Fuzzy Logic Tools
Reference Manual v1.0

Antonio Javier Barragán Piña
José Manuel Andújar Márquez
CEA Intelligent Control Group

Universidad de Huelva
PUBLICACIONES

Fuzzy Logic Tools
Reference Manual
v 1.0

Antonio Javier Barragán Piña
José Manuel Andújar Márquez

Universidad
de Huelva

2012
©
Servicio de Publicaciones
Universidad de Huelva

©
Antonio Javier Barragán Piña
José Manuel Andújar Márquez

Papel
Offset Ahuesado de 90 g/m²
Certificado FSC

Encuadernación
Rústica, cosido con hilo vegetal
Printed in Spain. Impreso en España.

I.S.B.N.
978-84-15147-32-9

Depósito legal
H xxx-2012

Imprime
Artes Gráficas Bonanza, S.L.

Reservados todos los derechos. Ni la totalidad ni parte de este libro puede reproducirse o transmitirse por ningún procedimiento electrónico o mecánico, incluyendo fotocopia, grabación magnética o cualquier almacenamiento de información y sistema de recuperación, sin permiso escrito del Servicio de Publicaciones de la Universidad de Huelva.

C.E.P.
Biblioteca Universitaria

Paisajes, tiempo y memoria / Juan Aurelio Pérez Macías, Juan Luis Camiazo Rubio, Beatriz Gavilán Ceballos (eds) -- Huelva : Universidad de Huelva, 2012
228 p.; 21 cm. – (Collectanea (Universidad de Huelva) ; 171)
ISBN 978-84-15147-86-2
1. Arqueología del paisaje. 2. Asentamientos humanos. I. Universidad de Huelva. II. Camiazo Rubio, Juan Luis. III. Gavilán Ceballos, Beatriz. IV. Título. V. Serie.
902

Preface

This manual documents the use of Fuzzy Logic Tools (FLT), a C++ framework for storage, analysis and design of fully general multiple-input multiple-output (MIMO) Takagi-Sugeno fuzzy control systems, without constraints in the order of either the inputs or the output vectors.

This reference manual is intended as a reference work for those developers wishing to use the tools provided by the FLT. Therefore, the text is structured following the typical pattern of reference manuals. Firstly, a general description of the variables, functions, classes, methods and attributes included in the software is presented. Then each of these items is studied in depth. Finally, some examples of using the FLT are included. These functions can be used for the analysis and design of TS-type fuzzy control.

With the intention of making our work available to the entire scientific community, FLT is licensed under GPLv3, so you can use it freely if it meets the requirements of such license (see http://www.gnu.org/licenses/gpl.html). With the same intention, this document is licensed under a Creative Commons Attribution-ShareAlike 3.0 License, approved for Free Cultural Works initiative.

This work is in continuous evolution and improvement. If you are interested can stay informed of new versions, bugs, and other information about the project at http://uhu.es/antonio.barragan/flt

José Manuel Andújar Márquez
Febrary 2012

Contents

Fuzzy Logic Tools

Fuzzy Logic Tools (FLT) is a C++ framework for storage, analysis and design of fully general multiple-input multiple-output (MIMO) Takagi-Sugeno fuzzy control systems, without constraints in the order of either the inputs or the output vectors.

FLT has been developed from research works of the Dr. Antonio Javier Barragán Piña under the direction of the Professor of Engineering Systems and Automation, Dr. José Manuel Andújar Márquez, both belonging to the "Control and Robotics" research group of the University of Huelva.

FLT is designed to store general Takagi Sugeno (TS) fuzzy systems, without restriction on size of input and output vectors. FLT nor limit the type or distribution of the membership functions used, even it allows mixing of different membership functions in the same antecedent. The storage of fuzzy models is done using the Membership, Rule and System classes. There are many tools that can store fuzzy models similarly, and even more flexible as it does FLT. However, the biggest advantage of FLT are the analysis and design functions that are implemented for the type of fuzzy system defined above:

In utilities.hpp some functions that may be of interest to the handling of fuzzy systems are defined, for example, the reading/writing of fuzzy systems from/in text files, the evaluation of a fuzzy control system in closed loop, the extraction of a subsystem from a given fuzzy system, etc.

Functions grouped in derivatives.hpp calculate several the derivatives of a fuzzy system. These calculations may be of interest for the analysis and design of fuzzy control systems. For example, it is possible to calculate the derivatives of a fuzzy system with respect to its inputs or parameters, or obtain a linearized model of a fuzzy system at a given point.

Since the functions provided by derivative.hpp, several methods for the parametric

adaptaition of fuzzy systems have been developed based on the extended Kalman filter. These algorithms are in Kalman.hpp.

Kalman.hpp implements the adaptation of a fuzzy model by the extended Kalman filter from the derivatives included in derivatives.hpp.

In aux_matlab.hpp some util functions to use with MATLAB© API are defined, as the reading/writing of fuzzy models from/to MATLAB©, and the conversion of row-major matrices in column-major and vice versa.

Finally, several examples of programming MEX functions are presented, which can be useful for the analysis and design of fuzzy control systems using MATLAB©.

0.1 Obtaining FLT, Help and Bugs reports:

You can download the latest versions of the FLT and access to forums and bugs tracking system at http://sourceforge.net/projects/fuzzylogictools.

Additional information, news and related projects are available from the FLT website: http://uhu.es/antonio.barragan/flt.

0.2 Publications/references

If you would like to refer the Fuzzy Logic Tools (FLT) in a journal paper or similar, the recommended way is to cite this reference manual or the following papers:

A.J. Barragán and J.M. Andújar, *Fuzzy Logic Tools Reference Manual*, ISBN XXXXX, http://uhu.es/antonio.barragan/flt.

Andújar, J. M. and Barragán, A. J. (2005). A methodology to design stable nonlinear fuzzy control systems. *Fuzzy Sets and Systems*, 154(2):157–181.

Andújar, J. M., Barragán, A. J., and Gegúndez, M. E. (2009). A general and formal methodology for designing stable nonlinear fuzzy control systems. *IEEE Transactions on Fuzzy Systems*, 17(5):1081–1091.

Barragán Piña, A. J. (2009). *Síntesis de sistemas de control borroso estables por diseño*. ISBN 978-84-92944-72-9. University of Huelva. http://uhu.es/antonio.barragan/thesis

Andújar, J. M. and Barragán, A. J. (2010). A formal methodology for the analysis and design of nonlinear fuzzy control systems. In *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, number 1, pages 66–74, Barcelona, Spain.

Barragán Piña, A. J., Andújar Márquez, J. M., Aznar Torres, M., and Jiménez Avello, A. (2011a). Methodology for adapting the parameters of a fuzzy system using the

extended Kalman filter. In *7th conference of the European Society for Fuzzy Logic and Technology, EUSFLAT (LFA-2011)*, Aix-les-Bains, France.

Barragán Piña, A. J., Andújar Márquez, J. M., Aznar Torres, M., and Jiménez Avello, A. (2011b). Practical application of the extended Kalman filter to fuzzy modeling. In *7th conference of the European Society for Fuzzy Logic and Technology, EUSFLAT (LFA-2011)*, Aix-les-Bains, France.

0.3 Installing and using FLT

FLT has been written in C++. This software has been compiled and tested, in 32 and 64 bits architectures on GNU/Linux using GCC, and on Microsoft Windows© operating systems using MinGW (http://www.mingw.org). FLT can be used to make MATLAB© function via MATLAB© API. We recommend that you use GNUMex (http://gnumex.sourceforge.net) with MinGW to compile the MEX files in Microsoft Windows© enviroments.

FLT is source code, so you do not need to be installed. If you wish, you can copy the directory "flt" to a route included in the search path of your compiler, for example, "/usr/include" in GNU/Linux or "C:\MinGW\include" if you use MinGW in Microsoft Windows©.

This example program obtains the output of a closed loop fuzzy system and its jacobian matrix.

```
#include <stdio.h>
#include <tnt/tnt.h>
#include <flt/system.hpp>
#include <flt/derivatives.hpp>

using namespace std;
using namespace TNT;
using namespace FLT;

int main(int argc, char **argv)
{
  if (argc < 4)
  {
    cout << "Error. Some input aguments are missing.\n\n";
    cout << "example plant.txt controller.txt x1 x2 ... xn\n";
    cout << "\t where 'n' is the order of the plant (number of outputs)" << endl;

    return 1;
  }

  char *plant = argv[1];
  System P = TXT2System(plant);
  size_t n = P.outputs();
  if (!n)
  {
    cout << "Error reading the plant file." << endl;
    return 1;
  }

  if (argc != 3+n)
  {
```

```
    cout << "Error. Some input aguments are missing.\n\n";
    cout << "example plant.txt controller.txt x1 x2 ... xn\n";
    cout << "\t where 'n' is the order of the plant (number of outputs)" << endl;

    return 1;
  }

  char *controller = argv[2];
  System C = TXT2System(controller);
  size_t m = C.outputs();
  if (!m)
  {
    cout << "Error reading the controller file." << endl;
    return 1;
  }

  if (P.inputs() != n+m)
  {
    cout << E_NoCoherent << endl;
    return 1;
  }

  Array1D<double> X(n);
  for (size_t i=0; i<n; i++)
    X[i] = atof(argv[i+3]);

  Array1D<double> dX = evaluate(&X[0], P, C);

  cout << "\nClosed loop output:\n";
  for (size_t i=0; i<n; i++)
    cout << dX[i] << "\n";

  Array2D<double> J = jacobian(P, C, &X[0]);
  cout << "\nClosed loop Jacobian matrix:\n";
  for (size_t i=0; i<n; i++)
  {
    for (size_t j=0; j<n; j++)
      cout << J[i][j] << "  ";
    cout << "\n";
  }
  cout << endl;

  return 0;
}
```

The example.cpp file is in the docs folder. You can compile it from this folder using this sentence:

```
g++ example.cpp ../src/membership.cpp ../src/rule.cpp ../src/system.cpp
    ../src/utilities.cpp ../src/derivatives.cpp -o example
```

For test this example function with the Plant.txt and Controller.txt from matlab_-utilities folder, use:

```
./example ../matlab_utilities/Plant.txt ../matlab_utilities/Controller.txt 0 0
```

0.4 No Warranty

Fuzzy Logic Tools (FLT) software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MER-

CHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. It is your responsibility to validate the behavior of the routines and their accuracy using the source code provided, or to purchase support and warranties from commercial redistributors. Consult the GNU General Public license for further details.

0.5 Namespace List

Here is a list of all documented namespaces with brief descriptions:

0.6 Class Hierarchy

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

0.7 Class List

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

0.8 File List

Here is a list of all documented files with brief descriptions:

Namespace Documentation

0.9 FLT Namespace Reference

Fuzzy Logic Tools (FLT) namespace.

Classes

- class Membership

  This class contains methods and attributes common to all Membership functions.

- class Anymf

  Any Membership function.

- class Constmf

  Constant Membership function.

- class Gaussmf

  Gaussian Membership function.

- class Gauss2mf

  Double gaussian Membership function.

- class GBellmf

  Bell Membership function.

- class Pimf

Pi (S-Z) Membership function.

- class PSigmf

  Product of sigmoidals Membership function.

- class Smf

  S Membership function.

- class Sigmf

  Sigmoidal Membership function.

- class Sig2mf

  Difference of sigmoidals Membership function.

- class Trapmf

  Trapezoidal Membership function.

- class Trimf

  Triangular Membership function.

- class Zmf

  Z Membership function.

- class Rule

  This class contains methods and attributes common to fuzzy Rules.

- class System

  This class contains methods and attributes common to fuzzy Systems.

Enumerations

- enum TYPE_MF {
  ANYMF = 0, CONSTMF, GAUSSMF, GAUSS2MF,
  GBELLMF, PIMF, PSIGMF, SMF,
  SIGMF, SIG2MF, TRAPMF, TRIMF,
  ZMF }

  Enumeration with the implemented Membership functions.

Functions

- TNT::Array2D< double > jacobian (System &S, const double *const x, const double *const u, TNT::Array2D< double > &B, TNT::Array1D< double > &F)

  Computes the open-loop jacobian matrix in `x` for the Plant `S`.

- TNT::Array2D< double > jacobian (System &S, System &C, const double *const x)

  Computes the closed-loop jacobian matrix in `x` for the Plant `S` and the Controller `C`.

- TNT::Array2D< double > jacobianAprox (System &S, System &C, const double *const x, double h=0.001)

  Computes the approximation of the closed-loop jacobian matrix in `x` for the Plant `S` and the Controller `C` using finite differences with a step `h`.

- TNT::Array2D< double > jacobianAprox (System &S, const double *const x, const double *const u, TNT::Array2D< double > &B, TNT::Array1D< double > &F, double h=0.001)

  Computes the approximation of the open-loop jacobian matrix in `x` for the Plant `S` using finite differences with a step `h`.

- double derconseq (System &S, double *x, size_t output, size_t rule, size_t parameter, int &error)

  Obtains the derivative of a fuzzy model with respect to a consequent.

- TNT::Array2D< double > derconseq (System &S, double *x)

  Obtains the jacobian matrix of a fuzzy model with respect to its consequents.

- double derantec (System &S, double *x, size_t input, size_t output, size_t rule, size_t parameter, int &error)

  Obtains the derivative of a fuzzy model with respect to a parameter of an antecedent.

- TNT::Array2D< double > derantec (System &S, double *x)

  Obtains the jacobian matrix of a fuzzy model with respect to its antecedents.

- TNT::Array2D< double > derfuzzy (System &S, double *x)

  Obtains the jacobian matrix of a fuzzy model with respect to all of its parameters.

- TNT::Array1D< double > KalmanAntec (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P, TNT::Array2D< double > &Phi)

Computes antecedets adjust by the discrete extended Kalman filter.

- TNT::Array1D< double > KalmanAntec (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P)

  Computes antecedets adjust by the discrete extended Kalman filter where Phi is assumed to be the identity matrix.

- TNT::Array1D< double > KalmanConseq (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P, TNT::Array2D< double > &Phi)

  Computes consequents adjust by the discrete extended Kalman filter.

- TNT::Array1D< double > KalmanConseq (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P)

  Computes consequents adjust by the discrete extended Kalman filter where Phi is assumed to be the identity matrix.

- TNT::Array1D< double > KalmanFuz (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P, TNT::Array2D< double > &Phi)

  Computes the simultaneous adjustment of antecedets and consequents by the discrete extended Kalman filter.

- TNT::Array1D< double > KalmanFuz (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P)

  Computes the simultaneous adjustment of antecedets and consequents by the discrete extended Kalman filter where Phi is assumed to be the identity matrix.

- int sign (double x)

  Implementation of the sign function.

- Membership * createMF (TYPE_MF t)

  Virtual constructor for Membership functions.

- int System2TXT (System &S, const char *file)

  Saves the fuzzy System S in a text file.

- System TXT2System (const char *file)

Reads a fuzzy System from a text file.

- int printSystem (const char *file, System &S, char *inputs[ ]=NULL, char *outputs[ ]=NULL, int accuracy=10)

  Writes a fuzzy system in its linguistic form.

- TNT::Array1D< double > evaluate (const double *const point, System &S, System &C)

  Evaluates a closed loop fuzzy system in the given `point`.

- System initialController (System &Plant)

  Creates a fuzzy Controller from a given plant.

- System subSystem (System &S, size_t nrules, size_t *outputs, size_t *rules)

  Extracts a subsystem from a fuzzy System.

- TNT::Array2D< double > extractPoints (System &S, unsigned int numMeshPoints=5, double precision=1E-3, bool addMesh=false, bool onlyStatePoints=true)

  Extracts the representative points of a fuzzy system.

- void col2row_major (const double *const colM, double *rowM, size_t num_rows, size_t num_cols)

  Converts a column-major matrix in a row-major one.

- TNT::Array2D< double > col2row_major (const double *const colM, size_t num_rows, size_t num_cols)

  Converts a column-major matrix in a row-major one.

- void row2col_major (const double *const rowM, double *colM, size_t num_rows, size_t num_cols)

  Converts a row-major matrix in a column-major one.

- int readModel (const mxArray *model, System &S)

  Reads a Fuzzy Model (TXT, FIS file or FIS variable).

- int FIS2System (const mxArray *FIS, System &S)

  Reads the Fuzzy Model from a FIS variable.

- int System2FIS (System &S, mxArray *FIS)

  Writes the Fuzzy Model in a FIS variable.

Variables

- static const char * MF_NAMES [ ]

  Names of the Membership functions.

- static const size_t MF_PARAM_NUMBER [ ]

  Initial number of parameters of each Membership function in FLT::TYPE_MF.

0.9.1 Detailed Description

Fuzzy Logic Tools (FLT) namespace.

0.9.2 Enumeration Type Documentation

0.9.2.1 enum FLT::TYPE_MF

Enumeration with the implemented Membership functions.

FLT::TYPE_MF is used to determinate the Membership function type, FLT::MF_-NAMES are the names of the Membership functions, and FLT::MF_PARAM_-NUMBER are the initial number of parameters of the Membership functions.

Remarks

FLT::TYPE_MF, FLT::MF_NAMES and FLT::MF_PARAM_NUMBER must be consistent.
Important: If the first or last element of FLT::TYPE_MF are changed, you needd to check membership.cpp (and perhaps others files), because these elements are used to make some comparisons.

Enumerator:

ANYMF Any Membership function (FLT::Anymf).

CONSTMF Constant Membership function (FLT::Constmf).

GAUSSMF Gaussian Membership function (FLT::Gaussmf).

GAUSS2MF Double gaussian Membership function (FLT::Gauss2mf).

GBELLMF Bell Membership function (FLT::GBellmf).

PIMF Pi (S-Z) Membership function (FLT::Pimf).

PSIGMF Product of sigmoidals Membership function (FLT::PSigmf).

SMF S Membership function (FLT::Smf).

SIGMF Sigmoidal Membership function (FLT::Sigmf).

SIG2MF Difference of sigmoidals Membership function (FLT::Sig2mf).

TRAPMF Trapezoidal Membership function (FLT::Trapmf).

TRIMF Triangular Membership function (FLT::Trimf).

ZMF Z Membership function (FLT::Zmf).

0.9.3 Function Documentation

0.9.3.1 void FLT::col2row_major (const double *const *colM*, double * *rowM*, size_t *num_rows*, size_t *num_cols*) [inline]

Converts a column-major matrix in a row-major one.

MATLAB© uses column-major matrices, but C uses row-major.

Examples:

matlab_utilities/Kalmanantec.cpp, matlab_utilities/Kalmanconseq.cpp, and matlab_utilities/Kalmanfuz.cpp.

0.9.3.2 TNT::Array2D<double> FLT::col2row_major (const double *const *colM*, size_t *num_rows*, size_t *num_cols*) [inline]

Converts a column-major matrix in a row-major one.

MATLAB© uses column-major matrices, but C uses row-major.

0.9.3.3 Membership * FLT::createMF (TYPE_MF *t*)

Virtual constructor for Membership functions.

This function create and return a Membership function of the selected type.

0.9.3.4 double FLT::derantec (System & *S*, double * *x*, size_t *input*, size_t *output*, size_t *rule*, size_t *parameter*, int & *error*)

Obtains the derivative of a fuzzy model with respect to a parameter of an antecedent.

It is assumed that the output of the system is the same as the parameter, otherwise it should be considered derivative equal to 0.

Returns

If `output`, `rule` or `parameter` are out of System limits, the function returns 0 and `error` = 1.

Examples:

matlab_utilities/fuzderantec.cpp.

0.9.3.5 TNT::Array2D< double > FLT::derantec (System & *S,* double * *x*)

Obtains the jacobian matrix of a fuzzy model with respect to its antecedents.

Returns

Returns an empty array if error.

0.9.3.6 double FLT::derconseq (System & *S,* double * *x,* size_t *output,* size_t *rule,* size_t *parameter,* int & *error*)

Obtains the derivative of a fuzzy model with respect to a consequent.

It is assumed that the output of the system is the same as the parameter, otherwise it should be considered derivative equal to 0.

Returns

If `output`, `rule` or `parameter` are out of System limits, the function returns 0 and `error` = 1.

Examples:

matlab_utilities/fuzderconseq.cpp.

0.9.3.7 TNT::Array2D< double > FLT::derconseq (System & *S,* double * *x*)

Obtains the jacobian matrix of a fuzzy model with respect to its consequents.

Returns

Returns an empty array if error.

0.9.3.8 TNT::Array2D< double > FLT::derfuzzy (System & *S,* double * *x*)

Obtains the jacobian matrix of a fuzzy model with respect to all of its parameters.

Returns

Returns an empty array if error.

Examples:

matlab_utilities/fuzderparam.cpp.

0.9.3.9 Array1D< double > FLT::evaluate (const double *const *point, System & *S*, System & *C*)

Evaluates a closed loop fuzzy system in the given `point`.

`S` is the plant and `C` is a fuzzy controller.

Returns

Returns an empty array if error.

Examples:

example.cpp, and matlab_utilities/fuzeval.cpp.

0.9.3.10 Array2D< double > FLT::extractPoints (System & *S*, unsigned int *numMeshPoints* = `5`, double *precision* = `1E-3`, bool *addMesh* = `false`, bool *onlyStatePoints* = `true`)

Extracts the representative points of a fuzzy system.

Parameters

S Is a fuzzy System.

numMeshPoints Is used in Memberships functions without significative points, like ANYMF, CONSTMF, SMF, ...

precision Is used to remove similar points.

addMesh If `addMesh` is true, a mesh of `numMeshPoints` for each dimmension will be added to System points.

onlyStatePoints If `onlyStatePoints` is true, this function only extracts the state vector points, not the control vector points.

Examples:

matlab_utilities/extractPoints.cpp.

0.9.3.11 int FLT::FIS2System (const mxArray * *FIS*, System & *S*)

Reads the Fuzzy Model from a FIS variable.

Returns 0 if no errors.

0.9.3.12 System FLT::initialController (System & *Plant*)

Creates a fuzzy Controller from a given plant.

The fuzzy controller will have all the rules that the plant owns in each of its outputs.

Consequents all initialized to 0.

Returns

Returns an empty System if error.

Examples:

matlab_utilities/initializeController.cpp.

0.9.3.13 Array2D< double > FLT::jacobian (System & *S*, const double *const *x*, const double *const *u*, TNT::Array2D< double > & *B*, TNT::Array1D< double > & *F*)

Computes the open-loop jacobian matrix in x for the Plant S.

Obtains the linearization $\mathbf{f}(\mathbf{x}, \mathbf{u}) \approx \mathbf{F} + \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$.

Returns

Returns the dynamic matrix, $\mathbf{A}$, or an empty array if error.

Examples:

example.cpp, matlab_utilities/fuzderparam.cpp, matlab_utilities/fuzjac.cpp, and matlab_utilities/fuzlinear.cpp.

0.9.3.14 TNT::Array2D< double > FLT::jacobian (System & *S*, System & *C*, const double *const *x*)

Computes the closed-loop jacobian matrix in x for the Plant S and the Controller C.

Returns

Returns the jacobian matrix or an empty array if error.

0.9.3.15 TNT::Array2D< double > FLT::jacobianAprox (System & *S*, const double *const *x*, const double *const *u*, TNT::Array2D< double > & *B*, TNT::Array1D< double > & *F*, double *h* = 0.001)

Computes the approximation of the open-loop jacobian matrix in x for the Plant S using finite differences with a step h.

Obtains the linearization $\mathbf{f}(\mathbf{x}, \mathbf{u}) \approx \mathbf{F} + \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$.

Returns

Returns the dynamic matrix, $\mathbf{A}$, or an empty array if error.

0.9.3.16 TNT::Array2D< double > FLT::jacobianAprox (System & *S*, System & *C*, const double *const *x*, double *h* = 0.001)

Computes the approximation of the closed-loop jacobian matrix in x for the Plant S and the Controller C using finite differences with a step h.

Returns

Returns an empty array if error.

Examples:

matlab_utilities/aproxjac.cpp, and matlab_utilities/aproxlinear.cpp.

0.9.3.17 TNT::Array1D< double > FLT::KalmanAntec (System & *Model*, TNT::Array1D< double > & *input*, TNT::Array1D< double > & *output*, TNT::Array2D< double > & *covariance*, TNT::Array2D< double > & *P*, TNT::Array2D< double > & *Phi*)

Computes antecedets adjust by the discrete extended Kalman filter.

Parameters

Model Is the fuzzy system whose antecedents will be adjusted.

input Is the input applied to the system in the current iteration.

output Is the real output of the system in the current iteration.

covariance Is the noise covariance matrix estimated from the hope operator.

P Is the covariance matrix of the filter.

Phi Is the Jacobian matrix that relates the parameters to be set with the following value of these parameters. Usually this will be an identity matrix of appropriate dimensions.

Returns

Returns the error vector of the iteration, or an empty vector if there was some error.

Model and P will be changed according to the Kalman discrete filte algorithm.

Examples:

matlab_utilities/Kalmanantec.cpp.

0.9.3.18 TNT::Array1D< double > FLT::KalmanAntec (System & *Model,* TNT::Array1D< double > & *input,* TNT::Array1D< double > & *output,* TNT::Array2D< double > & *covariance,* TNT::Array2D< double > & *P*)

Computes antecedets adjust by the discrete extended Kalman filter where Phi is assumed to be the identity matrix.

Parameters

Model Is the fuzzy system whose antecedents will be adjusted.

input Is the input applied to the system in the current iteration.

output Is the real output of the system in the current iteration.

covariance Is the noise covariance matrix estimated from the hope operator.

P Is the covariance matrix of the filter.

Returns

Returns the error vector of the iteration, or an empty vector if there was some error.

Model and P will be changed according to the Kalman discrete filte algorithm.

0.9.3.19 TNT::Array1D< double > FLT::KalmanConseq (System & *Model,* TNT::Array1D< double > & *input,* TNT::Array1D< double > & *output,* TNT::Array2D< double > & *covariance,* TNT::Array2D< double > & *P,* TNT::Array2D< double > & *Phi*)

Computes consequents adjust by the discrete extended Kalman filter.

Parameters

Model Is the fuzzy system whose consequents will be adjusted.

input Is the input applied to the system in the current iteration (current State vector, x(k)).

output Is the real output of the system in the current iteration.

covariance Is the noise covariance matrix estimated from the hope operator.

P Is the covariance matrix of the filter.

Phi Is the Jacobian matrix that relates the parameters to be set with the following value of these parameters. Usually this will be an identity matrix of appropriate dimensions.

Returns

Returns the error vector of the iteration, or an empty vector if there was some error.

Model and P will be changed according to the Kalman discrete filte algorithm.

Examples:

matlab_utilities/Kalmanconseq.cpp.

0.9.3.20 TNT::Array1D< double > FLT::KalmanConseq (System & *Model,* TNT::Array1D< double > & *input,* TNT::Array1D< double > & *output,* TNT::Array2D< double > & *covariance,* TNT::Array2D< double > & *P*)

Computes consequents adjust by the discrete extended Kalman filter where Phi is assumed to be the identity matrix.

Parameters

Model Is the fuzzy system whose consequents will be adjusted.

input Is the input applied to the system in the current iteration (current State vector, x(k)).

output Is the real output of the system in the current iteration.

covariance Is the noise covariance matrix estimated from the hope operator.

P Is the covariance matrix of the filter.

Returns

Returns the error vector of the iteration, or an empty vector if there was some error.

Model and P will be changed according to the Kalman discrete filte algorithm.

0.9.3.21 TNT::Array1D< double > FLT::KalmanFuz (System & *Model*, TNT::Array1D< double > & *input*, TNT::Array1D< double > & *output*, TNT::Array2D< double > & *covariance*, TNT::Array2D< double > & *P*)

Computes the simultaneous adjustment of antecedets and consequents by the discrete extended Kalman filter where Phi is assumed to be the identity matrix.

Parameters

Model Is the fuzzy system whose parameters will be adjusted.

input Is the input applied to the system in the current iteration.

output Is the real output of the system in the current iteration.

covariance Is the noise covariance matrix estimated from the hope operator.

P Is the covariance matrix of the filter.

Returns

Returns the error vector of the iteration, or an empty vector if there was some error.

Model and P will be changed according to the Kalman discrete filte algorithm.

0.9.3.22 TNT::Array1D< double > FLT::KalmanFuz (System & *Model*, TNT::Array1D< double > & *input*, TNT::Array1D< double > & *output*, TNT::Array2D< double > & *covariance*, TNT::Array2D< double > & *P*, TNT::Array2D< double > & *Phi*)

Computes the simultaneous adjustment of antecedets and consequents by the discrete extended Kalman filter.

Parameters

Model Is the fuzzy system whose parameters will be adjusted.

input Is the input applied to the system in the current iteration.

output Is the real output of the system in the current iteration.

covariance Is the noise covariance matrix estimated from the hope operator.

P Is the covariance matrix of the filter.

Phi Is the Jacobian matrix that relates the parameters to be set with the following value of these parameters. Usually this will be an identity matrix of appropriate dimensions.

Returns

Returns the error vector of the iteration, or an empty vector if there was some error.

Model and P will be changed according to the Kalman discrete filte algorithm.

Examples:

matlab_utilities/Kalmanfuz.cpp.

0.9.3.23 int FLT::printSystem (const char * *file,* System & *S,* char * *inputs[ ]* = NULL, char * *outputs[ ]* = NULL, int *accuracy* = 10)

Writes a fuzzy system in its linguistic form.

"IF temperature is high and pressure is ... THEN valve is ..."

You can specify the name of the variables and the number of decimals that will be used to represent the system.

Examples:

matlab_utilities/fuzprint.cpp.

0.9.3.24 int FLT::readModel (const mxArray * *model,* System & *S*)

Reads a Fuzzy Model (TXT, FIS file or FIS variable).

Returns 0 if no errors.

Examples:

matlab_utilities/activation.cpp, matlab_utilities/antec2mat.cpp, matlab_utilities/aproxjac.cpp, matlab_utilities/aproxlinear.cpp, matlab_utilities/conseq2mat.cpp, matlab_utilities/extractPoints.cpp, matlab_utilities/fis2txt.cpp, matlab_utilities/fuz2mat.cpp, matlab_utilities/fuzcomb.cpp, matlab_utilities/fuzderantec.cpp, matlab_utilities/fuzderconseq.cpp, matlab_utilities/fuzderparam.cpp, matlab_utilities/fuzeval.cpp, matlab_utilities/fuzjac.cpp, matlab_utilities/fuzlinear.cpp, matlab_utilities/fuzprint.cpp, matlab_utilities/initializeController.cpp, matlab_utilities/Kalmanantec.cpp, matlab_utilities/Kalmanconseq.cpp, matlab_utilities/Kalmanfuz.cpp, matlab_utilities/mat2antec.cpp, matlab_utilities/mat2conseq.cpp, matlab_utilities/mat2fuz.cpp, and matlab_utilities/subsystem.cpp.

0.9.3.25 void FLT::row2col_major (const double *const *rowM,* double * *colM,* size_t *num_rows,* size_t *num_cols*) [inline]

Converts a row-major matrix in a column-major one.

MATLAB© uses column-major matrices, but C uses row-major.

Examples:

matlab_utilities/Kalmanantec.cpp, matlab_utilities/Kalmanconseq.cpp, and matlab_utilities/Kalmanfuz.cpp.

0.9.3.26 int FLT::sign (double *x*) [inline]

Implementation of the sign function.

Returns

Returns 0 if x=0, 1 if x>0 and -1 if x<0.

0.9.3.27 System FLT::subSystem (System & *S,* size_t *nrules,* size_t * *outputs,* size_t * *rules*)

Extracts a subsystem from a fuzzy System.

Returns

Returns an empty System if error.

Examples:

matlab_utilities/subsystem.cpp.

0.9.3.28 int FLT::System2FIS (System & *S,* mxArray * *FIS*)

Writes the Fuzzy Model in a FIS variable.

Returns 0 if no errors.

Examples:

matlab_utilities/fuzcomb.cpp, matlab_utilities/initializeController.cpp, matlab_utilities/Kalmanantec.cpp, matlab_utilities/Kalmanconseq.cpp, matlab_utilities/Kalmanfuz.cpp, matlab_utilities/mat2antec.cpp, matlab_utilities/mat2conseq.cpp, matlab_utilities/mat2fuz.cpp, matlab_utilities/subsystem.cpp, and matlab_utilities/txt2fis.cpp.

0.9.3.29 int FLT::System2TXT (System & *S*, const char * *file*)

Saves the fuzzy System `S` in a text file.

Returns

Returns 0 if no errors.

See also

See TXT2System to see the template for the TXT file.

Examples:

matlab_utilities/fis2txt.cpp, and matlab_utilities/fuzcomb.cpp.

0.9.3.30 System FLT::TXT2System (const char * *file*)

Reads a fuzzy System from a text file.

Returns

Returns an empty System if error.

Template for the TXT file

The text in italics is only helpful, should not appear in the template.
'Number of inputs' 'Number of outputs'
'Number of rules for the 1st ouput' 'Number of rules for the 2nd ouput' ...
Rules: For each output, for each rule in this output:
Antecedents. For each input:
'Type membership function' 'Parameters separated by spaces'
Consequents:
'Affine term' *and for each input* **'consequents terms separated by spaces'**
'lower limit for input 1' 'upper limit for input 1'
'lower limit for input 2' 'upper limit for input 2'
... for all the inputs
'lower limit for output 1' 'upper limit for output 1'
'lower limit for output 2' 'upper limit for output 2'
... for all the outputs

Examples:

example.cpp, and matlab_utilities/txt2fis.cpp.

0.9.4 Variable Documentation

0.9.4.1 FLT::MF_NAMES [static]

Initial value:

```
{
  "ANYMF",
  "CONSTMF",
  "GAUSSMF",
  "GAUSS2MF",
  "GBELLMF",
  "PIMF",
  "PSIGMF",
  "SMF",
  "SIGMF",
  "SIG2MF",
  "TRAPMF",
  "TRIMF",
  "ZMF"
}
```

Names of the Membership functions.

Remarks

FLT::TYPE_MF, FLT::MF_NAMES and FLT::MF_PARAM_NUMBER must be consistent.

0.9.4.2 FLT::MF_PARAM_NUMBER [static]

Initial value:

```
{
  0,
  1,
  2,
  4,
  3,
  4,
  4,
  2,
  2,
  4,
  4,
  3,
  2
}
```

Initial number of parameters of each Membership function in FLT::TYPE_MF.

Remarks

These values are used for initialization, in the code is better use the `num_params` function from Membership class.

FLT::TYPE_MF, FLT::MF_NAMES and FLT::MF_PARAM_NUMBER must be consistent.

Class Documentation

0.10 FLT::Membership Class Reference

This class contains methods and attributes common to all Membership functions.

`#include <membership.hpp>`

Public Member Functions

- virtual double eval (double x) const =0

  Evaluates the Membership function.

- virtual double evalder (double x) const =0

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- virtual double paramder (size_t parameter, double x) const =0

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- virtual int test (void) const

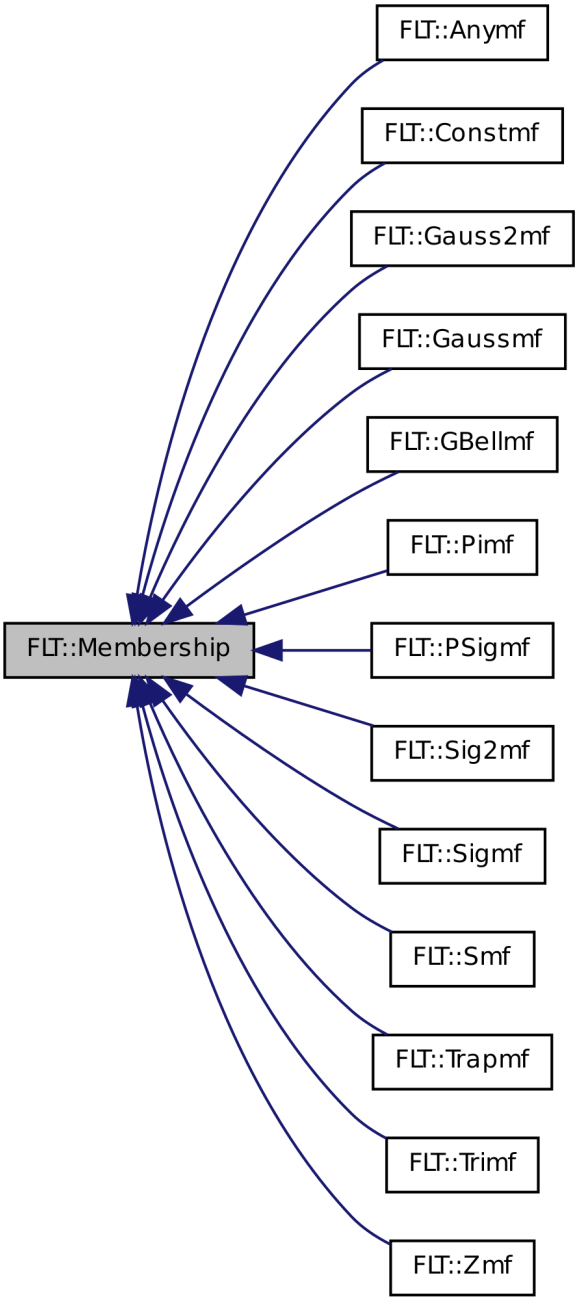
  This function checks parameters in the Membership function, and corrects them if possible.

- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

Inheritance diagram for FLT::Membership:



Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

- Membership & **operator=** (const Membership &P)
- **Membership** (const Membership &P)

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.10.1 Detailed Description

This class contains methods and attributes common to all Membership functions. This class can store the type of Membership function and all its parameters and evaluates the function and its derivatives.

See also

FLT::TYPE_MF for Membership functions details.
FLT::MF_NAMES for Membership functions names.

FLT::MF_PARAM_NUMBER for initial number of parameters.
For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt
FLT::createMF virtual constructor.

Examples:

matlab_utilities/fuzderantec.cpp.

0.10.2 Member Function Documentation

0.10.2.1 int Membership::edit (size_t *index,* double *value*)

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.10.2.2 void Membership::edit (const double *const *const *parameters*)

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.10.2.3 virtual double FLT::Membership::eval (double *x*) const [pure virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implemented in FLT::Anymf, FLT::Constmf, FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Pimf, FLT::PSigmf, FLT::Smf, FLT::Sigmf, FLT::Sig2mf, FLT::Trapmf, FLT::Trimf, and FLT::Zmf.

0.10.2.4 virtual double FLT::Membership::evalder (double *x*) const [pure virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implemented in FLT::Anymf, FLT::Constmf, FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Pimf, FLT::PSigmf, FLT::Smf, FLT::Sigmf, FLT::Sig2mf, FLT::Trapmf, FLT::Trimf, and FLT::Zmf.

0.10.2.5 size_t Membership::num_params (void) const

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.10.2.6 virtual double FLT::Membership::paramder (size_t *parameter,* double *x*) const [pure virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implemented in FLT::Anymf, FLT::Constmf, FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Pimf, FLT::PSigmf, FLT::Smf, FLT::Sigmf, FLT::Sig2mf, FLT::Trapmf, FLT::Trimf, and FLT::Zmf.

0.10.2.7 double Membership::read (size_t *index*) const

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.10.2.8 int Membership::test (void) const [virtual]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented in FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Trapmf, and FLT::Trimf.

0.10.2.9 int Membership::type (TYPE_MF *type_mf*)

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.10.2.10 TYPE_MF Membership::type (void) const

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

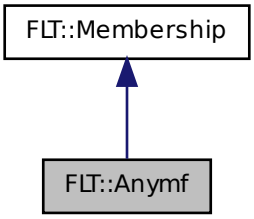
- membership.hpp
- membership.cpp

0.11 FLT::Anymf Class Reference

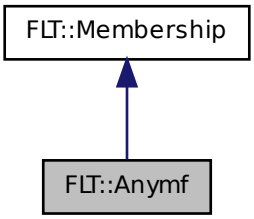
Any Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Anymf:



Collaboration diagram for FLT::Anymf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- virtual int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.11.1 Detailed Description

Any Membership function. Anymf function is used to remove a variable from the antecedent of a Rule. Anymf function is used to remove a variable from the antecedent of a Rule. It doesn't have parameters.

For example: *If x1 is A1 and x3 is A2 ...* is created using *x2 is ANYMF*.

Remarks

Anymf never must be evaluated, use `if` to avoid its evaluation.

0.11.2 Member Function Documentation

0.11.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.11.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.11.2.3 double Anymf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.11.2.4 double Anymf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.11.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.11.2.6 double Anymf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.11.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.11.2.8 int Membership::test (void) const [virtual, inherited]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented in FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Trapmf, and FLT::Trimf.

0.11.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.11.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

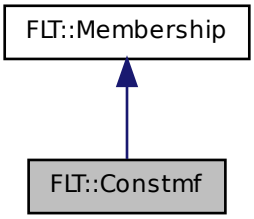
- membership.hpp
- membership.cpp

0.12 FLT::Constmf Class Reference

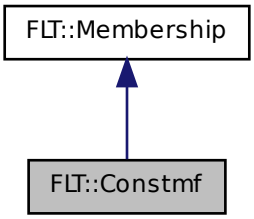
Constant Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Constmf:



Collaboration diagram for FLT::Constmf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- **Constmf** (double c=1.0)
- virtual int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.12.1 Detailed Description

Constant Membership function. Constmf is a constant Membership function with 1 parameter.

This function always return its parameter.

$$\mu[c](x) = c,\ 0 \leq c \leq 1$$

0.12.2 Member Function Documentation

0.12.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.12.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.12.2.3 double Constmf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.12.2.4 double Constmf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.12.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.12.2.6 double Constmf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.12.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.12.2.8 int Membership::test (void) const [virtual, inherited]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented in FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Trapmf, and FLT::Trimf.

0.12.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.12.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

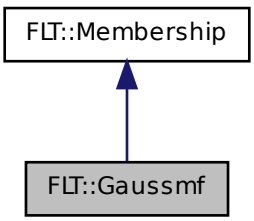
- membership.hpp
- membership.cpp

0.13 FLT::Gaussmf Class Reference

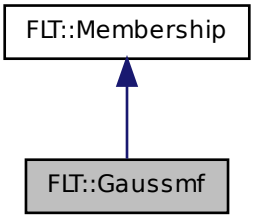
Gaussian Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Gaussmf:



Collaboration diagram for FLT::Gaussmf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- **Gaussmf** (double Center=0.5, double Beta=0.3)
- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.13.1 Detailed Description

Gaussian Membership function. Gaussmf is a Gaussian Membership function with 2 parameters.

$$\mu[c,\beta](x) = e^{-\left(\frac{x-c}{\beta}\right)^2},\ \beta > 0$$

0.13.2 Member Function Documentation

0.13.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.13.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.13.2.3 double Gaussmf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.13.2.4 double Gaussmf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.13.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.13.2.6 double Gaussmf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.13.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.13.2.8 int Gaussmf::test (void) const [virtual]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented from FLT::Membership.

0.13.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.13.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

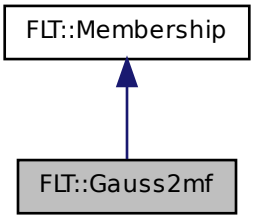
- membership.hpp
- membership.cpp

0.14 FLT::Gauss2mf Class Reference

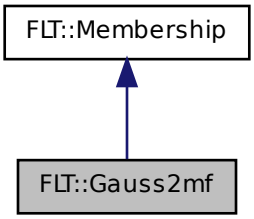
Double gaussian Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Gauss2mf:



Collaboration diagram for FLT::Gauss2mf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- **Gauss2mf** (double Center1=0.45, double Beta1=0.25, double Center2=0.55, double Beta2=0.25)
- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.14.1 Detailed Description

Double gaussian Membership function. Gauss2mf is a Double Gaussian Membership function with 4 parameters.

It is obtained by the product of 2 Gaussmf functions.

$$\mu_{Gauss2mf}(c_1,\beta_1,c_2,\beta_2) = \begin{cases} \mu_{Gaussmf}(c_1,\beta_1) & \text{if } x < c_1 \leq c_2 \\ \mu_{Gaussmf}(c_2,\beta_2) & \text{if } x > c_2, c_1 \leq c_2 \\ 1 & \text{if } c_1 \leq x \leq c_2 \text{ and } c_1 \leq c_2 \\ \mu_{Gaussmf}(c_1,\beta_1)\mu_{Gaussmf}(c_2,\beta_2) & \text{if } c_1 > c_2 \end{cases}$$

0.14.2 Member Function Documentation

0.14.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.14.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.14.2.3 double Gauss2mf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.14.2.4 double Gauss2mf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.14.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.14.2.6 double Gauss2mf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.14.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.14.2.8 int Gauss2mf::test (void) const [virtual]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented from FLT::Membership.

0.14.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.14.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

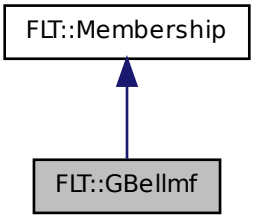
- membership.hpp
- membership.cpp

0.15 FLT::GBellmf Class Reference

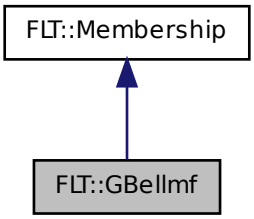
Bell Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::GBellmf:



Collaboration diagram for FLT::GBellmf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- **GBellmf** (double a=0.25, double b=2.5, double c=0.5)
- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.15.1 Detailed Description

Bell Membership function. GBellmf is a bell Membership function with 3 parameters.

$$\mu[\alpha,\beta,c](x) = \frac{1}{1+\left|\frac{x-c}{\alpha}\right|^{2\beta}},\ \alpha \neq 0$$

0.15.2 Member Function Documentation

0.15.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.15.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.15.2.3 double GBellmf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.15.2.4 double GBellmf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.15.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.15.2.6 double GBellmf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.15.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.15.2.8 int GBellmf::test (void) const [virtual]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented from FLT::Membership.

0.15.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.15.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

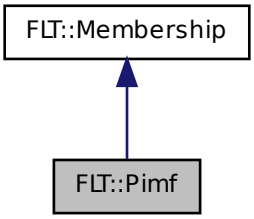
- membership.hpp
- membership.cpp

0.16 FLT::Pimf Class Reference

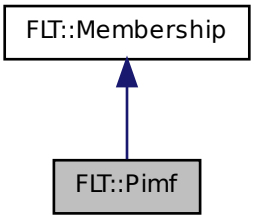
Pi (S-Z) Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Pimf:



Collaboration diagram for FLT::Pimf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- **Pimf** (double a=0.05, double b=0.45, double c=0.55, double d=0.95)
- virtual int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.16.1 Detailed Description

Pi (S-Z) Membership function. Pimf is a S-Z Membership function with 4 parameters.

It is obtained by the product of a Smf and a Zmf Membership functions.

$$\mu[a,b,c,d](x) = \mu_S[a,b](x) \cdot \mu_Z[c,d](x)$$

0.16.2 Member Function Documentation

0.16.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.16.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.16.2.3 double Pimf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.16.2.4 double Pimf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.16.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.16.2.6 double Pimf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.16.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.16.2.8 int Membership::test (void) const [virtual, inherited]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented in FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Trapmf, and FLT::Trimf.

0.16.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.16.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

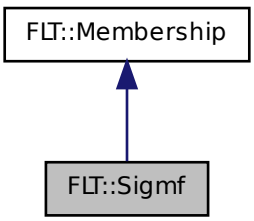
- membership.hpp
- membership.cpp

0.17 FLT::Sigmf Class Reference

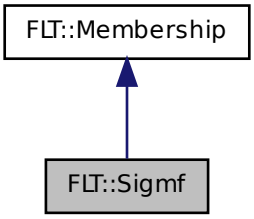
Sigmoidal Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Sigmf:



Collaboration diagram for FLT::Sigmf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- **Sigmf** (double a=14, double c=0.25)
- virtual int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.17.1 Detailed Description

Sigmoidal Membership function. Sigmf is a sigmoidal Membership function with 2 parameters.

$$\mu[\alpha, c](x) = \frac{1}{1 + e^{\alpha(c-x)}}$$

0.17.2 Member Function Documentation

0.17.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.17.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.17.2.3 double Sigmf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.17.2.4 double Sigmf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.17.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.17.2.6 double Sigmf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.17.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.17.2.8 int Membership::test (void) const [virtual, inherited]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented in FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Trapmf, and FLT::Trimf.

0.17.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.17.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

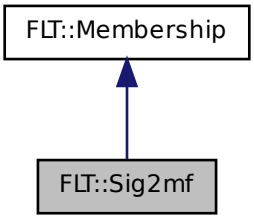
- membership.hpp
- membership.cpp

0.18 FLT::Sig2mf Class Reference

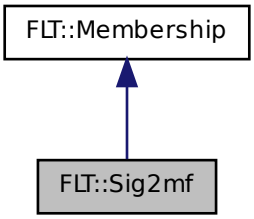
Difference of sigmoidals Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Sig2mf:



Collaboration diagram for FLT::Sig2mf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- **Sig2mf** (double a1=11, double c1=0.25, double a2=11, double c2=0.75)
- virtual int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double ∗const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double ∗ param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.18.1 Detailed Description

Difference of sigmoidals Membership function. Sig2mf is a double sigmoidal Membership function with 4 parameters.

It is obtained by the difference between 2 Sigmf functions.

$$\mu[\alpha_1, c_1, \alpha_2, c_2](x) = \mu_{Sigm}[\alpha_1, c_1](x) - \mu_{Sigm}[\alpha_2, c_2](x)$$

0.18.2 Member Function Documentation

0.18.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.18.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.18.2.3 double Sig2mf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.18.2.4 double Sig2mf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.18.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.18.2.6 double Sig2mf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.18.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.18.2.8 int Membership::test (void) const [virtual, inherited]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented in FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Trapmf, and FLT::Trimf.

0.18.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.18.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

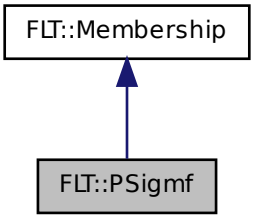
- membership.hpp
- membership.cpp

0.19 FLT::PSigmf Class Reference

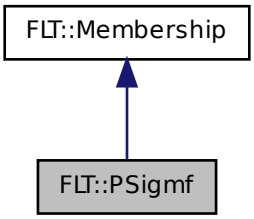
Product of sigmoidals Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::PSigmf:



Collaboration diagram for FLT::PSigmf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- **PSigmf** (double a1=11, double c1=0.25, double a2=-11, double c2=0.75)
- virtual int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.19.1 Detailed Description

Product of sigmoidals Membership function. PSigmf is the product of 2 sigmoidal Membership functions (Sigmf), and it has 4 parameters.

$$\mu[\alpha_1, c_1, \alpha_2, c_2](x) = \mu_{Sigm}[\alpha_1, c_1](x) \cdot \mu_{Sigm}[\alpha_2, c_2](x)$$

0.19.2 Member Function Documentation

0.19.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.19.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.19.2.3 double PSigmf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.19.2.4 double PSigmf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.19.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.19.2.6 double PSigmf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.19.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.19.2.8 int Membership::test (void) const [virtual, inherited]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented in FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Trapmf, and FLT::Trimf.

0.19.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.19.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

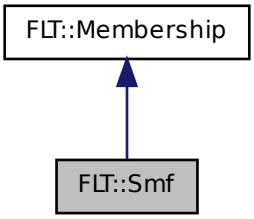
- membership.hpp
- membership.cpp

0.20 FLT::Smf Class Reference

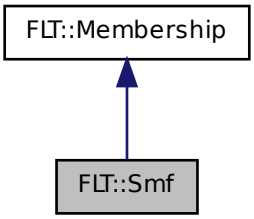
S Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Smf:



Collaboration diagram for FLT::Smf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- **Smf** (double a=0.05, double b=0.45)
- virtual int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.20.1 Detailed Description

S Membership function. Smf is a S Membership function with 2 parameters.

$$\mu_{Smf}[a,b](x) = \begin{cases} 0 & \text{if } x \leq a \\ 2\left(\frac{x-a}{b-a}\right)^2 & \text{if } a < x \leq \frac{a+b}{2} \\ 1 - 2\left(\frac{b-x}{b-a}\right)^2 & \text{if } \frac{a+b}{2} < x < b \\ 1 & \text{if } x \geq b \end{cases}$$

0.20.2 Member Function Documentation

0.20.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.20.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.20.2.3 double Smf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.20.2.4 double Smf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.20.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.20.2.6 double Smf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.20.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.20.2.8 int Membership::test (void) const [virtual, inherited]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented in FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Trapmf, and FLT::Trimf.

0.20.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.20.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

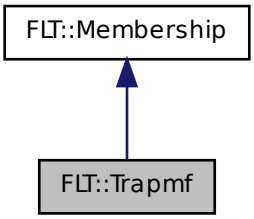
- membership.hpp
- membership.cpp

0.21 FLT::Trapmf Class Reference

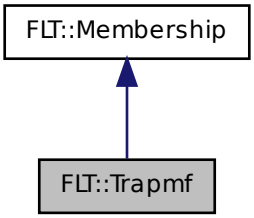
Trapezoidal Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Trapmf:



Collaboration diagram for FLT::Trapmf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- **Trapmf** (double a=0.05, double b=0.45, double c=0.55, double d=0.95)
- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.21.1 Detailed Description

Trapezoidal Membership function. Trapmf is a trapeziodal Membership function with 4 parameters.

$$\mu_{Trapmf}[a,b,c,d](x) = \begin{cases} \frac{x-a}{b-a} & \text{if } a < x < b \\ 1 & \text{if } b \leq x \leq c \\ \frac{d-x}{d-c} & \text{if } c < x < d \\ 1 & \text{if } x \geq d \end{cases}$$

$$a < b < c < d$$

0.21.2 Member Function Documentation

0.21.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.21.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.21.2.3 double Trapmf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.21.2.4 double Trapmf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.21.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.21.2.6 double Trapmf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.21.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If index is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.21.2.8 int Trapmf::test (void) const [virtual]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented from FLT::Membership.

0.21.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.21.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

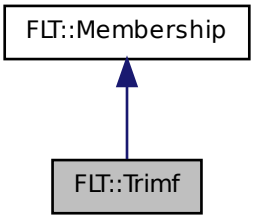
- membership.hpp
- membership.cpp

0.22 FLT::Trimf Class Reference

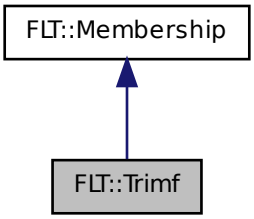
Triangular Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Trimf:



Collaboration diagram for FLT::Trimf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- **Trimf** (double a=0.0, double b=0.5, double c=1.0)
- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.22.1 Detailed Description

Triangular Membership function. Trimf is a triangular Membership function with 3 parameters.

$$\mu_{Trimf}[a,b,c](x) = \begin{cases} \frac{x-a}{b-a} & \text{if } a < x \leq b \\ \frac{c-x}{c-b} & \text{if } b < x < c \\ 0 & \text{if } x \geq c \end{cases}$$

$$a < b < c$$

0.22.2 Member Function Documentation

0.22.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.22.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.22.2.3 double Trimf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.22.2.4 double Trimf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.22.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.22.2.6 double Trimf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.22.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.22.2.8 int Trimf::test (void) const [virtual]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented from FLT::Membership.

0.22.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.22.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

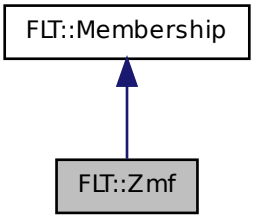
- membership.hpp
- membership.cpp

0.23 FLT::Zmf Class Reference

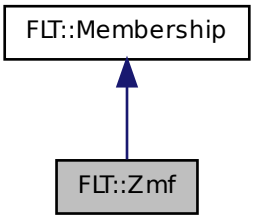
Z Membership function.

`#include <membership.hpp>`

Inheritance diagram for FLT::Zmf:



Collaboration diagram for FLT::Zmf:



Public Member Functions

- double eval (double x) const

  Evaluates the Membership function.

- double evalder (double x) const

  Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

- double paramder (size_t parameter, double x) const

  Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

- **Zmf** (double a=0.55, double b=0.95)
- virtual int test (void) const

  This function checks parameters in the Membership function, and corrects them if possible.

- size_t num_params (void) const

  Reads the number of parameters that define the Membership function.

- TYPE_MF type (void) const

  Returns the type of Membership function.

- int type (TYPE_MF type_mf)

  Changes the Membership function type.

- int edit (size_t index, double value)

  Changes a parameter.

- void edit (const double *const parameters)

  Changes the vector of parameters.

- double read (size_t index) const

  Reads a parameter of the Membership function.

- TNT::Array1D< double > read (void) const

  Returns the vector of parameters.

Protected Attributes

- double * param

  Vector of parameters that define the Membership function.

- size_t n

  Number of parameters of the Membership function.

- TYPE_MF type_mf

  Type of Membership function.

0.23.1 Detailed Description

Z Membership function. Zmf is a Z Membership function with 2 parameters.

$$\mu_{Zmf}[a,b](x) = \begin{cases} 1 & \text{if } x \leq a \\ 1 - 2\left(\frac{x-a}{a-b}\right)^2 & \text{if } a < x \leq \frac{a+b}{2} \\ 2\left(\frac{b-x}{a-b}\right)^2 & \text{if } \frac{a+b}{2} < x < b \\ 0 & \text{if } x \geq b \end{cases}$$

0.23.2 Member Function Documentation

0.23.2.1 int Membership::edit (size_t *index,* double *value*) [inherited]

Changes a parameter.

Returns

Returns 0 if no errors occurred.
An error means the index value is greater than the number of function parameters.

Note

The parameter value is not checked. If you want to check it, you must use Membership::test method.

0.23.2.2 void Membership::edit (const double *const *parameters*) [inherited]

Changes the vector of parameters.

Note

The values of the parameters are not checked. If you want to check it, you must use Membership::test method.

0.23.2.3 double Zmf::eval (double *x*) const [virtual]

Evaluates the Membership function.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.23.2.4 double Zmf::evalder (double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to x, $d\mu(x)/dx$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.23.2.5 size_t Membership::num_params (void) const [inherited]

Reads the number of parameters that define the Membership function.

We recommend using this function instead of the constant FLT::MF_PARAM_-NUMBER, since it is possible to create Membership functions with a variable number of parameters.

Examples:

matlab_utilities/fuzderantec.cpp.

0.23.2.6 double Zmf::paramder (size_t *parameter,* double *x*) const [virtual]

Evaluates the derivative of the Membership function with respect to a parameter, $d\mu(x)/dparam$.

Remarks

You must first ensure that the function is not Anymf.

Implements FLT::Membership.

0.23.2.7 double Membership::read (size_t *index*) const [inherited]

Reads a parameter of the Membership function.

Returns

If `index` is greater than the number of function parameters, an error is sent to the standard error stream and return 0.

0.23.2.8 int Membership::test (void) const [virtual, inherited]

This function checks parameters in the Membership function, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

Reimplemented in FLT::Gaussmf, FLT::Gauss2mf, FLT::GBellmf, FLT::Trapmf, and FLT::Trimf.

0.23.2.9 int Membership::type (TYPE_MF *type_mf*) [inherited]

Changes the Membership function type.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized.

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.23.2.10 TYPE_MF Membership::type (void) const [inherited]

Returns the type of Membership function.

The type of Membership function is represented with the enumeration FLT::TYPE_-MF.

Examples:

matlab_utilities/fuzcomb.cpp, and matlab_utilities/fuzderantec.cpp.

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

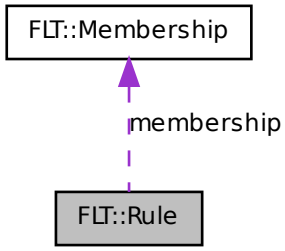
- membership.hpp
- membership.cpp

0.24 FLT::Rule Class Reference

This class contains methods and attributes common to fuzzy Rules.

`#include <rule.hpp>`

Collaboration diagram for FLT::Rule:



Public Member Functions

- size_t n_inputs (void) const

  Extract the number of inputs of the Rule.

- int test (void) const

  This function checks the parameters of all Membership functions in the Rule, and corrects them if possible.

- int changeTSK (double value, size_t input)

  Changes a parameter in the TSK consequent.

- void changeTSK (const double ∗const TSK)

Changes the TSK consequent.

- double readTSK (size_t input) const

  Reads a TSK parameter.

- TNT::Array1D< double > readTSK (void) const

  Reads the TSK consequent.

- Membership * readFunction (size_t input)

  Extracts a pointer to a Membership function.

- int changeFunction (const Membership &P, size_t input)

  Changes a Membership function of the Rule.

- int changeFunction (TYPE_MF type, size_t input)

  Changes the type af a Membership function.

- void initialize (size_t in)

  Initializes the Rule for `in` *inputs.*

- size_t NumberOfAntecedents (void)

  Gets the number of parameters stored in the antecedents of the Rule.

- int setAntecedents (const double *const parameters)

  Sets the parameters of the antecedents of the Rule.

- TNT::Array1D< double > getAntecedents (void)

  Gets the parameters of the antecedents of the Rule.

- void setConsequents (const double *const parameters)

  Sets the consequents of the Rule.

- TNT::Array1D< double > getConsequents (void)

  Gets the consequents of the Rule.

- double activation (const double *const point, double *dW=NULL) const

  Calculates the matching degree of the rule in `point` *and, optionally, its derivative.*

- Rule & **operator=** (const Rule &R)
- **Rule** (const Rule &R)
- **Rule** (size_t in)

Private Member Functions

- void clean (void)

  Cleans the dynamic memory.

Private Attributes

- size_t in

  Number of inputs of the Rule.

- double ∗ TSK

  TSK consequent of the Rule.

- Membership ∗∗ membership

  Membership functions of the Rule.

0.24.1 Detailed Description

This class contains methods and attributes common to fuzzy Rules. This class stores a completely general Takagi-Sugeno Rule.

See also

You can use the Anymf Membership function to remove a variable from the antecedent of a Rule.
For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt

Examples:

matlab_utilities/fuz2mat.cpp, matlab_utilities/fuzcomb.cpp, and matlab_utilities/mat2fuz.cpp.

0.24.2 Member Function Documentation

0.24.2.1 double Rule::activation (const double ∗const *point,* double ∗ *dW =* NULL) const

Calculates the matching degree of the rule in `point` and, optionally, its derivative.

Optionally, this function also calculates the derivative of the degree of activation of the rule with respect to the input at `point`.

dW represents dW/dX, and its length is equal to the number of entries in the rule.

The length of `point` must match the number of inputs in the rule.

0.24.2.2 int Rule::changeFunction (const Membership & *P,* size_t *input*)

Changes a Membership function of the Rule.

Returns

Returns 0 if no errors occurred.
An error means the input is greater than the number of Rule inputs.

Examples:

matlab_utilities/fuzcomb.cpp.

0.24.2.3 int Rule::changeFunction (TYPE_MF *type,* size_t *input*)

Changes the type af a Membership function.

Returns

Returns 0 if no errors occurred.
An error means the type of Membership function is not recognized (1) or `input` is greater than the number of Rule inputs (2).

Note

The parameters of the new Membership function are not assigned, you must do so by Membership::edit method.

0.24.2.4 int Rule::changeTSK (double *value,* size_t *input*)

Changes a parameter in the TSK consequent.

Returns

Returns 0 if no errors occurred.
An error means the input is greater than the number of parameter of the consequent.

Note

TSK(0) is the affine term, TSK(1) is the term associated to the 1st input variable, ...

Examples:

matlab_utilities/fuzcomb.cpp.

0.24.2.5 void Rule::initialize (size_t *in*)

Initializes the Rule for `in` inputs.

Empty membership functions are used. You can change them with `changeFunction` method or using the Membership class methods.

0.24.2.6 size_t Rule::NumberOfAntecedents (void)

Gets the number of parameters stored in the antecedents of the Rule.

Note

The function `NumberOfConsequents(void)` is not necessary because this value is `in+1`.

Examples:

matlab_utilities/mat2fuz.cpp.

0.24.2.7 Membership * Rule::readFunction (size_t *input*)

Extracts a pointer to a Membership function.

Returns

Returns NULL if the input is greater than the number of Rule inputs.

Examples:

matlab_utilities/fuzcomb.cpp.

0.24.2.8 double Rule::readTSK (size_t *input*) const

Reads a TSK parameter.

Returns

Returns the selected parameter. If `input` is greater than the number of Rule inputs, an error is sent to the standard error stream and return 0.

Examples:

matlab_utilities/fuzcomb.cpp.

0.24.2.9 int Rule::setAntecedents (const double *const *parameters*)

Sets the parameters of the antecedents of the Rule.

Returns

Returns the result of running the Rule::test() method.

Examples:

matlab_utilities/mat2fuz.cpp.

0.24.2.10 int Rule::test (void) const

This function checks the parameters of all Membership functions in the Rule, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

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

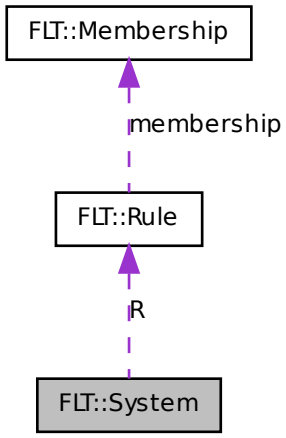
- rule.hpp
- rule.cpp

0.25 FLT::System Class Reference

This class contains methods and attributes common to fuzzy Systems.

`#include <system.hpp>`

Collaboration diagram for FLT::System:



Public Member Functions

- size_t inputs (void) const

  Returns the number of inputs of the System.

- size_t outputs (void) const

  Returns the number of outputs of the System.

- size_t rules (size_t output) const

  Returns the number of rules for the output `output` *of the System.*

- TNT::Array1D< size_t > rules (void) const

  Returns the number of rules for each output of the System.

- TNT::Array1D< double > in_low (void) const
  Returns the low limits of the System inputs.
- double in_low (size_t input) const
  Returns the low limit of the input `input`.
- TNT::Array1D< double > in_high (void) const
  Returns the high limits of the System inputs.
- double in_high (size_t input) const
  Returns the high limit of the input `input`.
- TNT::Array1D< double > out_low (void) const
  Returns the low limits of the System outputs.
- double out_low (size_t output) const
  Returns the low limit of the output `output`.
- TNT::Array1D< double > out_high (void) const
  Returns the high limits of the System outputs.
- double out_high (size_t output) const
  Returns the high limit of the output `output`.
- void in_low (double *limits)
  Changes the low limits of the System inputs.
- void in_high (double *limits)
  Changes the high limits of the System inputs.
- void out_low (double *limits)
  Changes the low limits of the System outputs.
- void out_high (double *limits)
  Changes the high limits of the System outputs.
- int checkInputLimits (const double *const point)
  Checks if `point` *meets the input limits of the system.*
- int checkOutputLimits (const double *const output)
  Checks if `output` *meets the output limits of the system.*

- int test (void) const

  This function checks the parameters of all Membership functions of the System, and corrects them if possible.

- int changeRule (const Rule &R, size_t r, size_t output)

  Changes a Rule in the System.

- int readRule (Rule &R, size_t output, size_t r) const

  Reads a Rule from the System.

- Rule ∗ readRule (size_t output, size_t r)

  Extracts a pointer to a Rule from the System.

- void initialize (size_t in, size_t out, size_t ∗N)

  Initializes the System with `in` *inputs,* `out` *outputs and* `N`*[out] rules for each output.*

- size_t NumberOfAntecedents (void)

  Gets the number of parameters stored in the antecedents of the System.

- int setAntecedents (const double ∗const parameters)

  Sets all the parameters of the antecedents of the System.

- TNT::Array1D< double > getAntecedents (void)

  Gets all the parameters of the antecedents of the System.

- size_t NumberOfConsequents (void)

  Gets the number of parameters stored in the consequents of the System.

- void setConsequents (const double ∗const parameters)

  Sets all the consequets of the System.

- TNT::Array1D< double > getConsequents (void)

  Gets the consequets of the System.

- TNT::Array1D< double > evaluate (const double ∗const point)

  Evaluates the System for input `point`*.*

- System & **operator=** (const System &S)
- **System** (const System &S)
- **System** (size_t in, size_t out, size_t ∗N)

Private Attributes

- size_t in

  Number of inputs of the System.

- size_t out

  Number of outputs of the System.

- size_t * N

  Number of rules of the System for each output.

- double * low_in

  Low limits for the inputs of the System.

- double * low_out

  Low limits for the outputs of the System.

- double * high_in

  High limits for the inputs of the System.

- double * high_out

  High limits for outputs of the System.

- Rule ** R

  System Rules.

0.25.1 Detailed Description

This class contains methods and attributes common to fuzzy Systems. This class stores a completely general Takagi-Sugeno fuzzy System.

See also

For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt

Examples:

example.cpp, matlab_utilities/activation.cpp, matlab_utilities/antec2mat.cpp, matlab_utilities/aproxjac.cpp, matlab_utilities/aproxlinear.cpp, matlab_utilities/conseq2mat.cpp, matlab_utilities/extractPoints.cpp, matlab_utilities/fis2txt.cpp, matlab_utilities/fuz2mat.cpp,

matlab_utilities/fuzcomb.cpp, matlab_utilities/fuzderantec.cpp, matlab_utilities/fuzderconseq.cpp, matlab_utilities/fuzderparam.cpp, matlab_utilities/fuzeval.cpp, matlab_utilities/fuzjac.cpp, matlab_utilities/fuzlinear.cpp, matlab_utilities/fuzprint.cpp, matlab_utilities/initializeController.cpp, matlab_utilities/Kalmanantec.cpp, matlab_utilities/Kalmanconseq.cpp, matlab_utilities/Kalmanfuz.cpp, matlab_utilities/mat2antec.cpp, matlab_utilities/mat2conseq.cpp, matlab_utilities/mat2fuz.cpp, matlab_utilities/subsystem.cpp, and matlab_utilities/txt2fis.cpp.

0.25.2 Member Function Documentation

0.25.2.1 int System::changeRule (const Rule & *R,* size_t *r,* size_t *output*)

Changes a Rule in the System.

Returns

Returns 0 if no errors.
An error means that the rule or the output indicated are beyond the limits of the system.

0.25.2.2 int System::checkInputLimits (const double *const *point*)

Checks if `point` meets the input limits of the system.

Returns

Returns 0 if `low_in[i] <= point[i] <= high_in[i]`, for `i = 1..in`.
If `point[i] < low_in[i]` or `point[i] > high_in[i]`, returns 1 + the lowest value of `i` that breaks limits.

0.25.2.3 int System::checkOutputLimits (const double *const *output*)

Checks if `output` meets the output limits of the system.

Returns

Returns 0 if `low_out[j] <= output[j] <= high_out[j]`, for `i = j..out`.
If `output[j] < low_out[j]` or `output[j] > high_out[j]`, returns 1 + the lowest value of `j` that breaks limits.

0.25.2.4 Array1D< double > System::evaluate (const double *const *point*)

Evaluates the System for input `point`.

If `point` or `output` are beyond the limits of the system, a warning message is sent to the standard error stream.

Examples:

matlab_utilities/fuzeval.cpp.

0.25.2.5 Array1D< double > System::getAntecedents (void)

Gets all the parameters of the antecedents of the System.

The antecedents must be sorted for each output, for each rule.

Examples:

matlab_utilities/antec2mat.cpp, and matlab_utilities/fuz2mat.cpp.

0.25.2.6 Array1D< double > System::getConsequents (void)

Gets the consequets of the System.

The consequets are sorted for each output, for each rule, the affine term and the consequents of each input.

Examples:

matlab_utilities/conseq2mat.cpp, and matlab_utilities/fuz2mat.cpp.

0.25.2.7 int System::readRule (Rule & *R,* size_t *output,* size_t *r*) const

Reads a Rule from the System.

Returns

Returns 0 if no errors.
An error means that the rule or the output indicated are beyond the limits of the system.

Examples:

matlab_utilities/activation.cpp, matlab_utilities/antec2mat.cpp, matlab_utilities/conseq2mat.cpp, matlab_utilities/fuz2mat.cpp,

matlab_utilities/fuzcomb.cpp, matlab_utilities/fuzderantec.cpp, matlab_utilities/mat2antec.cpp, matlab_utilities/mat2conseq.cpp, and matlab_utilities/mat2fuz.cpp.

0.25.2.8 Rule * System::readRule (size_t *output,* size_t *r*)

Extracts a pointer to a Rule from the System.

Returns

Returns NULL if the rule or the output indicated are beyond the limits of the system.

0.25.2.9 int System::setAntecedents (const double *const *parameters*)

Sets all the parameters of the antecedents of the System.

The antecedents must be sorted for each output, for each rule.

Returns

Returns the result of running the System::test() method.

Examples:

matlab_utilities/mat2antec.cpp, and matlab_utilities/mat2fuz.cpp.

0.25.2.10 void System::setConsequents (const double *const *parameters*)

Sets all the consequets of the System.

The consequets must be sorted for each output, for each rule, the affine term and the consequents of each input.

Examples:

matlab_utilities/mat2conseq.cpp, and matlab_utilities/mat2fuz.cpp.

0.25.2.11 int System::test (void) const

This function checks the parameters of all Membership functions of the System, and corrects them if possible.

Returns

Returns 0 if no errors, 1 if an error occurs, and -1 if the parameters were corrected.

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

- system.hpp
- system.cpp

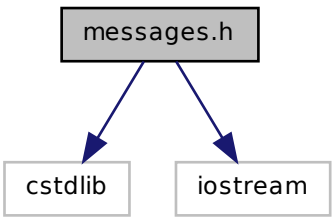
File Documentation

0.26 messages.h File Reference

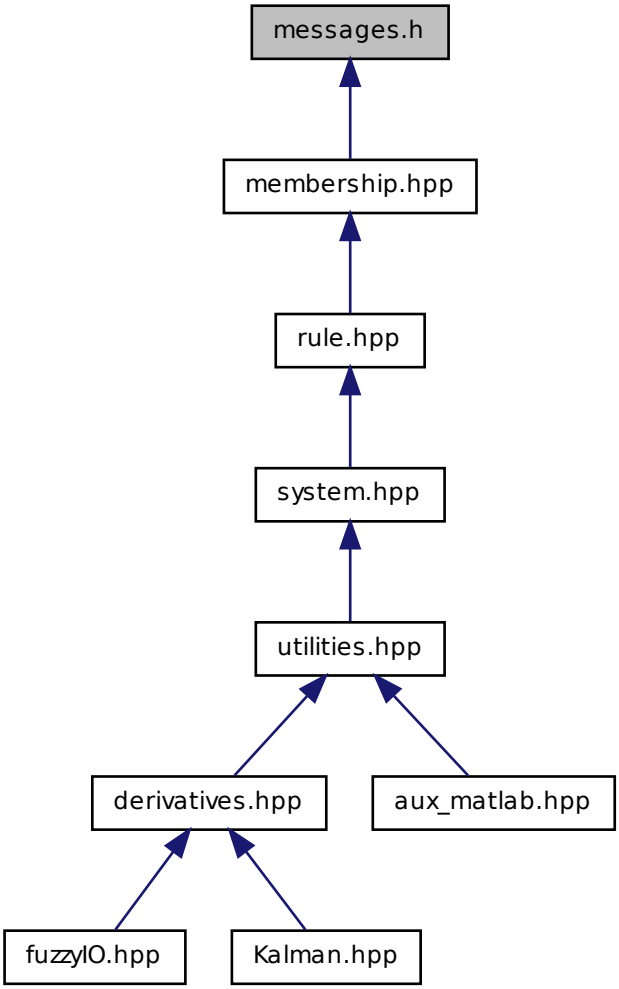
Defines output messages (informational messages, warnings, errors, ...).

```
#include <cstdlib>
#include <iostream>
```

Include dependency graph for messages.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace FLT

Fuzzy Logic Tools (FLT) namespace.

Defines

- #define WARNINGMSG(menssage) {std::cerr << menssage << std::endl;}

  Generates the given wanring message.

- #define ERRORMSG(menssage) {std::cerr << menssage << std::endl; return;}

  Generates the given error and return.

- #define ERRORMSGVAL(menssage, value) {std::cerr << menssage << std::endl; return value;}

  Generates the given error and return `value`.

- #define MAX_CHAR_LONG 256

  Maximum length of messages strings.

- #define MAX_FILE_NAME 256

  Maximum length of file names.

- #define PRECISION 10

  Precision for data extraction (for use with f/i/o/stream).

- #define **W_InputsLimitsExceeded** "Warning, the input exceeds the limits of the system."
- #define **W_OutputsLimitsExceeded** "Warning, the output exceeds the limits of the system."
- #define **E_Model** "Error reading model."
- #define **E_NoSugeno** "Model 'type' should be 'sugeno'."
- #define **E_NoOutputs** "Error, the model has not any Output."
- #define **E_NoInputs** "Error, the model has not any Input."
- #define **E_NoRules** "Error, some Outputs of the model have not any Rule."
- #define **E_No1Conseq** "Only one consequent for rule is allowed."
- #define **E_SameIns** "All models must have the same inputs."
- #define **E_NoCoherent** "The Controller is not coherent with the Plant."
- #define **E_InNoCoherent** "Inputs arguments are not coherent with fuzzy model."
- #define **E_BadModel** "Undefined or wrong fuzzy model.\nCheck rules, membership functions and theirs parameters"

- #define **E_NoCoherent_BadX** "The Controller is not coherent with the Plant, or the point is not coherent with the closed loop system."
- #define **E_ArgNoValid** "Empty and NaN matrices are not valid inputs arguments."
- #define **E_AccuracyNoNum** "The accuracy should be a number."
- #define **E_Acquire** "Error reading model or incorrect initial model."
- #define **E_Store** "Error storing data. Wrong vector size or any data not valid for membership function, i.e. a zero width in Gaussmf function."
- #define **E_NoFIS** "Fuzzy Model must be a Sugeno FIS structure."
- #define **E_CreateFIS** "Error at create FIS."
- #define **E_NoProd** "Implication method must be 'prod'."
- #define **E_FISOut** "Error writing FIS."
- #define **E_InputExceded** "Input limit exceeded."
- #define **E_OutputExceded** "Output limit exceeded."
- #define **E_RulesExceded** "Rule limit exceeded."
- #define **E_ParamExceded** "Parameter number exceeded the number of parameters of the membership function."
- #define **E_MFType** "Error reading membership function type:"
- #define **E_BadMF** "Membership function was not identified."
- #define **E_MFTypeDef** "Error in membership function type."
- #define **E_ParamMF** "Error reading membership function parameters."
- #define **E_NoValidFP** "Any parameter of membership function is not valid."
- #define **E_MFDef** "Error in membership function definition."
- #define **E_AnymfUse** "ANYMF function must not be evaluated. It is possible that there is an error in the sources, contact with authors, please."
- #define **E_Conseq** "It is not posible read consequent type in rule:"
- #define **E_ParamConseq** "Error reading TSK's consequent parameters."
- #define **E_BadConseq** "Not valid cosequent."
- #define **E_FileInput** "Input file error."
- #define **E_FileOutput** "Output file error."
- #define **E_Syntax** "Syntax error."
- #define **E_NumberParam** "Number of parameters incorrect."
- #define **E_NumberArg** "Error in number of arguments."
- #define **E_NumberArgIn** "Error in inputs arguments"
- #define **E_NumberArgOut** "Error in outputs arguments."
- #define **E_NumberRulesOut** "Error, the number of rules must be a vector of size equal to the number of system outputs."
- #define **E_Column** "The state vector must be a column vector."
- #define **E_In_Out_Column** "Input and output vector must be column vectors."

- #define **E_InNames** "Error reading names of inputs."
- #define **E_OutNames** "Error reading names of outputs."

- #define **E_Controller_Init** "Controller initialization error."
- #define **E_No_Points_Extracted** "No points were extracted."
- #define **E_Jacobian_Calculation** "It happened an error during the calculation of the Jacobian."
- #define **E_ArrayExceded** "Array limit exceded."
- #define **E_ChangingParameter** "Error changing the parameter."
- #define **E_WritingRule** "Error writing the rule"
- #define **E_NoSumProm** "Error, 'defuzzMethod' should be 'wtaver'."
- #define **E_RuleOutputFileVector** "Rules and Outputs should be a file vector."
- #define **E_PointCoherent** "The point/s should be coherent with fuzzy models."

- #define **E_In_No_Scalar** "The number of input must be a scalar."
- #define **E_Out_No_Scalar** "The number of output must be a scalar."
- #define **E_R_No_Scalar** "The number of rule must be a scalar."
- #define **E_Param_No_Scalar** "The number of the parameter must be a scalar."

- #define **E_Epoch_No_Scalar** "The number of epoch must be a scalar."
- #define **E_Alfa_No_Scalar** "Alfa must be a scalar."
- #define **E_Error_No_Scalar** "The error must be a scalar."
- #define **E_ModelError** "Modeling error."
- #define **E_H_GE_0** "'h' must be >= 0."
- #define **E_N_MESH_P** "Number of point to make the mesh must be >1."
- #define **E_PRECISION** "Precision must be >0."
- #define **U_FISName** "Unnamed"
- #define **U_FISInput** "Input"
- #define **U_FISOutput** "Output"
- #define **U_RuleAbbreviation** "R"
- #define **U_InputAbbreviation** "-In"
- #define **U_OutputAbreviation** "-Out"
- #define **U_If** "IF"
- #define **U_Is** "is"
- #define **U_And** "and"
- #define **U_OR** "or"
- #define **U_Then** "THEN"
- #define **U_SeeManual** "See also Fuzzy Logic Toolbox Manual."
- #define **U_SeeHelp** "see the help."
- #define **U_CheckModel** "Check model's definition."
- #define **U_MathException** "Warning. Math exception generated."
- #define **U_Overflow** "Happened an overflow."
- #define **U_FewData** "There are few data for modeling the system."
- #define **O_Input** "Input:"
- #define **O_OfIn** "of in"

- #define **O_OfOut** "of out"
- #define **O_RuleWeigh** "The rule's weigh:"
- #define **O_RuleConnection** "The rule's connection:"
- #define **O_AntecedentSize** "The size of rule's antecedent"
- #define **O_ConsequentSize** "The size of rule's consequent"
- #define **O_Rule** "Rule:"
- #define **O_DifferentOf** "is different of"
- #define **O_IsNotCorrect** "is not correct."
- #define **O_MF** "Membership Function:"
- #define **O_OfPlant** "of the Plant"
- #define **O_OfController** "of Controller"
- #define **O_GeneralError** "An error occurred. Please, check all parameters or contact with authors."

0.26.1 Detailed Description

Defines output messages (informational messages, warnings, errors, ...). Some macros for generating error messages are also defined to facilitate generation console or MATLAB© applications.

This file defines the messages used in all functions according to the following schedule:

E_* -> Error messages.

U_* -> User Messages.

O_* -> Other messages.

0.26.2 Define Documentation

0.26.2.1 #define ERRORMSG(*menssage*) {std::cerr << menssage << std::endl; return;}

Generates the given error and return.

This macro is used to generate an error.

The standard error stream is used, but if the code is being compiled in MATLAB©, uses the `mexErrMsgTxt` function from the MATLAB© API.

Examples:

matlab_utilities/activation.cpp, matlab_utilities/antec2mat.cpp, matlab_utilities/aproxjac.cpp, matlab_utilities/aproxlinear.cpp, matlab_utilities/conseq2mat.cpp, matlab_utilities/extractPoints.cpp,

matlab_utilities/fis2txt.cpp, matlab_utilities/fuz2mat.cpp, matlab_utilities/fuzcomb.cpp, matlab_utilities/fuzderantec.cpp, matlab_utilities/fuzderconseq.cpp, matlab_utilities/fuzderparam.cpp, matlab_utilities/fuzeval.cpp, matlab_utilities/fuzjac.cpp, matlab_utilities/fuzlinear.cpp, matlab_utilities/fuzprint.cpp, matlab_utilities/initializeController.cpp, matlab_utilities/Kalmanantec.cpp, matlab_utilities/Kalmanconseq.cpp, matlab_utilities/Kalmanfuz.cpp, matlab_utilities/mat2antec.cpp, matlab_utilities/mat2conseq.cpp, matlab_utilities/mat2fuz.cpp, matlab_utilities/subsystem.cpp, and matlab_utilities/txt2fis.cpp.

0.26.2.2 #define ERRORMSGVAL(*menssage, value*) {std::cerr << menssage << std::endl; return value;}

Generates the given error and return `value`.

This macro is used to generate an error and return a value.

The standard error stream is used, but if the code is being compiled in MATLAB©, uses the `mexErrMsgTxt` function from the MATLAB© API.

0.26.2.3 #define WARNINGMSG(*menssage*) {std::cerr << menssage << std::endl;}

Generates the given wanring message.

This macro is used to generate a warning message.

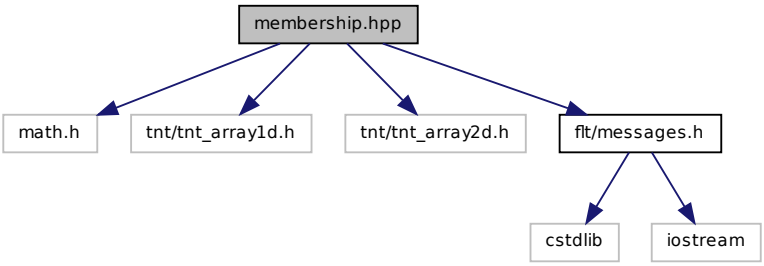
The standard error stream is used, but if the code is being compiled in MATLAB©, uses the `mexErrMsgTxt` function from the MATLAB© API.

0.27 membership.hpp File Reference

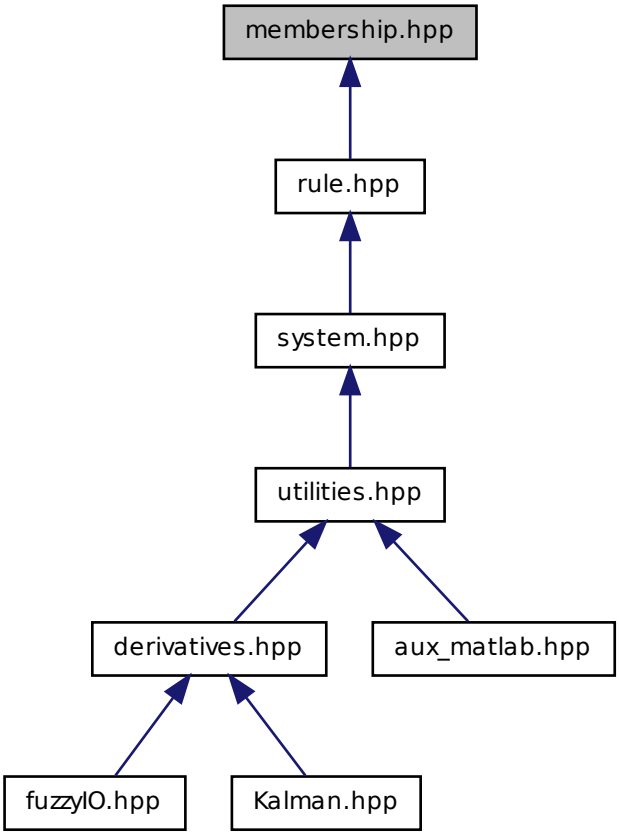
Defines Membership functions.

```
#include <math.h>
#include <tnt/tnt_array1d.h>
#include <tnt/tnt_array2d.h>
#include <flt/messages.h>
```

Include dependency graph for membership.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class FLT::Membership

  This class contains methods and attributes common to all Membership functions.

- class FLT::Anymf

  Any Membership function.

- class FLT::Constmf
  Constant Membership function.
- class FLT::Gaussmf
  Gaussian Membership function.
- class FLT::Gauss2mf
  Double gaussian Membership function.
- class FLT::GBellmf
  Bell Membership function.
- class FLT::Pimf
  Pi (S-Z) Membership function.
- class FLT::PSigmf
  Product of sigmoidals Membership function.
- class FLT::Smf
  S Membership function.
- class FLT::Sigmf
  Sigmoidal Membership function.
- class FLT::Sig2mf
  Difference of sigmoidals Membership function.
- class FLT::Trapmf
  Trapezoidal Membership function.
- class FLT::Trimf
  Triangular Membership function.
- class FLT::Zmf
  Z Membership function.

Namespaces

- namespace FLT
  Fuzzy Logic Tools (FLT) namespace.

Defines

- #define M_EPS 1e-16

  Epsilon value.

- #define MAX_SIZE_TYPE_NAME 16

  Maximum size for the names of the Membership functions.

Enumerations

- enum FLT::TYPE_MF {
  FLT::ANYMF = 0, FLT::CONSTMF, FLT::GAUSSMF, FLT::GAUSS2MF,
  FLT::GBELLMF, FLT::PIMF, FLT::PSIGMF, FLT::SMF,
  FLT::SIGMF, FLT::SIG2MF, FLT::TRAPMF, FLT::TRIMF,
  FLT::ZMF }

  Enumeration with the implemented Membership functions.

Functions

- int FLT::sign (double x)

  Implementation of the sign function.

- Membership * FLT::createMF (TYPE_MF t)

  Virtual constructor for Membership functions.

Variables

- static const char * FLT::MF_NAMES [ ]

  Names of the Membership functions.

- static const size_t FLT::MF_PARAM_NUMBER [ ]

  Initial number of parameters of each Membership function in FLT::TYPE_MF.

0.27.1 Detailed Description

Defines Membership functions.

See also

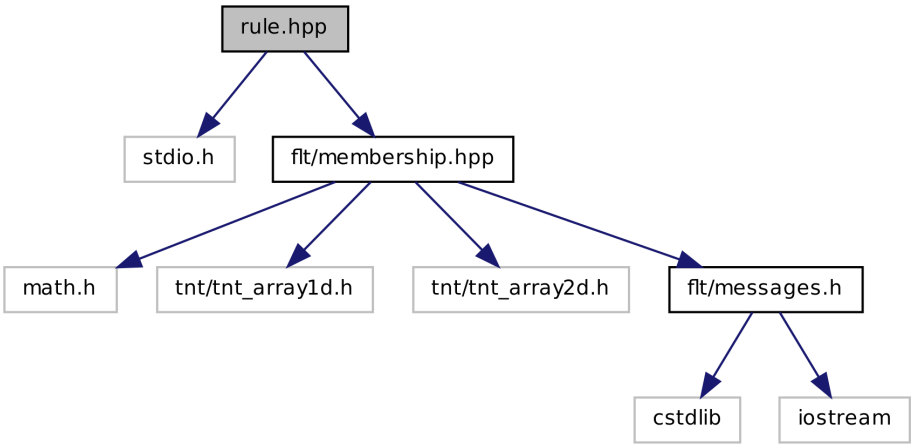
For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt

0.28 rule.hpp File Reference

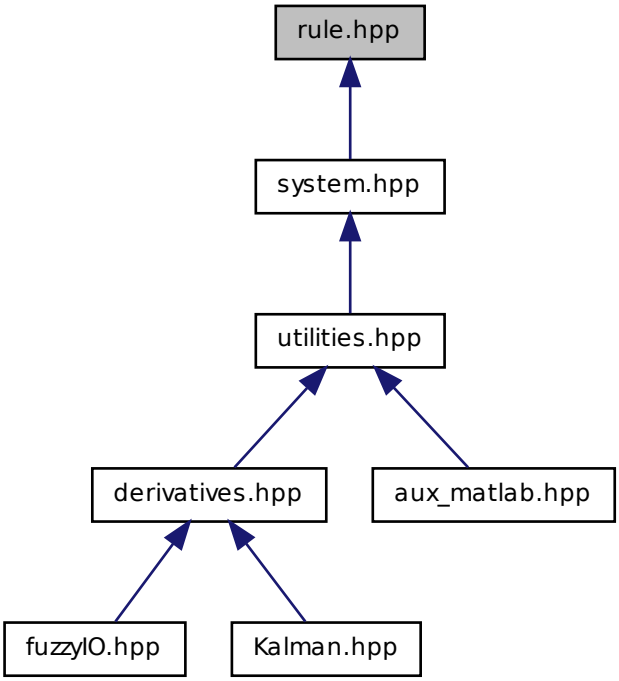
Defines a Takagi-Sugeno fuzzy Rule.

```
#include <stdio.h>
#include <flt/membership.hpp>
```

Include dependency graph for rule.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class FLT::Rule

  This class contains methods and attributes common to fuzzy Rules.

Namespaces

- namespace FLT

  Fuzzy Logic Tools (FLT) namespace.

0.28.1 Detailed Description

Defines a Takagi-Sugeno fuzzy Rule.

See also

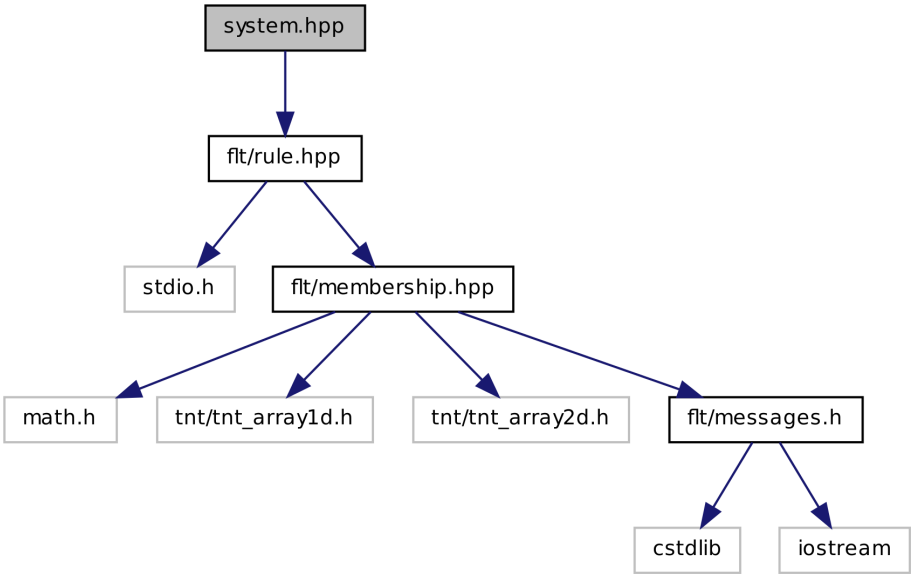
For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt

0.29 system.hpp File Reference

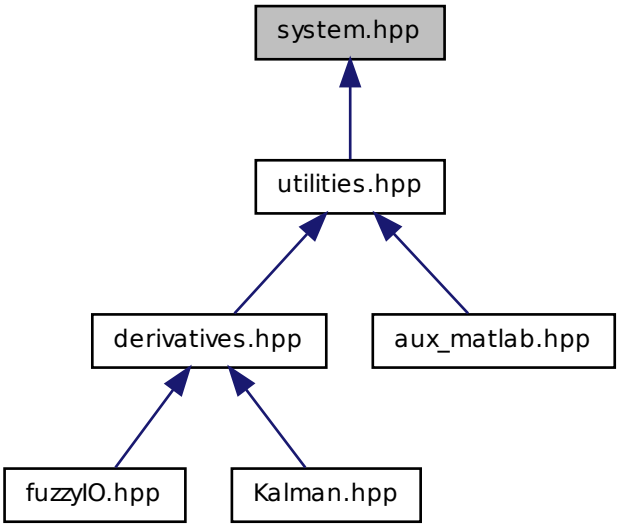
Defines a general Takagi-Sugeno fuzzy System.

```
#include <flt/rule.hpp>
```

Include dependency graph for system.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class FLT::System

  This class contains methods and attributes common to fuzzy Systems.

Namespaces

- namespace FLT

  Fuzzy Logic Tools (FLT) namespace.

Defines

- #define LOW_DEFAULT_LIMIT -100.0

Default minimum of the universe of discourse.

- #define HIGH_DEFAULT_LIMIT 100.0

Default maximum of the universe of discourse.

0.29.1 Detailed Description

Defines a general Takagi-Sugeno fuzzy System.

See also

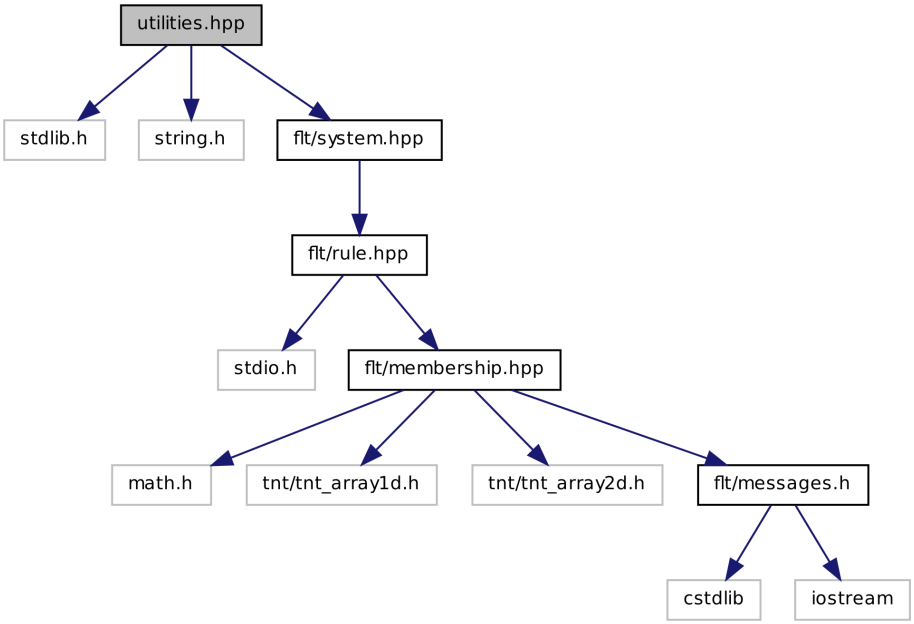
For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt

0.30 utilities.hpp File Reference

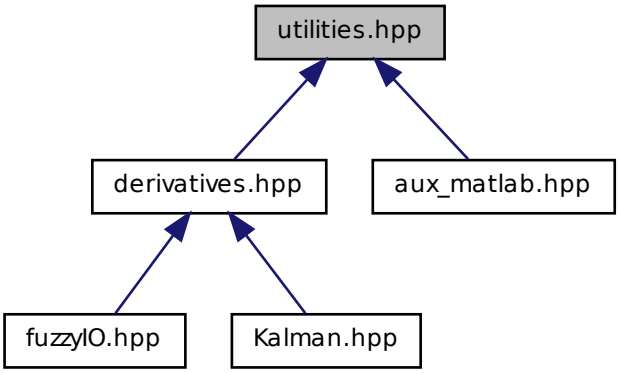
Implements useful functions for fuzzy systems.

```
#include <stdlib.h>
#include <string.h>
#include <flt/system.hpp>
```

Include dependency graph for utilities.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace FLT

  Fuzzy Logic Tools (FLT) namespace.

Defines

- #define **strcmpi** strcasecmp

Functions

- int FLT::System2TXT (System &S, const char *file)

  Saves the fuzzy System S in a text file.

- System FLT::TXT2System (const char *file)

  Reads a fuzzy System from a text file.

- int FLT::printSystem (const char *file, System &S, char *inputs[ ]=NULL, char *outputs[ ]=NULL, int accuracy=10)

Writes a fuzzy system in its linguistic form.

- TNT::Array1D< double > FLT::evaluate (const double *const point, System &S, System &C)

  Evaluates a closed loop fuzzy system in the given `point`.

- System FLT::initialController (System &Plant)

  Creates a fuzzy Controller from a given plant.

- System FLT::subSystem (System &S, size_t nrules, size_t *outputs, size_t *rules)

  Extracts a subsystem from a fuzzy System.

- TNT::Array2D< double > FLT::extractPoints (System &S, unsigned int numMeshPoints=5, double precision=1E-3, bool addMesh=false, bool onlyStatePoints=true)

  Extracts the representative points of a fuzzy system.

0.30.1 Detailed Description

Implements useful functions for fuzzy systems.

See also

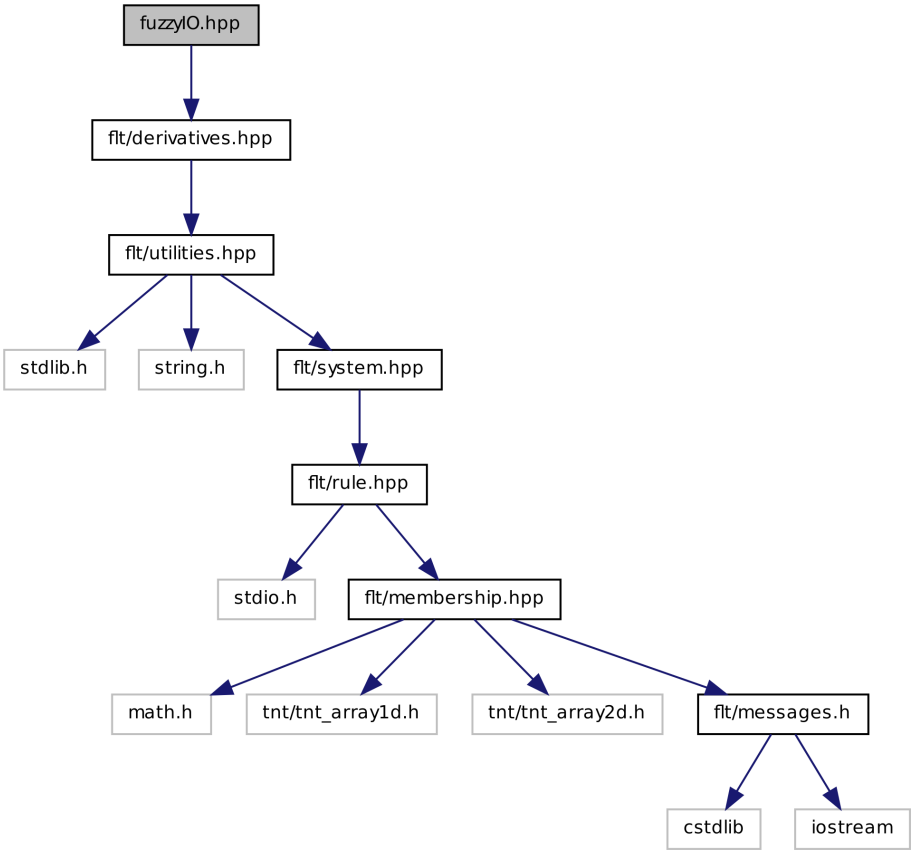
For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt

0.31 fuzzyIO.hpp File Reference

Defines Fuzzy Logic Tools streams operators.

```
#include <flt/derivatives.hpp>
```

Include dependency graph for fuzzyIO.hpp:



Functions

- ostream & std::operator<< (ostream &F, FLT::TYPE_MF &T)

  Ostream operator for TYPE_MF enumeration.

- istream & std::operator>> (istream &F, FLT::TYPE_MF &T)

  Istream operator for TYPE_MF enumeration.

- ostream & std::operator<< (ostream &F, FLT::Membership &P)

  Ostream operator for Membership class.

- istream & std::operator>> (istream &F, FLT::Membership &P)

  Istream operator for Membership class.

- ostream & std::operator<< (ostream &F, FLT::Rule &R)

  Ostream operator for Rule class.

- istream & std::operator>> (istream &F, FLT::Rule &R)

  Istream operator for Rule class.

- ostream & std::operator<< (ostream &F, FLT::System &S)

  Ostream operator for System class.

- istream & std::operator>> (istream &F, FLT::System &S)

  Istream operator for System class.

0.31.1 Detailed Description

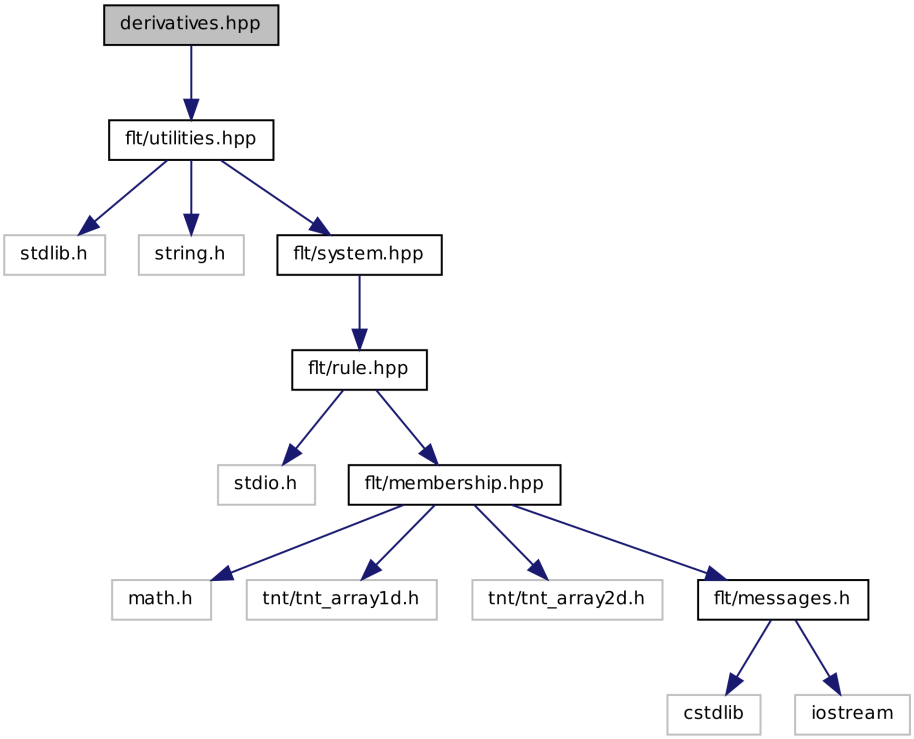
Defines Fuzzy Logic Tools streams operators.

0.32 derivatives.hpp File Reference

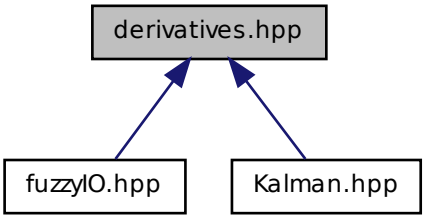
Calculates several derivatives for a fuzzy system.

```
#include <flt/utilities.hpp>
```

Include dependency graph for derivatives.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace FLT

  Fuzzy Logic Tools (FLT) namespace.

Functions

- TNT::Array2D< double > FLT::jacobian (System &S, const double *const x, const double *const u, TNT::Array2D< double > &B, TNT::Array1D< double > &F)

  Computes the open-loop jacobian matrix in `x` *for the Plant* `S`*.*

- TNT::Array2D< double > FLT::jacobian (System &S, System &C, const double *const x)

  Computes the closed-loop jacobian matrix in `x` *for the Plant* `S` *and the Controller* `C`*.*

- TNT::Array2D< double > FLT::jacobianAprox (System &S, System &C, const double *const x, double h=0.001)

  Computes the approximation of the closed-loop jacobian matrix in `x` *for the Plant* `S` *and the Controller* `C` *using finite differences with a step* `h`*.*

- TNT::Array2D< double > FLT::jacobianAprox (System &S, const double *const x, const double *const u, TNT::Array2D< double > &B, TNT::Array1D< double > &F, double h=0.001)

Computes the approximation of the open-loop jacobian matrix in `x` for the Plant `S` using finite differences with a step `h`.

- double FLT::derconseq (System &S, double *x, size_t output, size_t rule, size_t parameter, int &error)

  Obtains the derivative of a fuzzy model with respect to a consequent.

- TNT::Array2D< double > FLT::derconseq (System &S, double *x)

  Obtains the jacobian matrix of a fuzzy model with respect to its consequents.

- double FLT::derantec (System &S, double *x, size_t input, size_t output, size_t rule, size_t parameter, int &error)

  Obtains the derivative of a fuzzy model with respect to a parameter of an antecedent.

- TNT::Array2D< double > FLT::derantec (System &S, double *x)

  Obtains the jacobian matrix of a fuzzy model with respect to its antecedents.

- TNT::Array2D< double > FLT::derfuzzy (System &S, double *x)

  Obtains the jacobian matrix of a fuzzy model with respect to all of its parameters.

0.32.1 Detailed Description

Calculates several derivatives for a fuzzy system.

See also

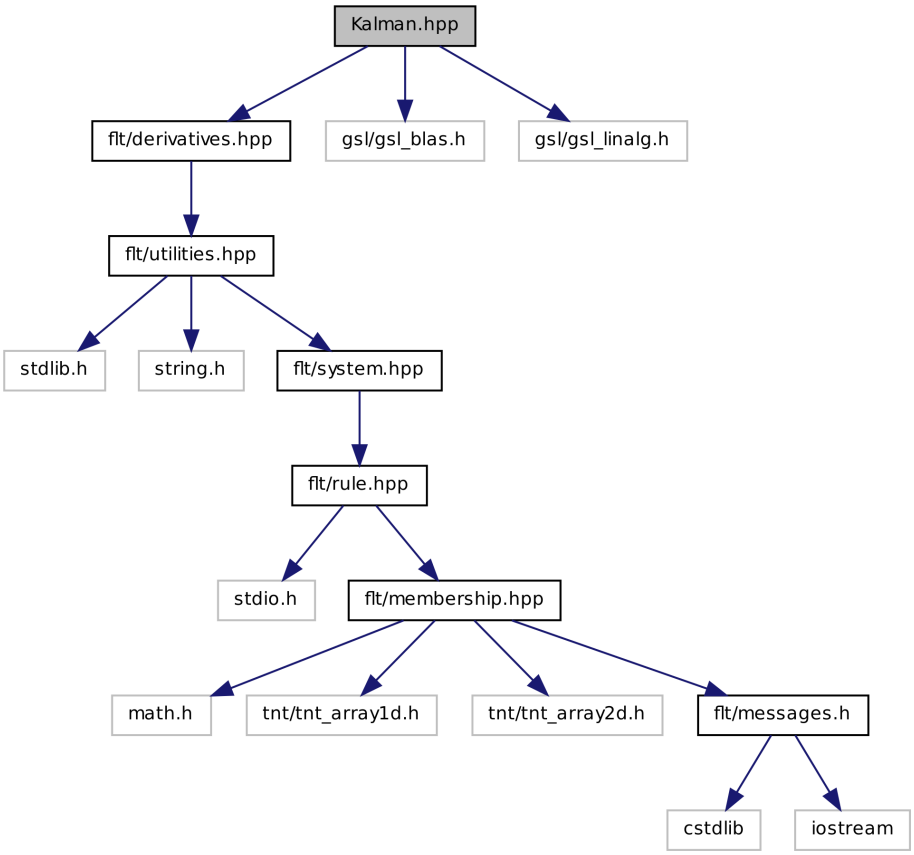
For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt

0.33 Kalman.hpp File Reference

Implements the adaptation of a fuzzy model by the extended Kalman filter.

```
#include <flt/derivatives.hpp>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_linalg.h>
```

Include dependency graph for Kalman.hpp:



Namespaces

- namespace FLT

  Fuzzy Logic Tools (FLT) namespace.

Functions

- TNT::Array1D< double > FLT::KalmanAntec (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P, TNT::Array2D< double > &Phi)

  Computes antecedets adjust by the discrete extended Kalman filter.

- TNT::Array1D< double > FLT::KalmanAntec (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P)

  Computes antecedets adjust by the discrete extended Kalman filter where Phi is assumed to be the identity matrix.

- TNT::Array1D< double > FLT::KalmanConseq (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P, TNT::Array2D< double > &Phi)

  Computes consequents adjust by the discrete extended Kalman filter.

- TNT::Array1D< double > FLT::KalmanConseq (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P)

  Computes consequents adjust by the discrete extended Kalman filter where Phi is assumed to be the identity matrix.

- TNT::Array1D< double > FLT::KalmanFuz (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P, TNT::Array2D< double > &Phi)

  Computes the simultaneous adjustment of antecedets and consequents by the discrete extended Kalman filter.

- TNT::Array1D< double > FLT::KalmanFuz (System &Model, TNT::Array1D< double > &input, TNT::Array1D< double > &output, TNT::Array2D< double > &covariance, TNT::Array2D< double > &P)

  Computes the simultaneous adjustment of antecedets and consequents by the discrete extended Kalman filter where Phi is assumed to be the identity matrix.

0.33.1 Detailed Description

Implements the adaptation of a fuzzy model by the extended Kalman filter.

See also

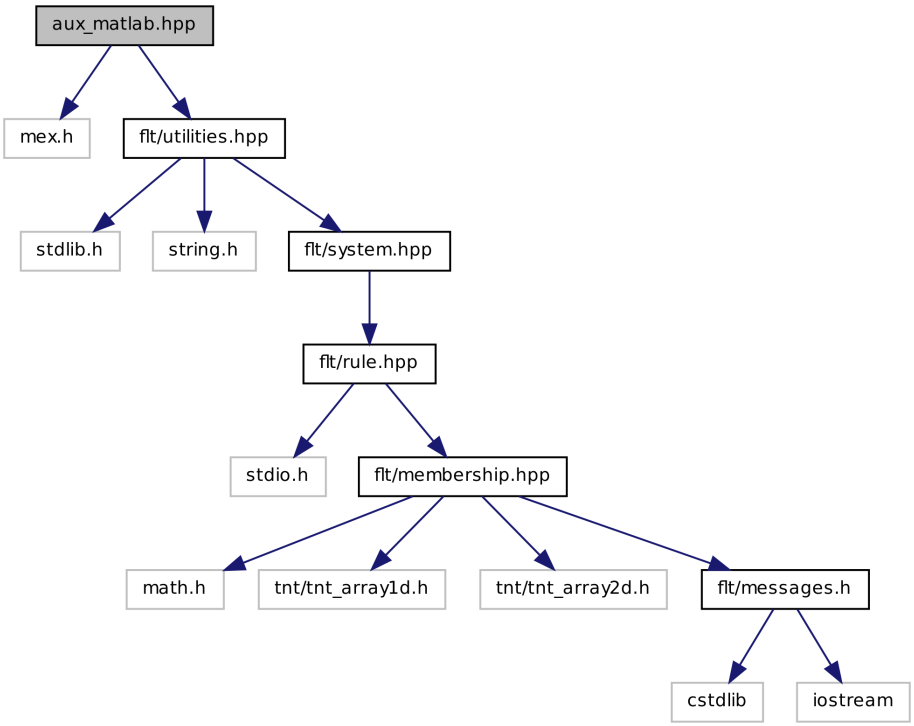
For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt
For GNU Scientific Library (GSL) documentation see http://gnu.org/software/gsl

0.34 aux_matlab.hpp File Reference

Defines some util functions to use with MATLAB© API.

```
#include "mex.h"
#include <flt/utilities.hpp>
```

Include dependency graph for aux_matlab.hpp:



Namespaces

- namespace FLT

  Fuzzy Logic Tools (FLT) namespace.

Functions

- void FLT::col2row_major (const double ∗const colM, double ∗rowM, size_t num_rows, size_t num_cols)

Converts a column-major matrix in a row-major one.

- TNT::Array2D< double > FLT::col2row_major (const double *const colM, size_t num_rows, size_t num_cols)

  Converts a column-major matrix in a row-major one.

- void FLT::row2col_major (const double *const rowM, double *colM, size_t num_rows, size_t num_cols)

  Converts a row-major matrix in a column-major one.

- int FLT::readModel (const mxArray *model, System &S)

  Reads a Fuzzy Model (TXT, FIS file or FIS variable).

- int FLT::FIS2System (const mxArray *FIS, System &S)

  Reads the Fuzzy Model from a FIS variable.

- int FLT::System2FIS (System &S, mxArray *FIS)

  Writes the Fuzzy Model in a FIS variable.

0.34.1 Detailed Description

Defines some util functions to use with MATLAB© API.

See also

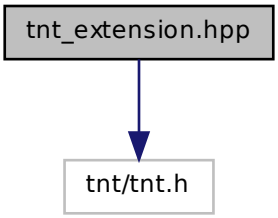
MATLAB© is a registered trademark of The MathWorks, Inc. http://www.mathworks.com

0.35 tnt_extension.hpp File Reference

Implements useful functions for the management of types defined in the TNT library.

```
#include <tnt/tnt.h>
```

Include dependency graph for tnt_extension.hpp:



Functions

- template<class T >
  Vector< T > TNT::Array1D2Vector (Array1D< T > A)

  Creates a TNT::Vector from a TNT::Array1D.

- template<class T >
  Array1D< T > TNT::Vector2Array1D (Vector< T > V)

  Creates a TNT::Array1D from a TNT::Vector.

- template<class T >
  Matrix< T > TNT::Array2D2Matrix (Array2D< T > A)

  Creates a TNT::Matrix from a TNT::Array2D.

- template<class T >
  Array2D< T > TNT::Matrix2Array2D (Matrix< T > M)

  Creates a TNT::Array2D from a TNT::Matrix.

- template<class T >
  Matrix< T > TNT::Array1D2Matrix (Array1D< T > A)

  Creates a TNT::Matrix from a TNT::Array1D.

- template<class T >
  Array1D< T > TNT::Matrix2Array1D (Matrix< T > M)

  Creates a TNT::Array1D from a TNT::Matrix.

- Array2D< double > TNT::makeIdentity (int order)

  Creates an Identity Array2D of the given `order`.

- template<class T >
  Array1D< T > TNT::copyrow (const Array2D< T > &Array, int r)

  Extracts the `r` *row from an Array2D.*

- template<class T >
  Array1D< T > TNT::copycolumn (const Array2D< T > &Array, int c)

  Extracts the `c` *column from an Array2D.*

- template<class T >
  Array2D< T > TNT::copyArray2D (const Array3D< T > &Array, int m)

  Extracts the `m` *Array2D from an Array3D.*

- template<class T >
  int TNT::injectrow (Array2D< T > &A2D, const Array1D< T > &A1D, int r)

  Injects a row into an Array2D in the position `r`.

- template<class T >
  int TNT::injectcolumn (Array2D< T > &A2D, const Array1D< T > &A1D, int c)

  Injects a column into an Array2D in the position `c`.

- template<class T >
  int TNT::injectArray2D (Array3D< T > &A3D, const Array2D< T > &A2D, int m)

  Injects an Array2D into an Array3D in the position `m`.

- template<class T >
  Matrix< T > TNT::mult_transpose (const Matrix< T > &A, const Matrix< T > &B)

  Matrix-Matrix tranpose multiplication, i.e. compute A∗tranpose(B).

0.35.1 Detailed Description

Implements useful functions for the management of types defined in the TNT library.

See also

For Template Numerical Toolkit (TNT) documentation see http://math.nist.gov/tnt

Example Documentation

0.36 example.cpp

Obtains the output of a closed loop fuzzy system and its jacobian matrix.

Compile: g++ example.cpp ../src/membership.cpp ../src/rule.cpp ../src/system.cpp ../src/utilities.cpp ../src/derivatives.cpp -o example

Use: example plant.txt controller.txt x1 x2 ... xn ('n' is the order of the plant)

Test: ./example ../matlab_utilities/Plant.txt ../matlab_utilities/Controller.txt 0 0

Output:

Closed loop output:

0.0409887

0.340054

Closed loop Jacobian matrix:

-2.58935 1.98356

2.30224 -7.94545

```
#include <stdio.h>
#include <tnt/tnt.h>
#include <flt/system.hpp>
#include <flt/derivatives.hpp>

using namespace std;
using namespace TNT;
using namespace FLT;

int main(int argc, char **argv)
{
```

```
  if (argc < 4)
  {
    cout << "Error. Some input aguments are missing.\n\n";
    cout << "example plant.txt controller.txt x1 x2 ... xn\n";
    cout << "\t where 'n' is the order of the plant (number of outputs)" << endl;

    return 1;
  }

  char *plant = argv[1];
  System P = TXT2System(plant);
  size_t n = P.outputs();
  if (!n)
  {
    cout << "Error reading the plant file." << endl;
    return 1;
  }

  if (argc != 3+n)
  {
    cout << "Error. Some input aguments are missing.\n\n";
    cout << "example plant.txt controller.txt x1 x2 ... xn\n";
    cout << "\t where 'n' is the order of the plant (number of outputs)" << endl;

    return 1;
  }

  char *controller = argv[2];
  System C = TXT2System(controller);
  size_t m = C.outputs();
  if (!m)
  {
    cout << "Error reading the controller file." << endl;
    return 1;
  }

  if (P.inputs() != n+m)
  {
    cout << E_NoCoherent << endl;
    return 1;
  }

  Array1D<double> X(n);
  for (size_t i=0; i<n; i++)
    X[i] = atof(argv[i+3]);

  Array1D<double> dX = evaluate(&X[0], P, C);

  cout << "\nClosed loop output:\n";
  for (size_t i=0; i<n; i++)
    cout << dX[i] << "\n";

  Array2D<double> J = jacobian(P, C, &X[0]);
  cout << "\nClosed loop Jacobian matrix:\n";
  for (size_t i=0; i<n; i++)
  {
    for (size_t j=0; j<n; j++)
      cout << J[i][j] << "  ";
    cout << "\n";
  }
  cout << endl;

  return 0;
}
```

0.37 matlab_utilities/activation.cpp

MEX file that calculates the fulfillment degree and the derivative of a rule.
MATLAB help:

```
% ACTIVATION Calculates the fulfillment degree and the derivative of a rule.
%
%   W = activation(Fuzzy_Model, Point, out)
%
%   [W, SumW] = activation(Fuzzy_Model, Point, out)
%
% For a specific rule:
%
%   W = activation(Fuzzy_Model, Point, out, rule)
%
%   [W, dW] = activation(Fuzzy_Model, Point, out, rule)
%
% Arguments:
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
%   out -> Output of the model for select the rule.
%
%   rule -> The rule for the output 'out'. If 'rule' is omitted will be
%             calculated the degree of activation of all rules of 'out'.
%
%   Point -> Point where the fulfillment degree will be calculate.
%
%   W -> Matching degree of the rule/s.
%
%   SumW -> Sumatory of all fulfillment degrees for the output 'out' => sum(W)
%
%   dW -> Derivative of the fulfillment degree. 'rule' must be used.
%   Note: dW will be correct only if all inputs of Fuzzy_Model are independent.
%
% See also antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt, fuz2mat,
%          fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i, output, rule, m;
  double *Point,*W,*dW, *SumW;

  if(nrhs<3 || nrhs>4)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>2)
    ERRORMSG(E_NumberArgOut)
  for (i=0;i<nrhs;i++)
  {
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
  }

  System S;
  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)

  if (1!=mxGetN(prhs[1]))
    ERRORMSG(E_Column)
  if (S.inputs()!=mxGetM(prhs[1]))
    ERRORMSG(E_PointCoherent)
  if ( (mxGetN(prhs[2])!=1) || (mxGetM(prhs[2])!=1))
    ERRORMSG(E_Out_No_Scalar)
  if (nrhs==4)
  {
    if ((mxGetN(prhs[3])!=1) || (mxGetM(prhs[3])!=1))
      ERRORMSG(E_R_No_Scalar)
    rule = ((size_t)*mxGetPr(prhs[3]))-1;
    if (rule>=S.rules(output))
      ERRORMSG(E_RulesExceded)
  }

  output = ((size_t)*mxGetPr(prhs[2]))-1;
  if (output>=S.outputs())
    ERRORMSG(E_OutputExceded)
  Point = mxGetPr(prhs[1]);

  if (nrhs==4)
  {// Only one rule
    plhs[0] = mxCreateDoubleScalar(1);
    W = mxGetPr(plhs[0]);
    if (!plhs[0])
      ERRORMSG(E_NumberArgOut)
    if (nlhs==1)
      (*W) = S.readRule(output,rule)->activation(Point);
    else
    {
      m = S.inputs();
      plhs[1] = mxCreateDoubleMatrix(1,m,mxREAL);
      dW = mxGetPr(plhs[1]);
      (*W) = S.readRule(output,rule)->activation(Point, dW);
    }
  }
  else
  {// All rules. dW can't be returned
```

```
    plhs[0] = mxCreateDoubleMatrix(1,S.rules(output),mxREAL);
    W = mxGetPr(plhs[0]);
    if (!plhs[0])
      ERRORMSG(E_NumberArgOut)
    if (nlhs==2)
    {
      plhs[1] = mxCreateDoubleScalar(1);
      if (!plhs[1])
        ERRORMSG(E_NumberArgOut)
      SumW = mxGetPr(plhs[1]);
      *SumW = 0.0;
    }
    for (rule=0;rule<S.rules(output);rule++)
    {
      (*W) = S.readRule(output,rule)->activation(Point);
      if (nlhs==2)
        (*SumW) += (*W);
      W++;
    }
  }
}
```

0.38 matlab_utilities/antec2mat.cpp

MEX file that extracts data from antecedent of a fuzzy model.

MATLAB help:

```
% ANTEC2MAT Extracts data from antecedent of a fuzzy model.
%
%   Vector = antec2mat(Fuzzy_Model)
%
%   Vector = antec2mat(Fuzzy_Model,Output,Rule)
%
%   [Vector, length] = antec2mat(Fuzzy_Model)
%
%   [Vector, length] = antec2mat(Fuzzy_Model,Output,Rule)
%
% Arguments:
%
%   Vector -> Vector with data from fuzzy model antecedent.
%
%   Output -> Use to select the antecedent of one output.
%
%   Rule -> Use to select the antecedent of one rule for the given output.
%
%   length -> The length of Vector, 0 if error.
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
% See also activation, aproxjac, aproxlinear, conseq2mat, fis2txt, fuz2mat,
%          fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan
```

```
  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i, length, out, rule;

  if(nrhs<1)
    ERRORMSG(E_NumberArgIn)
  if (nrhs>1 && nrhs!=3)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>2)
    ERRORMSG(E_NumberArgOut)
  if (mxIsEmpty(prhs[0]) || mxIsNaN(*mxGetPr(prhs[0])))
    ERRORMSG(E_ArgNoValid)
  if (nrhs==3)
  {
    if (mxGetM(prhs[1])>1 || mxGetM(prhs[2])>1)
      ERRORMSG(E_RuleOutputFileVector)
    if (mxGetN(prhs[1])!=mxGetN(prhs[2]))
      ERRORMSG(E_NumberArg)
  }

  System S;
  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)

  if (nrhs==1)
  {
    length = S.NumberOfAntecedents();
    if (length==0)
      ERRORMSG(E_BadModel)
  }
  else
  {
    length = 0;
    for (i=0;i<mxGetN(prhs[1]);i++)
    {
      out = ((size_t)*(mxGetPr(prhs[1])+i))-1; // MATLAB is index-1 but C/C++ is
      index-0
      rule = ((size_t)*(mxGetPr(prhs[2])+i))-1;
      length += S.readRule(out,rule)->NumberOfAntecedents();
```

```
    }
  }

  plhs[0] = mxCreateDoubleMatrix(length,1,mxREAL);
  Array1D<double> Data(length);

  if (nrhs==1)
    Data = S.getAntecedents();
  else
  {
    length = 0;
    for (i=0;i<mxGetN(prhs[1]);i++)
    {
      out = ((size_t)*(mxGetPr(prhs[1])+i))-1;
      rule = ((size_t)*(mxGetPr(prhs[2])+i))-1;
      Array1D<double> tempData = S.readRule(out,rule)->getAntecedents();

      for (size_t d=0;d<tempData.dim();d++)
        Data[length+d] = tempData[d];
      length += tempData.dim();
    }
  }

  double *MATLABData = mxGetPr(plhs[0]);
  for (i=0; i<length; i++)
    MATLABData[i] = Data[i];

  if (nlhs==2)
    plhs[1] = mxCreateDoubleScalar(length);

  if (length==0)
  {
    if (nlhs==1)
      ERRORMSG(E_Acquire)
    else
      plhs[0] = mxCreateDoubleMatrix(0,0,mxREAL); // Empty matrix
  }
}
```

0.39 matlab_utilities/aproxjac.cpp

MEX file that calculates the aproximation to the Jacobian matrix of the closed loop fuzzy system usign finite differences.

MATLAB help:

```
% APROXJAC Calculates the closed loop Jacobian matrix usign finite differences.
%
%   J = aproxjac(X, Plant, Controller)
%
%   J = aproxjac(X, Plant, Controller, h)
%
% Arguments:
%
%   X -> State vector.
%
%   Plant -> Fuzzy model of the Plant.
%
%   Controlador -> Fuzzy model of the Controller.
%
%   h -> Increment. Default value is 0.001;
%
```

```
%   J -> Jacobian matrix of the closed loop fuzzy system.
%
%                                   (  df1          dfn  )
%    dx1                            | -----   ...  ----- |
%   ----- = f1(x)                   |  dx1          dxn  |
%    dt                             |                    |
%    ...                      J =   |  ...    ...   ...  |
%    dxn                            |                    |
%   ----- = fn(x)                   |  dfn          dfn  |
%    dt                             | -----   ...  ----- |
%                                   (  dx1          dxn  )
%   x = [ x1 ; x2 ; ... ; xn]
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
% See also activation, antec2mat, aproxlinear, conseq2mat, fis2txt, fuz2mat,
%          fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"
#include <flt/derivatives.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i,q,n,model;
  double *X,*J,*d,h;
  System S[2];
  char Sist[MAX_FILE_NAME];
  char error[MAX_FILE_NAME];

  if(nrhs!=3 && nrhs!=4)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>1)
```

```
    ERRORMSG(E_NumberArgOut)
  for (i=0;i<nrhs;i++)
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)

  X = mxGetPr(prhs[0]);
  if (!X)
    ERRORMSG(E_NumberArgIn)

  n = mxGetM(prhs[0]);

  if (mxGetN(prhs[0])!=1)
    ERRORMSG(E_Column)

  for (model=1;model<3;model++)
  {
    if (readModel(prhs[model],S[model-1]))
    {
      sprintf(error,"%s %llu.\n",E_Model, model);
      ERRORMSG(error)
    }
  }

  if ((nrhs==3) && n!=S[0].outputs())
    ERRORMSG(E_PointCoherent)

  if ((nrhs==2) && n!=S[0].inputs())
    ERRORMSG(E_PointCoherent)

  Array2D<double> Jac(n,n);
  if (nrhs==4)
  {
    h = *mxGetPr(prhs[3]);
    if (h<=0)
      ERRORMSG(E_H_GE_0)
    Jac = jacobianAprox(S[0], S[1], X, h);
  }
  else
    Jac = jacobianAprox(S[0], S[1], X);

  if (!Jac.dim1())
    ERRORMSG(E_NoCoherent)

  plhs[0] = mxCreateDoubleMatrix(n,n,mxREAL);
  J = mxGetPr(plhs[0]);
  if (!J)
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_NumberArgOut)
  }

  for (i=0;i<n;i++)
  {
    for (q=0;q<n;q++)
    {
      *J = Jac[q][i];
      J++;
    }
  }
}
```

0.40 matlab_utilities/aproxlinear.cpp

MEX file that calculates the linealization of a Fuzzy Model in a point usign finite differences.

MATLAB help:

```
% APROXLINEAR Linealizes a Fuzzy Model usign finite differences.
%
%   A = aproxlinear(Fuzzy_Model, X, U)
%
%   A = aproxlinear(Fuzzy_Model, X, U, h)
%
%   [A,B] = aproxlinear(Fuzzy_Model, X, U, h)
%
%   [A,B,F] = aproxlinear(Fuzzy_Model, X, U, h)
%
% Solve the matrices the linear representation of the 'Fuzzy_Model' model
% usign finite differences in the point X with signal control U.
% F is the value of the function at the point (X,U).
%
%                           Y = F + Ax + Bu
%
% Arguments:
%
%   Fuzzy_Model -> Input Fuzzy model.
%
%   X -> State vector.
%
%   U -> Control vector.
%
%   h -> Increment. Default value is 0.001;
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
% See also activation, antec2mat, aproxjac, conseq2mat, fis2txt, fuz2mat,
%          fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.
```

```
  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"
#include <flt/derivatives.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i,q,j,n,m=0;
  size_t countA,countB;
  double *X,*U=NULL,*M_A,*M_B,*M_F,h;
  System S;
  char Sist[MAX_FILE_NAME];
  char error[MAX_FILE_NAME];

  if(nrhs<2 || nrhs>4)
    ERRORMSG(E_NumberArgIn)

  if (nlhs>3)
    ERRORMSG(E_NumberArgOut)
  for (i=0;i<nrhs;i++)
  {
    if ((i!=2 && mxIsEmpty(prhs[i])) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
  }

  X = mxGetPr(prhs[1]);
  if (!X)
    ERRORMSG(E_NumberArgIn)

  if (mxGetN(prhs[1])!=1)
    ERRORMSG(E_Column)
  n = mxGetM(prhs[1]);

  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)

  if (n!=S.outputs())
    ERRORMSG(E_PointCoherent)

  if (nrhs>=3)
  {
    U = mxGetPr(prhs[2]);
    if (!U)
      ERRORMSG(E_NumberArgIn)
    if (mxGetN(prhs[2])!=1)
      ERRORMSG(E_Column)
    m = mxGetM(prhs[2]);
    if (m!=(S.inputs()-n))
      ERRORMSG(E_PointCoherent)
  }
  else if(m!=0)
  {
    U = new double[m];
    for (i=0;i<m;i++)
      U[i]=0;
  }
  else
    U = NULL;

  plhs[0] = mxCreateDoubleMatrix(n,n,mxREAL);
  M_A = mxGetPr(plhs[0]);
```

```
if (!plhs[0] || !M_A)
{
  if (nrhs==2 && m!=0)
    delete []U;
  mxDestroyArray(plhs[0]);
  ERRORMSG(E_NumberArgOut)
}
if (nlhs>1)
{
  plhs[1] = mxCreateDoubleMatrix(n,m,mxREAL);
  M_B = mxGetPr(plhs[1]);
  if (!plhs[1] || !M_B)
  {
    if (nrhs==2 && m!=0)
      delete []U;
    mxDestroyArray(plhs[0]);
    mxDestroyArray(plhs[1]);
    ERRORMSG(E_NumberArgOut)
  }
}
if (nlhs==3)
{
  plhs[2] = mxCreateDoubleMatrix(n,1,mxREAL);
  M_F = mxGetPr(plhs[2]);
  if (!plhs[2] || !M_F)
  {
    if (nrhs==2 && m!=0)
      delete []U;
    mxDestroyArray(plhs[0]);
    mxDestroyArray(plhs[1]);
    mxDestroyArray(plhs[2]);
    ERRORMSG(E_NumberArgOut)
  }
}

Array2D<double> A(n,n), B(n,m);
Array1D<double> F(n);
if (nrhs==4)
{
  h=mxGetScalar(prhs[3]);
  if (h<=0)
    ERRORMSG(E_H_GE_0)
  A = jacobianAprox (S, X, U, B, F, h);
}
else
  A = jacobianAprox (S, X, U, B, F);

if(!A.dim1())
{
  if (nrhs==2 && m!=0)
    delete []U;
  mxDestroyArray(plhs[0]);
  if (nlhs>1)
    mxDestroyArray(plhs[1]);
  if (nlhs==3)
    mxDestroyArray(plhs[2]);
  ERRORMSG(U_Overflow)
}

for (q=0,countA=0;q<n;q++)
{
  for (i=0;i<n;i++,countA++)
    *(M_A + countA) = A[i][q];
}

if (nlhs>1)
{
```

```
    for (j=0,countB=0;j<m;j++)
      for (i=0;i<n;i++,countB++)
        *(M_B + countB) = B[i][j];
  }

  if (nlhs==3)
    for (q=0;q<n;q++)
      M_F[q] = F[q];
}
```

0.41 matlab_utilities/conseq2mat.cpp

MEX file that extracts data from consequent of a fuzzy model.

MATLAB help:

```
% CONSEQ2MAT Extracts data from consequent of a fuzzy model.
%
%   Vector = conseq2mat(Fuzzy_Model)
%
%   Vector = conseq2mat(Fuzzy_Model,Output,Rule)
%
%   [Vector, length] = conseq2mat(Fuzzy_Model)
%
%   [Vector, length] = conseq2mat(Fuzzy_Model,Output,Rule)
%
% Arguments:
%
%   Vector -> Vector with data from fuzzy model consequent.
%
%   Output -> Use to select the consequent of one output.
%
%   Rule -> Use to select the consequent of one rule for given output.
%
%   length -> The length of Vector, 0 if error.
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
% See also activation, antec2mat, aproxjac, aproxlinear, fis2txt, fuz2mat,
%          fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
```

```
   (at your option) any later version.

   This software is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i, NoC, length, out, rule;
  double *data;

  if(nrhs<1)
    ERRORMSG(E_NumberArgIn)
  if (nrhs>1 && nrhs!=3)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>2)
    ERRORMSG(E_NumberArgOut)
  if (mxIsEmpty(prhs[0]) || mxIsNaN(*mxGetPr(prhs[0])))
    ERRORMSG(E_ArgNoValid)
  if (nrhs==3)
  {
    if (mxGetM(prhs[1])>1 || mxGetM(prhs[2])>1)
      ERRORMSG(E_RuleOutputFileVector)
    if (mxGetN(prhs[1])!=mxGetN(prhs[2]))
      ERRORMSG(E_NumberArg)
  }

  System S;
  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)
  size_t m = S.inputs();
  if (nrhs==1)
  {
    length = S.NumberOfConsequents();
    if (length==0)
      ERRORMSG(E_BadModel)
  }
  else
    length = mxGetN(prhs[1])*(m+1);

  plhs[0] = mxCreateDoubleMatrix(length,1,mxREAL);
  Array1D<double> Data(length);

  if (nrhs==1)
    Data = S.getConsequents();
  else
  {
    length = 0;
    for (i=0;i<mxGetN(prhs[1]);i++)
    {
      out = ((size_t)*(mxGetPr(prhs[1])+i))-1;
      rule = ((size_t)*(mxGetPr(prhs[2])+i))-1;
      Array1D<double> tempData = S.readRule(out,rule)->getConsequents();
      for (size_t d=0;d<tempData.dim();d++)
        Data[length+d] = tempData[d];
      length += tempData.dim();
    }
```

```
  }

  double *MATLABData = mxGetPr(plhs[0]);
  for (i=0; i<length; i++)
    MATLABData[i] = Data[i];

  if (nlhs==2)
    plhs[1] = mxCreateDoubleScalar(length);

  if (length==0)
  {
    if (nlhs==1)
      ERRORMSG(E_Acquire)
    else
      plhs[0] = mxCreateDoubleMatrix(0,0,mxREAL); // Empty matrix
  }
}
```

0.42 matlab_utilities/extractPoints.cpp

MEX file that extracts the significative points of a fuzzy model.

MATLAB help:

```
% EXTRACTPOINTS Extracts the significative points of a fuzzy model.
%
%   Points = extractPoints(System)
%
%   Points = extractPoints(System, numMeshPoints)
%
%   Points = extractPoints(System, numMeshPoints, precision)
%
%   Points = extractPoints(System, numMeshPoints, precision, controlPoints)
%
% Arguments:
%
%   System -> This fuzzy model could be a '.txt' file, a '.fis' file,
%             or a 'FIS' variable from MATLAB Workspace.
%
%   numMeshPoints -> Used in memberships functions without significative
%                    points (ANYMF, CONSTMF, ...). Default value is 5.
%
%   precision -> Used to remove similar points. Default value is 1e-3.
%
%   controlPoints -> If controlPoints>0 extract all point, otherwise, only
%               extracts the state vector points, not the control vector points.
%               Default value is 0 (do not extract the control vector points).
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat,fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint,
%          initializeController, mat2antec, mat2conseq, mat2fuz, subsystem
%          txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
```

```
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  char file[MAX_CHAR_LONG];
  size_t i,j;
  int error = 0;
  size_t numMeshPoints = 5;
  double precision = 1E-3;
  bool onlyStatePoints = true;
  double *d;

  if(nrhs<1 || nrhs>4)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>1)
    ERRORMSG(E_NumberArgOut)
  if (mxIsEmpty(prhs[0]) || mxIsNaN(*mxGetPr(prhs[0])))
    ERRORMSG(E_ArgNoValid)

  System S;
  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)

  if (nrhs>1)
  {
    d = mxGetPr(prhs[1]);
    if (!d)
      ERRORMSG(E_NumberArgIn)

    numMeshPoints = (size_t)(*d);
    if (numMeshPoints<2)
      ERRORMSG(E_N_MESH_P)
    if (nrhs>2)
    {
      d = mxGetPr(prhs[2]);
      if (!d)
        ERRORMSG(E_NumberArgIn)
      precision = *d;
      if (precision<=0.0)
        ERRORMSG(E_PRECISION)
      if (nrhs==4)
      {
        d=mxGetPr(prhs[3]);
        if (!d)
```

```
          ERRORMSG(E_NumberArgIn)
        onlyStatePoints = (*d<=0.0);
      }
    }
  }

  TNT::Array2D<double> aPoints = extractPoints(S,numMeshPoints,precision,onlyStat
      ePoints);
  plhs[0] = mxCreateDoubleMatrix(aPoints.dim1(),aPoints.dim2(),mxREAL);
  double *Points = mxGetPr(plhs[0]);
  if (!Points)
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_NumberArgOut)
  }
  for (j=0;j<aPoints.dim2();j++)
  {
    for (i=0;i<aPoints.dim1();i++)
    {
      *Points = aPoints[i][j];
      Points++;
    }
  }
}
```

0.43 matlab_utilities/fis2txt.cpp

MEX file that writes the fuzzy model in an ASCII text file.

MATLAB help:

```
% FIS2TXT Writes the fuzzy model in an ASCII text file.
%
%   fis2txt(Model,'File.txt')
%
% Arguments:
%
%   Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%            or a 'FIS' variable from MATLAB Workspace.
%
%   'File.txt' -> String with the name of file to write. This file will be
%                 overwrited without confirmation.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fuz2mat,
%          fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.
```

```
  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  char Fichero[MAX_CHAR_LONG];
  char error[MAX_CHAR_LONG];
  if(nrhs!=2)
  {
    sprintf(error,"%s %s, %s",E_NumberArg,O_OfIn,U_SeeHelp);
    ERRORMSG(error)
  }
  if (nlhs>0)
  {
    sprintf(error,"%s %s, %s",E_NumberArg,O_OfOut,U_SeeHelp);
    ERRORMSG(error)
  }
  for (int i=0;i<nrhs;i++)
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
  if(mxGetString(prhs[1],Fichero,MAX_CHAR_LONG))
    ERRORMSG(E_FileInput)

  System S;
  if(readModel(prhs[0],S))
    ERRORMSG(E_Model)

  if(System2TXT(S,Fichero))
    mexPrintf("Error using ==> fis2txt\n%s\n%s %s\n",E_FileOutput,U_OR,E_NoValidF
      P);
}
```

0.44 matlab_utilities/fuz2mat.cpp

MEX file that extracts all data from a fuzzy model.

MATLAB help:

```
% FUZ2MAT Extracts all data from a fuzzy model.
%
%   Vector = fuz2mat(Fuzzy_Model)
%
%   Vector = fuz2mat(Fuzzy_Model,Output,Rule)
%
%   [Vector, length] = fuz2mat(Fuzzy_Model)
%
```

```
%   [Vector, length] = fuz2mat(Fuzzy_Model,Output,Rule)
%
% Arguments:
%
%   Vector -> Vector with data from fuzzy model.
%
%   Output -> Use to select the output.
%
%   Rule -> Use to select the rule for given output.
%
%   length -> The length of Vector, 0 if error.
%
%   Fuzzy_Model -> This model could be a '.txt' file, a '.fis' file or a
%                  'FIS' variable from MATLAB Workspace.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i, length, out, rule;
  double *data;

  if(nrhs<1)
    ERRORMSG(E_NumberArgIn)
  if (nrhs>1)
  {
    if (nrhs!=3)
      ERRORMSG(E_NumberArgIn)
    if (mxGetM(prhs[1])>1 || mxGetM(prhs[2])>1)
      ERRORMSG(E_RuleOutputFileVector)
```

```
  if (mxGetN(prhs[1])!=mxGetN(prhs[2]))
    ERRORMSG(E_NumberArg)
}
if (nlhs>2)
  ERRORMSG(E_NumberArgOut)
if (mxIsEmpty(prhs[0]) || mxIsNaN(*mxGetPr(prhs[0])))
  ERRORMSG(E_ArgNoValid)

System S;
if (readModel(prhs[0],S))
  ERRORMSG(E_Model)

size_t NoC = S.inputs() + 1;
if (nrhs==1)
{
  length = S.NumberOfAntecedents() + S.NumberOfConsequents();
}
else
{
  length = 0;
  for (i=0;i<mxGetN(prhs[2]);i++)
  {
    out = ((size_t)*(mxGetPr(prhs[1])))-1; // MATLAB is index-1 but C/C++ is in
    dex-0
    rule = ((size_t)*(mxGetPr(prhs[2])))-1;
    size_t numAntec = S.readRule(out, rule)->NumberOfAntecedents();
    length += numAntec + NoC;
  }
}

Array1D<double> Data(length);

if (nrhs==1)
{
  Array1D<double> AntecData = S.getAntecedents();
  Array1D<double> ConseqData = S.getConsequents();
  for (size_t d=0;d<AntecData.dim();d++)
    Data[d] = AntecData[d];
  for (size_t d=0;d<ConseqData.dim();d++)
    Data[AntecData.dim()+d] = ConseqData[d];
}
else
{
  size_t dataIdx = 0;
  for (i=0;i<mxGetN(prhs[2]);i++)
  {
    out = ((size_t)*(mxGetPr(prhs[1])))-1; // MATLAB is index-1 but C/C++ is in
    dex-0
    rule = ((size_t)*(mxGetPr(prhs[2])))-1;
    Rule *R = S.readRule(out,rule);

    Array1D<double> AntecData = R->getAntecedents();
    for (size_t d=0;d<AntecData.dim();d++, dataIdx++)
      Data[dataIdx] = AntecData[d];

    Array1D<double> ConseqData = R->getConsequents();
    for (size_t d=0;d<ConseqData.dim();d++, dataIdx++)
      Data[dataIdx] = ConseqData[d];
  }
}

plhs[0] = mxCreateDoubleMatrix(length,1,mxREAL);
double *MATLABData = mxGetPr(plhs[0]);
for (i=0; i<length; i++)
  MATLABData[i] = Data[i];

if (nlhs==2)
```

```
    plhs[1] = mxCreateDoubleScalar(length);

  if (length==0)
  {
    if (nlhs==1)
      ERRORMSG(E_Acquire)
    else
      plhs[0] = mxCreateDoubleMatrix(0,0,mxREAL); // Empty matrix
  }
}
```

0.45 matlab_utilities/fuzcomb.cpp

MEX file that combines some fuzzy models in one.

MATLAB help:

```
% FUZCOMB Combines some fuzzy models in one.
%
%   fuzcomb(Combined_Model_TXT,Model1, Model2,...)
%
%   Combined_Model_FIS = fuzcomb(Model1, Model2,...)
%
% Arguments:
%
%   Combined_Model_TXT -> String with the name of TXT file to write with the
%                         combined model. This file will be overwrited without
%                         confirmation.
%
%   Combined_Model_FIS -> Variable 'FIS' to write the combined model.
%
%   ModelX -> Fuzzy models to combine. These models must have the same number
%             of inputs. They could be a '.txt' or '.fis' file, or a 'FIS'
%             variable from MATLAB Workspace.
%
% To simulate this combined model should use 'fuzeval' function, 'evalfis'
% doesn't work correctly.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.
```

```
  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i,j,r,s;
  char Output[MAX_FILE_NAME];
  char error[MAX_FILE_NAME];
  size_t sysnumber,in,outputs=0,*rules;

  if (nlhs>1)
  {
    sprintf(error,"%s %s, %s",E_NumberArg,O_OfOut,U_SeeHelp);
    ERRORMSG(error)
  }
  if ((nlhs==0 && nrhs<3) || (nlhs==1 && nrhs<2))
  {
    sprintf(error,"%s, %s",E_NumberArg,U_SeeHelp);
    ERRORMSG(error)
  }
  int operationtype = 1 - nlhs; // 0 if output = FIS, 1 if output = TXT
  sysnumber = nrhs - operationtype;
  for (size_t i=0;i<nrhs;i++)
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)

  // Read models
  System *S = new System[sysnumber];
  Rule *R, *Rall;
  for (i=operationtype;i<nrhs;i++)
  {
    if(readModel(prhs[i],S[i-operationtype]))
    {
      mexPrintf("%s %d.\n",U_CheckModel,1+i-operationtype);
      delete [] S;
      ERRORMSG(E_Model)
    }

    // Check if the models are compatible ( = number of inputs)
    if (i==operationtype)
    {
      in = S[0].inputs();
    }
    else if (in!=S[i-operationtype].inputs())
    {
      delete [] S;
      ERRORMSG(E_SameIns)
    }

    outputs += S[i-operationtype].outputs();
  }

  // Combine the models
  rules = new size_t[outputs];
  j = 0;
  for (s=0;s<sysnumber;s++)
```

```
  for (i=0;i<S[s].outputs();i++,j++)
    rules[j] = S[s].rules(i);

System *Sall = new System(in,outputs,rules);
outputs = 0;
for (s=0;s<sysnumber;s++)
{
  for (j=0;j<S[s].outputs();j++,outputs++)
  {

    for (r=0;r<S[s].rules(j);r++)
    {
      Rall = Sall->readRule(outputs,r);
      R = S[s].readRule(j,r);
      for (i=0;i<in;i++)
      {
        Rall->changeFunction(R->readFunction(i)->type(),i);
        (*Rall->readFunction(i)) = (*R->readFunction(i));
        Rall->changeTSK(R->readTSK(i),i);
      }
      Rall->changeTSK(R->readTSK(in),in);
    }
  }
}

double *limitsmin = new double[S[0].inputs()];
double *limitsmax = new double[S[0].inputs()];
for (i=0;i<S[0].inputs();i++)
{
  limitsmin[i] = S[0].in_low(i);
  limitsmax[i] = S[0].in_high(i);
}

Sall->in_low(limitsmin);
Sall->in_high(limitsmax);
delete [] limitsmin;
delete [] limitsmax;
limitsmin = new double[outputs];
limitsmax = new double[outputs];
outputs=0;
for (s=0;s<sysnumber;s++)
{
  for (j=0;j<S[s].outputs();j++,outputs++)
  {
    limitsmin[outputs] = S[s].out_low(j);
    limitsmax[outputs] = S[s].out_high(j);
  }
}

Sall->out_low(limitsmin);
Sall->out_high(limitsmax);
delete [] limitsmin;
delete [] limitsmax;

// Ending
if (nlhs==0)
{
  if(mxGetString(prhs[0],Output,MAX_FILE_NAME))
  {
    delete [] S;
    delete Sall;
    delete [] rules;
    ERRORMSG(E_FileOutput)
  }
  if(System2TXT(*Sall,Output))
  {
    delete [] S;
```

```
      delete Sall;
      delete [] rules;
      mexPrintf("??? Error using ==> fuzcomb\n%s\n%s %s\n",E_FileOutput,U_OR,E_No
      ValidFP);
    }
  }
  else
  {
    const char* names[]={"name","type","andMethod","orMethod","defuzzMethod","imp
      Method","aggMethod","input","output","rule"};
    int dim[]={1,1};
    plhs[0]=mxCreateStructArray(2,dim,10,names);
    if(System2FIS(*Sall, plhs[0]))
    {
      mxDestroyArray(plhs[0]);
      delete [] S;
      delete Sall;
      delete [] rules;
      ERRORMSG(E_CreateFIS)
    }
  }
  delete [] S;
  delete Sall;
  delete [] rules;
}
```

0.46 matlab_utilities/fuzderantec.cpp

MEX file that gets the derivative of a fuzzy model with respect to its antecedents.

MATLAB help:

```
% FUZDERANTEC Gets the derivatives of a fuzzy model respect its antecedents.
%
%   It is assumed that the output of the system is the same as the parameter,
%   otherwise it should be considered derivative equal to 0.
%
%   dh_dantec = fuzderconseq(Fuzzy_Model, Point)
%
%   dh_dantec = fuzderconseq(Fuzzy_Model, Point, input, out, rule, parameter)
%
% Arguments:
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
%   Point -> Point where the derivative will be calculate.
%
%   input -> Input for the parameter.
%
%   out -> Output for select the rule for the parameter.
%
%   rule -> Rule for the output 'out' for the parameter.
%
%   parameter -> Parameter of the antecedent.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzderconseq, fuzderparam, fuzeval, fuzjac,
%          fuzlinear, fuzprint, mat2antec, mat2conseq, mat2fuz, subsystem,
%          txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es
  MARIANO J. AZNAR, marianojose.aznar@alu.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"
#include <flt/derivatives.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t input, output, rule, param, m;
  int error=0;
  double *Point, *dModel_dantec;

  if(nrhs!=2 && nrhs!=6)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>1)
    ERRORMSG(E_NumberArgOut)
  for (int i=0;i<nrhs;i++)
  {
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
  }
  System S;
  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)

  if (1 != mxGetN(prhs[1]))
    ERRORMSG(E_Column)
  if (S.inputs() != mxGetM(prhs[1]))
    ERRORMSG(E_PointCoherent)
  Point = mxGetPr(prhs[1]);

  if (nrhs==6)
  {
    if ((mxGetN(prhs[2])!=1) || (mxGetM(prhs[2])!=1))
      ERRORMSG(E_In_No_Scalar)
    if ((mxGetN(prhs[3])!=1) || (mxGetM(prhs[3])!=1))
      ERRORMSG(E_Out_No_Scalar)
    if ((mxGetN(prhs[4])!=1) || (mxGetM(prhs[4])!=1))
      ERRORMSG(E_R_No_Scalar)
    if ((mxGetN(prhs[5])!=1) || (mxGetM(prhs[4])!=1))
```

```
      ERRORMSG(E_Param_No_Scalar)

    input = ((size_t)*mxGetPr(prhs[2]))-1;
    if (input >= S.inputs())
      ERRORMSG(E_InputExceded)

    output = ((size_t)*mxGetPr(prhs[3]))-1;
    if (output >= S.outputs())
      ERRORMSG(E_OutputExceded)

    rule = ((size_t)*mxGetPr(prhs[4]))-1;
    if (rule >= S.rules(output))
      ERRORMSG(E_RulesExceded)

    param = ((size_t)*mxGetPr(prhs[5]))-1;
    Membership *M = S.readRule(output, rule)->readFunction(input);
    if (M->type() == ANYMF) // ANYMF doesn't have parameters
    {
      plhs[0]=mxCreateDoubleMatrix(0,0,mxREAL);
      if (!plhs[0])
        ERRORMSG(E_NumberArgOut)
      return;
    }
    if (param >= M->num_params())
      ERRORMSG(E_ParamExceded)

    plhs[0] = mxCreateDoubleScalar(1);
    if (!plhs[0])
      ERRORMSG(E_NumberArgOut)
    dModel_dantec = mxGetPr(plhs[0]);

    (*dModel_dantec) = derantec(S, Point, input, output, rule, param, error);
    if (error)
    {
      mxDestroyArray(plhs[0]);
      ERRORMSG(O_GeneralError)
    }
  }
  else // nrhs=2
  {
    m = S.outputs();
    plhs[0] = mxCreateDoubleMatrix(m,S.NumberOfAntecedents(),mxREAL);
    if (!plhs[0])
      ERRORMSG(E_NumberArgOut)
    dModel_dantec=mxGetPr(plhs[0]);
    TNT::Array2D<double> dS_dantec = derantec(S, Point);
    if (dS_dantec.dim1()==0 || dS_dantec.dim2()==0 )
    {
      mxDestroyArray(plhs[0]);
      ERRORMSG(O_GeneralError)
    }
    for (size_t j=0;j<dS_dantec.dim2();j++)
    {
      for (size_t i=0;i<dS_dantec.dim1();i++)
      {
        *dModel_dantec = dS_dantec[i][j];
        dModel_dantec++;
      }
    }
  }
}
```

0.47 matlab_utilities/fuzderconseq.cpp

MEX file that gets the derivative of a fuzzy model with respect to its consequents.
MATLAB help:

```
% FUZDERCONSEQ Gets the derivative of a fuzzy model respect its consequents.
%
%   It is assumed that the output of the system is the same as the parameter,
%   otherwise it should be considered derivative equal to 0.
%
%   dh_dconseq = fuzderconseq(Fuzzy_Model, Point)
%
%   dh_dconseq = fuzderconseq(Fuzzy_Model, Point, out, rule, parameter)
%
% Arguments:
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
%   Point -> Point where the derivative will be calculate.
%
%   out -> Output for select the rule for the parameter.
%
%   rule -> Rule for the output 'out' for the parameter.
%
%   parameter -> Parameter of the consequent (0 -> affine consequent)
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzderantec, fuzderparam, fuzeval, fuzjac, fuzlinear
    ,
%          fuzprint, mat2antec, mat2conseq, mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es
  MARIANO J. AZNAR, marianojose.aznar@alu.uhu.es
  AGUSTÍN JIMÉNEZ AVELLO, agustin.jimenez@upm.es
  BASIL M. AL-HADITHI, basil.alhadithi@upm.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
```

```
#include "aux_matlab.hpp"
#include <flt/derivatives.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t output, rule, parameter;
  int error=0;
  double *Point, *dModel_dconseq;

  if(nrhs!=2 && nrhs!=5)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>1)
    ERRORMSG(E_NumberArgOut)

  for (int i=0;i<nrhs;i++)
  {
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
  }

  System S;
  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)

  size_t n = S.inputs();
  size_t m = S.outputs();

  if (mxGetN(prhs[1])!=1)
    ERRORMSG(E_Column)

  if (n != mxGetM(prhs[1]))
    ERRORMSG(E_PointCoherent)

  Point = mxGetPr(prhs[1]);

  if (nrhs==5)
  {
    if ( (mxGetN(prhs[2])!=1) || (mxGetM(prhs[2])!=1))
      ERRORMSG(E_Out_No_Scalar)
    if ((mxGetN(prhs[3])!=1) || (mxGetM(prhs[3])!=1))
      ERRORMSG(E_R_No_Scalar)
    if ( (mxGetN(prhs[4])!=1) || (mxGetM(prhs[4])!=1))
      ERRORMSG(E_Param_No_Scalar)

    output = ((size_t)*mxGetPr(prhs[2]))-1;
    if (output >= m)
      ERRORMSG(E_OutputExceded)

    rule = ((size_t)*mxGetPr(prhs[3]))-1;
    if (rule >= S.rules(output))
      ERRORMSG(E_RulesExceded)

    parameter = ((size_t)*mxGetPr(prhs[4]));
    if (parameter > S.inputs())
      ERRORMSG(E_InputExceded)

    plhs[0] = mxCreateDoubleScalar(1);
    if (!plhs[0])
      ERRORMSG(E_NumberArgOut)

    dModel_dconseq = mxGetPr(plhs[0]);
    (*dModel_dconseq) = derconseq(S, Point, output, rule, parameter, error);
    if (error)
```

```
    {
      mxDestroyArray(plhs[0]);
      ERRORMSG(O_GeneralError)
    }
  }
  else // nrhs==2
  {
    TNT::Array2D<double> dS_dconseq = derconseq(S, Point);
    if (dS_dconseq.dim1()==0 || dS_dconseq.dim2()==0 )
    {
      mxDestroyArray(plhs[0]);
      ERRORMSG(O_GeneralError)
    }

    plhs[0] = mxCreateDoubleMatrix(dS_dconseq.dim1(), dS_dconseq.dim2(),mxREAL);
    if (!plhs[0])
      ERRORMSG(E_NumberArgOut)
    dModel_dconseq = mxGetPr(plhs[0]);

    for (size_t j=0;j<dS_dconseq.dim2();j++)
    {
      for (size_t i=0;i<dS_dconseq.dim1();i++)
      {
        *dModel_dconseq = dS_dconseq[i][j];
        dModel_dconseq++;
      }
    }
  }
}
```

0.48 matlab_utilities/fuzderparam.cpp

MEX file that gets the jacobian matrix of a fuzzy model with respect to its parameters.
MATLAB help:

```
% FUZDERPARAM Gets the jacobian matrix of a fuzzy model respect its parameters.
%
%   dh_dparam = fuzderparam(Fuzzy_Model, Point)
%
% Arguments:
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
%   Point -> Point where the derivative will be calculate.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzderconseq, fuzeval, fuzjac, fuzlinear, fuzprint,
%          mat2antec, mat2conseq, mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es
  MARIANO J. AZNAR, marianojose.aznar@alu.uhu.es
```

```
  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"
#include <flt/derivatives.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t input, output, rule, param, m;
  int error=0;
  double *Point, *dModel_dparam;

  if(nrhs!=2)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>1)
    ERRORMSG(E_NumberArgOut)
  for (int i=0;i<nrhs;i++)
  {
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
  }

  System S;
  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)

  if (1 != mxGetN(prhs[1]))
    ERRORMSG(E_Column)
  if (S.inputs() != mxGetM(prhs[1]))
    ERRORMSG(E_PointCoherent)
  Point = mxGetPr(prhs[1]);

  m = S.outputs();
  plhs[0]=mxCreateDoubleMatrix(m,S.NumberOfAntecedents() + S.NumberOfConsequents(
      ),mxREAL);
  if (!plhs[0])
    ERRORMSG(E_NumberArgOut)

  dModel_dparam = mxGetPr(plhs[0]);
  TNT::Array2D<double> dS_dparam = derfuzzy(S, Point);
  if (dS_dparam.dim1()==0 || dS_dparam.dim2()==0 )
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(O_GeneralError)
  }

  for (size_t j=0;j<dS_dparam.dim2();j++)
```

```
  {
    for (size_t i=0;i<dS_dparam.dim1();i++)
    {
      *dModel_dparam = dS_dparam[i][j];
      dModel_dparam++;
    }
  }
}
```

0.49 matlab_utilities/fuzeval.cpp

MEX file that performs fuzzy inference calculations of open/closed loop fuzzy system.
MATLAB help:

```
% FUZEVAL Performs fuzzy inference calculation of open/closed loop fuzzy system.
%
%   Output = fuzeval(X_U,Fuzzy_Model)
%
%   Output = fuzeval(X,Plant,Controller)
%
%   X_U -> Input vector: [X;U] = [x1;x2;...;xn;u1;u2;...;um]
%
%   X -> State vector.
%
%   Plant -> Fuzzy model of the Plant.
%
%   Controller -> Fuzzy model of the Controller.
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzjac, fuzlinear, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.
```

```
  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i;
  int ode = 0;

  if(nrhs<2 || nrhs>4)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>1)
    ERRORMSG(E_NumberArgOut)
  for (i=0;i<nrhs;i++)
  {
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
  }

  double *X;
  Array1D<double> dX;

  System S,C;
  char error[MAX_FILE_NAME];

  if (nrhs!=2 && mxGetN(prhs[0])==1 && mxGetM(prhs[0])==1 && mxIsStruct(prhs[1])=
      =false && mxIsChar(prhs[1])==false)
    ode = 1; // If this function is called from 'ode', the 1st argument is the ti
      me

  // Read arguments
  X = mxGetPr(prhs[ode]);
  if (!X)
    ERRORMSG(E_NumberArgIn)
  if (mxGetN(prhs[ode])!=1)
    ERRORMSG(E_Column)
  if (readModel(prhs[1+ode],S))
  {
    mexPrintf("%s %s.\n",U_CheckModel,O_OfPlant);
    ERRORMSG(E_Model)
  }

  plhs[0] = mxCreateDoubleMatrix(S.outputs(),1,mxREAL);
  if (!plhs[0])
    ERRORMSG(E_NumberArgOut)

  // Run
  if (nrhs==(2+ode))
  {// Evaluate the plant
    if (mxGetM(prhs[ode])!=S.inputs())
    {
      mxDestroyArray(plhs[0]);
      ERRORMSG(E_PointCoherent)
    }
    dX = S.evaluate(X);
    if(!dX.dim())
    {
      mxDestroyArray(plhs[0]);
      mexPrintf("Fuzeval. %s\b or %s\n",U_Overflow,E_NoCoherent);
      mexErrMsgTxt("");
      return;
    }
```

```
  }
  else
  {// Evaluate the closed-loop system (plant + controller)
    if (mxGetM(prhs[ode])!=S.outputs())
    {
      mxDestroyArray(plhs[0]);
      ERRORMSG(E_PointCoherent)
    }
    if (readModel(prhs[2+ode],C))
    {
      mxDestroyArray(plhs[0]);
      mexPrintf("%s %s.\n",U_CheckModel,O_OfController);
      ERRORMSG(E_Model)
    }
    dX = evaluate(X, S, C);
    if (!dX.dim())
    {
      if (!error)
      {
        mxDestroyArray(plhs[0]);
        ERRORMSG(E_NoCoherent_BadX)
      }
      else
      {
        mxDestroyArray(plhs[0]);
        mexPrintf("Fuzeval. %s or",U_Overflow);
        ERRORMSG(E_NoCoherent)
      }
    }
  }

  double *MATLABOut = mxGetPr(plhs[0]);
  for (i=0;i<dX.dim();i++)
    MATLABOut[i] = dX[i];
}
```

0.50 matlab_utilities/fuzjac.cpp

MEX file that calculates the Jacobian matrix of the closed loop fuzzy system.

MATLAB help:

```
% FUZJAC Calculates the Jacobian matrix of the closed loop fuzzy system.
%
%   J = fuzjac(X, Plant, Controller)
%
% Arguments:
%
%   X -> State vector where the jacobian matrix is calculated.
%
%   Plant -> Fuzzy model of the Plant.
%
%   Controlador -> Fuzzy model of the Controller.
%
%   J -> Jacobian matrix of closed loop system.
%
%                                    (  df1           df1  )
%     dx1                            | -----   ...  ----- |
%    ----- = f1(x)                   |  dx1           dxn  |
%     dt                             |                     |
%     ...                      J =   |  ...    ...    ...  |
%     dxn                            |                     |
```

```
%    ----- = fn(x)                 |  dfn          dfn  |
%     dt                           | -----   ...  ----- |
%                                  (  dx1          dxn  )
%    x = [ x1 ; x2 ; ... ; xn]
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%         fuz2mat, fuzcomb, fuzeval, fuzlinear, fuzprint, mat2antec, mat2conseq,
%         mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"
#include <flt/derivatives.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i,q,n;
  int model;
  double *X,*J,*d;
  System S[2];
  char Sist[MAX_FILE_NAME];
  char error[MAX_FILE_NAME];

  if(nrhs!=3)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>1)
    ERRORMSG(E_NumberArgOut)
  for (i=0;i<nrhs;i++)
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)

  X = mxGetPr(prhs[0]);
  if (!X)
```

```
    ERRORMSG(E_NumberArgIn)

  n = mxGetM(prhs[0]);
  if (mxGetN(prhs[0])!=1)
    ERRORMSG(E_Column)

  for (model=1;model<3;model++)
  {
    if (readModel(prhs[model],S[model-1]))
    {
      sprintf(error,"%s %d.\n",E_Model, model);
      ERRORMSG(error)
    }
  }

  if ((nrhs==3) && n!=S[0].outputs())
    ERRORMSG(E_PointCoherent)
  if ((nrhs==2) && n!=S[0].inputs())
    ERRORMSG(E_PointCoherent)

  Array2D<double> Jac = jacobian(S[0], S[1], X);
  if (!Jac.dim1())
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_NoCoherent)
  }

  plhs[0] = mxCreateDoubleMatrix(n,n,mxREAL);
  J = mxGetPr(plhs[0]);
  if (!J)
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_NumberArgOut)
  }
  for (i=0;i<n;i++)
  {
    for (q=0;q<n;q++)
    {
      *J = Jac[q][i];
      J++;
    }
  }
}
```

0.51 matlab_utilities/fuzlinear.cpp

MEX file that calculartes the linealization of a Fuzzy Model in a point.

MATLAB help:

```
% FUZLINEAR Calculartes the linealization of a Fuzzy Model in a point.
%
%   A = fuzlinear(Fuzzy_Model, X, U)
%
%   [A,B] = fuzlinear(Fuzzy_Model, X, U)
%
%   [A,B,F] = fuzlinear(Fuzzy_Model, X, U)
%
% Solve the matrices the linear representation of the 'Fuzzy_Model' model
% according to Taylor series in the point X with signal control U.
% F is the value of the function at the point (X,U).
%
```

```
%                           Y = F + Ax + Bu
%
% Arguments:
%
%   Fuzzy_Model -> Input Fuzzy model.
%
%   X -> State vector.
%
%   U -> Control vector.
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzeval, fuzjac, fuzprint, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"
#include <flt/derivatives.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i,q,j,n,m=0,model,countA,countB;
  double *X,*U=NULL,*M_A,*M_B,*M_F;
  System S;

  if(nrhs<=2 || nrhs>3)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>3)
    ERRORMSG(E_NumberArgOut)
  for (i=0;i<nrhs;i++)
  {
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
```

```
}

X = mxGetPr(prhs[1]);
if (!X)
  ERRORMSG(E_NumberArgIn)

if (mxGetN(prhs[1])!=1)
  ERRORMSG(E_Column)

if (readModel(prhs[0],S))
  ERRORMSG(E_Model)

n = mxGetM(prhs[1]);
if (n!=S.outputs())
  ERRORMSG(E_PointCoherent)

if (nrhs==3)
{
  U = mxGetPr(prhs[2]);
  if (!U)
    ERRORMSG(E_NumberArgIn)
  if (mxGetN(prhs[2])!=1)
    ERRORMSG(E_Column)
  m = mxGetM(prhs[2]);
  if (m!=(S.inputs()-n))
    ERRORMSG(E_PointCoherent)
}
else if(m)
{
  U = new double[m];
  for (i=0;i<m;i++)
    U[i] = 0;
}
else
  U = NULL;

Array2D<double> B(n,m);
Array1D<double> F(n);
Array2D<double> A = jacobian(S,X,U,B,F);
if(!A.dim1())
{
  if (nrhs==2 && m!=0)
    delete []U;
  mxDestroyArray(plhs[0]);
  if (nlhs>1)
    mxDestroyArray(plhs[1]);
  if (nlhs==3)
    mxDestroyArray(plhs[2]);
  ERRORMSG(U_Overflow)
}

plhs[0] = mxCreateDoubleMatrix(n,n,mxREAL);
M_A = mxGetPr(plhs[0]);
if (!plhs[0] || !M_A)
{
  if (nrhs==2 && m!=0)
    delete []U;
  mxDestroyArray(plhs[0]);
  ERRORMSG(E_NumberArgOut)
}
if (nlhs>1 && nrhs==3)
{
  plhs[1] = mxCreateDoubleMatrix(n,m,mxREAL);
  M_B = mxGetPr(plhs[1]);
  if (!plhs[1] || !M_B)
  {
    if (nrhs==2 && m!=0)
```

```
        delete []U;
      mxDestroyArray(plhs[0]);
      mxDestroyArray(plhs[1]);
      ERRORMSG(E_NumberArgOut)
    }
  }
  if (nlhs==3)
  {
    plhs[2] = mxCreateDoubleMatrix(n,1,mxREAL);
    M_F = mxGetPr(plhs[2]);
    if (!plhs[2] || !M_F)
    {
      if (nrhs==2 && m!=0)
        delete []U;
      mxDestroyArray(plhs[0]);
      mxDestroyArray(plhs[1]);
      mxDestroyArray(plhs[2]);
      ERRORMSG(E_NumberArgOut)
    }
  }

  for (q=0,countA=0;q<n;q++)
  {
    for (i=0;i<n;i++,countA++)
      *(M_A+countA) = A[i][q];
  }
  if (nlhs>1)
  {
    for (j=0,countB=0;j<m;j++)
      for (i=0;i<n;i++,countB++)
        *(M_B+countB) = B[i][j];
    if (nlhs==3)
      for (i=0;i<n;i++,countB++)
        M_F[i] = F[i];
  }
  if (nrhs==2 && m!=0)
    delete []U;
}
```

0.52 matlab_utilities/fuzprint.cpp

MEX file that converts a fuzzy model in a text file with its linguistic representation.

MATLAB help:

```
% FUZPRINT Converts a fuzzy model in its linguistic representation.
%
% Use the linguistic form:
%   "IF Name_of_Input1 is...and Name_of_Input2 is...THEN Name_of_Output1 is..."
%
%   fuzprint('File.txt',Fuzzy_Model)
%
%   fuzprint('File.txt',Fuzzy_Model,Name_of_Input1,Name_of_Input2,...,
%            Name_of_Output1,Name_of_Output2,...)
%
%   fuzprint('File.txt',Fuzzy_Model,Name_of_Input1,Name_of_Input2,...,
%            Name_of_Output1,Name_of_Output2,...,Accuracy)
%
% Arguments:
%
%   'File.txt' -> String with the name of file to write. This file will be
%                 overwrited without confirmation.
```

```
%
%   Fuzzy_Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%                  or a 'FIS' variable from MATLAB Workspace.
%
%   Name_of_InputX -> String with the name of input X.
%
%   Name_of_OutputX -> String with the name of output X.
%
%   Accuracy -> Number of decimals to print (10 by defult).
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzeval, fuzjac, fuzlinear, mat2antec, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

#define ACCURACY 10 // Default accuracy to print the system

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  char File[MAX_CHAR_LONG];
  size_t i,j;
  int accuracy;

  if(nrhs<2)
    ERRORMSG(E_NumberArgIn)
  if(nlhs>0)
    ERRORMSG(E_NumberArgOut)
  for (int i=0;i<nrhs;i++)
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
  if(mxGetString(prhs[0],File,MAX_CHAR_LONG))
    ERRORMSG(E_FileOutput)

  System S;
```

```
  if(readModel(prhs[1],S))
    ERRORMSG(E_Model)

  if(nrhs>2)
  {
    if ( (nrhs<(2+S.inputs()+S.outputs())) || (nrhs>(3+S.inputs()+S.outputs())) )

      ERRORMSG(E_InNoCoherent)

    char **inputs=new char *[S.inputs()];
    char **outputs=new char *[S.outputs()];
    if (nrhs==(2+S.inputs()+S.outputs()))
      accuracy = ACCURACY;
    else
    {
      if (!mxIsNumeric(prhs[2+S.inputs()+S.outputs()]))
      {
        for (i=0;i<S.inputs();delete []inputs[i++]);
        for (i=0;i<S.outputs();delete []outputs[i++]);
        delete [] inputs;
        delete [] outputs;
        ERRORMSG(E_AccuracyNoNum)
      }
      accuracy = 1+(int)mxGetScalar(prhs[2+S.inputs()+S.outputs()]);
    }

    // Read inputs and outputs' names
    for (i=0,j=2;i<S.inputs();i++,j++)
    {
      inputs[i] = new char [MAX_CHAR_LONG];
      if(mxGetString(prhs[j],inputs[i],MAX_CHAR_LONG))
      {
        for (i=0;i<S.inputs();delete []inputs[i++]);
        for (i=0;i<S.outputs();delete []outputs[i++]);
        delete [] inputs;
        delete [] outputs;
        ERRORMSG(E_InNames)
      }
    }
    for (i=0,j=2+S.inputs();i<S.outputs();i++,j++)
    {
      outputs[i] = new char [MAX_CHAR_LONG];
      if(mxGetString(prhs[j],outputs[i],MAX_CHAR_LONG))
      {
        for (i=0;i<S.inputs();delete []inputs[i++]);
        for (i=0;i<S.outputs();delete []outputs[i++]);
        delete [] inputs;
        delete [] outputs;
        ERRORMSG(E_OutNames)
      }
    }
    if(printSystem(File,S,inputs,outputs,accuracy))
    {
      for (i=0;i<S.inputs();delete []inputs[i++]);
      for (i=0;i<S.outputs();delete []outputs[i++]);
      delete [] inputs;
      delete [] outputs;
      ERRORMSG(E_FileOutput)
    }
    for (i=0;i<S.inputs();delete []inputs[i++]);
    for (i=0;i<S.outputs();delete []outputs[i++]);
    delete [] inputs;
    delete [] outputs;
  }
  else if(printSystem(File,S))
    ERRORMSG(E_FileOutput)
}
```

0.53 matlab_utilities/initializeController.cpp

MEX file that initializes a fuzzy Controller from a fuzzy Plant.

MATLAB help:

```
% INITIALIZECONTROLLER Initializes a fuzzy Controller from a fuzzy Plant.
%
%   Controller = initializeController(Plant)
%
% Arguments:
%
%   Plant -> Plant model. This model could be a '.txt' file, a '.fis' file,
%            or a 'FIS' variable from MATLAB Workspace.
%
%   Controller -> An initial controller based in the Plant. This controller
%                 has all its consecuents to 0, and each of its outputs have
%                 so many rules as all of the plant.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat,fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec,
%          mat2conseq, mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  char file[MAX_CHAR_LONG];

  if(nrhs!=1)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>1)
    ERRORMSG(E_NumberArgOut)
```

```
  if (mxIsEmpty(prhs[0]) || mxIsNaN(*mxGetPr(prhs[0])))
    ERRORMSG(E_ArgNoValid)

  System P,C;
  if (readModel(prhs[0],P))
    ERRORMSG(E_Model)

  C = initialController(P);
  if (!C.outputs())
  {
    mexErrMsgTxt("The input fuzzy model is not a valid Plant\n");
    return;
  }
  const char* names[]={"name","type","andMethod","orMethod","defuzzMethod","impMe
      thod","aggMethod","input","output","rule"};
  int dim[] = {1,1};
  plhs[0] = mxCreateStructArray(2,dim,10,names);
  if(System2FIS(C,plhs[0]))
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_FISOut)
  }
}
```

0.54 matlab_utilities/Kalmanantec.cpp

MEX file that calculates an iteration of EKF only for antecedents.

MATLAB help:

```
function [Model, P, err] = Kalmanantec(Model, x, output, covariance, P)
% KALMANANTEC. Calculates an iteration of EKF only for antecedents.
%
%   [Model, P, err] = Kalmanantec(Model, x, output, covariance, P)
%
%   [Model, P, err] = Kalmanantec(Model, x, output, covariance, P, Phi)
%
% Arguments:
%
%   Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%            or a 'FIS' variable from MATLAB Workspace.
%
%   x -> Current State vector, x(k).
%
%   output -> Real output of the system in the current iteration.
%
%   covariance -> Noise covariance matrix estimated from the hope operator.
%
%   P -> Covariance matrix of the filter.
%
%   Phi -> Jacobian matrix that relates the parameters to be set with the
%          following value of these parameters.
%          If not specified, Phi is assumed to be the identity matrix.
%
% See also Kalmanconseq, Kalmanfuz, activation, antec2mat, aproxjac, aproxlinear
%          conseq2mat, fis2txt, fuz2mat, fuzcomb, fuzeval, fuzjac, fuzlinear,
%          fuzprint, mat2antec, mat2conseq, mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
```

```
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es
  MARIANO J. AZNAR, marianojose.aznar@alu.uhu.es
  AGUSTÍN JIMÉNEZ AVELLO, agustin.jimenez@upm.es
  BASIL M. AL-HADITHI, basil.alhadithi@upm.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"
#include <flt/Kalman.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  // Reads and checks arguments
  if(nrhs<5 | nrhs>6)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>3)
    ERRORMSG(E_NumberArgOut)
  for (size_t i=0;i<nrhs;i++)
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)

  System Model;
  if (readModel(prhs[0],Model))
    ERRORMSG(E_Model)

  int m = Model.outputs();

  double *p_input = mxGetPr(prhs[1]);
  if (!p_input)
    ERRORMSG(E_NumberArgIn)
  int n = mxGetM(prhs[1]);
  if (n==1)
    n = mxGetN(prhs[1]);
  Array1D<double> input(n, p_input);

  double *p_output = mxGetPr(prhs[2]);
  if (!p_output)
    ERRORMSG(E_NumberArgIn)
  n = mxGetM(prhs[2]);
  if (n==1)
    n = mxGetN(prhs[2]);
  Array1D<double> output(n, p_output);
```

```
double *p_covariance = mxGetPr(prhs[3]);
if (!p_covariance)
  ERRORMSG(E_NumberArgIn)
Array2D<double> covariance = col2row_major(p_covariance, m, m);

n = mxGetN(prhs[4]);
double *p_P = mxGetPr(prhs[4]);
if (!p_P)
  ERRORMSG(E_NumberArgIn)
Array2D<double> P = col2row_major(p_P, n, n);

// Does the iteration
Array1D<double> error;
if (nrhs==5)
  error = KalmanAntec(Model, input, output, covariance, P);
else
{
  double *p_Phi = mxGetPr(prhs[5]);
  if (!p_Phi)
    ERRORMSG(E_NumberArgIn)
  Array2D<double> Phi = col2row_major(p_Phi, n, n);

  error = KalmanAntec(Model, input, output, covariance, P, Phi);
}

if (error.dim() == 0)
  ERRORMSG(E_InNoCoherent)

// Generates outputs
const char* names[]={"name","type","andMethod","orMethod","defuzzMethod","impMe
    thod","aggMethod","input","output","rule"};
int dim[] = {1,1};
plhs[0] = mxCreateStructArray(2,dim,10,names);
if(System2FIS(Model, plhs[0]))
{
  mxDestroyArray(plhs[0]);
  ERRORMSG(E_CreateFIS)
}

if (nlhs>1)
{
  plhs[1] = mxCreateDoubleMatrix(n,n,mxREAL);
  double *p_newP = mxGetPr(plhs[1]);
  if (!p_newP)
  {
    mxDestroyArray(plhs[0]);
    mxDestroyArray(plhs[1]);
    ERRORMSG(E_NumberArgOut)
  }
  row2col_major(P[0], p_newP, n, n);

  if (nlhs>2)
  {
    plhs[2] = mxCreateDoubleMatrix(m,1,mxREAL);
    double *p_error = mxGetPr(plhs[2]);
    if (!p_error)
    {
      mxDestroyArray(plhs[0]);
      mxDestroyArray(plhs[1]);
      mxDestroyArray(plhs[2]);
      ERRORMSG(E_NumberArgOut)
    }
    for (size_t i=0;i<m;i++)
    {
      *p_error = error[i];
      p_error++;
    }
```

```
    }
  }
}
```

0.55 matlab_utilities/Kalmanconseq.cpp

MEX file that calculates an iteration of EKF only for consequents.

MATLAB help:

```
function [Model, P, err] = Kalmanconseq(Model, x, output, covariance, P)
% KALMANCONSEQ. Calculates an iteration of EKF only for consequents.
%
%   [Model, P, err] = Kalmanconseq(Model, x, output, covariance, P)
%
%   [Model, P, err] = Kalmanconseq(Model, x, output, covariance, P, Phi)
%
% Arguments:
%
%   Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%            or a 'FIS' variable from MATLAB Workspace.
%
%   x -> Current State vector, x(k).
%
%   output -> Real output of the system in the current iteration.
%
%   covariance -> Noise covariance matrix estimated from the hope operator.
%
%   P -> Covariance matrix of the filter.
%
%   Phi -> Jacobian matrix that relates the parameters to be set with the
%          following value of these parameters.
%          If not specified, Phi is assumed to be the identity matrix.
%
% See also Kalmanantec, Kalmanfuz, activation, antec2mat, aproxjac, aproxlinear
%          conseq2mat, fis2txt, fuz2mat, fuzcomb, fuzeval, fuzjac, fuzlinear,
%          fuzprint, mat2antec, mat2conseq, mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es
  MARIANO J. AZNAR, marianojose.aznar@alu.uhu.es
  AGUSTÍN JIMÉNEZ AVELLO, agustin.jimenez@upm.es
  BASIL M. AL-HADITHI, basil.alhadithi@upm.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.
```

```
  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"
#include <flt/Kalman.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  // Reads and checks arguments
  if(nrhs<5 | nrhs>6)
    ERRORMSG(E_NumberArgIn)

  if (nlhs>3)
    ERRORMSG(E_NumberArgOut)

  for (size_t i=0;i<nrhs;i++)
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)

  System Model;
  if (readModel(prhs[0],Model))
    ERRORMSG(E_Model)

  int m = Model.outputs();

  double *p_input = mxGetPr(prhs[1]);
  if (!p_input)
    ERRORMSG(E_NumberArgIn)
  int n = mxGetM(prhs[1]);
  if (n==1)
    n = mxGetN(prhs[1]);
  Array1D<double> input(n, p_input);

  double *p_output = mxGetPr(prhs[2]);
  if (!p_output)
    ERRORMSG(E_NumberArgIn)
  n = mxGetM(prhs[2]);
  if (n==1)
    n = mxGetN(prhs[2]);
  Array1D<double> output(n, p_output);

  double *p_covariance = mxGetPr(prhs[3]);
  if (!p_covariance)
    ERRORMSG(E_NumberArgIn)
  Array2D<double> covariance = col2row_major(p_covariance, m, m);

  double *p_P = mxGetPr(prhs[4]);
  if (!p_P)
    ERRORMSG(E_NumberArgIn)
  n = mxGetM(prhs[4]);
  Array2D<double> P = col2row_major(p_P, n, n);

  // Does the iteration
  Array1D<double> error;
  if (nrhs==5)
    error = KalmanConseq(Model, input, output, covariance, P);
  else
  {
```

```
    double *p_Phi = mxGetPr(prhs[5]);
    if (!p_Phi)
      ERRORMSG(E_NumberArgIn)
    Array2D<double> Phi = col2row_major(p_Phi, n, n);

    error = KalmanConseq(Model, input, output, covariance, P, Phi);
  }

  if (error.dim() == 0)
    ERRORMSG(E_NumberArgIn)

  // Generates outputs
  const char* names[]={"name","type","andMethod","orMethod","defuzzMethod","impMe
      thod","aggMethod","input","output","rule"};
  int dim[] = {1,1};
  plhs[0] = mxCreateStructArray(2,dim,10,names);
  if(System2FIS(Model, plhs[0]))
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_CreateFIS)
  }

  if (nlhs>1)
  {
    plhs[1] = mxCreateDoubleMatrix(n,n,mxREAL);
    double *p_newP = mxGetPr(plhs[1]);
    if (!p_newP)
    {
      mxDestroyArray(plhs[0]);
      mxDestroyArray(plhs[1]);
      ERRORMSG(E_NumberArgOut)
    }
    row2col_major(P[0], p_newP, n, n);

    if (nlhs>2)
    {
      plhs[2] = mxCreateDoubleMatrix(m,1,mxREAL);
      double *p_error = mxGetPr(plhs[2]);
      if (!p_error)
      {
        mxDestroyArray(plhs[0]);
        mxDestroyArray(plhs[1]);
        mxDestroyArray(plhs[2]);
        ERRORMSG(E_NumberArgOut)
      }
      for (size_t i=0;i<m;i++)
      {
        *p_error = error[i];
        p_error++;
      }
    }
  }
}
```

0.56 matlab_utilities/Kalmanfuz.cpp

MEX file that calculates an iteration of EKF for antecedents and consequents.

MATLAB help:

```
function [Model, P, err] = Kalmanfuz(Model, x, output, covariance, P)
% KALMANFUZ. Calculates an iteration of EKF for antecedents and consequents.
```

```
%
%   [Model, P, err] = Kalmanfuz(Model, x, output, covariance, P)
%
%   [Model, P, err] = Kalmanfuz(Model, x, output, covariance, P, Phi)
%
% Arguments:
%
%   Model -> This fuzzy model could be a '.txt' file, a '.fis' file,
%            or a 'FIS' variable from MATLAB Workspace.
%
%   x -> Current State vector, x(k).
%
%   output -> Real output of the system in the current iteration.
%
%   covariance -> Noise covariance matrix estimated from the hope operator.
%
%   P -> Covariance matrix of the filter.
%
%   Phi -> Jacobian matrix that relates the parameters to be set with the
%          following value of these parameters.
%          If not specified, Phi is assumed to be the identity matrix.
%
% See also Kalmanantec, Kalmanconseq, activation, antec2mat, aproxjac
%          aproxlinear, conseq2mat, fis2txt, fuz2mat, fuzcomb, fuzeval, fuzjac,
%          fulinear, fuzprint, mat2antec, mat2conseq, mat2fuz, subsystem,
%          txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es
  MARIANO J. AZNAR, marianojose.aznar@alu.uhu.es
  AGUSTÍN JIMÉNEZ AVELLO, agustin.jimenez@upm.es
  BASIL M. AL-HADITHI, basil.alhadithi@upm.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"
#include <flt/Kalman.hpp>

using namespace FLT;
using namespace TNT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
```

```
{
  // Reads and checks arguments
  if(nrhs<5 | nrhs>6)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>3)
    ERRORMSG(E_NumberArgOut)
  for (size_t i=0;i<nrhs;i++)
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)

  System Model;
  if (readModel(prhs[0],Model))
    ERRORMSG(E_Model)

  int m = Model.outputs();

  double *p_input = mxGetPr(prhs[1]);
  if (!p_input)
    ERRORMSG(E_NumberArgIn)
  int n = mxGetM(prhs[1]);
  if (n==1)
    n = mxGetN(prhs[1]);
  Array1D<double> input(n, p_input);

  double *p_output = mxGetPr(prhs[2]);
  if (!p_output)
    ERRORMSG(E_NumberArgIn)
  n = mxGetM(prhs[2]);
  if (n==1)
    n = mxGetN(prhs[2]);
  Array1D<double> output(n, p_output);

  double *p_covariance = mxGetPr(prhs[3]);
  if (!p_covariance)
    ERRORMSG(E_NumberArgIn)
  Array2D<double> covariance = col2row_major(p_covariance, m, m);

  n = mxGetN(prhs[4]);
  double *p_P = mxGetPr(prhs[4]);
  if (!p_P)
    ERRORMSG(E_NumberArgIn)
  Array2D<double> P = col2row_major(p_P, n, n);

  // Does the iteration
  Array1D<double> error;
  if (nrhs==5)
    error = KalmanFuz(Model, input, output, covariance, P);
  else
  {
    double *p_Phi = mxGetPr(prhs[5]);
    if (!p_Phi)
      ERRORMSG(E_NumberArgIn)
    Array2D<double> Phi = col2row_major(p_Phi, n, n);

    error = KalmanFuz(Model, input, output, covariance, P, Phi);
  }

  if (error.dim() == 0)
    ERRORMSG(E_InNoCoherent)

  // Generates outputs
  const char* names[]={"name","type","andMethod","orMethod","defuzzMethod","impMe
      thod","aggMethod","input","output","rule"};
  int dim[] = {1,1};
  plhs[0] = mxCreateStructArray(2,dim,10,names);
  if(System2FIS(Model, plhs[0]))
  {
```

```
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_CreateFIS)
  }

  if (nlhs>1)
  {
    plhs[1] = mxCreateDoubleMatrix(n,n,mxREAL);
    double *p_newP = mxGetPr(plhs[1]);
    if (!p_newP)
    {
      mxDestroyArray(plhs[0]);
      mxDestroyArray(plhs[1]);
      ERRORMSG(E_NumberArgOut)
    }
    row2col_major(P[0], p_newP, n, n);

    if (nlhs>2)
    {
      plhs[2] = mxCreateDoubleMatrix(m,1,mxREAL);
      double *p_error = mxGetPr(plhs[2]);
      if (!p_error)
      {
        mxDestroyArray(plhs[0]);
        mxDestroyArray(plhs[1]);
        mxDestroyArray(plhs[2]);
        ERRORMSG(E_NumberArgOut)
      }
      for (size_t i=0;i<m;i++)
      {
        *p_error = error[i];
        p_error++;
      }
    }
  }
}
```

0.57 matlab_utilities/mat2antec.cpp

MEX file that changes the antecedent of a fuzzy model.

MATLAB help:

```
% MAT2ANTEC Changes the antecedent of a fuzzy model.
%
%   Final_Model = mat2antec(Initial_Model,Vector)
%
%   Final_Model = mat2antec(Initial_Model,Vector,Output,Rule)
%
%   [Final_Model,length] = mat2antec(Initial_Model,Vector)
%
%   [Final_Model,length] = mat2antec(Initial_Model,Vector,Output,Rule)
%
% Arguments:
%
%   Final_Model -> Changed fuzzy model.
%
%   Output -> Use to select the antecedent of one output.
%
%   Rule -> Use to select the antecedent of one rule for given output.
%
%   length -> The length of Vector, 0 if error.
%
```

```
%   Initial_Model -> Fuzzy Model to change. This model could be a '.txt' file,
%                    a '.fis' file, or a 'FIS' variable from MATLAB Workspace.
%
%   Vector -> Vector with data for fuzzy model antecedent.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2conseq,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i, rule, out, length;
  double *data;

  if(nrhs<2)
    ERRORMSG(E_NumberArgIn)
  if (nrhs>2 && nrhs!=4)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>2)
    ERRORMSG(E_NumberArgOut)
  if (mxIsEmpty(prhs[0]) || mxIsNaN(*mxGetPr(prhs[0])))
    ERRORMSG(E_ArgNoValid)

  if (nrhs==4)
  {
    if (mxGetM(prhs[2])>1 || mxGetM(prhs[3])>1)
      ERRORMSG(E_RuleOutputFileVector)
    if (mxGetN(prhs[2])!=mxGetN(prhs[3]))
      ERRORMSG(E_NumberArg)
  }

  System S;
  if (readModel(prhs[0],S))
```

```
  ERRORMSG(E_Model)

if (nlhs==2)
{
  if (nrhs==2)
  {
    length = S.NumberOfAntecedents();
    if (!length)
      ERRORMSG(E_BadModel)
  }
  else
  {
    length = 0;
    for (i=0;i<mxGetN(prhs[2]);i++)
    {
      out = ((size_t)*(mxGetPr(prhs[2])+i))-1; // MATLAB is index-1 but C/C++ i
    s index-0
      rule = ((size_t)*(mxGetPr(prhs[3])+i))-1;
      length += S.readRule(out,rule)->NumberOfAntecedents();
    }

    if (!length)
      mexPrintf("??? Error using ==> mat2antec\n%s, or\n%s\n",E_BadModel,E_Numb
    erArgIn);
  }
}

// Save data in S
data = mxGetPr(prhs[1]);
int error = 0;

if (nrhs==2)
  error = S.setAntecedents(data);
else
{
  for (i=0;i<mxGetN(prhs[2]);i++)
  {
    out = ((size_t)*(mxGetPr(prhs[2])+i))-1; // MATLAB is index-1 but C/C++ is
    index-0
    rule = ((size_t)*(mxGetPr(prhs[3])+i))-1;
    size_t numAntec = S.readRule(out,rule)->NumberOfAntecedents();
    error += S.readRule(out,rule)->setAntecedents(data);
    data += numAntec;
  }
}

if (error>0)
{
  ERRORMSG(E_Store)
  plhs[0] = mxCreateDoubleMatrix(0,0,mxREAL); // Empty matrix
  return;
}

if (nlhs==2)
  plhs[1] = mxCreateDoubleScalar(length);

// Create the FIS variable
const char* names[] = {"name","type","andMethod","orMethod","defuzzMethod","imp
    Method","aggMethod","input","output","rule"};
int dim[] = {1,1};
plhs[0] = mxCreateStructArray(2,dim,10,names);

if(System2FIS(S, plhs[0]))
{
  mxDestroyArray(plhs[0]);
  ERRORMSG(E_CreateFIS)
}
```

```
}
```

0.58 matlab_utilities/mat2conseq.cpp

MEX file that changes the consequent of a fuzzy model.

MATLAB help:

```
% MAT2CONSEQ Changes the consequent of a fuzzy model.
%
%   Final_Model = mat2conseq(Initial_Model,Vector)
%
%   Final_Model = mat2conseq(Initial_Model,Vector,Output,Rule)
%
%   [Final_Model,length] = mat2conseq(Initial_Model,Vector)
%
%   [Final_Model,length] = mat2conseq(Initial_Model,Vector,Output,Rule)
%
% Arguments:
%
%   Final_Model -> Changed fuzzy model.
%
%   Output -> Use to select the consequent of one output.
%
%   Rule -> Use to select the consequent of one rule for given output.
%
%   length -> The length of Vector, 0 if error.
%
%   Initial_Model -> Fuzzy Model to change. This model could be a '.txt' file,
%                    a '.fis' file, or a 'FIS' variable from MATLAB Workspace.
%
%   Vector -> Vector with data for fuzzy model consequent.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec,
%          mat2fuz, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.
```

```
  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i, n_con, out,rule;
  double *data;
  if(nrhs<2)
    ERRORMSG(E_NumberArgIn)
  if (nrhs>2 && nrhs!=4)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>2)
    ERRORMSG(E_NumberArgOut)
  if (mxIsEmpty(prhs[0]) || mxIsNaN(*mxGetPr(prhs[0])))
    ERRORMSG(E_ArgNoValid)
  if (nrhs==4)
  {
    if (mxGetM(prhs[2])>1 || mxGetM(prhs[3])>1)
      ERRORMSG(E_RuleOutputFileVector)
    if (mxGetN(prhs[2])!=mxGetN(prhs[3]))
      ERRORMSG(E_NumberArg)
  }

  System S;
  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)
  size_t NoC = S.inputs() + 1;

  if (nlhs==2)
  {
    if (nrhs==2)
    {
      n_con = S.NumberOfConsequents();
    }
    else
    {
      n_con = 0;
      for (i=0;i<mxGetN(prhs[2]);i++)
      {
        out = ((size_t)*(mxGetPr(prhs[2])+i))-1; // MATLAB is index-1 but C/C++ i
      s index-0
        rule = ((size_t)*(mxGetPr(prhs[3])+i))-1;
        n_con += NoC;
      }
    }
    if (!n_con)
      ERRORMSG(E_BadModel)
  }

  // Save data in S
  data = mxGetPr(prhs[1]);
  if (nrhs==2)
    S.setConsequents(data);
  else
  {
    for (i=0;i<mxGetN(prhs[2]);i++)
    {
      out = ((size_t)*(mxGetPr(prhs[2])+i))-1;
      rule = ((size_t)*(mxGetPr(prhs[3])+i))-1;
      S.readRule(out,rule)->setConsequents(data+NoC*i);
    }
```

```
  }
  if (nlhs==2)
    plhs[1] = mxCreateDoubleScalar(n_con);

  // Create the FIS variable
  const char* names[] = {"name","type","andMethod","orMethod","defuzzMethod","imp
      Method","aggMethod","input","output","rule"};
  int dim[] = {1,1};
  plhs[0]= mxCreateStructArray(2,dim,10,names);
  if(System2FIS(S, plhs[0]))
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_CreateFIS)
  }
}
```

0.59 matlab_utilities/mat2fuz.cpp

MEX file that changes all data of a fuzzy model.

MATLAB help:

```
% MAT2FUZ Changes all data of a fuzzy model.
%
%   Final_Model = mat2fuz(Initial_Model,Vector)
%
%   Final_Model = mat2fuz(Initial_Model,Vector,Output,Rule)
%
%   [Final_Model,length] = mat2fuz(Initial_Model,Vector)
%
%   [Final_Model,length] = mat2fuz(Initial_Model,Vector,Output,Rule)
%
% Arguments:
%
%   Final_Model -> Changed fuzzy model.
%
%   Output -> Use to select the output.
%
%   Rule -> Use to select the rule for given output.
%
%   length -> The length of Vector, 0 if error.
%
%   Initial_Model -> Fuzzy model to change. This model could be a '.txt' file,
%                    a '.fis' file, or a 'FIS' variable from MATLAB Workspace.
%
%   Vector -> Vector with data for fuzzy model.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec,
%          mat2conseq, subsystem, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es
```

```
  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i, length, out, rule;
  double *data;

  if(nrhs<2)
    ERRORMSG(E_NumberArgIn)
  if (nrhs>2)
  {
    if (nrhs!=4)
      ERRORMSG(E_NumberArgIn)
    if (mxGetM(prhs[2])>1 || mxGetM(prhs[3])>1)
      ERRORMSG(E_RuleOutputFileVector)
    if (mxGetN(prhs[2])!=mxGetN(prhs[3]))
      ERRORMSG(E_NumberArg)
  }

  if (nlhs>2)
    ERRORMSG(E_NumberArgOut)
  if (mxIsEmpty(prhs[0]) || mxIsNaN(*mxGetPr(prhs[0])))
    ERRORMSG(E_ArgNoValid)

  System S;
  if (readModel(prhs[0],S))
    ERRORMSG(E_Model)

  size_t NoC = S.inputs() + 1;

  // Save data in S
  data = mxGetPr(prhs[1]);
  int error = 0;
  if (nrhs==2)
  {
    error = S.setAntecedents(data);
    length = S.NumberOfAntecedents();
    S.setConsequents(data + length);
    length += S.NumberOfConsequents();
  }
  else
  {
    length = 0;
    for (i=0;i<mxGetN(prhs[2]);i++)
    {
      out = ((size_t)*(mxGetPr(prhs[2])+i))-1; // MATLAB is index-1 but C/C++ is
```

```
      index-0
      rule = ((size_t)*(mxGetPr(prhs[3])+i))-1;

      Rule *R = S.readRule(out,rule);

      size_t numAntec = R->NumberOfAntecedents();
      error += R->setAntecedents(data);
      data += numAntec;

      R->setConsequents(data);
      data += NoC;

      length += numAntec + NoC;
    }

  }

  if (error>0)
  {
    ERRORMSG(E_Store)
    plhs[0] = mxCreateDoubleMatrix(0,0,mxREAL); // Empty matrix
    return;
  }

  if (nlhs==2)
    plhs[1] = mxCreateDoubleScalar(length);


   // Create the FIS variable
  const char* names[] = {"name","type","andMethod","orMethod","defuzzMethod","imp
      Method","aggMethod","input","output","rule"};
  int dim[] = {1,1};
  plhs[0] = mxCreateStructArray(2,dim,10,names);

  if(System2FIS(S, plhs[0]))
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_CreateFIS)
  }
}
```

0.60 matlab_utilities/subsystem.cpp

MEX file that generates a subsystem (submodel) from a fuzzy model.

MATLAB help:

```
% SUBSYSTEM Generates a subsystem (submodel) from a fuzzy model.
%
%   SubFIS = subsystem(FIS_Model,Outputs,Rules)
%
% Arguments:
%
%   FIS_Model -> Original fuzzy model.
%
%   Outputs -> List of outputs for selected rules.
%
%   Rules -> Selected rules (for each output) to make the subsystem.
%
%   SubFIS -> Generated subsystem.
%
% For example:
```

```
%
%   P = readfis('Plant.fis')
%   subP = subsystem(P,[1,2,2],[1,1,2])
%
%  'subP' has 3 rules, the 1st rule for the 1st output of P, and the 1st and
%  2nd rule for the 2nd output of P.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec,
%          mat2conseq, mat2fuz, txt2fis
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  size_t i,j,n1,n2,m1,m2,n;
  size_t *outputs,*rules;
  size_t nOutputs, nRules;
  double *poutput,*prule;
  System S, Snew;
  char error[MAX_CHAR_LONG];
  if(nrhs!=3)
  {
    sprintf(error,"%s %s, %s",E_NumberArg,O_OfIn,U_SeeHelp);
    ERRORMSG(error)
  }
  if (nlhs>1)
  {
    sprintf(error,"%s %s, %s",E_NumberArg,O_OfOut,U_SeeHelp);
    ERRORMSG(error)
  }
  for (int i=0;i<nrhs;i++)
    if (mxIsEmpty(prhs[i]) || mxIsNaN(*mxGetPr(prhs[i])))
      ERRORMSG(E_ArgNoValid)
```

```
  n1 = mxGetN(prhs[1]);
  n2 = mxGetN(prhs[2]);
  m1 = mxGetM(prhs[1]);
  m2 = mxGetM(prhs[2]);
  if (n1!=n2 || m1!=m2)
  {
    sprintf(error,"%s The sizes of Outputs and Rules must be the same, %s",E_Numb
      erArgIn,U_SeeHelp);
    ERRORMSG(error)
  }
  if (n1==1 && n2==1 && m1==m2)
    n=m1;
  else if (m1==1 && m2==1 && n1==n2)
    n=n1;
  else
  {
    sprintf(error,"%s, %s",E_NumberArgIn,U_SeeHelp);
    ERRORMSG(error)
  }

  if(readModel(prhs[0], S))
    ERRORMSG(E_Model)

  poutput = mxGetPr(prhs[1]);
  prule = mxGetPr(prhs[2]);
  outputs = new size_t[n];
  rules = new size_t[n];
  for (i=0;i<n;i++)
  {
    *(outputs+i) = ((size_t)*poutput)-1; // MATLAB is index-1 but C/C++ is index-
      0
    *(rules+i) = ((size_t)*prule)-1;
    poutput++;
    prule++;
  }

  Snew = subSystem(S,n,outputs,rules);
  if(Snew.outputs()==0)
  {
    delete []outputs;
    delete []rules;
    sprintf(error,"%s, %s\nAt least, 1 rule for each output is needed.\nCheck ind
      ices.",E_NumberArgIn,U_SeeHelp);
    ERRORMSG(error)
  }

  delete [] outputs;
  delete [] rules;
  const char* names[]={"name","type","andMethod","orMethod","defuzzMethod","impMe
      thod","aggMethod","input","output","rule"};
  int dim[] = {1,1};
  plhs[0] = mxCreateStructArray(2,dim,10,names);
  if(System2FIS(Snew,plhs[0]))
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_FISOut)
  }
}
```

0.61 matlab_utilities/txt2fis.cpp

MEX file that reads a TXT fuzzy model and creates a FIS variable.

MATLAB help:

```
% TXT2FIS Reads a TXT fuzzy model and creates a FIS variable.
%
%   FIS_Model = txt2fis('File.txt')
%
% Arguments:
%
%   'File.txt' -> String with the name of TXT-file to read.
%
%   FIS_Model -> 'FIS' fuzzy model.
%
% See also activation, antec2mat, aproxjac, aproxlinear, conseq2mat, fis2txt,
%          fuz2mat, fuzcomb, fuzeval, fuzjac, fuzlinear, fuzprint, mat2antec,
%          mat2conseq, mat2fuz, subsystem
```

Source code:

```
/*  Copyright (C) 2004-2011
  ANTONIO JAVIER BARRAGAN, antonio.barragan@diesia.uhu.es
  http://uhu.es/antonio.barragan

  Collaborators:
  JOSE MANUEL ANDUJAR, andujar@diesia.uhu.es

  DPTO. DE ING. ELECTRONICA, DE SISTEMAS INFORMATICOS Y AUTOMATICA
  ETSI, UNIVERSITY OF HUELVA (SPAIN)

  For more information, please contact with authors.

  This software is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This software is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aux_matlab.hpp"

#define MATLAB_BUFLEN ((mxGetM(prhs[0])*mxGetN(prhs[0])*sizeof(mxChar))+1)

using namespace FLT;

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  char file[MAX_CHAR_LONG];
  System S;

  if(nrhs!=1)
    ERRORMSG(E_NumberArgIn)
  if (nlhs>1)
    ERRORMSG(E_NumberArgOut)
  if (mxIsEmpty(prhs[0]) || mxIsNaN(*mxGetPr(prhs[0])))
    ERRORMSG(E_ArgNoValid)
  if(mxGetString(prhs[0],file,MATLAB_BUFLEN))
    ERRORMSG(E_NumberArgIn)
  S = TXT2System(file);
  if(!S.outputs())
```

```
    ERRORMSG(E_BadModel)

  const char* names[] = {"name","type","andMethod","orMethod","defuzzMethod","imp
      Method","aggMethod","input","output","rule"};
  int dim[] = {1,1};

  plhs[0] = mxCreateStructArray(2,dim,10,names);

  if(System2FIS(S,plhs[0]))
  {
    mxDestroyArray(plhs[0]);
    ERRORMSG(E_FISOut)
  }
}
```

Index

FUZZY LOGIC TOOLS REFERENCE MANUAL v1.0

©2011 Antonio Javier Barragan Piña & José Manuel Andújar Márquez

Publication Service of the University of Huelva

Further information about this book is available at http://uhu.es/antonio.barragan/flt

Permission is granted to copy, distribute and/or modify this document under the terms of the Creative Commons Attribution 3.0 License. You are free to share, copy, distribute, transmit, remix and adapt this work, but you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

With the understanding that:

- Waiver: Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain: Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- Other Rights: In no way are any of the following rights affected by the license:
  - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations.
  - The author's moral rights.
  - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- Notice: For any reuse or distribution, you must make clear to others the license terms of this work.

To view a copy of the full license, please go to http://creativecommons.org/licenses/by/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

This book has been generated by Doxygen (http://www.stack.nl/~dimitri/doxygen) and it has subsequently been revised and edited by the authors.

Fuzzy Logic Tools (FLT) is licensed under the GNU/GPLv3 or higher[1]. You can view this license at: http://www.gnu.org/licenses/gpl.html

FLT has been written in C++. This software has been compiled and tested in 32 and 64 bits architectures on GNU/Linux using GCC (http://gcc.gnu.org), and in 32 bits architectures on Microsoft Windows® operating systems[2] using MinGW (http://www.mingw.org).

This text provides some useful functions for use with MATLAB® API[3]. The authors recommend that you use GNUMex (http://gnumex.sourceforge.net) to compile MEX files in Microsoft Windows® operating systems.

ISBN: 978-84-15147-32-9.

Legal Deposit: H 125-2012.

[1] The GNU General Public License is a free, copyleft license for software and other kinds of works.

[2] The Microsoft brand name and is owned by Microsoft Corporation (http://www.microsoft.com).

[3] Currently the license of MATLAB® is owned by MathWorks MATLAB Inc.

respect to its inputs or parameters, or obtain a linearized model of a fuzzy system at a given point.

Since the functions provided by derivative.hpp, several methods for the parametric adaptaition of fuzzy systems have been developed based on the extended Kalman filter. These algorithms are in Kalman.hpp.

Kalman.hpp implements the adaptation of a fuzzy model by the extended Kalman filter from the derivatives included in derivatives.hpp.

In aux_matlab.hpp some util functions to use with MATLAB© API are defined, as the reading/writing of fuzzy models from/to MATLAB©, and the conversion of row-major matrices in column-major and vice versa.

Finally, several examples of programming MEX functions are presented, which can be useful for the analysis and design of fuzzy control systems using MATLAB©.

1.1 Obtaining FLT, Help and Bugs reports:

You can download the latest versions of the FLT and access to forums and bugs tracking system at http://sourceforge.net/projects/fuzzylogictools.

Additional information, news and related projects are available from the FLT website: http://uhu.es/antonio.barragan/flt.

1.2 Publications/references

If you would like to refer the Fuzzy Logic Tools (FLT) in a journal paper or similar, the recommended way is to cite this reference manual or the following papers:

A.J. Barragán and J.M. Andújar, *Fuzzy Logic Tools Reference Manual*, ISBN 978-84-15147-32-9, http://uhu.es/antonio.barragan/flt.

Andújar, J. M. and Barragán, A. J. (2005). A methodology to design stable nonlinear fuzzy control systems. *Fuzzy Sets and Systems*, 154(2):157–181.

Andújar, J. M., Barragán, A. J., and Gegúndez, M. E. (2009). A general and formal methodology for designing stable nonlinear fuzzy control systems. *IEEE Transactions on Fuzzy Systems*, 17(5):1081–1091.

Barragán Piña, A. J. (2009). *Síntesis de sistemas de control borroso estables por diseño*. ISBN 978-84-92944-72-9. University of Huelva. http://uhu.es/antonio.barragan/thesis

"Fuzzy Logic Tools Reference Manual v 1.0"
se acabó de imprimir el 23 de Mayo de 2012
día de la festividad de San Basileo en los
talleres de Imprenta Bonanza, S. L. y
estando al cuidado de la edición el
Servicio de Publicaciones de
la Universidad de
Huelva