

# **BFilt**

## **1.0**

Generated by Doxygen 1.5.7

Wed Feb 25 14:57:28 2009



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	Description . . . . .	1
1.2	Dependances . . . . .	1
1.3	Installation . . . . .	1
1.4	Auteur . . . . .	2
<b>2</b>	<b>An AR process</b>	<b>3</b>
2.1	Define the AR model . . . . .	4
2.2	The main program . . . . .	5
2.3	The CMakeList.txt . . . . .	6
<b>3</b>	<b>An Ornstien-Uhlenbeck process</b>	<b>7</b>
3.1	The main program . . . . .	9
3.2	The CMakeList.txt . . . . .	10
<b>4</b>	<b>Van Der Pol oscillator</b>	<b>13</b>
<b>5</b>	<b>Terrain navigation</b>	<b>17</b>
<b>6</b>	<b>Class Index</b>	<b>23</b>
6.1	Class Hierarchy . . . . .	23
<b>7</b>	<b>Class Index</b>	<b>25</b>
7.1	Class List . . . . .	25
<b>8</b>	<b>Class Documentation</b>	<b>27</b>
8.1	Bootstrap_Filter Class Reference . . . . .	27
8.1.1	Constructor & Destructor Documentation . . . . .	28
8.1.1.1	Bootstrap_Filter . . . . .	28
8.1.1.2	~Bootstrap_Filter . . . . .	28
8.1.1.3	Bootstrap_Filter . . . . .	28

8.1.1.4	Bootstrap_Filter	28
8.1.2	Member Data Documentation	28
8.1.2.1	sim	28
8.2	Bootstrap_Sampler Class Reference	29
8.2.1	Detailed Description	29
8.2.2	Constructor & Destructor Documentation	29
8.2.2.1	Bootstrap_Sampler	29
8.2.2.2	Bootstrap_Sampler	29
8.2.3	Member Function Documentation	29
8.2.3.1	Draw	29
8.2.3.2	DrawInitCloud	30
8.2.3.3	Weight	30
8.3	CD_Bootstrap_Filter Class Reference	31
8.3.1	Constructor & Destructor Documentation	31
8.3.1.1	CD_Bootstrap_Filter	31
8.3.1.2	~CD_Bootstrap_Filter	31
8.3.1.3	CD_Bootstrap_Filter	31
8.3.1.4	CD_Bootstrap_Filter	31
8.3.1.5	CD_Bootstrap_Filter	31
8.3.2	Member Function Documentation	31
8.3.2.1	Save_X	31
8.3.3	Member Data Documentation	32
8.3.3.1	sim	32
8.4	CD_Extended_Kalman_Filter Class Reference	33
8.4.1	Constructor & Destructor Documentation	34
8.4.1.1	CD_Extended_Kalman_Filter	34
8.4.1.2	CD_Extended_Kalman_Filter	34
8.4.2	Member Function Documentation	34
8.4.2.1	_euler_prediction	34
8.4.2.2	_heun_prediction	34
8.4.2.3	_rk4_prediction	34
8.4.2.4	_rk4_prediction_FM	34
8.4.2.5	_thgl_prediction	34
8.4.2.6	_update	34
8.4.3	Member Data Documentation	34
8.4.3.1	Scheme	34

8.5	CD_Filter Class Reference	35
8.5.1	Detailed Description	36
8.5.2	Constructor & Destructor Documentation	36
8.5.2.1	CD_Filter	36
8.5.2.2	CD_Filter	36
8.5.3	Member Function Documentation	36
8.5.3.1	_init	36
8.5.3.2	Expected_Get	36
8.5.3.3	Save_X	37
8.5.4	Member Data Documentation	37
8.5.4.1	M	37
8.5.4.2	R	37
8.5.4.3	Rp	37
8.5.4.4	Xp	37
8.6	CD_Kalman Class Reference	38
8.6.1	Detailed Description	38
8.6.2	Constructor & Destructor Documentation	38
8.6.2.1	CD_Kalman	38
8.6.2.2	CD_Kalman	38
8.6.3	Member Function Documentation	38
8.6.3.1	_update	38
8.7	CD_Simulator Class Reference	40
8.7.1	Constructor & Destructor Documentation	41
8.7.1.1	CD_Simulator	41
8.7.1.2	CD_Simulator	41
8.7.2	Member Function Documentation	41
8.7.2.1	_update	41
8.7.2.2	Draw_Init	41
8.7.2.3	Draw_Observation	41
8.7.2.4	draw_state	41
8.7.2.5	Draw_Transition	42
8.7.2.6	Observation_Density	42
8.7.2.7	Save_X	42
8.7.2.8	Save_Y	42
8.7.2.9	Set_Alpha	43
8.7.2.10	Simulate	43

8.7.3	Member Data Documentation	43
8.7.3.1	_a	43
8.7.3.2	Dx	43
8.7.3.3	Dy	43
8.7.3.4	scheme	43
8.8	CD_Simulator_WT Class Reference	44
8.8.1	Constructor & Destructor Documentation	44
8.8.1.1	CD_Simulator_WT	44
8.8.1.2	CD_Simulator_WT	44
8.8.2	Member Function Documentation	44
8.8.2.1	Draw_Init	44
8.8.3	Member Data Documentation	45
8.8.3.1	T	45
8.8.3.2	TB	45
8.8.3.3	Xt	45
8.9	Continuous_Discrete_Model Class Reference	46
8.9.1	Detailed Description	46
8.9.2	Constructor & Destructor Documentation	47
8.9.2.1	Continuous_Discrete_Model	47
8.9.2.2	~Continuous_Discrete_Model	47
8.9.3	Member Function Documentation	47
8.9.3.1	Diffusion_Function	47
8.9.3.2	Drift_Function	47
8.9.3.3	Init	47
8.9.3.4	J_Drift_Function	47
8.9.4	Member Data Documentation	48
8.9.4.1	Ts	48
8.10	DD_Kalman Class Reference	49
8.10.1	Detailed Description	49
8.10.2	Constructor & Destructor Documentation	49
8.10.2.1	DD_Kalman	49
8.10.2.2	DD_Kalman	49
8.10.3	Member Function Documentation	49
8.10.3.1	_update	49
8.11	Discrete_Approximation_CD_Model Class Reference	51
8.11.1	Detailed Description	52

---

8.11.2	Constructor & Destructor Documentation	52
8.11.2.1	Discrete_Approximation_CD_Model	52
8.11.2.2	~Discrete_Approximation_CD_Model	52
8.11.2.3	Discrete_Approximation_CD_Model	52
8.11.2.4	Discrete_Approximation_CD_Model	52
8.11.3	Member Function Documentation	53
8.11.3.1	Get_Alpha	53
8.11.3.2	Get_Linear_Parameters	53
8.11.3.3	Get_Linear_Scheme	53
8.11.3.4	Init	53
8.11.3.5	J_Observation_Function	53
8.11.3.6	Jw_Scheme	54
8.11.3.7	Jw_State_Function	54
8.11.3.8	Jx_Scheme	54
8.11.3.9	Jx_State_Function	54
8.11.3.10	Observation_Function	55
8.11.3.11	Scheme	55
8.11.3.12	Set_Alpha	55
8.11.3.13	State_Function	55
8.11.4	Member Data Documentation	55
8.11.4.1	alpha	55
8.11.4.2	cd_model	55
8.12	Discrete_Observed_Model Class Reference	56
8.12.1	Detailed Description	57
8.12.2	Constructor & Destructor Documentation	57
8.12.2.1	Discrete_Observed_Model	57
8.12.2.2	~Discrete_Observed_Model	57
8.12.3	Member Function Documentation	57
8.12.3.1	Get_Init_Parameters	57
8.12.3.2	J_Observation_Function	58
8.12.3.3	Observation_Function	58
8.12.4	Member Data Documentation	58
8.12.4.1	Qv	58
8.12.4.2	Qw	58
8.12.4.3	R0	58
8.12.4.4	X0	58

---

8.13 Euler_CD_Model Class Reference . . . . .	59
8.13.1 Detailed Description . . . . .	59
8.13.2 Constructor & Destructor Documentation . . . . .	59
8.13.2.1 Euler_CD_Model . . . . .	59
8.13.2.2 Euler_CD_Model . . . . .	59
8.13.3 Member Function Documentation . . . . .	59
8.13.3.1 Get_Linear_Scheme . . . . .	59
8.13.3.2 Jw_Scheme . . . . .	60
8.13.3.3 Jx_Scheme . . . . .	60
8.13.3.4 Scheme . . . . .	60
8.14 Extended_Kalman_Filter Class Reference . . . . .	61
8.14.1 Constructor & Destructor Documentation . . . . .	61
8.14.1.1 Extended_Kalman_Filter . . . . .	61
8.14.1.2 Extended_Kalman_Filter . . . . .	61
8.14.2 Member Function Documentation . . . . .	61
8.14.2.1 _update . . . . .	61
8.15 Filter Class Reference . . . . .	62
8.15.1 Detailed Description . . . . .	63
8.15.2 Constructor & Destructor Documentation . . . . .	63
8.15.2.1 Filter . . . . .	63
8.15.2.2 ~Filter . . . . .	63
8.15.3 Member Function Documentation . . . . .	63
8.15.3.1 _init . . . . .	63
8.15.3.2 _update . . . . .	64
8.15.3.3 Expected_Get . . . . .	64
8.15.3.4 Filtering . . . . .	64
8.15.3.5 Init . . . . .	64
8.15.3.6 Likelihood_Get . . . . .	65
8.15.3.7 Save_X . . . . .	65
8.15.3.8 Update . . . . .	65
8.15.4 Member Data Documentation . . . . .	65
8.15.4.1 Likelihood . . . . .	65
8.15.4.2 model . . . . .	65
8.15.4.3 X . . . . .	65
8.16 G_Simulator Class Reference . . . . .	66
8.16.1 Constructor & Destructor Documentation . . . . .	67



8.16.1.1	G_Simulator	67
8.16.1.2	G_Simulator	67
8.16.2	Member Function Documentation	67
8.16.2.1	Draw_Init	67
8.16.2.2	Draw_Observation	67
8.16.2.3	Draw_Optimal	67
8.16.2.4	Draw_Transition	68
8.16.2.5	Obs_Optimal_Density	68
8.16.2.6	Observation_Density	68
8.17	G_Simulator_WT Class Reference	69
8.17.1	Constructor & Destructor Documentation	69
8.17.1.1	G_Simulator_WT	69
8.17.1.2	G_Simulator_WT	69
8.17.2	Member Function Documentation	69
8.17.2.1	Draw_Init	69
8.17.3	Member Data Documentation	70
8.17.3.1	N	70
8.17.3.2	NB	70
8.17.3.3	Xt	70
8.18	GA_Filter Class Reference	71
8.18.1	Detailed Description	72
8.18.2	Constructor & Destructor Documentation	72
8.18.2.1	GA_Filter	72
8.18.2.2	GA_Filter	72
8.18.3	Member Function Documentation	72
8.18.3.1	_init	72
8.18.3.2	Expected_Get	72
8.18.4	Member Data Documentation	73
8.18.4.1	M	73
8.18.4.2	R	73
8.18.4.3	Rp	73
8.18.4.4	Xp	73
8.19	Gaussian_Linear_Model Class Reference	74
8.19.1	Detailed Description	75
8.19.2	Constructor & Destructor Documentation	75
8.19.2.1	Gaussian_Linear_Model	75

8.19.3	Member Function Documentation	75
8.19.3.1	Get_Cov_Prediction	75
8.19.3.2	Get_Mean_Prediction	75
8.19.3.3	J_Observation_Function	75
8.19.3.4	Jw_State_Function	75
8.19.3.5	Jx_State_Function	75
8.19.3.6	Observation_Function	76
8.19.3.7	State_Function	76
8.19.4	Member Data Documentation	76
8.19.4.1	f	76
8.19.4.2	F	76
8.19.4.3	G	76
8.19.4.4	H	76
8.19.4.5	h	76
8.20	Gaussian_Nonlinear_Model Class Reference	77
8.20.1	Detailed Description	77
8.20.2	Constructor & Destructor Documentation	78
8.20.2.1	Gaussian_Nonlinear_Model	78
8.20.2.2	~Gaussian_Nonlinear_Model	78
8.20.3	Member Function Documentation	78
8.20.3.1	Get_Linear_Parameters	78
8.20.3.2	Init	78
8.20.3.3	Jw_State_Function	78
8.20.3.4	Jx_State_Function	78
8.20.3.5	State_Function	79
8.21	Heun_CD_Model Class Reference	80
8.21.1	Detailed Description	80
8.21.2	Constructor & Destructor Documentation	80
8.21.2.1	Heun_CD_Model	80
8.21.2.2	Heun_CD_Model	80
8.21.3	Member Function Documentation	80
8.21.3.1	Get_Linear_Scheme	80
8.21.3.2	Jw_Scheme	81
8.21.3.3	Jx_Scheme	81
8.21.3.4	Scheme	81
8.22	Linear_CD_Model Class Reference	82

8.22.1 Detailed Description . . . . .	83
8.22.2 Constructor & Destructor Documentation . . . . .	83
8.22.2.1 Linear_CD_Model . . . . .	83
8.22.3 Member Function Documentation . . . . .	83
8.22.3.1 Diffusion_Function . . . . .	83
8.22.3.2 Drift_Function . . . . .	83
8.22.3.3 Get_Cov_Prediction . . . . .	83
8.22.3.4 Get_Mean_Prediction . . . . .	83
8.22.3.5 Init . . . . .	83
8.22.3.6 J_Drift_Function . . . . .	83
8.22.3.7 J_Observation_Function . . . . .	84
8.22.3.8 Observation_Function . . . . .	84
8.22.4 Member Data Documentation . . . . .	84
8.22.4.1 A . . . . .	84
8.22.4.2 B . . . . .	84
8.22.4.3 C . . . . .	84
8.22.4.4 H . . . . .	84
8.22.4.5 h . . . . .	84
8.23 LL_Filter Class Reference . . . . .	85
8.23.1 Constructor & Destructor Documentation . . . . .	85
8.23.1.1 LL_Filter . . . . .	85
8.23.1.2 LL_Filter . . . . .	85
8.23.2 Member Function Documentation . . . . .	85
8.23.2.1 _update . . . . .	85
8.24 LTI_CD_Simulator Class Reference . . . . .	86
8.24.1 Constructor & Destructor Documentation . . . . .	86
8.24.1.1 LTI_CD_Simulator . . . . .	86
8.24.1.2 LTI_CD_Simulator . . . . .	86
8.24.2 Member Function Documentation . . . . .	86
8.24.2.1 draw_state . . . . .	86
8.25 LTI_CD_Simulator_WT Class Reference . . . . .	87
8.25.1 Constructor & Destructor Documentation . . . . .	87
8.25.1.1 LTI_CD_Simulator_WT . . . . .	87
8.25.1.2 LTI_CD_Simulator_WT . . . . .	87
8.25.2 Member Function Documentation . . . . .	87
8.25.2.1 Draw_Init . . . . .	87

8.25.3	Member Data Documentation	88
8.25.3.1	T	88
8.25.3.2	TB	88
8.25.3.3	Xt	88
8.26	Model Class Reference	89
8.26.1	Detailed Description	90
8.26.2	Constructor & Destructor Documentation	90
8.26.2.1	Model	90
8.26.2.2	~Model	90
8.26.3	Member Function Documentation	90
8.26.3.1	Clear	90
8.26.3.2	Get_Time	90
8.26.3.3	Update	90
8.26.4	Member Data Documentation	90
8.26.4.1	_k	90
8.27	Opt_Simulator Class Reference	91
8.27.1	Constructor & Destructor Documentation	91
8.27.1.1	Opt_Simulator	91
8.27.2	Member Function Documentation	91
8.27.2.1	Draw_Optimal	91
8.27.2.2	Obs_Optimal_Density	92
8.28	Optimal_Sampler Class Reference	93
8.28.1	Detailed Description	93
8.28.2	Constructor & Destructor Documentation	93
8.28.2.1	Optimal_Sampler	93
8.28.2.2	Optimal_Sampler	93
8.28.3	Member Function Documentation	93
8.28.3.1	Draw	93
8.28.3.2	DrawInitCloud	94
8.28.3.3	Weight	94
8.29	OptSISR_Filter Class Reference	95
8.29.1	Constructor & Destructor Documentation	95
8.29.1.1	OptSISR_Filter	95
8.29.1.2	~OptSISR_Filter	95
8.29.1.3	OptSISR_Filter	95
8.29.1.4	OptSISR_Filter	95

8.29.2	Member Data Documentation	95
8.29.2.1	sim	95
8.30	Ozaki_CD_Model Class Reference	96
8.30.1	Detailed Description	96
8.30.2	Constructor & Destructor Documentation	96
8.30.2.1	Ozaki_CD_Model	96
8.30.2.2	Ozaki_CD_Model	96
8.30.3	Member Function Documentation	96
8.30.3.1	Get_Linear_Scheme	96
8.30.3.2	Jw_Scheme	97
8.30.3.3	Jx_Scheme	97
8.30.3.4	Scheme	97
8.31	SI_Sampler Class Reference	98
8.31.1	Detailed Description	98
8.31.2	Constructor & Destructor Documentation	98
8.31.2.1	SI_Sampler	98
8.31.2.2	SI_Sampler	99
8.31.3	Member Function Documentation	99
8.31.3.1	Draw	99
8.31.3.2	DrawInitCloud	99
8.31.3.3	Weight	99
8.31.4	Member Data Documentation	100
8.31.4.1	model	100
8.32	Simulator Class Reference	101
8.32.1	Constructor & Destructor Documentation	102
8.32.1.1	Simulator	102
8.32.1.2	~Simulator	102
8.32.2	Member Function Documentation	102
8.32.2.1	_update	102
8.32.2.2	Clear	102
8.32.2.3	Draw_Init	102
8.32.2.4	Draw_Observation	102
8.32.2.5	Draw_Transition	103
8.32.2.6	Observation_Density	103
8.32.2.7	Save_X	103
8.32.2.8	Save_Y	104

8.32.2.9	Set_Seed . . . . .	104
8.32.2.10	Simulate . . . . .	104
8.32.2.11	Update . . . . .	104
8.32.3	Member Data Documentation . . . . .	104
8.32.3.1	b . . . . .	104
8.32.3.2	model . . . . .	104
8.32.3.3	r . . . . .	104
8.32.3.4	X . . . . .	104
8.32.3.5	Y . . . . .	104
8.33	SISR_Filter Class Reference . . . . .	105
8.33.1	Constructor & Destructor Documentation . . . . .	106
8.33.1.1	SISR_Filter . . . . .	106
8.33.1.2	~SISR_Filter . . . . .	106
8.33.1.3	SISR_Filter . . . . .	106
8.33.1.4	SISR_Filter . . . . .	107
8.33.2	Member Function Documentation . . . . .	107
8.33.2.1	_init . . . . .	107
8.33.2.2	_update . . . . .	107
8.33.2.3	CloudGet . . . . .	107
8.33.2.4	Expected_Get . . . . .	107
8.33.2.5	Resampling . . . . .	108
8.33.2.6	SetRc . . . . .	108
8.33.2.7	SetSeed . . . . .	108
8.33.3	Member Data Documentation . . . . .	108
8.33.3.1	cloud . . . . .	108
8.33.3.2	cloud_km1 . . . . .	108
8.33.3.3	NbSample . . . . .	108
8.33.3.4	r . . . . .	108
8.33.3.5	Rc . . . . .	108
8.33.3.6	seed . . . . .	108
8.33.3.7	Sys . . . . .	108
8.34	SRK4_CD_Model Class Reference . . . . .	109
8.34.1	Detailed Description . . . . .	109
8.34.2	Constructor & Destructor Documentation . . . . .	109
8.34.2.1	SRK4_CD_Model . . . . .	109
8.34.2.2	SRK4_CD_Model . . . . .	109

8.34.3	Member Function Documentation	109
8.34.3.1	Get_Linear_Scheme	109
8.34.3.2	Jw_Scheme	110
8.34.3.3	Jx_Scheme	110
8.34.3.4	Scheme	110
8.35	THGL_Filter Class Reference	111
8.35.1	Constructor & Destructor Documentation	111
8.35.1.1	THGL_Filter	111
8.35.1.2	THGL_Filter	111
8.35.2	Member Function Documentation	111
8.35.2.1	_update	111
8.36	Unscented_Kalman_Filter Class Reference	112
8.36.1	Detailed Description	113
8.36.2	Constructor & Destructor Documentation	113
8.36.2.1	Unscented_Kalman_Filter	113
8.36.2.2	Unscented_Kalman_Filter	113
8.36.3	Member Function Documentation	114
8.36.3.1	_init	114
8.36.3.2	_update	114
8.36.3.3	SP_Init	114
8.36.3.4	U_Cov	114
8.36.3.5	U_Mean	114
8.36.4	Member Data Documentation	115
8.36.4.1	lambda	115
8.36.4.2	sqrt_Qv	115
8.36.4.3	sqrt_Qw	115
8.36.4.4	sW	115
8.36.4.5	sX	115
8.36.4.6	sY	115
8.36.4.7	w	115
8.36.4.8	w_0	115
8.36.4.9	w_0c	115
8.37	Weighted_Sample Class Reference	116
8.37.1	Member Data Documentation	116
8.37.1.1	Value	116
8.37.1.2	Weight	116





# 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-discrete (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/include

```
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 :

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

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 simulated 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");

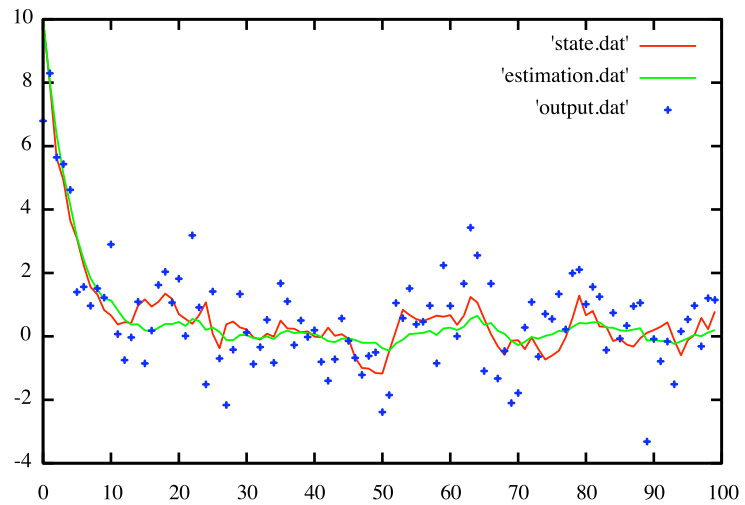
```

and the estimated state :

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

After compiling 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 Ornstein-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  $R_0 = \text{diag}[0, 3]$ .

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 simulated 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 :

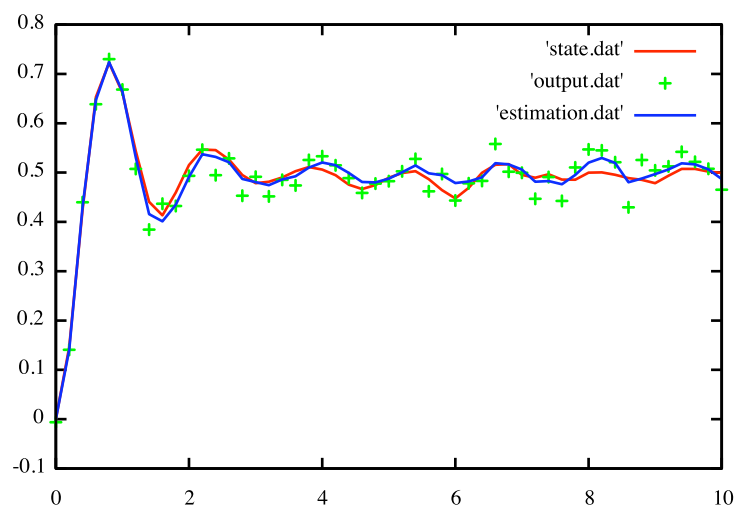
```
sim.Save_Y("output.dat");
sim.Save_X("state.dat");
```

and the estimated state :

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

After compiling 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.1);

    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.1,X.1);
```

```

        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.);

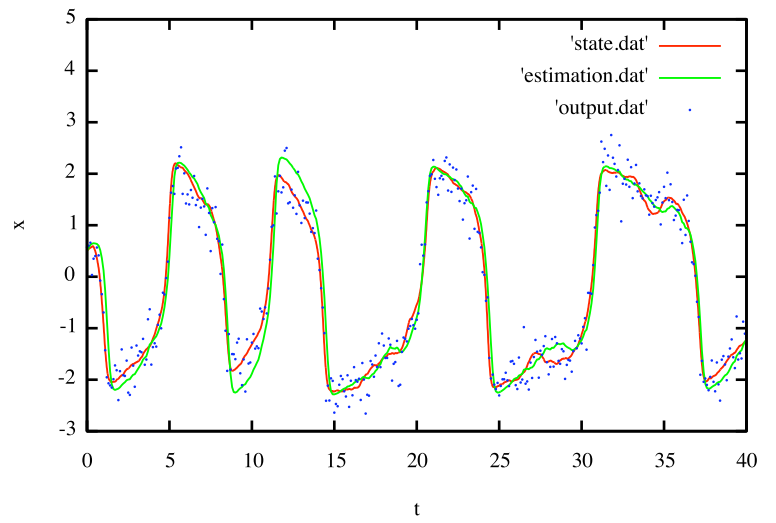
    // Filtering from the simulated output sim.Y
    filter.Filtering(sim.Y);

    // 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;
}

```



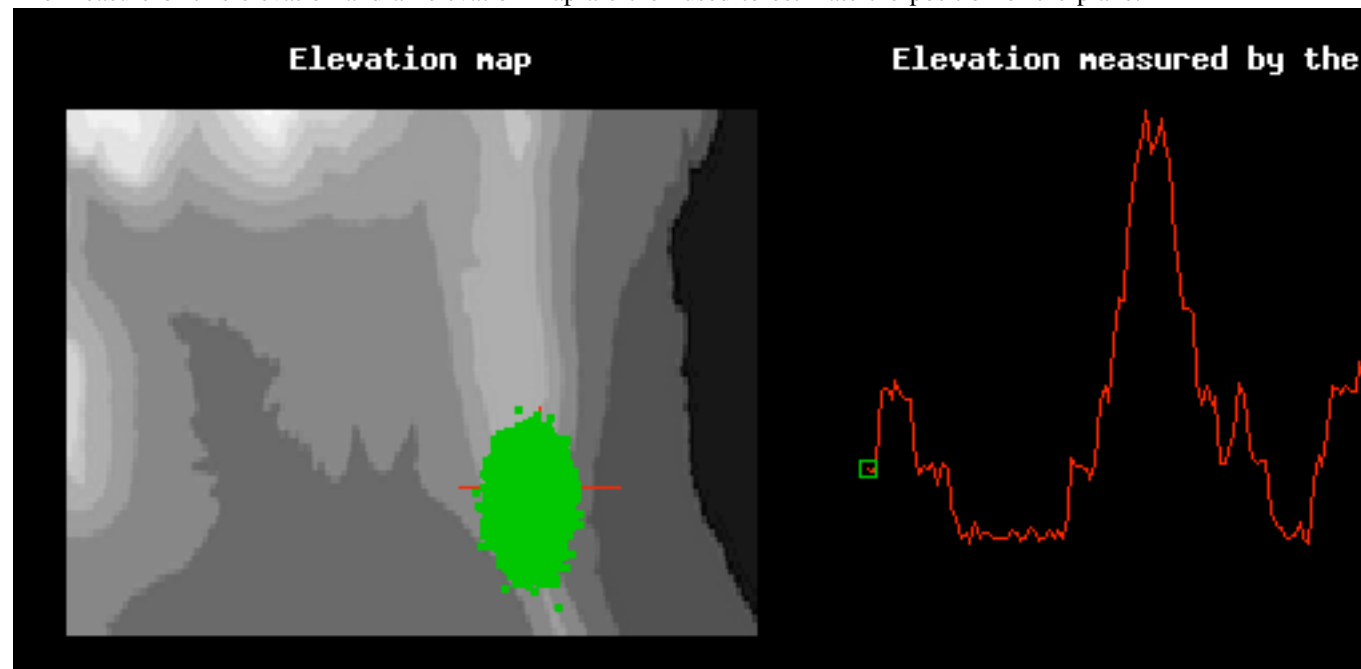
Results can be plotted (here with gnuplot):



## **Chapter 5**

# **Terrain navigation**

This example illustrates the performance of a particle filter compared to a highly non-linear filter. The problem here involves a plane whose trajectory is a brownian motion. This aircraft measures 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>ymin)
            ymin=y;
        if(y<ymin)
            ymax=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/sirs_filter.h>
#include "plane.h"

// This example illustrate performances of particle filter to highly non-linear
// filter. The problem 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 bootstrap 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:

Filter . . . . .	62
CD_Filter . . . . .	35
CD_Extended_Kalman_Filter . . . . .	33
CD_Kalman . . . . .	38
LL_Filter . . . . .	85
THGL_Filter . . . . .	111
GA_Filter . . . . .	71
DD_Kalman . . . . .	49
Extended_Kalman_Filter . . . . .	61
Unscented_Kalman_Filter . . . . .	112
SISR_Filter . . . . .	105
Bootstrap_Filter . . . . .	27
CD_Bootstrap_Filter . . . . .	31
OptSISR_Filter . . . . .	95
Model . . . . .	89
Discrete_Observed_Model . . . . .	56
Continuous_Discrete_Model . . . . .	46
Linear_CD_Model . . . . .	82
Gaussian_Nonlinear_Model . . . . .	77
Discrete_Approximation_CD_Model . . . . .	51
Euler_CD_Model . . . . .	59
Heun_CD_Model . . . . .	80
Ozaki_CD_Model . . . . .	96
SRK4_CD_Model . . . . .	109
Gaussian_Linear_Model . . . . .	74
SI_Sampler . . . . .	98
Bootstrap_Sampler . . . . .	29
Optimal_Sampler . . . . .	93
Simulator . . . . .	101
CD_Simulator . . . . .	40
CD_Simulator_WT . . . . .	44

LTI_CD_Simulator . . . . .	86
LTI_CD_Simulator_WT . . . . .	87
Opt_Simulator . . . . .	91
G_Simulator . . . . .	66
G_Simulator_WT . . . . .	69
Weighted_Sample . . . . .	116



# Chapter 7

## Class Index

### 7.1 Class List

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

<a href="#">Bootstrap_Filter</a> . . . . .	27
<a href="#">Bootstrap_Sampler</a> (This sampler use the transition as importance density ) . . . . .	29
<a href="#">CD_Bootstrap_Filter</a> . . . . .	31
<a href="#">CD_Extended_Kalman_Filter</a> . . . . .	33
<a href="#">CD_Filter</a> (Abstract class of continuous-discrete filters ) . . . . .	35
<a href="#">CD_Kalman</a> (The continuous-discrete kalman filter ) . . . . .	38
<a href="#">CD_Simulator</a> . . . . .	40
<a href="#">CD_Simulator_WT</a> . . . . .	44
<a href="#">Continuous_Discrete_Model</a> (Continuous Discrete <a href="#">Model</a> : The continuous state : $dX(t) = F(X)dt + G() * d\beta$ The discrete Observation $Y_k = H(X(tk)) + V_k$ ) . . . . .	46
<a href="#">DD_Kalman</a> (The discrete-discrete kalman filter ) . . . . .	49
<a href="#">Discrete_Approximation_CD_Model</a> (Continuous state equation is discretely approximate by $X(tk) = f'(X(tk-1), W_k)$ ) . . . . .	51
<a href="#">Discrete_Observed_Model</a> (Class of discretely observed model ) . . . . .	56
<a href="#">Euler_CD_Model</a> (Continuous discret model: the state SDE is discretely approximate by an Euler method ) . . . . .	59
<a href="#">Extended_Kalman_Filter</a> . . . . .	61
<a href="#">Filter</a> (Abstract class of all filters ) . . . . .	62
<a href="#">G_Simulator</a> . . . . .	66
<a href="#">G_Simulator_WT</a> . . . . .	69
<a href="#">GA_Filter</a> (Abstract class of Gaussian Approximation filters ) . . . . .	71
<a href="#">Gaussian_Linear_Model</a> (Gaussian Linear <a href="#">Model</a> : ) . . . . .	74
<a href="#">Gaussian_Nonlinear_Model</a> (Gaussian Nonlinear <a href="#">Model</a> The state : $X(k) = F(X_{k-1}, W_k)$ The Observation $Y(k) = H(X(k)) + V$ ) . . . . .	77
<a href="#">Heun_CD_Model</a> (Continuous discret model: the state SDE is discretely approximate by an Sstochastic Heun method ) . . . . .	80
<a href="#">Linear_CD_Model</a> (Linear continuous discrete model class of the form $dx = AX dt + Bdt + CdW$ $Y_k = HX(t_k) + h + V_k$ ) . . . . .	82
<a href="#">LL_Filter</a> . . . . .	85
<a href="#">LTI_CD_Simulator</a> . . . . .	86
<a href="#">LTI_CD_Simulator_WT</a> . . . . .	87
<a href="#">Model</a> (The class of time varying-models ) . . . . .	89
<a href="#">Opt_Simulator</a> . . . . .	91

<a href="#">Optimal_Sampler</a> (This sampler use the optimal importance density ) . . . . .	93
<a href="#">OptSISR_Filter</a> . . . . .	95
<a href="#">Ozaki_CD_Model</a> (Continuous discret model: the state SDE is discretly approximate by Ozaki method ) . . . . .	96
<a href="#">SI_Sampler</a> (Sequential importance sampler used for sisr filter (bootstrap,optimal ...) ) . . . . .	98
<a href="#">Simulator</a> . . . . .	101
<a href="#">SISR_Filter</a> . . . . .	105
<a href="#">SRK4_CD_Model</a> (Continuous discret model: the state SDE is discretly approximate by an Sstochastic runge kutta method ) . . . . .	109
<a href="#">THGL_Filter</a> . . . . .	111
<a href="#">Unscented_Kalman_Filter</a> (The Discrete Unscented Kalman <a href="#">Filter</a> (UKF) ) . . . . .	112
<a href="#">Weighted_Sample</a> . . . . .	116

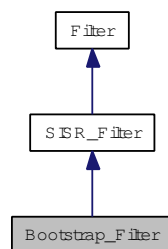
# Chapter 8

## 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

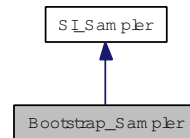
8.1.2.1 `Simulator* Bootstrap_Filter::sim` [private]

## 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  $q(X_k|Y_{0:k}, X_{0:k-1})$

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 sirs.

**Parameters:**

*cloud* The curent coud at k

*Y\_k* The observation at k

*X\_km1* the cloud from at km1

**Returns:**

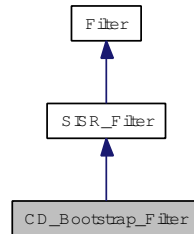
The sum of the weights

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

**Parameters:**

*filename*

**Returns:**

0 if everything is ok

Reimplemented from [Filter](#).

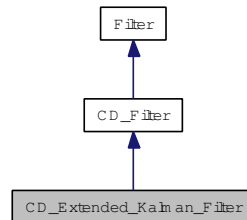
**8.3.3 Member Data Documentation****8.3.3.1** `CD_Simulator* CD_Bootstrap_Filter::sim` `[private]`



## 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

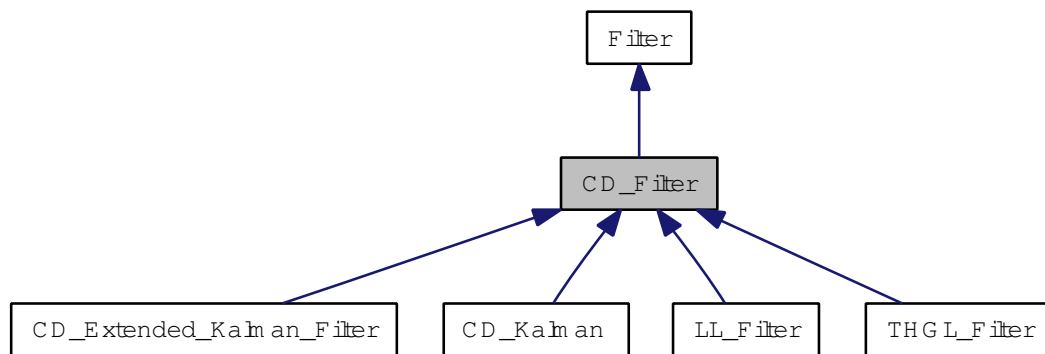
8.4.3.1 `int CD_Extended_Kalman_Filter::Scheme`

## 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 continuous-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 continuous-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 continuous-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  $\{\hat{X}_{k|k}, k = 0, \dots, N\}$

**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**

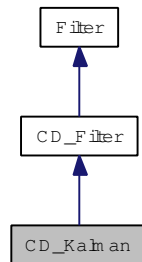
The prediction  $\hat{X}_{k-1|k} = E[X_{k-1} | Y_{0:k}]$ .

## 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:**

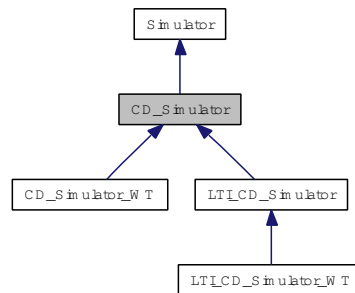
0 if no problem

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 &alpha=10)
- virtual dcovector [Draw\\_Init](#) (void)  
*Draw a sample from  $p(X_0)$ .*
- dcovector [Draw\\_Transition](#) (const dcovector &Xkm1)  
*Draw a sample from the transition density  $p(X_k|X_{k-1})$ .*
- 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  $p(X_0)$ .

### Returns:

A sample from  $p(X_0)$

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 density  $p(X_k|X_{k-1})$ .

**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 observation

*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

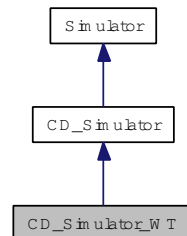
**8.7.3.3** double CD\_Simulator::Dy

**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  $p(X_0)$ .*

### 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  $p(X_0)$ .

#### Returns:

A sample from  $p(X_0)$

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]

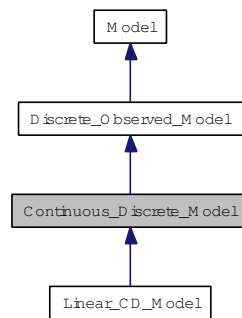
**8.8.3.3** `vector<dcovector> CD_Simulator_WT::Xt` [private]

## 8.9 Continuous\_Discrete\_Model Class Reference

Continuous Discrete [Model](#): The continuous state :  $dX(t) = F(X)dt + G() * d\beta$  The discrete Observation  $Y_k = H(X(tk)) + V_k$ .

```
#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  $T_s = t_k - t_{k-1}$ .*

#### 8.9.1 Detailed Description

Continuous Discrete [Model](#): The continuous state :  $dX(t) = F(X)dt + G() * d\beta$  The discrete Observation  $Y_k = H(X(tk)) + V_k$ .

## 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  $T_s = t_k - t_{k-1}$ .

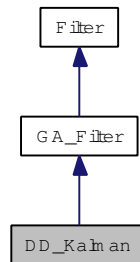


## 8.10 DD\_Kalman Class Reference

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

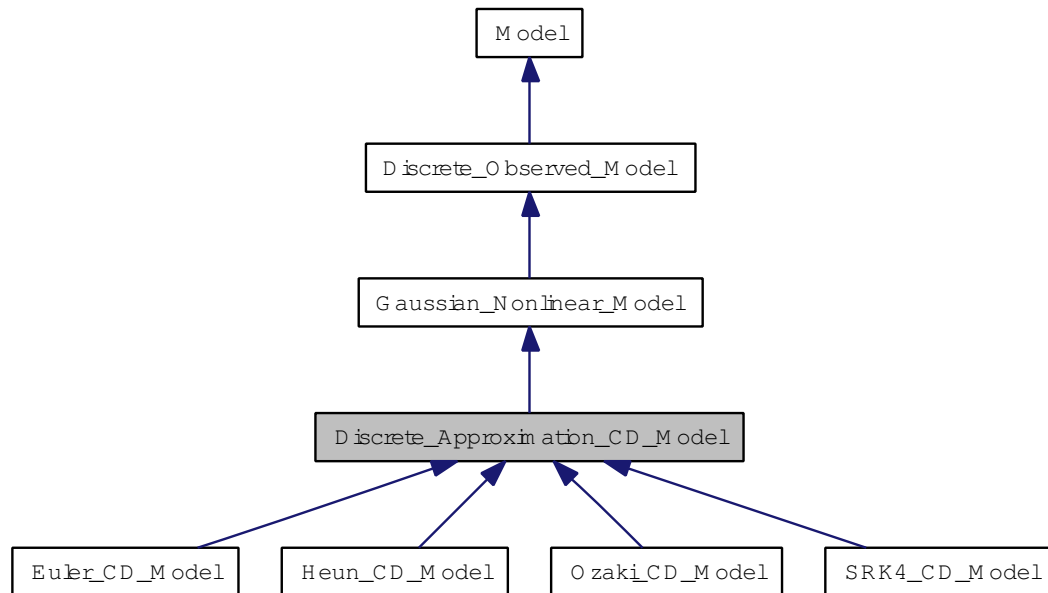
Implements [Filter](#).

## 8.11 Discrete\_Approximation\_CD\_Model Class Reference

the continuous state equation is discretely 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  $X_k = F(X_{k-1}, W_k)$ .*
- [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  $Y_k = H(X_k) + V_k$ .*

- 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  $T_d = T_s * a$  ( $T_s$  = sample duration of discrete observation)*

### 8.11.1 Detailed Description

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

### 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  $T_d = T_s * a$  ( $T_s$  = 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  $Y_k = H(X_k) + V_k$ .

#### 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  $X_k = F(X_{k-1}, W_k)$ .

#### 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  $T_d = T_s * a$  ( $T_s$  = sample duration of discrete observation)

### 8.11.4.2 `Continuous_Discrete_Model* Discrete_Approximation_CD_Model::cd_model` [protected]

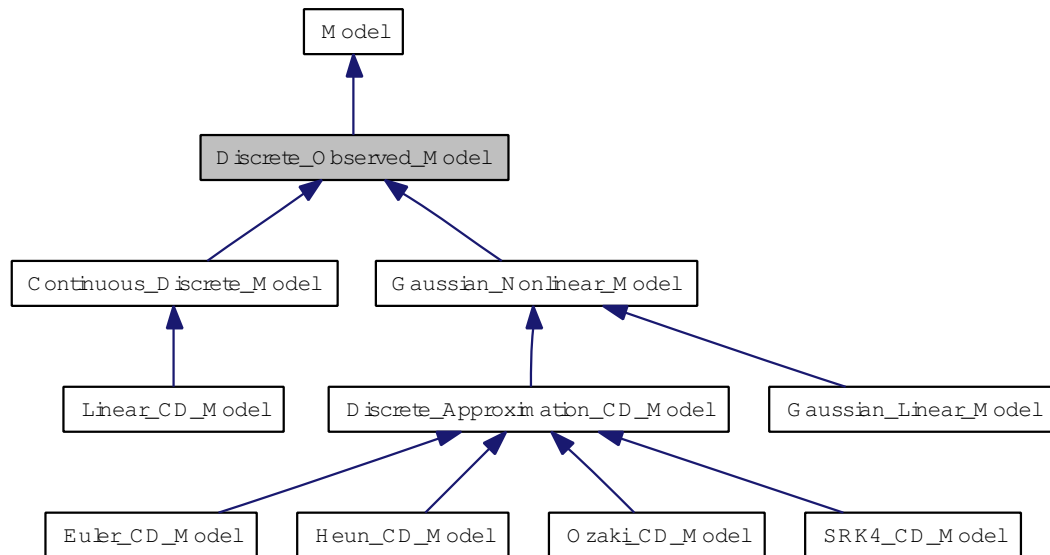
the continuous discrete model

## 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  $Y_k=H(X_k) + V_k$ .*
- 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  $p(X_0)$ .*

### 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  $W_k, V_k$  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 and 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  $Y_k = H(X_k) + V_k$ .

#### 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  $p(X_0)$ .

### 8.12.4.4 `dcovector Discrete_Observed_Model::X0 [protected]`

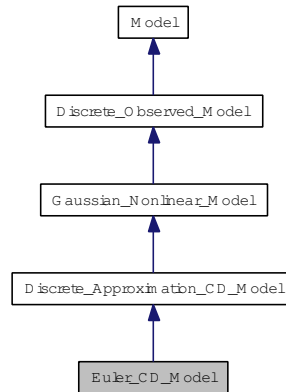
The mean of  $p(X_0)$ .

## 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](#).

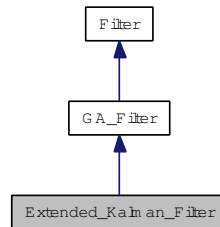
**8.13.3.4** `dcovector Euler_CD_Model::Scheme (const dcovector & X, const dcovector & W)`  
[virtual]

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

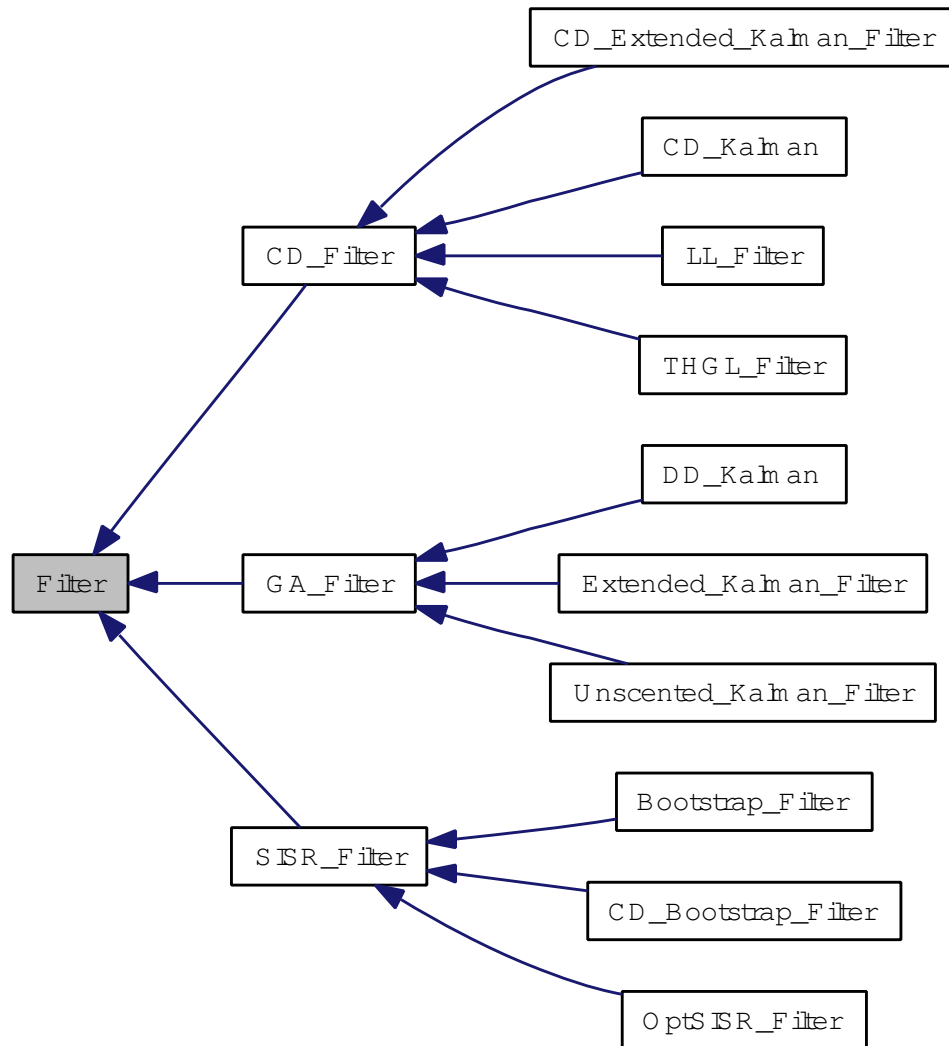
Implements [Filter](#).

## 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](#)  
 $\{\hat{X}_{k|k}, k = 0, \dots, N\}$

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

**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**

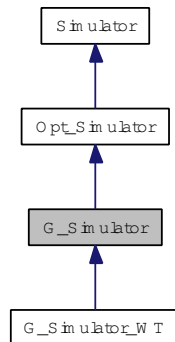
$$\{\hat{X}_{k|k}, k = 0, \dots, N\}$$

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  $p(X_0)$ .*
- dcovector [Draw\\_Transition](#) (const dcovector &Xkm1)  
*Draw a sample from the transition density  $p(X_k|X_{k-1})$ .*
- 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 density  $p(X_k|Y_k, X_{k-1})$ .*
- long double [Obs\\_Optimal\\_Density](#) (const dcovector &Yk, const dcovector &Xkm1)  
*calculate the value of the density of probability of Yk given Xk-1 :  $p(Y_k|X_{k-1})$*

## 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  $p(X_0)$ .

**Returns:**

A sample from  $p(X_0)$

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 density  $p(X_k|Y_k, X_{k-1})$ .

**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 density  $p(X_k|X_{k-1})$ .

**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  $Y_k$  given  $X_{k-1}$  :  $p(Y_k|X_{k-1})$

**Parameters:**

*Yk* the observation at  $k$

*Xkm1* The state at  $k-1$

**Returns:**

The value of the density  $p(Y_k|X_{k-1})$

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 observation

*X* The state

**Returns:**

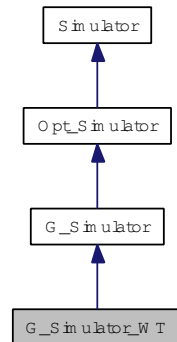
The value of the density

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  $p(X_0)$ .*

### 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  $p(X_0)$ .

#### Returns:

A sample from  $p(X_0)$

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]`

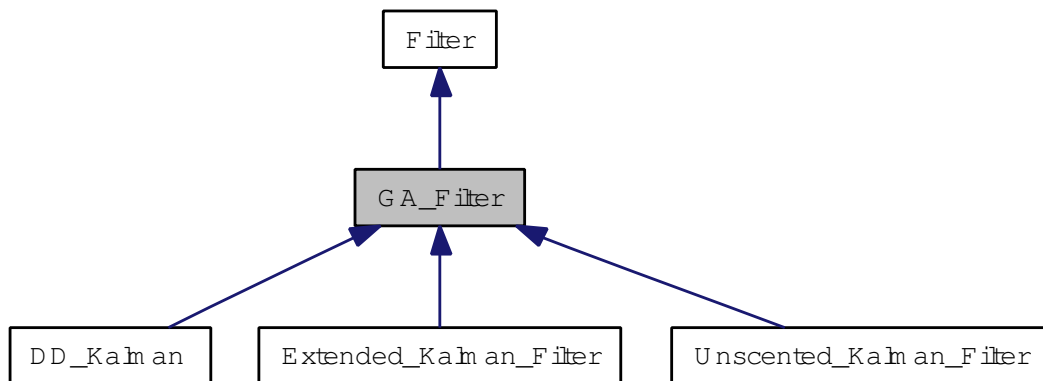
**8.17.3.3** `vector<dcovector> G_Simulator_WT::Xt` `[private]`

## 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

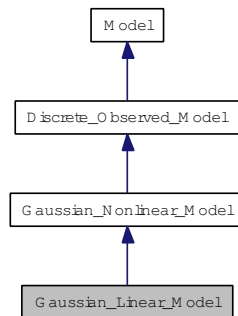
The prediction  $\hat{X}_{k-1|k} = E[X_{k-1} | Y_{0:k}]$ .

## 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  $X_k = F(X_{k-1}, W_k)$ .*
- 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  $Y_k = H(X_k) + V_k$ .*
- 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 :  $X(k) = F X(k-1) + f + G * W_k$  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

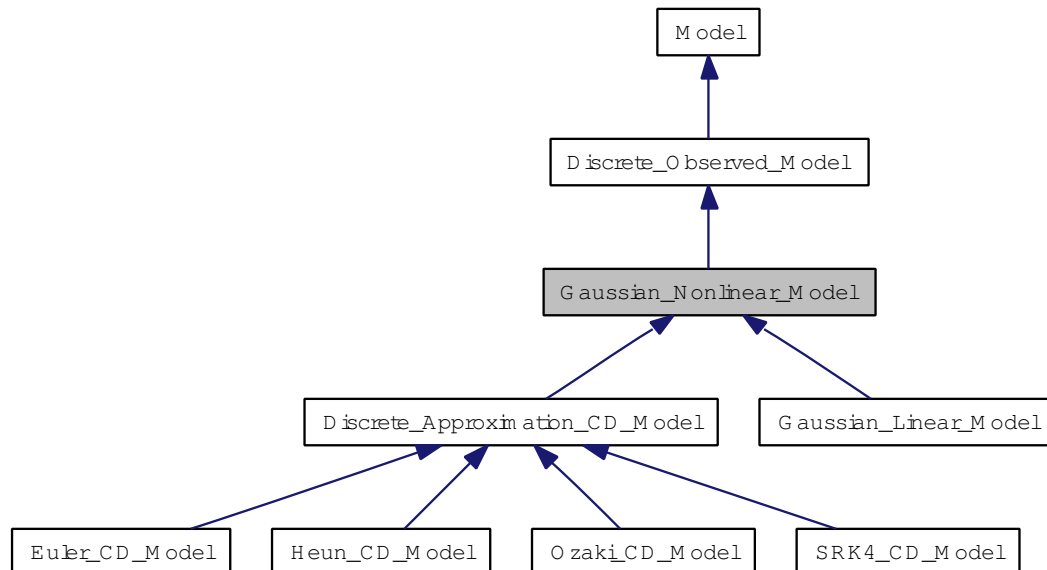
Reimplemented from [Gaussian\\_Nonlinear\\_Model](#).**8.19.3.6 dcovector Gaussian\_Linear\_Model::Observation\_Function (const dcovector &  $X$ )**  
[virtual]The observation  $Y_k = H(X_k) + V_k$ .**Parameters:** $X$  The state at  $k$ **Returns:**The observation at  $k$ Implements [Discrete\\_Observed\\_Model](#).**8.19.3.7 dcovector Gaussian\_Linear\_Model::State\_Function (const dcovector &  $X$ , const dcovector &  $W$ )** [virtual]The state  $X_k = F(X_{k-1}, W_k)$ .**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 :  $X(k) = F(X_{k-1}, W_k)$  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  $X_k = F(X_{k-1}, W_k)$ .*
- 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 :  $X(k) = F(X_{k-1}, W_k)$  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  $X_p = f(X,W)$

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  $X$

$W$  evaluate 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  $X_k = F(X_{k-1}, W_k)$ .

**Parameters:**

$X$  The state at  $k-1$

$W$  The Noise

**Returns:**

The state at  $k$

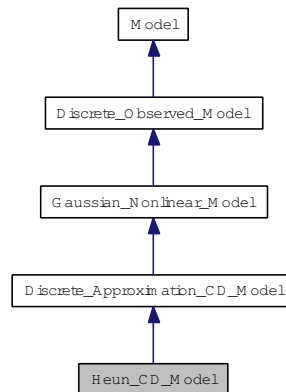
Implemented in [Gaussian\\_Linear\\_Model](#), and [Discrete\\_Approximation\\_CD\\_Model](#).

## 8.21 Heun\_CD\_Model Class Reference

continuous discret model: the state SDE is discretely 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 discretely 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]

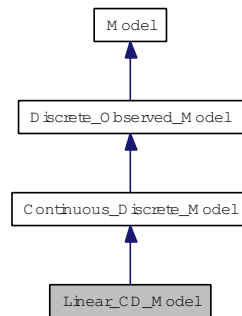
Implements [Discrete\\_Approximation\\_CD\\_Model](#).

## 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  $Y_k = H(X_k) + V_k$ .*
- 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:**

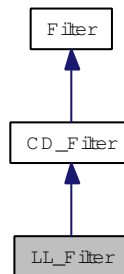
The jacobian matrix

Reimplemented from [Discrete\\_Observed\\_Model](#).**8.22.3.8 dcovector Linear\_CD\_Model::Observation\_Function (const dcovector &  $X$ )**  
[virtual]The observation  $Y_k = H(X_k) + V_k$ .**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:

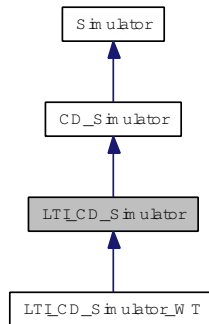
0 if no problem

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

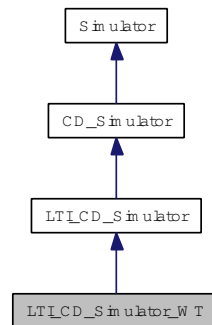
8.24.2.1 dcovector [LTI\\_CD\\_Simulator::draw\\_state](#) (const dcovector & *X*) [protected, virtual]

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  $p(X_0)$ .*

### 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  $p(X_0)$ .

#### Returns:

A sample from  $p(X_0)$

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]

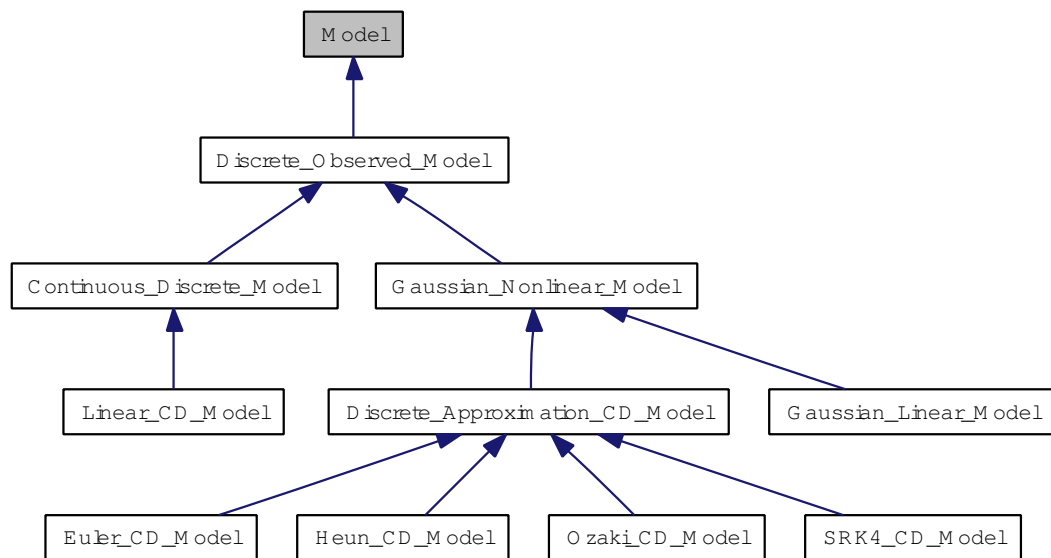


## 8.26 Model Class Reference

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

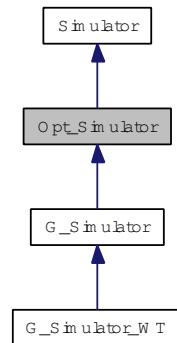
#### 8.26.4.1 `int Model::_k` `[protected]`

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  $p(X_k|Y_k, X_{k-1})$ .*
- 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 :  $p(Y_k|X_{k-1})$*

### 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  $p(X_k|Y_k, X_{k-1})$ .

#### 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  $Y_k$  given  $X_{k-1}$  :  $p(Y_k|X_{k-1})$

**Parameters:**

*Yk* the osbervation at k

*Xkm1* The state at k-1

**Returns:**

The value of the density  $p(Y_k|X_{k-1})$

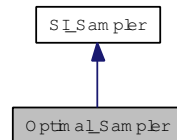
Implemented in [G\\_Simulator](#).

## 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  $q(X_k|Y_{0:k}, X_{0:k-1})$

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 sirs.

**Parameters:**

*cloud* The curent coud at k

*Y\_k* The observation at k

*X\_km1* the cloud from at km1

**Returns:**

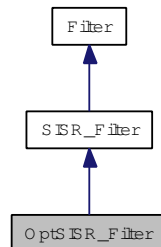
The sum of the weights

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

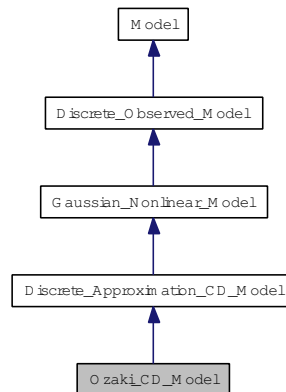
8.29.2.1 `Opt_Simulator* OptSISR_Filter::sim` [private]

## 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]

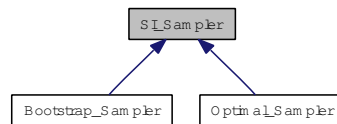
Implements [Discrete\\_Approximation\\_CD\\_Model](#).

## 8.31 SI\_Sampler Class Reference

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

```
#include <sirs_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  $X_k$  given  $Y_{0:k}$   $X_{0:k-1}$ .*
- 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 sirs.*

### Public Attributes

- [Simulator](#) \* model

### 8.31.1 Detailed Description

the sequential importance sampler used for sirs 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  $X_k$  given  $Y_{0:k}$   $X_{0:k-1}$ .

**Parameters:**

*Y\_k* The observation from 0 to k

*X\_km1* The cloud from 0 to km1

**Returns:**

A cloud representing the importance density  $q(X_k|Y_{0:k}, X_{0:k-1})$

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](#).

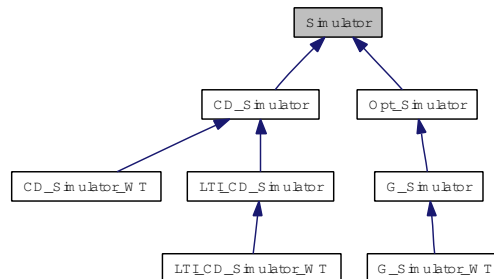
## 8.31.4 Member Data Documentation

### 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  $p(X_0)$ .*
- virtual dcovector [Draw\\_Transition](#) (const dcovector &Xkm1)=0  
*Draw a sample from the transition density  $p(X_k|X_{k-1})$ .*
- 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  $p(X_0)$ .

### Returns:

A sample from  $p(X_0)$

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 density  $p(X_k|X_{k-1})$ .

**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 observation

*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.

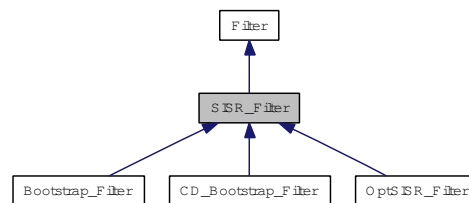
**8.32.3.2 Model\* Simulator::model****8.32.3.3 gsl\_rng\* Simulator::r [protected]****8.32.3.4 vector<dcovector> Simulator::X****8.32.3.5 vector<dcovector> Simulator::Y**



## 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\\_kml](#)  
*The particle clouds at kml.*
- 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  $p(X_0)$*

## 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 current 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**

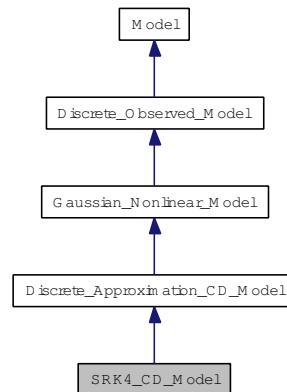
the sampler

## 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](#).

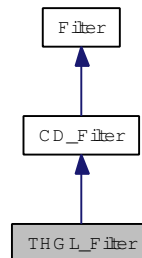
**8.34.3.4** `dcovector SRK4_CD_Model::Scheme (const dcovector & X, const dcovector & W)`  
[virtual]

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

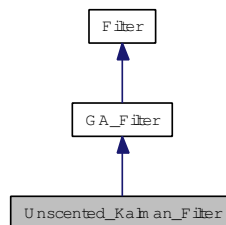
Implements [Filter](#).

## 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  $Q_w$ .*
- dgematrix [sqrt\\_Qv](#)  
*The square root matrix (cholesky) of  $Q_v$ .*
- 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

- ~Bootstrap\_Filter
  - Bootstrap\_Filter, 28
- ~CD\_Bootstrap\_Filter
  - CD\_Bootstrap\_Filter, 31
- ~Continuous\_Discrete\_Model
  - Continuous\_Discrete\_Model, 47
- ~Discrete\_Approximation\_CD\_Model
  - Discrete\_Approximation\_CD\_Model, 52
- ~Discrete\_Observed\_Model
  - Discrete\_Observed\_Model, 57
- ~Filter
  - Filter, 63
- ~Gaussian\_Nonlinear\_Model
  - Gaussian\_Nonlinear\_Model, 78
- ~Model
  - Model, 90
- ~OptSISR\_Filter
  - OptSISR\_Filter, 95
- ~SISR\_Filter
  - SISR\_Filter, 106
- ~Simulator
  - Simulator, 102
- \_a
  - CD\_Simulator, 43
- \_euler\_prediction
  - CD\_Extended\_Kalman\_Filter, 34
- \_heun\_prediction
  - CD\_Extended\_Kalman\_Filter, 34
- \_init
  - CD\_Filter, 36
  - Filter, 63
  - GA\_Filter, 72
  - SISR\_Filter, 107
  - Unscented\_Kalman\_Filter, 114
- \_k
  - Model, 90
- \_rk4\_\_prediction
  - CD\_Extended\_Kalman\_Filter, 34
- \_rk4\_\_prediction\_FM
  - CD\_Extended\_Kalman\_Filter, 34
- \_thgl\_prediction
  - CD\_Extended\_Kalman\_Filter, 34
- \_update
  - CD\_Extended\_Kalman\_Filter, 34
  - CD\_Kalman, 38

- CD\_Simulator, 41
- DD\_Kalman, 49
- Extended\_Kalman\_Filter, 61
- Filter, 64
- LL\_Filter, 85
- Simulator, 102
- SISR\_Filter, 107
- THGL\_Filter, 111
- Unscented\_Kalman\_Filter, 114

## A

- Linear\_CD\_Model, 84

## alpha

- Discrete\_Approximation\_CD\_Model, 55

## B

- Linear\_CD\_Model, 84

## b

- Simulator, 104

## Bootstrap\_Filter, 27

- ~Bootstrap\_Filter, 28
- Bootstrap\_Filter, 28
- Bootstrap\_Filter, 28
- sim, 28

## Bootstrap\_Sampler, 29

- Bootstrap\_Sampler, 29
- Bootstrap\_Sampler, 29
- Draw, 29
- DrawInitCloud, 30
- Weight, 30

## C

- Linear\_CD\_Model, 84

## CD\_Bootstrap\_Filter, 31

- ~CD\_Bootstrap\_Filter, 31
- CD\_Bootstrap\_Filter, 31
- CD\_Bootstrap\_Filter, 31
- Save\_X, 31
- sim, 32

## CD\_Extended\_Kalman\_Filter, 33

- \_euler\_prediction, 34
- \_heun\_prediction, 34
- \_rk4\_\_prediction, 34
- \_rk4\_\_prediction\_FM, 34
- \_thgl\_prediction, 34

- \_update, 34
  - CD\_Extended\_Kalman\_Filter, 34
  - CD\_Extended\_Kalman\_Filter, 34
  - Scheme, 34
- CD\_Filter, 35
  - \_init, 36
  - CD\_Filter, 36
  - CD\_Filter, 36
  - Expected\_Get, 36
  - M, 37
  - R, 37
  - Rp, 37
  - Save\_X, 36
  - Xp, 37
- CD\_Kalman, 38
  - \_update, 38
  - CD\_Kalman, 38
  - CD\_Kalman, 38
- cd\_model
  - Discrete\_Approximation\_CD\_Model, 55
- CD\_Simulator, 40
  - \_a, 43
  - \_update, 41
  - CD\_Simulator, 41
  - CD\_Simulator, 41
  - Draw\_Init, 41
  - Draw\_Observation, 41
  - draw\_state, 41
  - Draw\_Transition, 41
  - Dx, 43
  - Dy, 43
  - Observation\_Density, 42
  - Save\_X, 42
  - Save\_Y, 42
  - scheme, 43
  - Set\_Alpha, 42
  - Simulate, 43
- CD\_Simulator\_WT, 44
  - CD\_Simulator\_WT, 44
  - CD\_Simulator\_WT, 44
  - Draw\_Init, 44
  - T, 45
  - TB, 45
  - Xt, 45
- Clear
  - Model, 90
  - Simulator, 102
- cloud
  - SISR\_Filter, 108
- cloud\_kml
  - SISR\_Filter, 108
- CloudGet
  - SISR\_Filter, 107
- Continuous\_Discrete\_Model, 46
  - ~Continuous\_Discrete\_Model, 47
  - Continuous\_Discrete\_Model, 47
  - Continuous\_Discrete\_Model, 47
  - Diffusion\_Function, 47
  - Drift\_Function, 47
  - Init, 47
  - J\_Drift\_Function, 47
  - Ts, 48
- DD\_Kalman, 49
  - \_update, 49
  - DD\_Kalman, 49
  - DD\_Kalman, 49
- Diffusion\_Function
  - Continuous\_Discrete\_Model, 47
  - Linear\_CD\_Model, 83
- Discrete\_Approximation\_CD\_Model, 51
  - ~Discrete\_Approximation\_CD\_Model, 52
  - alpha, 55
  - cd\_model, 55
  - Discrete\_Approximation\_CD\_Model, 52
  - Discrete\_Approximation\_CD\_Model, 52
  - Get\_Alpha, 53
  - Get\_Linear\_Parameters, 53
  - Get\_Linear\_Scheme, 53
  - Init, 53
  - J\_Observation\_Function, 53
  - Jw\_Scheme, 54
  - Jw\_State\_Function, 54
  - Jx\_Scheme, 54
  - Jx\_State\_Function, 54
  - Observation\_Function, 54
  - Scheme, 55
  - Set\_Alpha, 55
  - State\_Function, 55
- Discrete\_Observed\_Model, 56
  - ~Discrete\_Observed\_Model, 57
  - Discrete\_Observed\_Model, 57
  - Discrete\_Observed\_Model, 57
  - Get\_Init\_Parameters, 57
  - J\_Observation\_Function, 57
  - Observation\_Function, 58
  - Qv, 58
  - Qw, 58
  - R0, 58
  - X0, 58
- Draw
  - Bootstrap\_Sampler, 29
  - Optimal\_Sampler, 93
  - SI\_Sampler, 99
- Draw\_Init
  - CD\_Simulator, 41
  - CD\_Simulator\_WT, 44
  - G\_Simulator, 67

- G\_Simulator\_WT, 69
- LTI\_CD\_Simulator\_WT, 87
- Simulator, 102
- Draw\_Observation
  - CD\_Simulator, 41
  - G\_Simulator, 67
  - Simulator, 102
- Draw\_Optimal
  - G\_Simulator, 67
  - Opt\_Simulator, 91
- draw\_state
  - CD\_Simulator, 41
  - LTI\_CD\_Simulator, 86
- Draw\_Transition
  - CD\_Simulator, 41
  - G\_Simulator, 67
  - Simulator, 103
- DrawInitCloud
  - Bootstrap\_Sampler, 30
  - Optimal\_Sampler, 94
  - SI\_Sampler, 99
- Drift\_Function
  - Continuous\_Discrete\_Model, 47
  - Linear\_CD\_Model, 83
- Dx
  - CD\_Simulator, 43
- Dy
  - CD\_Simulator, 43
- Euler\_CD\_Model, 59
  - Euler\_CD\_Model, 59
  - Euler\_CD\_Model, 59
  - Get\_Linear\_Scheme, 59
  - Jw\_Scheme, 60
  - Jx\_Scheme, 60
  - Scheme, 60
- Expected\_Get
  - CD\_Filter, 36
  - Filter, 64
  - GA\_Filter, 72
  - SISR\_Filter, 107
- Extended\_Kalman\_Filter, 61
  - \_update, 61
  - Extended\_Kalman\_Filter, 61
  - Extended\_Kalman\_Filter, 61
- F
  - Gaussian\_Linear\_Model, 76
- f
  - Gaussian\_Linear\_Model, 76
- Filter, 62
  - ~Filter, 63
  - \_init, 63
  - \_update, 64
- Expected\_Get, 64
- Filter, 63
- Filtering, 64
- Init, 64
- Likelihood, 65
- Likelihood\_Get, 64
- model, 65
- Save\_X, 65
- Update, 65
- X, 65
- Filtering
  - Filter, 64
- G
  - Gaussian\_Linear\_Model, 76
- G\_Simulator, 66
  - Draw\_Init, 67
  - Draw\_Observation, 67
  - Draw\_Optimal, 67
  - Draw\_Transition, 67
  - G\_Simulator, 67
  - G\_Simulator, 67
  - Obs\_Optimal\_Density, 68
  - Observation\_Density, 68
- G\_Simulator\_WT, 69
  - Draw\_Init, 69
  - G\_Simulator\_WT, 69
  - G\_Simulator\_WT, 69
  - N, 70
  - NB, 70
  - Xt, 70
- GA\_Filter, 71
  - \_init, 72
  - Expected\_Get, 72
  - GA\_Filter, 72
  - GA\_Filter, 72
  - M, 73
  - R, 73
  - Rp, 73
  - Xp, 73
- Gaussian\_Linear\_Model, 74
  - F, 76
  - f, 76
  - G, 76
  - Gaussian\_Linear\_Model, 75
  - Gaussian\_Linear\_Model, 75
  - Get\_Cov\_Prediction, 75
  - Get\_Mean\_Prediction, 75
  - H, 76
  - h, 76
  - J\_Observation\_Function, 75
  - Jw\_State\_Function, 75
  - Jx\_State\_Function, 75
  - Observation\_Function, 76

- State\_Function, 76
- Gaussian\_Nonlinear\_Model, 77
  - ~Gaussian\_Nonlinear\_Model, 78
  - Gaussian\_Nonlinear\_Model, 78
  - Gaussian\_Nonlinear\_Model, 78
  - Get\_Linear\_Parameters, 78
  - Init, 78
  - Jw\_State\_Function, 78
  - Jx\_State\_Function, 78
  - State\_Function, 79
- Get\_Alpha
  - Discrete\_Approximation\_CD\_Model, 53
- Get\_Cov\_Prediction
  - Gaussian\_Linear\_Model, 75
  - Linear\_CD\_Model, 83
- Get\_Init\_Parameters
  - Discrete\_Observed\_Model, 57
- Get\_Linear\_Parameters
  - Discrete\_Approximation\_CD\_Model, 53
  - Gaussian\_Nonlinear\_Model, 78
- Get\_Linear\_Scheme
  - Discrete\_Approximation\_CD\_Model, 53
  - Euler\_CD\_Model, 59
  - Heun\_CD\_Model, 80
  - Ozaki\_CD\_Model, 96
  - SRK4\_CD\_Model, 109
- Get\_Mean\_Prediction
  - Gaussian\_Linear\_Model, 75
  - Linear\_CD\_Model, 83
- Get\_Time
  - Model, 90
- H
  - Gaussian\_Linear\_Model, 76
  - Linear\_CD\_Model, 84
- h
  - Gaussian\_Linear\_Model, 76
  - Linear\_CD\_Model, 84
- Heun\_CD\_Model, 80
  - Get\_Linear\_Scheme, 80
  - Heun\_CD\_Model, 80
  - Heun\_CD\_Model, 80
  - Jw\_Scheme, 81
  - Jx\_Scheme, 81
  - Scheme, 81
- Init
  - Continuous\_Discrete\_Model, 47
  - Discrete\_Approximation\_CD\_Model, 53
  - Filter, 64
  - Gaussian\_Nonlinear\_Model, 78
  - Linear\_CD\_Model, 83
- J\_Drift\_Function
  - Continuous\_Discrete\_Model, 47
  - Linear\_CD\_Model, 83
- J\_Observation\_Function
  - Discrete\_Approximation\_CD\_Model, 53
  - Discrete\_Observed\_Model, 57
  - Gaussian\_Linear\_Model, 75
  - Linear\_CD\_Model, 84
- Jw\_Scheme
  - Discrete\_Approximation\_CD\_Model, 54
  - Euler\_CD\_Model, 60
  - Heun\_CD\_Model, 81
  - Ozaki\_CD\_Model, 97
  - SRK4\_CD\_Model, 110
- Jw\_State\_Function
  - Discrete\_Approximation\_CD\_Model, 54
  - Gaussian\_Linear\_Model, 75
  - Gaussian\_Nonlinear\_Model, 78
- Jx\_Scheme
  - Discrete\_Approximation\_CD\_Model, 54
  - Euler\_CD\_Model, 60
  - Heun\_CD\_Model, 81
  - Ozaki\_CD\_Model, 97
  - SRK4\_CD\_Model, 110
- Jx\_State\_Function
  - Discrete\_Approximation\_CD\_Model, 54
  - Gaussian\_Linear\_Model, 75
  - Gaussian\_Nonlinear\_Model, 78
- lambda
  - Unscented\_Kalman\_Filter, 115
- Likelihood
  - Filter, 65
- Likelihood\_Get
  - Filter, 64
- Linear\_CD\_Model, 82
  - A, 84
  - B, 84
  - C, 84
  - Diffusion\_Function, 83
  - Drift\_Function, 83
  - Get\_Cov\_Prediction, 83
  - Get\_Mean\_Prediction, 83
  - H, 84
  - h, 84
  - Init, 83
  - J\_Drift\_Function, 83
  - J\_Observation\_Function, 84
  - Linear\_CD\_Model, 83
  - Linear\_CD\_Model, 83
  - Observation\_Function, 84
- LL\_Filter, 85
  - \_update, 85
  - LL\_Filter, 85
  - LL\_Filter, 85



- LTI\_CD\_Simulator, 86
  - draw\_state, 86
  - LTI\_CD\_Simulator, 86
  - LTI\_CD\_Simulator, 86
- LTI\_CD\_Simulator\_WT, 87
  - Draw\_Init, 87
  - LTI\_CD\_Simulator\_WT, 87
  - LTI\_CD\_Simulator\_WT, 87
  - T, 88
  - TB, 88
  - Xt, 88
- M
  - CD\_Filter, 37
  - GA\_Filter, 73
- Model, 89
  - ~Model, 90
  - \_k, 90
  - Clear, 90
  - Get\_Time, 90
  - Model, 90
  - Update, 90
- model
  - Filter, 65
  - SI\_Sampler, 100
  - Simulator, 104
- N
  - G\_Simulator\_WT, 70
- NB
  - G\_Simulator\_WT, 70
- NbSample
  - SISR\_Filter, 108
- Obs\_Optimal\_Density
  - G\_Simulator, 68
  - Opt\_Simulator, 91
- Observation\_Density
  - CD\_Simulator, 42
  - G\_Simulator, 68
  - Simulator, 103
- Observation\_Function
  - Discrete\_Approximation\_CD\_Model, 54
  - Discrete\_Observed\_Model, 58
  - Gaussian\_Linear\_Model, 76
  - Linear\_CD\_Model, 84
- Opt\_Simulator, 91
  - Draw\_Optimal, 91
  - Obs\_Optimal\_Density, 91
  - Opt\_Simulator, 91
  - Opt\_Simulator, 91
- Optimal\_Sampler, 93
  - Draw, 93
  - DrawInitCloud, 94
  - Optimal\_Sampler, 93
  - Optimal\_Sampler, 93
  - Weight, 94
- OptSISR\_Filter, 95
  - ~OptSISR\_Filter, 95
  - OptSISR\_Filter, 95
  - OptSISR\_Filter, 95
  - sim, 95
- Ozaki\_CD\_Model, 96
  - Get\_Linear\_Scheme, 96
  - Jw\_Scheme, 97
  - Jx\_Scheme, 97
  - Ozaki\_CD\_Model, 96
  - Ozaki\_CD\_Model, 96
  - Scheme, 97
- Qv
  - Discrete\_Observed\_Model, 58
- Qw
  - Discrete\_Observed\_Model, 58
- R
  - CD\_Filter, 37
  - GA\_Filter, 73
- r
  - Simulator, 104
  - SISR\_Filter, 108
- R0
  - Discrete\_Observed\_Model, 58
- Rc
  - SISR\_Filter, 108
- Resampling
  - SISR\_Filter, 107
- Rp
  - CD\_Filter, 37
  - GA\_Filter, 73
- Save\_X
  - CD\_Bootstrap\_Filter, 31
  - CD\_Filter, 36
  - CD\_Simulator, 42
  - Filter, 65
  - Simulator, 103
- Save\_Y
  - CD\_Simulator, 42
  - Simulator, 103
- Scheme
  - CD\_Extended\_Kalman\_Filter, 34
  - Discrete\_Approximation\_CD\_Model, 55
  - Euler\_CD\_Model, 60
  - Heun\_CD\_Model, 81
  - Ozaki\_CD\_Model, 97
  - SRK4\_CD\_Model, 110
- scheme

- CD\_Simulator, 43
- seed
  - SISR\_Filter, 108
- Set\_Alpha
  - CD\_Simulator, 42
  - Discrete\_Approximation\_CD\_Model, 55
- Set\_Seed
  - Simulator, 104
- SetRc
  - SISR\_Filter, 108
- SetSeed
  - SISR\_Filter, 108
- SI\_Sampler, 98
  - Draw, 99
  - DrawInitCloud, 99
  - model, 100
  - SI\_Sampler, 98
  - SI\_Sampler, 98
  - Weight, 99
- sim
  - Bootstrap\_Filter, 28
  - CD\_Bootstrap\_Filter, 32
  - OptSISR\_Filter, 95
- Simulate
  - CD\_Simulator, 43
  - Simulator, 104
- Simulator, 101
  - ~Simulator, 102
  - \_update, 102
  - b, 104
  - Clear, 102
  - Draw\_Init, 102
  - Draw\_Observation, 102
  - Draw\_Transition, 103
  - model, 104
  - Observation\_Density, 103
  - r, 104
  - Save\_X, 103
  - Save\_Y, 103
  - Set\_Seed, 104
  - Simulate, 104
  - Simulator, 102
  - Update, 104
  - X, 104
  - Y, 104
- SISR\_Filter, 105
  - ~SISR\_Filter, 106
  - \_init, 107
  - \_update, 107
  - cloud, 108
  - cloud\_km1, 108
  - CloudGet, 107
  - Expected\_Get, 107
  - NbSample, 108
  - r, 108
  - Rc, 108
  - Resampling, 107
  - seed, 108
  - SetRc, 108
  - SetSeed, 108
  - SISR\_Filter, 106
  - SISR\_Filter, 106
  - Sys, 108
- SP\_Init
  - Unscented\_Kalman\_Filter, 114
- sqrt\_Qv
  - Unscented\_Kalman\_Filter, 115
- sqrt\_Qw
  - Unscented\_Kalman\_Filter, 115
- SRK4\_CD\_Model, 109
  - Get\_Linear\_Scheme, 109
  - Jw\_Scheme, 110
  - Jx\_Scheme, 110
  - Scheme, 110
  - SRK4\_CD\_Model, 109
  - SRK4\_CD\_Model, 109
- State\_Function
  - Discrete\_Approximation\_CD\_Model, 55
  - Gaussian\_Linear\_Model, 76
  - Gaussian\_Nonlinear\_Model, 79
- sW
  - Unscented\_Kalman\_Filter, 115
- sX
  - Unscented\_Kalman\_Filter, 115
- sY
  - Unscented\_Kalman\_Filter, 115
- Sys
  - SISR\_Filter, 108
- T
  - CD\_Simulator\_WT, 45
  - LTI\_CD\_Simulator\_WT, 88
- TB
  - CD\_Simulator\_WT, 45
  - LTI\_CD\_Simulator\_WT, 88
- THGL\_Filter, 111
  - \_update, 111
  - THGL\_Filter, 111
  - THGL\_Filter, 111
- Ts
  - Continuous\_Discrete\_Model, 48
- U\_Cov
  - Unscented\_Kalman\_Filter, 114
- U\_Mean
  - Unscented\_Kalman\_Filter, 114
- Unscented\_Kalman\_Filter, 112
  - \_init, 114

- [\\_update](#), 114
  - [lambda](#), 115
  - [SP\\_Init](#), 114
  - [sqrt\\_Qv](#), 115
  - [sqrt\\_Qw](#), 115
  - [sW](#), 115
  - [sX](#), 115
  - [sY](#), 115
  - [U\\_Cov](#), 114
  - [U\\_Mean](#), 114
  - [Unscented\\_Kalman\\_Filter](#), 113
  - [Unscented\\_Kalman\\_Filter](#), 113
  - [w](#), 115
  - [w\\_0](#), 115
  - [w\\_0c](#), 115
- Update
  - [Filter](#), 65
  - [Model](#), 90
  - [Simulator](#), 104
- Value
  - [Weighted\\_Sample](#), 116
- w
  - [Unscented\\_Kalman\\_Filter](#), 115
- w\_0
  - [Unscented\\_Kalman\\_Filter](#), 115
- w\_0c
  - [Unscented\\_Kalman\\_Filter](#), 115
- Weight
  - [Bootstrap\\_Sampler](#), 30
  - [Optimal\\_Sampler](#), 94
  - [SI\\_Sampler](#), 99
  - [Weighted\\_Sample](#), 116
- Weighted\_Sample, 116
  - [Value](#), 116
  - [Weight](#), 116
- X
  - [Filter](#), 65
  - [Simulator](#), 104
- X0
  - [Discrete\\_Observed\\_Model](#), 58
- Xp
  - [CD\\_Filter](#), 37
  - [GA\\_Filter](#), 73
- Xt
  - [CD\\_Simulator\\_WT](#), 45
  - [G\\_Simulator\\_WT](#), 70
  - [LTI\\_CD\\_Simulator\\_WT](#), 88
- Y
  - [Simulator](#), 104