcovector & *Y*, const dcovector & *X*) `[virtual]`

calculate the value of the density of probability of Y given X : p(Y|X)

Parameters:

Y The osbervation

X The state

Returns:

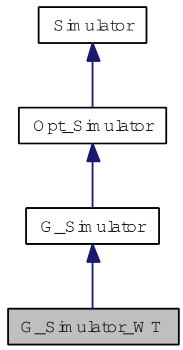
The value of the density

Implements Simulator.

8.17 G_Simulator_WT Class Reference

```
#include <simulator.h>
```

Inheritance diagram for G_Simulator_WT:



Public Member Functions

- G_Simulator_WT (void)
- G_Simulator_WT (Gaussian_Nonlinear_Model ∗m, const int &NB, const int &N)
- dcovector Draw_Init (void)

  Draw a sample from p(X0).

Private Attributes

- vector< dcovector > Xt
- int NB
- int N

8.17.1 Constructor & Destructor Documentation

8.17.1.1 G_Simulator_WT::G_Simulator_WT (void)

8.17.1.2 G_Simulator_WT::G_Simulator_WT (Gaussian_Nonlinear_Model ∗ *m*, const int & *NB*, const int & *N*)

8.17.2 Member Function Documentation

8.17.2.1 dcovector G_Simulator_WT::Draw_Init (void) `[virtual]`

Draw a sample from p(X0).

Returns:

A sample from p(X0)

Reimplemented from G_Simulator.

8.17.3 Member Data Documentation

8.17.3.1 int G_Simulator_WT::N [private]

8.17.3.2 int G_Simulator_WT::NB [private]

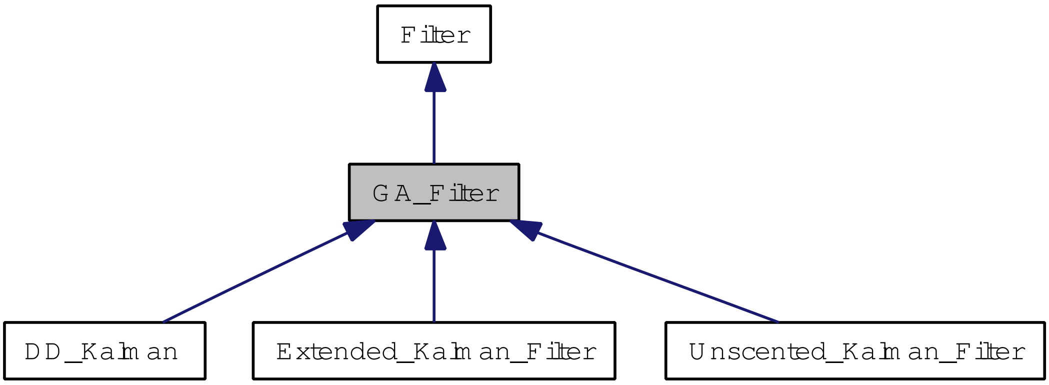
8.17.3.3 vector<dcovector> G_Simulator_WT::Xt [private]

8.18 GA_Filter Class Reference

Abstract class of Gaussian Approximation filters.

`#include <filter.h>`

Inheritance diagram for GA_Filter:



Public Member Functions

- GA_Filter (void)

  A constructor.

- GA_Filter (Gaussian_Nonlinear_Model ∗m)
- dcovector Expected_Get (void)

Public Attributes

- dcovector M

  The current mean $\hat{X}_{k|k} = E[X_k|Y_{0:k}]$.

- dgematrix R

  The current covariance $\hat{P}_{k|k} = E[(X_k - \hat{X}_{k|k})(X_k - \hat{X}_{k|k})]$.

- dcovector Xp

  The prediction $\hat{X}_{k-1|k} = E[X_{k-1}|Y_{0:k}]$.

- dgematrix Rp

  The prediction covariance $\hat{P}_{k-1|k} = E[(X_k - \hat{X}_{k-1|k})(X_k - \hat{X}_{k-1|k})]$.

Protected Member Functions

- virtual int _init (void)

8.18.1 Detailed Description

Abstract class of Gaussian Approximation filters.

For discrete-discrete models (Gaussian_Nonlinear_Model), these filters approximate the probability density of the state transition $p_{X_k|X_{k-1}}$ and the probability of the observation $p_{Y_k|X_k}$ by gaussian densities. The approximation is exact in the case of linear model (Gaussian_Linear_Model) and lead to the discrete-discrete Kalman Filter (DD_Kalman). For other non-linear models (Gaussian_Nonlinear_Model) UKF (Unscented_Kalman_Filter) or EKF (Extended_Kalman_Filter) can be used.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 GA_Filter::GA_Filter (void)

A constructor.

8.18.2.2 GA_Filter::GA_Filter (Gaussian_Nonlinear_Model $*$ *m*)

A constructor

Parameters:

m A discrete-discrete gaussian non-linear model

8.18.3 Member Function Documentation

8.18.3.1 virtual int GA_Filter::_init (void) `[protected, virtual]`

Specific init for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

Implements Filter.

Reimplemented in Unscented_Kalman_Filter.

8.18.3.2 dcovector GA_Filter::Expected_Get (void) `[virtual]`

Get the current estimation $\hat{X}_{k|k}$

Returns:

$\hat{X}_{k|k}$

Implements Filter.

8.18.4 Member Data Documentation

8.18.4.1 dcovector GA_Filter::M

The current mean $\hat{X}_{k|k} = E[X_k|Y_{0:k}]$.

8.18.4.2 dgematrix GA_Filter::R

The current covariance $\hat{P}_{k|k} = E[(X_k - \hat{X}_{k|k})(X_k - \hat{X}_{k|k})]$.

8.18.4.3 dgematrix GA_Filter::Rp

The prediction covariance $\hat{P}_{k-1|k} = E[(X_k - \hat{X}_{k-1|k})(X_k - \hat{X}_{k-1|k})]$.

8.18.4.4 dcovector GA_Filter::Xp

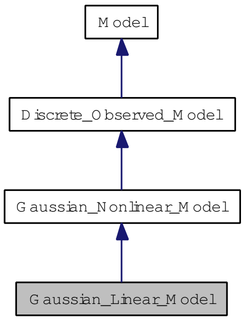
The prediction $\hat{X}_{k-1|k} = E[X_{k-1}|Y_{0:k}]$.

8.19 Gaussian_Linear_Model Class Reference

Gaussian Linear Model :.

`#include <gaussian_model.h>`

Inheritance diagram for Gaussian_Linear_Model:



Public Member Functions

- Gaussian_Linear_Model (void)
- dcovector State_Function (const dcovector &X, const dcovector &W)

  The state Xk=F(Xk-1,Wk).

- dgematrix Jx_State_Function (const dcovector &X, const dcovector &W)

  the X jacobian of the State function evaluate at X,W

- dgematrix Jw_State_Function (const dcovector &X, const dcovector &W)

  the W jacobian of the State function evaluate at X,W

- dcovector Get_Mean_Prediction (const dcovector &M)
- dgematrix Get_Cov_Prediction (const dgematrix &P)
- dcovector Observation_Function (const dcovector &X)

  The observation Yk=H(Xk) + Vk.

- dgematrix J_Observation_Function (const dcovector &X)

  the jacobian of the observation function evaluate at X

Public Attributes

- dgematrix F
- dgematrix G
- dcovector f
- dcovector h
- dgematrix H

8.19.1 Detailed Description

Gaussian Linear Model :.

The state : X(k) = F X(k-1) + f + G $*$ Wk The Observation Y(k) = H X(k) + h + V

8.19.2 Constructor & Destructor Documentation

8.19.2.1 Gaussian_Linear_Model::Gaussian_Linear_Model (void)

8.19.3 Member Function Documentation

8.19.3.1 dgematrix Gaussian_Linear_Model::Get_Cov_Prediction (const dgematrix & *P*)

8.19.3.2 dcovector Gaussian_Linear_Model::Get_Mean_Prediction (const dcovector & *M*)

8.19.3.3 dgematrix Gaussian_Linear_Model::J_Observation_Function (const dcovector & *X*) `[virtual]`

the jacobian of the observation function evaluate at X

Parameters:

X

Returns:

The jacobian matrix

Reimplemented from Discrete_Observed_Model.

8.19.3.4 dgematrix Gaussian_Linear_Model::Jw_State_Function (const dcovector & *X*, const dcovector & *W*) `[virtual]`

the W jacobian of the State function evaluate at X,W

Parameters:

X evaluate at X

W evalate at W

Returns:

The jacobian matrix

Reimplemented from Gaussian_Nonlinear_Model.

8.19.3.5 dgematrix Gaussian_Linear_Model::Jx_State_Function (const dcovector & *X*, const dcovector & *W*) `[virtual]`

the X jacobian of the State function evaluate at X,W

Parameters:

X evaluate at X

W evalate at W

Returns:

The jacobian matrix

Reimplemented from Gaussian_Nonlinear_Model.

8.19.3.6 dcovector Gaussian_Linear_Model::Observation_Function (const dcovector & *X*) [virtual]

The observation Yk=H(Xk) + Vk.

Parameters:

X The state at k

Returns:

The observation at k

Implements Discrete_Observed_Model.

8.19.3.7 dcovector Gaussian_Linear_Model::State_Function (const dcovector & *X*, const dcovector & *W*) [virtual]

The state Xk=F(Xk-1,Wk).

Parameters:

X The state at k-1

W The Noise

Returns:

The state at k

Implements Gaussian_Nonlinear_Model.

8.19.4 Member Data Documentation

8.19.4.1 dcovector Gaussian_Linear_Model::f

8.19.4.2 dgematrix Gaussian_Linear_Model::F

8.19.4.3 dgematrix Gaussian_Linear_Model::G

8.19.4.4 dgematrix Gaussian_Linear_Model::H

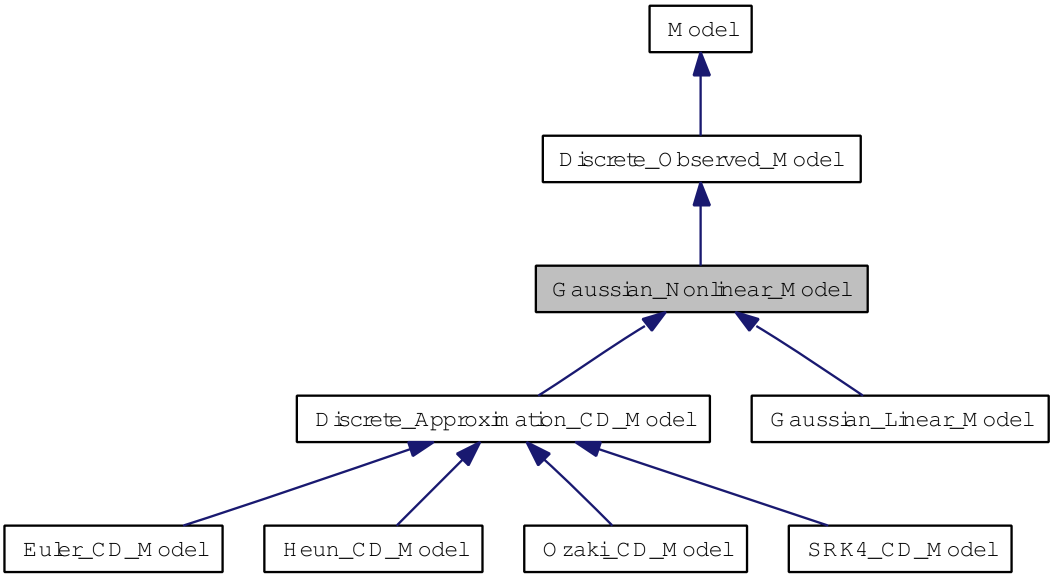
8.19.4.5 dcovector Gaussian_Linear_Model::h

8.20 Gaussian_Nonlinear_Model Class Reference

Gaussian Nonlinear Model The state : X(k) = F (Xk-1, Wk) The Observation Y(k) = H (X(k)) + V.

`#include <gaussian_model.h>`

Inheritance diagram for Gaussian_Nonlinear_Model:



Public Member Functions

- Gaussian_Nonlinear_Model (void)
- virtual ~Gaussian_Nonlinear_Model (void)
- virtual void Init (void)

  Init The model if needed.

- virtual dcovector State_Function (const dcovector &X, const dcovector &W)=0

  The state Xk=F(Xk-1,Wk).

- virtual dgematrix Jx_State_Function (const dcovector &X, const dcovector &W)

  the X jacobian of the State function evaluate at X,W

- virtual dgematrix Jw_State_Function (const dcovector &X, const dcovector &W)

  the W jacobian of the State function evaluate at X,W

- virtual void Get_Linear_Parameters (const dcovector &X, const dcovector &W, dgematrix &F, dgematrix &G, dcovector &Xp)

  computed linearized parameter for EKF in X,W

8.20.1 Detailed Description

Gaussian Nonlinear Model The state : X(k) = F (Xk-1, Wk) The Observation Y(k) = H (X(k)) + V.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 Gaussian_Nonlinear_Model::Gaussian_Nonlinear_Model (void)

8.20.2.2 virtual Gaussian_Nonlinear_Model::~Gaussian_Nonlinear_Model (void) `[virtual]`

8.20.3 Member Function Documentation

8.20.3.1 virtual void Gaussian_Nonlinear_Model::Get_Linear_Parameters (const dcovector & *X*, const dcovector & *W*, dgematrix & *F*, dgematrix & *G*, dcovector & *Xp*) `[virtual]`

computed linearized parameter for EKF in X,W

Parameters:

X The state value

W The noise value

F The jacobian of f(X,W) in X

G The jacobian in f(X,W) in W

Xp The prediction Xp = f(X,W)

Reimplemented in Discrete_Approximation_CD_Model.

8.20.3.2 virtual void Gaussian_Nonlinear_Model::Init (void) `[virtual]`

Init The model if needed.

Reimplemented in Discrete_Approximation_CD_Model.

8.20.3.3 virtual dgematrix Gaussian_Nonlinear_Model::Jw_State_Function (const dcovector & *X*, const dcovector & *W*) `[virtual]`

the W jacobian of the State function evaluate at X,W

Parameters:

X evaluate at X

W evalate at W

Returns:

The jacobian matrix

Reimplemented in Gaussian_Linear_Model, and Discrete_Approximation_CD_Model.

8.20.3.4 virtual dgematrix Gaussian_Nonlinear_Model::Jx_State_Function (const dcovector & *X*, const dcovector & *W*) `[virtual]`

the X jacobian of the State function evaluate at X,W

Parameters:

X evaluate at X

W evalate at W

Returns:

The jacobian matrix

Reimplemented in Gaussian_Linear_Model, and Discrete_Approximation_CD_Model.

8.20.3.5 virtual dcovector Gaussian_Nonlinear_Model::State_Function (const dcovector & *X*, const dcovector & *W*) `[pure virtual]`

The state Xk=F(Xk-1,Wk).

Parameters:

X The state at k-1

W The Noise

Returns:

The state at k

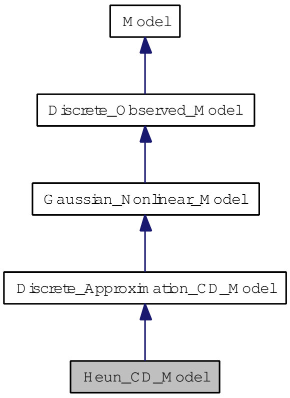
Implemented in Gaussian_Linear_Model, and Discrete_Approximation_CD_Model.

8.21 Heun_CD_Model Class Reference

continuous discret model: the state SDE is discretly approximate by an Sstochastic Heun method

`#include <gaussian_model.h>`

Inheritance diagram for Heun_CD_Model:



Public Member Functions

- Heun_CD_Model (void)
- Heun_CD_Model (Continuous_Discrete_Model *m, const int &a)
- dcovector Scheme (const dcovector &X, const dcovector &W)
- dgematrix Jx_Scheme (const dcovector &X, const dcovector &W)
- dgematrix Jw_Scheme (const dcovector &X, const dcovector &W)
- void Get_Linear_Scheme (const dcovector &X, const dcovector &W, dgematrix &F, dgematrix &J, dcovector &Xp)

    Get the Linearized parameters Scheme in X,W.

8.21.1 Detailed Description

continuous discret model: the state SDE is discretly approximate by an Sstochastic Heun method

8.21.2 Constructor & Destructor Documentation

8.21.2.1 Heun_CD_Model::Heun_CD_Model (void)

8.21.2.2 Heun_CD_Model::Heun_CD_Model (Continuous_Discrete_Model * *m*, const int & *a*)

8.21.3 Member Function Documentation

8.21.3.1 void Heun_CD_Model::Get_Linear_Scheme (const dcovector & *X*, const dcovector & *W*, dgematrix & *F*, dgematrix & *J*, dcovector & *Xp*) `[virtual]`

Get the Linearized parameters Scheme in X,W.

Parameters:

- ***X*** The state value
- ***W*** The noise value
- ***F*** The jacobian of f(X,W) in X
- ***G*** The jacobian in f(X,W) in W
- ***Xp*** The prediction Xp = f(X,W)

Implements Discrete_Approximation_CD_Model.

8.21.3.2 dgematrix Heun_CD_Model::Jw_Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

8.21.3.3 dgematrix Heun_CD_Model::Jx_Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

8.21.3.4 dcovector Heun_CD_Model::Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

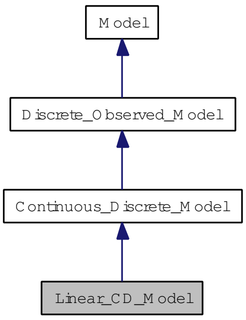
Implements Discrete_Approximation_CD_Model.

8.22 Linear_CD_Model Class Reference

Linear continuous discrete model class of the form dx = AX dt + Bdt + CdW Y_k = HX(t_k) + h + V_k.

`#include <gaussian_model.h>`

Inheritance diagram for Linear_CD_Model:



Public Member Functions

- Linear_CD_Model (void)
- dcovector Drift_Function (const dcovector &X)

  the drift function of dX(t) = F(X)dt + G(X)∗dW

- dgematrix J_Drift_Function (const dcovector &X)

  the jacobian of the drift function evaluate at X

- dgematrix Diffusion_Function (void)

  the diffusion function

- dcovector Observation_Function (const dcovector &X)

  The observation Yk=H(Xk) + Vk.

- dgematrix J_Observation_Function (const dcovector &X)

  the jacobian of the observation function evaluate at X

- dcovector Get_Mean_Prediction (const dcovector &M)
- dgematrix Get_Cov_Prediction (const dgematrix &P)
- virtual void Init (void)

Public Attributes

- dgematrix A
- dcovector B
- dgematrix C
- dcovector h
- dgematrix H

8.22.1 Detailed Description

Linear continuous discrete model class of the form dx = AX dt + Bdt + CdW Y_k = HX(t_k) + h + V_k.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 Linear_CD_Model::Linear_CD_Model (void)

8.22.3 Member Function Documentation

8.22.3.1 dgematrix Linear_CD_Model::Diffusion_Function (void) `[virtual]`

the diffusion function

Parameters:

X the state

Returns:

G(X).

Implements Continuous_Discrete_Model.

8.22.3.2 dcovector Linear_CD_Model::Drift_Function (const dcovector & *X*) `[virtual]`

the drift function of dX(t) = F(X)dt + G(X)∗dW

Parameters:

X the state.

Returns:

f(X)

Implements Continuous_Discrete_Model.

8.22.3.3 dgematrix Linear_CD_Model::Get_Cov_Prediction (const dgematrix & *P*)

8.22.3.4 dcovector Linear_CD_Model::Get_Mean_Prediction (const dcovector & *M*)

8.22.3.5 virtual void Linear_CD_Model::Init (void) `[inline, virtual]`

Initialized CD model

Reimplemented from Continuous_Discrete_Model.

8.22.3.6 dgematrix Linear_CD_Model::J_Drift_Function (const dcovector & *X*) `[virtual]`

the jacobian of the drift function evaluate at X

Parameters:

X

Returns:

the jacobian matrix

Reimplemented from Continuous_Discrete_Model.

8.22.3.7 dgematrix Linear_CD_Model::J_Observation_Function (const dcovector & *X*) [virtual]

the jacobian of the observation function evaluate at X

Parameters:

X

Returns:

The jacobian matrix

Reimplemented from Discrete_Observed_Model.

8.22.3.8 dcovector Linear_CD_Model::Observation_Function (const dcovector & *X*) [virtual]

The observation Yk=H(Xk) + Vk.

Parameters:

X The state at k

Returns:

The observation at k

Implements Discrete_Observed_Model.

8.22.4 Member Data Documentation

8.22.4.1 dgematrix Linear_CD_Model::A

8.22.4.2 dcovector Linear_CD_Model::B

8.22.4.3 dgematrix Linear_CD_Model::C

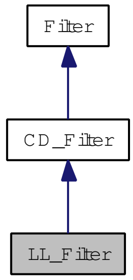
8.22.4.4 dgematrix Linear_CD_Model::H

8.22.4.5 dcovector Linear_CD_Model::h

8.23 LL_Filter Class Reference

`#include <local_linearization_filter.h>`

Inheritance diagram for LL_Filter:



Public Member Functions

- LL_Filter (void)
- LL_Filter (Continuous_Discrete_Model ∗m)

Protected Member Functions

- int _update (const dcovector &Y)

8.23.1 Constructor & Destructor Documentation

8.23.1.1 LL_Filter::LL_Filter (void)

8.23.1.2 LL_Filter::LL_Filter (Continuous_Discrete_Model ∗ *m*)

8.23.2 Member Function Documentation

8.23.2.1 int LL_Filter::_update (const dcovector & *Y*) `[protected, virtual]`

Specific update for each filter

Parameters:

Y The observed sample

Returns:

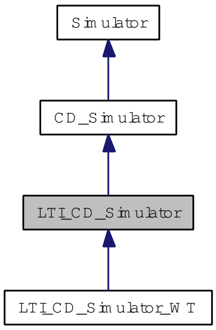
0 if no problem

Implements Filter.

8.24 LTI_CD_Simulator Class Reference

`#include <simulator.h>`

Inheritance diagram for LTI_CD_Simulator:



Public Member Functions

- LTI_CD_Simulator (void)
- LTI_CD_Simulator (Linear_CD_Model ∗cd_m, const int &apha=1)

Protected Member Functions

- dcovector draw_state (const dcovector &X)

8.24.1 Constructor & Destructor Documentation

8.24.1.1 LTI_CD_Simulator::LTI_CD_Simulator (void)

8.24.1.2 LTI_CD_Simulator::LTI_CD_Simulator (Linear_CD_Model ∗ *cd_m*, const int & *apha* = `1`)

8.24.2 Member Function Documentation

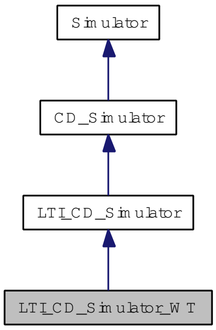
8.24.2.1 dcovector LTI_CD_Simulator::draw_state (const dcovector & *X*) `[protected, virtual]`

Reimplemented from CD_Simulator.

8.25 LTI_CD_Simulator_WT Class Reference

`#include <simulator.h>`

Inheritance diagram for LTI_CD_Simulator_WT:



Public Member Functions

- LTI_CD_Simulator_WT (void)
- LTI_CD_Simulator_WT (Linear_CD_Model ∗cd_m, const int &apha, const double &tb, const double &t)
- dcovector Draw_Init (void)

    Draw a sample from p(X0).

Private Attributes

- vector< dcovector > Xt
- double TB
- double T

8.25.1 Constructor & Destructor Documentation

8.25.1.1 LTI_CD_Simulator_WT::LTI_CD_Simulator_WT (void)

8.25.1.2 LTI_CD_Simulator_WT::LTI_CD_Simulator_WT (Linear_CD_Model ∗ *cd_m*, const int & *apha*, const double & *tb*, const double & *t*)

8.25.2 Member Function Documentation

8.25.2.1 dcovector LTI_CD_Simulator_WT::Draw_Init (void) `[virtual]`

Draw a sample from p(X0).

Returns:

A sample from p(X0)

Reimplemented from CD_Simulator.

8.25.3 Member Data Documentation

8.25.3.1 double LTI_CD_Simulator_WT::T [private]

8.25.3.2 double LTI_CD_Simulator_WT::TB [private]

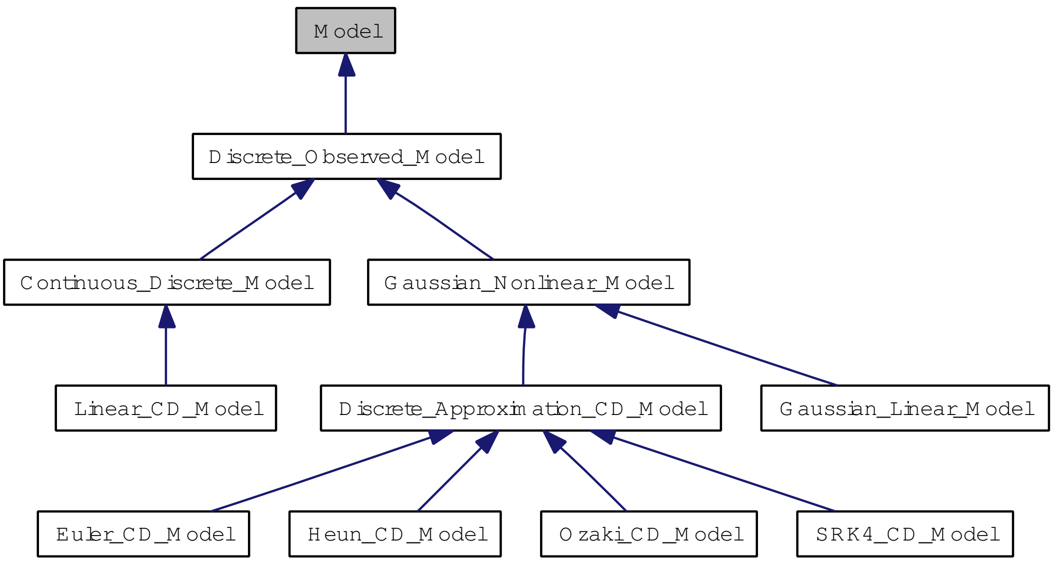
8.25.3.3 vector<dcovector> LTI_CD_Simulator_WT::Xt [private]

8.26 Model Class Reference

The class of time varying-models.

`#include <gaussian_model.h>`

Inheritance diagram for Model:



Public Member Functions

- Model (void)
- virtual ∼Model (void)

  The Destructor.

- int Update (void)

  Update the time.

- int Clear (void)

  Set the time to 0.

- int Get_Time (void)

  Get The current time.

Protected Attributes

- int _k

  The time.

8.26.1 Detailed Description

The class of time varying-models.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 Model::Model (void)

8.26.2.2 virtual Model::~Model (void) `[virtual]`

The Destructor.

Returns:

8.26.3 Member Function Documentation

8.26.3.1 int Model::Clear (void)

Set the time to 0.

Returns:

0 if it's Ok;

8.26.3.2 int Model::Get_Time (void)

Get The current time.

Returns:

8.26.3.3 int Model::Update (void)

Update the time.

Returns:

0 if it's Ok;

8.26.4 Member Data Documentation

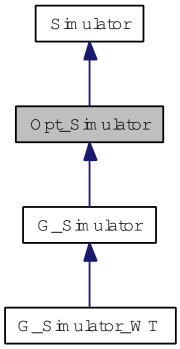
8.26.4.1 int Model::_k `[protected]`

The time.

8.27 Opt_Simulator Class Reference

`#include <simulator.h>`

Inheritance diagram for Opt_Simulator:



Public Member Functions

- Opt_Simulator (void)
- virtual dcovector Draw_Optimal (const dcovector &Yk, const dcovector &Xkm1)=0

  Draw a sample from the optimal densisty p(Xk|Yk,Xk-1).

- virtual long double Obs_Optimal_Density (const dcovector &Yk, const dcovector &Xkm1)=0

  calculate the value of the density of probability of Yk given Xk-1 : p(Yk|Xk-1)

8.27.1 Constructor & Destructor Documentation

8.27.1.1 Opt_Simulator::Opt_Simulator (void)

8.27.2 Member Function Documentation

8.27.2.1 virtual dcovector Opt_Simulator::Draw_Optimal (const dcovector & *Yk*, const dcovector & *Xkm1*) `[pure virtual]`

Draw a sample from the optimal densisty p(Xk|Yk,Xk-1).

Parameters:

Yk The obseration at k

Xkm1 X(k-1) the state value at k-1

Returns:

A sample from the optimal importance density

Implemented in G_Simulator.

8.27.2.2 virtual long double Opt_Simulator::Obs_Optimal_Density (const dcovector & *Yk*, const dcovector & *Xkm1*) `[pure virtual]`

calculate the value of the density of probability of Yk given Xk-1 : p(Yk|Xk-1)

Parameters:

Yk the osbervation at k

Xkm1 The state at k-1

Returns:

The value of the density p(Yk|Xk-1)

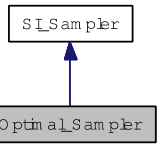
Implemented in G_Simulator.

8.28 Optimal_Sampler Class Reference

This sampler use the optimal importance density.

```
#include <sisr_filter.h>
```

Inheritance diagram for Optimal_Sampler:



Public Member Functions

- Optimal_Sampler (void)
- Optimal_Sampler (Opt_Simulator ∗m)
- vector< Weighted_Sample > DrawInitCloud (const int &NbSample)

    draw a set of possible init state

- vector< Weighted_Sample > Draw (const dcovector &Y_k, const vector< Weighted_Sample > &X_km1)

    Draw a set of samples from the importance density Xk given Y0:k X0:k-1.

- long double Weight (vector< Weighted_Sample > &cloud, const dcovector &Y_k, const vector< Weighted_Sample > &X_km1)

    Modify the weights of cloud for the weighting step in the sisr.

8.28.1 Detailed Description

This sampler use the optimal importance density.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 Optimal_Sampler::Optimal_Sampler (void)

8.28.2.2 Optimal_Sampler::Optimal_Sampler (Opt_Simulator ∗ *m*)

8.28.3 Member Function Documentation

8.28.3.1 vector<Weighted_Sample > Optimal_Sampler::Draw (const dcovector & *Y_k*, const vector< Weighted_Sample > & *X_km1*) [virtual]

Draw a set of samples from the importance density Xk given Y0:k X0:k-1.

Parameters:

Y_k The observation from 0 to k

X_km1 The cloud from 0 to km1

Returns:

A cloud representing the importance density q(Xk|Y0:k,X0:k-1)

Implements SI_Sampler.

8.28.3.2 vector<Weighted_Sample > Optimal_Sampler::DrawInitCloud (const int & *NbSample*) `[virtual]`

draw a set of possible init state

Parameters:

NbSample Number of sample

Returns:

A set of weighted samples

Implements SI_Sampler.

8.28.3.3 long double Optimal_Sampler::Weight (vector< Weighted_Sample > & *cloud*, const dcovector & *Y_k*, const vector< Weighted_Sample > & *X_km1*) `[virtual]`

Modify the weights of cloud for the weighting step in the sisr.

Parameters:

cloud The curent coud at k

Y_k The observation at k

X_km1 the cloud from at km1

Returns:

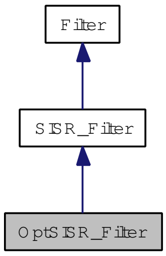
The sum of the weights

Implements SI_Sampler.

8.29 OptSISR_Filter Class Reference

#include <sisr_filter.h>

Inheritance diagram for OptSISR_Filter:



Public Member Functions

- OptSISR_Filter (void)
- ∼OptSISR_Filter (void)
- OptSISR_Filter (const int &Ns, Opt_Simulator ∗m)
- OptSISR_Filter (const int &Ns, Gaussian_Nonlinear_Model ∗m)

Private Attributes

- Opt_Simulator ∗ sim

8.29.1 Constructor & Destructor Documentation

8.29.1.1 OptSISR_Filter::OptSISR_Filter (void)

8.29.1.2 OptSISR_Filter::∼OptSISR_Filter (void)

8.29.1.3 OptSISR_Filter::OptSISR_Filter (const int & *Ns*, Opt_Simulator ∗ *m*)

8.29.1.4 OptSISR_Filter::OptSISR_Filter (const int & *Ns*, Gaussian_Nonlinear_Model ∗ *m*)

8.29.2 Member Data Documentation

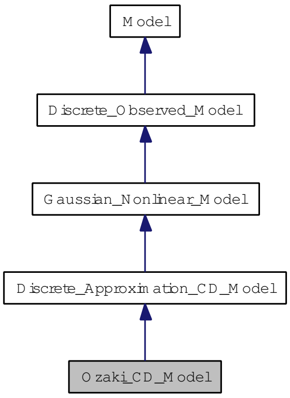
8.29.2.1 Opt_Simulator∗ OptSISR_Filter::sim [private]

8.30 Ozaki_CD_Model Class Reference

continuous discret model: the state SDE is discretly approximate by Ozaki method

`#include <gaussian_model.h>`

Inheritance diagram for Ozaki_CD_Model:



Public Member Functions

- Ozaki_CD_Model (void)
- Ozaki_CD_Model (Continuous_Discrete_Model *m, const int &a)
- dcovector Scheme (const dcovector &X, const dcovector &W)
- dgematrix Jx_Scheme (const dcovector &X, const dcovector &W)
- dgematrix Jw_Scheme (const dcovector &X, const dcovector &W)
- void Get_Linear_Scheme (const dcovector &X, const dcovector &W, dgematrix &F, dgematrix &J, dcovector &Xp)

  Get the Linearized parameters Scheme in X,W.

8.30.1 Detailed Description

continuous discret model: the state SDE is discretly approximate by Ozaki method

8.30.2 Constructor & Destructor Documentation

8.30.2.1 Ozaki_CD_Model::Ozaki_CD_Model (void)

8.30.2.2 Ozaki_CD_Model::Ozaki_CD_Model (Continuous_Discrete_Model * *m*, const int & *a*)

8.30.3 Member Function Documentation

8.30.3.1 void Ozaki_CD_Model::Get_Linear_Scheme (const dcovector & *X*, const dcovector & *W*, dgematrix & *F*, dgematrix & *J*, dcovector & *Xp*) `[virtual]`

Get the Linearized parameters Scheme in X,W.

Parameters:

X The state value

W The noise value

F The jacobian of f(X,W) in X

G The jacobian in f(X,W) in W

Xp The prediction Xp = f(X,W)

Implements Discrete_Approximation_CD_Model.

8.30.3.2 dgematrix Ozaki_CD_Model::Jw_Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

8.30.3.3 dgematrix Ozaki_CD_Model::Jx_Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

8.30.3.4 dcovector Ozaki_CD_Model::Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

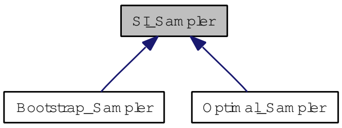
Implements Discrete_Approximation_CD_Model.

8.31 SI_Sampler Class Reference

the sequential importance sampler used for sisr filter (bootstrap,optimal ...)

```
#include <sisr_filter.h>
```

Inheritance diagram for SI_Sampler:



Public Member Functions

- SI_Sampler (void)

  The constructor.

- SI_Sampler (Simulator ∗m)

  constructor

- virtual vector< Weighted_Sample > DrawInitCloud (const int &NbSample)=0

  draw a set of possible init state

- virtual vector< Weighted_Sample > Draw (const dcovector &Y_k, const vector< Weighted_Sample > &X_km1)=0

  Draw a set of samples from the importance density Xk given Y0:k X0:k-1.

- virtual long double Weight (vector< Weighted_Sample > &cloud, const dcovector &Y_k, const vector< Weighted_Sample > &X_km1)=0

  Modify the weights of cloud for the weighting step in the sisr.

Public Attributes

- Simulator ∗ model

8.31.1 Detailed Description

the sequential importance sampler used for sisr filter (bootstrap,optimal ...)

8.31.2 Constructor & Destructor Documentation

8.31.2.1 SI_Sampler::SI_Sampler (void)

The constructor.

8.31.2.2 SI_Sampler::SI_Sampler (Simulator $*$ *m*)

constructor

The constructor

Parameters:

> ***m*** A discrete model

8.31.3 Member Function Documentation

8.31.3.1 virtual vector<Weighted_Sample > SI_Sampler::Draw (const dcovector & *Y_k*, const vector< Weighted_Sample > & *X_km1*) `[pure virtual]`

Draw a set of samples from the importance density Xk given Y0:k X0:k-1.

Parameters:

> ***Y_k*** The observation from 0 to k
>
> ***X_km1*** The cloud from 0 to km1

Returns:

> A cloud representing the importance density q(Xk|Y0:k,X0:k-1)

Implemented in Bootstrap_Sampler, and Optimal_Sampler.

8.31.3.2 virtual vector<Weighted_Sample > SI_Sampler::DrawInitCloud (const int & *NbSample*) `[pure virtual]`

draw a set of possible init state

Parameters:

> ***NbSample*** Number of sample

Returns:

> A set of weighted samples

Implemented in Bootstrap_Sampler, and Optimal_Sampler.

8.31.3.3 virtual long double SI_Sampler::Weight (vector< Weighted_Sample > & *cloud*, const dcovector & *Y_k*, const vector< Weighted_Sample > & *X_km1*) `[pure virtual]`

Modify the weights of cloud for the weighting step in the sisr.

Parameters:

> ***cloud*** The curent coud at k
>
> ***Y_k*** The observation at k
>
> ***X_km1*** the cloud from at km1

Returns:

The sum of the weights

Implemented in Bootstrap_Sampler, and Optimal_Sampler.

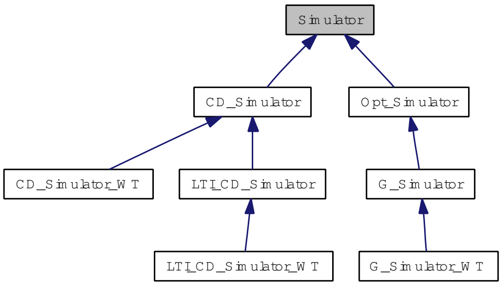
8.31.4 Member Data Documentation

8.31.4.1 Simulator∗ SI_Sampler::model

8.32 Simulator Class Reference

`#include <simulator.h>`

Inheritance diagram for Simulator:



Public Member Functions

- Simulator (void)
- ∼Simulator (void)
- void Set_Seed (const int &s)
- virtual dcovector Draw_Init (void)=0

  Draw a sample from p(X0).

- virtual dcovector Draw_Transition (const dcovector &Xkm1)=0

  Draw a sample from the transition densisty p(Xk|Xk-1).

- virtual dcovector Draw_Observation (const dcovector &Xk)=0

  Calculate the value of the density of probability of Y given X : p(Y|X).

- virtual long double Observation_Density (const dcovector &Y, const dcovector &X)=0

  calculate the value of the density of probability of Y given X : p(Y|X)

- virtual void Simulate (const int &N)

  simulate the markovian model

- virtual void Update (void)

  Update the simulation of the markovian model.

- virtual int Save_X (const char ∗filename)

  Save the simulated state trajectory in filename.

- virtual int Save_Y (const char ∗filename)

  Save the simulated observation trajectory in filename.

- void Clear (void)

  Clear the simulated trajectory X and Y.

Public Attributes

- Model ∗ model
- vector< dcovector > X
- vector< dcovector > Y
- dcovector(∗ b)(void ∗p, gsl_rng ∗rng)

  A pointer for stochastic input.

Protected Member Functions

- virtual void _update (void)

Protected Attributes

- gsl_rng ∗ r

8.32.1 Constructor & Destructor Documentation

8.32.1.1 Simulator::Simulator (void)

8.32.1.2 Simulator::∼Simulator (void)

8.32.2 Member Function Documentation

8.32.2.1 virtual void Simulator::_update (void) `[protected, virtual]`

Reimplemented in CD_Simulator.

8.32.2.2 void Simulator::Clear (void)

Clear the simulated trajectory X and Y.

8.32.2.3 virtual dcovector Simulator::Draw_Init (void) `[pure virtual]`

Draw a sample from p(X0).

Returns:

A sample from p(X0)

Implemented in G_Simulator, G_Simulator_WT, CD_Simulator, CD_Simulator_WT, and LTI_CD_-Simulator_WT.

8.32.2.4 virtual dcovector Simulator::Draw_Observation (const dcovector & *Xk*) `[pure virtual]`

Calculate the value of the density of probability of Y given X : p(Y|X).

Parameters:

Xk The state at k

Returns:

The simulated observation

Implemented in G_Simulator, and CD_Simulator.

8.32.2.5 virtual dcovector Simulator::Draw_Transition (const dcovector & *Xkm1*) `[pure virtual]`

Draw a sample from the transition densisty p(Xk|Xk-1).

Parameters:

Xkm1 X(k-1) the preceding state

Returns:

Implemented in G_Simulator, and CD_Simulator.

8.32.2.6 virtual long double Simulator::Observation_Density (const dcovector & *Y*, const dcovector & *X*) `[pure virtual]`

calculate the value of the density of probability of Y given X : p(Y|X)

Parameters:

Y The osbervation

X The state

Returns:

The value of the density

Implemented in G_Simulator, and CD_Simulator.

8.32.2.7 virtual int Simulator::Save_X (const char ∗ *filename*) `[virtual]`

Save the simulated state trajectory in filename.

Parameters:

filename The file

Returns:

0 if it's ok

Reimplemented in CD_Simulator.

8.32.2.8 virtual int Simulator::Save_Y (const char * *filename*) [virtual]

Save the simulated observation trajectory in filename.

Parameters:

filename The file

Returns:

0 if it's ok

Reimplemented in CD_Simulator.

8.32.2.9 void Simulator::Set_Seed (const int & *s*)

8.32.2.10 virtual void Simulator::Simulate (const int & *N*) [virtual]

simulate the markovian model

Parameters:

N The duration

X The state trajectory

Y The output

8.32.2.11 virtual void Simulator::Update (void) [virtual]

Update the simulation of the markovian model.

Parameters:

N The duration

X The state trajectory

Y The output

8.32.3 Member Data Documentation

8.32.3.1 dcovector(* Simulator::b)(void *p, gsl_rng *rng)

A pointer for stochastic input.

8.32.3.2 Model* Simulator::model

8.32.3.3 gsl_rng* Simulator::r [protected]

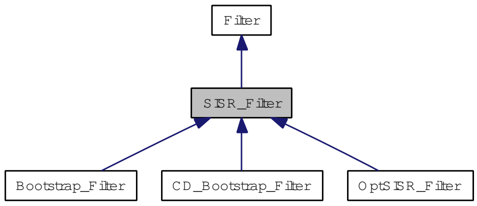
8.32.3.4 vector<dcovector> Simulator::X

8.32.3.5 vector<dcovector> Simulator::Y

8.33 SISR_Filter Class Reference

`#include <sisr_filter.h>`

Inheritance diagram for SISR_Filter:



Public Member Functions

- SISR_Filter (void)

  A constructor.

- ~SISR_Filter (void)

  The destructor.

- SISR_Filter (const int &Ns, SI_Sampler ∗s)

  A constructor.

- SISR_Filter (const int &Ns, const double &rc, const int &seed, SI_Sampler ∗s)

  A constructor.

- void SetSeed (const int &s)

  Set the seed of the random number generator of the discret pdf.

- void Resampling (const int &Ns)

  The resampling step.

- vector< Weighted_Sample > CloudGet (void)

  get The current cloud

- void SetRc (const float &rc)
- dcovector Expected_Get (void)

Public Attributes

- vector< Weighted_Sample > cloud_km1

  The particle clouds at km1.

- vector< Weighted_Sample > cloud

  The curent particle cloud.

- int NbSample

Number of particle.

- float Rc

  the resampling criterion

- SI_Sampler ∗ Sys

  the sampler

Protected Member Functions

- int _update (const dcovector &Yk)
- int _init (void)

  to initialized the first particle cloud of p(X0)

Protected Attributes

- gsl_rng ∗ r
- int seed

8.33.1 Constructor & Destructor Documentation

8.33.1.1 SISR_Filter::SISR_Filter (void)

A constructor.

8.33.1.2 SISR_Filter::∼SISR_Filter (void)

The destructor.

8.33.1.3 SISR_Filter::SISR_Filter (const int & *Ns*, SI_Sampler ∗ *s*)

A constructor.

Parameters:

Ns number of sample

s a sampler

Returns:

8.33.1.4 SISR_Filter::SISR_Filter (const int & *Ns*, const double & *rc*, const int & *seed*, SI_Sampler $*$ *s*)

A constructor.

Parameters:

Ns number of sample

rc The resampling criterion

seed The seed

s a sampler

Returns:

8.33.2 Member Function Documentation

8.33.2.1 int SISR_Filter::_init (void) `[protected, virtual]`

to initialized the first particle cloud of p(X0)

Implements Filter.

8.33.2.2 int SISR_Filter::_update (const dcovector & *Y*) `[protected, virtual]`

Specific update for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

Implements Filter.

8.33.2.3 vector$<$Weighted_Sample $>$ SISR_Filter::CloudGet (void)

get The current cloud

8.33.2.4 dcovector SISR_Filter::Expected_Get (void) `[virtual]`

Evaluate the current estimation of the state

Returns:

$\hat{X}_{k|k}$

Implements Filter.

8.33.2.5 void SISR_Filter::Resampling (const int & *Ns*)

The resampling step.

Parameters:

Ns

8.33.2.6 void SISR_Filter::SetRc (const float & *rc*)

Set the Resampling Criterion

Parameters:

rc The resampling Criterion

8.33.2.7 void SISR_Filter::SetSeed (const int & *s*)

Set the seed of the random number generator of the discret pdf.

Parameters:

s The seed

8.33.3 Member Data Documentation

8.33.3.1 vector<Weighted_Sample > SISR_Filter::cloud

The curent particle cloud.

8.33.3.2 vector<Weighted_Sample > SISR_Filter::cloud_km1

The particle clouds at km1.

8.33.3.3 int SISR_Filter::NbSample

Number of particle.

8.33.3.4 gsl_rng* SISR_Filter::r `[protected]`

8.33.3.5 float SISR_Filter::Rc

the resampling criterion

8.33.3.6 int SISR_Filter::seed `[protected]`

8.33.3.7 SI_Sampler* SISR_Filter::Sys

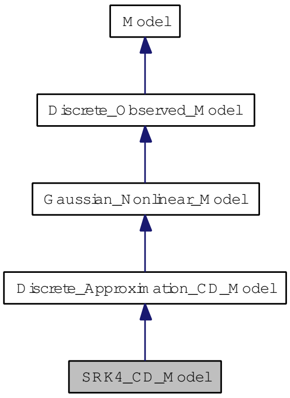
the sampler

8.34 SRK4_CD_Model Class Reference

continuous discret model: the state SDE is discretly approximate by an Sstochastic runge kutta method

`#include <gaussian_model.h>`

Inheritance diagram for SRK4_CD_Model:



Public Member Functions

- SRK4_CD_Model (void)
- SRK4_CD_Model (Continuous_Discrete_Model *m, const int &a)
- dcovector Scheme (const dcovector &X, const dcovector &W)
- void Get_Linear_Scheme (const dcovector &X, const dcovector &W, dgematrix &F, dgematrix &J, dcovector &Xp)

    Get the Linearized parameters Scheme in X,W.

- dgematrix Jx_Scheme (const dcovector &X, const dcovector &W)
- dgematrix Jw_Scheme (const dcovector &X, const dcovector &W)

8.34.1 Detailed Description

continuous discret model: the state SDE is discretly approximate by an Sstochastic runge kutta method

8.34.2 Constructor & Destructor Documentation

8.34.2.1 SRK4_CD_Model::SRK4_CD_Model (void)

8.34.2.2 SRK4_CD_Model::SRK4_CD_Model (Continuous_Discrete_Model * *m*, const int & *a*)

8.34.3 Member Function Documentation

8.34.3.1 void SRK4_CD_Model::Get_Linear_Scheme (const dcovector & *X*, const dcovector & *W*, dgematrix & *F*, dgematrix & *J*, dcovector & *Xp*) [virtual]

Get the Linearized parameters Scheme in X,W.

Parameters:

- ***X*** The state value
- ***W*** The noise value
- ***F*** The jacobian of f(X,W) in X
- ***G*** The jacobian in f(X,W) in W
- ***Xp*** The prediction Xp = f(X,W)

Implements Discrete_Approximation_CD_Model.

8.34.3.2 dgematrix SRK4_CD_Model::Jw_Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

8.34.3.3 dgematrix SRK4_CD_Model::Jx_Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

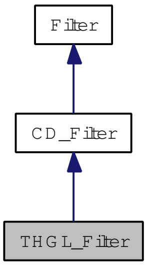
8.34.3.4 dcovector SRK4_CD_Model::Scheme (const dcovector & *X*, const dcovector & *W*) `[virtual]`

Implements Discrete_Approximation_CD_Model.

8.35 THGL_Filter Class Reference

#include <thgl_filter.h>

Inheritance diagram for THGL_Filter:



Public Member Functions

- THGL_Filter (void)
- THGL_Filter (Continuous_Discrete_Model ∗m)

Protected Member Functions

- int _update (const dcovector &Y)

8.35.1 Constructor & Destructor Documentation

8.35.1.1 THGL_Filter::THGL_Filter (void)

8.35.1.2 THGL_Filter::THGL_Filter (Continuous_Discrete_Model ∗ *m*)

8.35.2 Member Function Documentation

8.35.2.1 int THGL_Filter::_update (const dcovector & *Y*) [protected, virtual]

Specific update for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

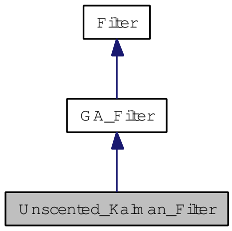
Implements Filter.

8.36 Unscented_Kalman_Filter Class Reference

The Discrete Unscented Kalman Filter (UKF).

`#include <unscented_kalman_filter.h>`

Inheritance diagram for Unscented_Kalman_Filter:



Public Member Functions

- Unscented_Kalman_Filter (void)

  A constructor.

- Unscented_Kalman_Filter (Gaussian_Nonlinear_Model *model)

  The constructor.

Public Attributes

- float lambda

  A scaled parameter.

Protected Member Functions

- int SP_Init (void)

  Initialize the sigma points at each update step.

- int U_Cov (const vector< dcovector > &sP1, const dcovector &m1, const vector< dcovector > &sP2, const dcovector &m2, dgematrix &cov)

  Calculate the covaraince between two sets of sigma points.

- int U_Mean (const vector< dcovector > &sP, dcovector &mean)

  Calculate the mean of a set of sigma points.

- int _update (const dcovector &Y)
- int _init (void)

  Itialization of the UKF.

Private Attributes

- dgematrix sqrt_Qw

  The square root matrix (cholesky) of Qw.

- dgematrix sqrt_Qv

  The square root matrix (cholesky) of Qv.

- vector< dcovector > sX

  The sigma points for the state X.

- vector< dcovector > sW

  The sigma points for the state noise W.

- vector< dcovector > sY

  The sigma points for the observation.

- double w_0

  The first weight to compute the mean.

- double w_0c

  The first weight to compute the covariance.

- double w

  Other weights.

8.36.1 Detailed Description

The Discrete Unscented Kalman Filter (UKF).

8.36.2 Constructor & Destructor Documentation

8.36.2.1 Unscented_Kalman_Filter::Unscented_Kalman_Filter (void)

A constructor.

8.36.2.2 Unscented_Kalman_Filter::Unscented_Kalman_Filter (Gaussian_Nonlinear_Model * *model*)

The constructor.

Parameters:

model A gaussian non linear model

8.36.3 Member Function Documentation

8.36.3.1 int Unscented_Kalman_Filter::_init (void) `[protected, virtual]`

Itialization of the UKF.

Reimplemented from GA_Filter.

8.36.3.2 int Unscented_Kalman_Filter::_update (const dcovector & *Y*) `[protected, virtual]`

Specific update for each filter

Parameters:

Y The observed sample

Returns:

0 if no problem

Implements Filter.

8.36.3.3 int Unscented_Kalman_Filter::SP_Init (void) `[protected]`

Initialize the sigma points at each update step.

8.36.3.4 int Unscented_Kalman_Filter::U_Cov (const vector< dcovector > & *sP1*, const dcovector & *m1*, const vector< dcovector > & *sP2*, const dcovector & *m2*, dgematrix & *cov*) `[protected]`

Calculate the covaraince between two sets of sigma points.

Parameters:

sP1 The first set of sigma point

m1 The mean of the sigma point

sP2 The second set of sigma point

m2 The mean of the second set of sigma point

cov Return the empirical covariance matrix between two sets

Returns:

0 if dimensions are ok

8.36.3.5 int Unscented_Kalman_Filter::U_Mean (const vector< dcovector > & *sP*, dcovector & *mean*) `[protected]`

Calculate the mean of a set of sigma points.

Parameters:

sP a set of sigma point

mean Return the mean

Returns:

0 if dimensions are ok

8.36.4 Member Data Documentation

8.36.4.1 float Unscented_Kalman_Filter::lambda

A scaled parameter.

8.36.4.2 dgematrix Unscented_Kalman_Filter::sqrt_Qv `[private]`

The square root matrix (cholesky) of Qv.

8.36.4.3 dgematrix Unscented_Kalman_Filter::sqrt_Qw `[private]`

The square root matrix (cholesky) of Qw.

8.36.4.4 vector<dcovector> Unscented_Kalman_Filter::sW `[private]`

The sigma points for the state noise W.

8.36.4.5 vector<dcovector> Unscented_Kalman_Filter::sX `[private]`

The sigma points for the state X.

8.36.4.6 vector<dcovector> Unscented_Kalman_Filter::sY `[private]`

The sigma points for the observation.

8.36.4.7 double Unscented_Kalman_Filter::w `[private]`

Other weights.

8.36.4.8 double Unscented_Kalman_Filter::w_0 `[private]`

The first weight to compute the mean.

8.36.4.9 double Unscented_Kalman_Filter::w_0c `[private]`

The first weight to compute the covariance.

8.37 Weighted_Sample Class Reference

```
#include <sisr_filter.h>
```

Public Attributes

- dcovector Value

    The position.

- long double Weight

    The weight of the sample.

8.37.1 Member Data Documentation

8.37.1.1 dcovector Weighted_Sample::Value

The position.

8.37.1.2 long double Weighted_Sample::Weight

The weight of the sample.

Index