ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG
INSTITUT FÜR INFORMATIK
Lehrstuhl für Mustererkennung und Bildverarbeitung

Presto-Box 1.1
Library Documentation

Internal Report 2/03

Bernard Haasdonk, Bhaskara Reddy Poluru, Alexandra Teynor

November 2003

This work was funded by the *Bundesministerium für Bildung und Forschung*, grant number 08NM074C

Presto-Box 1.1
Library Documentation

Bernard Haasdonk, Bhaskara Reddy Poluru, Alexandra Teynor
Computer Science Department
Albert-Ludwigs-University Freiburg
79110 Freiburg, Germany
haasdonk,poluru,teynor@informatik.uni-freiburg.de

November 12, 2003

Abstract

The library *Presto-Box* is presented, which is a collection of functions and demos dealing with pattern recognition. It aims at providing an experimental toolbox for interactive exploration of basic concepts which are presented in a corresponding lecture. This document serves as introduction and documentation for students, supervisors and other users. Additionally the complete function list is given, which can be used as a reference manual.

1 Introduction

University teaching by giving traditional pure oral or black-board lectures is not suited for all fields of science, especially not for natural end engineering sciences. Instead of this one-way method often a second reflection phase is provided for students by theoretical or practical exercises. This also applies to the lecture *pattern recognition* which is taught every winter at the university of Freiburg. It is an obligatory lecture for 3rd year students of computer science, cf.

`http://lmb.informatik.uni-freiburg.de/lectures/mustererkennung`.

This toolbox targets at representing the different algorithms presented in that lecture. On one hand it is basis for the exercise component of the lecture by presenting a pool of available functions that can be applied/modified in the exercise tasks. On the other hand the toolbox provides an individual experimental component. Students can apply algorithms and play with interactive demos, which can deepen the insight into the concepts.

This toolbox is a by-product of the project *Universitärer Lehrverbund Informatik* (ULI) during the period of 2001-2003, cf. `www.uli-campus.de`. It was a joint project of

18 computer science groups in Germany and Switzerland. The aim was to develop web-based multimedia-lectures for enabling time- and place-independent studying. The main contribution of our sub-project is the collection of lecture-recordings [1]. It represents the complete material of the course, and can be used with *Presto-Box* for self-study.

The toolbox is based on the scientific computing programming language *Scilab*, comparable to the commercial product *MATLAB*. It is free software which is available for various platforms at `http://scilabsoft.inria.fr/`. Additionally, support is provided, several books and articles are dealing with the topic, the package is under continuous further development and an increasing number of user-provided toolboxes are available. These were the reasons for the choice of Scilab. Similar as Scilab itself being free, *Presto-Box* is released under the *GNU Lesser General Public Licence (LGPL)* copying policy, see

`http://www.gnu.org/copyleft/lesser.html`.

We do not require much knowledge of Scilab in the following except knowing how to install resp. start it and that it is an interpeter language providing a command window that accepts commands. The most important of these are the **help** and **apropos** commands which enable exploration of the language and extent of its functionality. For details see the introduction and documentations on the Scilab-site.

The structure of this document is the following. Most important is the next section, which gives an introduction into the installation, usage and functionality of the toolbox. It can e.g. be read while reproducing the steps at a computer and is meant for general first-time usage. The remaining part of the document consists of a complete function list and detailed descriptions which can serve as a reference manual.

2 User's Guide

We now start with the most important aspects for using the library. First this is the installation procedure, followed by an explanation of the toolbox's structure and an initial contact with the toolbox by presentation of the included demonstrations. We end with some information on the available support.

2.1 Installation

The toolbox is platform-independent and only requires a working version of Scilab 2.6 or higher. The package however only has been tested with Scilab versions ranging from 2.6 to 2.7.2 on Windows XP, Windows NT and Linux (Debian distribution). On other platforms no serious problems are expected, as the toolbox does not make use of platform specific aspects. The installation is performed by downloading the archive from the *Presto-Box* site [3] or the "contributions"-section on the Scilab-Homepage `http://scilabsoft.inria.fr/`. Extraction of the package generates a subfolder `presto-box`. Among others this contains two installation scripts. (See `README.txt` for details on further files.)

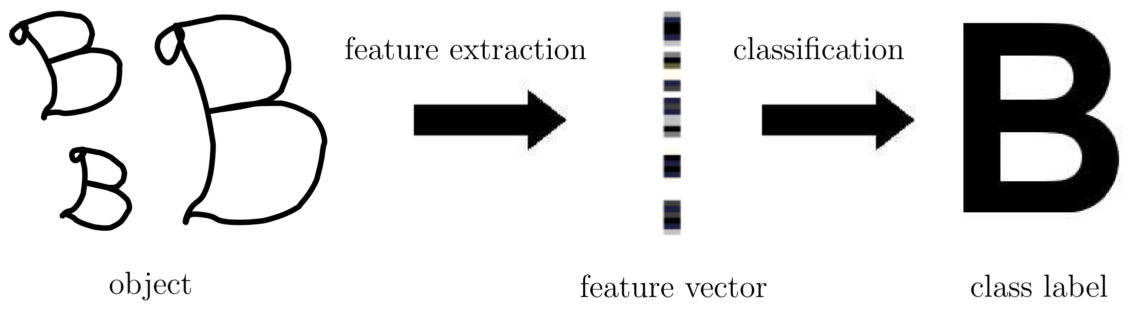


Figure 1: Illustration of a typical pattern recognition process.

`builder.sce` This is a script for building the library. It is to be called once by **exec builder.sce** within Scilab. It creates `.bin` files for all source files ending with `.sci` in the `macros`-subtree. Additionally `lib` and `names` are generated for each of the directories. Each time `builder.sce` is executed it checks for modifications/changes in `.sci` files and updates `.bin` files if it finds any.

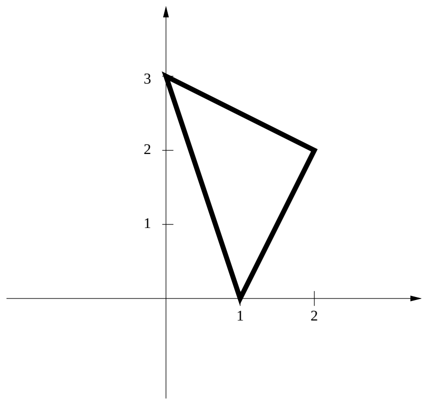
`loader.sce` This is a script for loading the built library. This has to be called at every start of Scilab by **exec loader.sce**. Alternatively the call of this can be put in a `.scilab` file in the user's home-directory. It reads the `names` and `lib` files in the sub-directories of `macros` and generates one library-variable for each directory, e.g. *basic_lib* for the functions in the subdirectory `basic`.

After execution of these two files, *Presto-Box* is ready to be used. This can be checked by pressing the *Demos* or *Help* button and finding a corresponding *Presto-Box* entry. If this should not work, see Section 2.4 for additional hints on usage.

2.2 Structure of the Toolbox

We recall the basic components of a pattern recognition system, which naturally leads to the structure of *Presto-Box* . For the following italic notions we refer to Figure 1. For details on this very interesting field we recommend excellent standard text-books like [2, 4].

The typical pattern recognition task starts with some type of *objects* (e.g. handwritten letters as digitized point sequences) and some finite set of target classes (e.g. the 26 classes of letters). The goal is to have a system that assigns an estimated *class label* to any formerly unseen object in the best possible way. Such a system can be separated in (at least) two separate modules. The first is the so called *feature extraction* stage, where an arbitrary structured object is converted in a vector of numerical values, the *features.* This so called *feature vector* is a simple object that represents the original object. Of course there are uncountable ways of obtaining such a feature vector, however there are some properties that are favorable. The first property is good *separability* (or in the best case *completeness*) of the features. This basically means, that objects of different classes should not result in the same feature vector as they could not be discriminated based



p1 = [1 2 0 ; 0 2 3]   cp1 = [1 ; 2+2i ; 3i]

p2 = [2 0 1 ; 2 3 0]   cp2 = [2+2i ; 3i ; 1]

Figure 2: Illustration of a polygon and different representations in Scilab notation.

on their features. The second property is that the feature extraction should take care of known *invariances* with respect to certain *transformations*. This means that in certain problems the class membership of an object is independent of certain transformations of the object. e.g. scaling or rotation of an A again results in an object that belongs to class A. This knowledge should be incorporated in the feature extraction stage such that transformed objects of the same class result in identical feature vectors. In this case the features are called *invariant*. If a good feature representation of the object is achieved, these are fed into a component called *classifier* which produces an estimated class label. In order to result in a good estimate, such classifiers often need a learning stage. This is often based on a set of objects with known labels, the so called training set. This decomposition was basis for the design of *Presto-Box* .

Object Types and Basic Transformations

We focussed on 3 types of objects and corresponding simple transformations. Details for the Presto-Box commands are also provided by the commands **help** and **apropos** or can be found in Section 3.

Polygons: These are the natural "digital" representation of boundaries of 2d-real world objects. For convenience *Presto-Box* supports two kinds of polygon representations. The first is a matrix representation as a $2 \times n$ matrix with real values. Each column represents the two coordinates of a vertex of the polygon. Points that correspond to neighbouring columns are connected. The first column and last column are also connected. The second representation of polygons is a complex $n \times 1$ vector. Similarly every entry corresponds to a vertex in the plane which is connected to the next/previous vertex.

So a cyclic shift of a matrix or vector representing a polygon results in an equivalent representation of the polygon, cf. Figure 2. The routines **polygon** and **cpolygon** convert between the complex vector/real matrix representation of a polygon.

The following functions differ between the required polygon types. For details see the function list in Section 3. Beside explicitly specifying the matrix/vector by entering coordinates, we provide a function **polstar** for generating a default polygon representing an arbitrary shaped star, and a function **inputpol** for interactive drawing of a polygon. The first supported polygon transformation is coarsening, i.e. reducing superfluous points, by the command **coarsenpol**. Further similarity transformations are provided by **simtrans**. These consist of angle-preserving transformations such as shifts, rotations and scalings. For visualizing polygons one can make use of **plotpol** plotting single polygons or **plotpolseq** for simultaneously plotting whole polygon sequences with individual titles.

coarsenpol	coarsens (removes points of) a polygon with respect to a threshold angle
cpolygon	converts a real-valued 2*n matrix to a complex-valued vector
inputpol	generates a polygon interactively (by mouse clicks or dragging)
plotpol	plots a polygon represented by a 2*n real matrix
plotpolseq	plots the given list of polygons in sequence
polstar	generation of a star-shaped origin-centered polygon.
polygon	converts a complex-valued vector to a real-valued matrix
simtrans	performs simple similarity-transformation of a polygon.

Vectors: The second object type is the class of general real valued vectors. These are for example the digital representation of time signals. During sampling a periodic signal with different starting times, the resulting vectors are cyclically shifted. Consequently a useful transformation is (cyclic) shift of such vectors, which can be performed after calling **cyclmat**. Other simple transformations like scaling, shifting the mean value etc. are simple vector operations supported as standard Scilab operations.

Matrices: These are digital representation of grey-value objects, e.g. an image of an object taken by a camera. Again simple transformations like intensity scaling, shifting the mean etc. are standard Scilab commands. More interesting transformations are rotations or translations of the image. These are provided by **cyclrot** resp. **cycltrans** which perform a cyclic rotation resp. translation of an image. A fast version of cyclic translation is **fcycltrans**.

cyclmat	implementation of cyclic translation matrix of dimension n*n
cyclrot	implements cyclic rotation of a matrix
cycltrans	implementation of cyclic translation for matrices.
fcycltrans	fast implementation of cyclic translation for matrices.

Feature Extraction

Polygons: Useful feature extraction routines for polygons are simple ones like polygon area **polarea**, length of the polygon boundary **polboundary**, center of gravity **polcog** or line center of gravity **pollinecog**. The latter is the center of gravity after the boundary (instead of the area) of the polygon is uniformly allocated with mass. More sophisticated features are complex Fourier-coefficients by **computeFc**. Based on this the rotational symmetry degree can be detected by **detect_symmetry** or the pose-invariant and complete features called *Fourier-descriptors* can be computed by **computeFd**. The Fourier-coefficients/descriptors have the nice property, that the original polygon can be (approximately) recovered (up to a similarity transformation). This is provided by **Fsynthesis**.

computeFc	Fourier-coefficients of closed polygons in the complex plane
computeFd	Fourier-descriptors of closed polygons in the complex plane
detect_symmetry	detection of rotational symmetry degree of a complex polygon
Fsynthesis	Fourier-synthesis of a polygon using its Fourier-coefficients
polarea	area of a real polygon
polboundary	length of boundary of a real polygon
polcog	center of gravity of a real polygon
pollinecog	line center of gravity of a real polygon

Vectors: Useful feature extraction methods for vectors are provided by Scilab itself, e.g. mean-calculations, min/max etc. More sophisticated, translation-invariant feature extraction can be performed by the class of fast transformations $\mathbb{C}T$ which are provided by **CT**, **RT**, **QT** and **MT**.

Matrices: Simple feature extraction operations for matrices are again available as standard Scilab-commands like **mean**, **min** or **max**. Nontrivial translation-invariant feature extraction is performed by the class of fast transformations $\mathbb{C}T_{2D}$. These are provided by **CT_SZ**, **RT_SZ**,**MT_SZ**,**QT_SZ**, and the corresponding modifications with suffix **_ZS**, **_DI**.

CT	general translation-invariant CT-transformation of vectors
CT_DI	general 2d translation-invariant CT_DI-transformation of matrices
CT_SZ	general 2d-translation-invariant CT_SZ-transformation of matrices
CT_ZS	general 2d-translation-invariant CT_ZS-transformation of matrices
MT	1d-translation-invariant CT-transformation using max and min.
MT_DI	2d-translation-invariant CT_DI-Transformation using max and min
MT_SZ	2d-translation-invariant CT_SZ-transformation using max and min.
MT_ZS	2d-translation-invariant CT_ZS-transformation using max and min.
QT	1d-translation-invariant CT-transformation using + and squared-difference
QT_DI	2d-translation-invariant CT_DI-transformation using + and squared-difference
QT_SZ	2d-translation-invariant CT_SZ-transformation using + and squared-difference
QT_ZS	2d-translation-invariant CT_ZS-transformation using + and squared-difference
RT	1d-translation-invariant CT-transformation using + and difference
RT_DI	2d-translation-invariant CT_DI-transformation using + and difference
RT_SZ	2d-translation-invariant CT_SZ-transformation using + and difference
RT_ZS	2d-translation-invariant CT_ZS-transformation using + and difference

These object types and related functions represent the first stage of the pattern-recognition chain, which was abstracting concrete objects in numerical valued feature vectors.

Classification

The following stage of learning from these vector representations is implemented by the block of functions dealing with classifiers. The classifiers all follow a common syntax which is

$$\textbf{[cl,cert] = some\_class(v,...)}$$

The name **some_class** is to be replaced by the according classifier name. A classifier performs estimation of class labels for a set of vectors. The vectors to be classified are given columnwise in the matrix **v**. Every useful classifier needs to have some further parameters specifying its behaviour, e.g. parameters obtained from learning on training observations. These are passed in the variable argument list ... The result of the classification is a vector of integer class labels in the result-vector **cl**, each entry corresponds to the predicted class label for the corresponding input-vector. Additionally many classifiers return a vector **cert** indicating the certainty with which the classification is correct. Some classifiers only accept 2-dimensional input-data, these have the suffix **_2d** in their function names, e.g. **dumb_class_2d**. The first useful classifier is the nearest-neighbour classifier **nneigh_class_2d**. It needs a set of labeled training vectors. Classification of a new example is done by taking the label of the closest training point. The corresponding function for arbitrary dimensional input-data is **nneigh_class**.

A very good parametric classification method consists of estimating normal distributions based on training data, and taking a *Bayes decision* for classification of new points. The estimation of mean and covariance matrices can be performed by **class_statistics**, these can be plugged in the classifier **bayes_class_2d** for solving a 2 dimensional 2-class problem, or in **bayes_class** for general multiclass problems in arbitrary dimensions. For experimental purposes, normally distributed data can be generated by **randnormal**.

A further well known parametric approach is the polynomial classifier. This classifier is based on learning a polynomial regression function from the training data and using this function for classification. The whole procedure is performed by **polynom_class**. Additionally the training and prediction steps of the regression can be performed separately by **regress_matrix** and **polynom_regress**. For visualization purposes one can make use of **visclass_2d**, which draws classification boundaries in 2 dimensions. The dimension-independent version **visclass** can draw 2d-cuts in the feature space and the resulting classification regions. The included interactive GUI allows very easy exploration of the classification behaviour of a certain classifier.

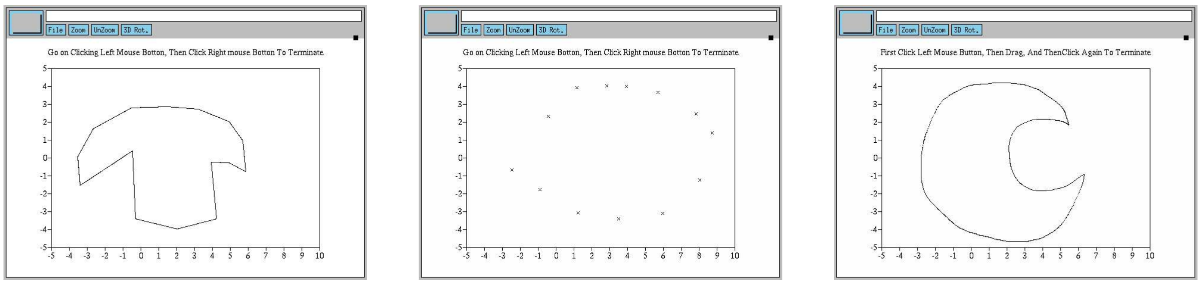

File
Zoom
UnZoom
3D Rot.
Go on Clicking Left Mouse Botton, Then Click Right mouse Botton To Terminate
Go on Clicking Left Mouse Botton, Then Click Right mouse Botton To Terminate
First Click Left Mouse Button, Then Drag, And ThenClick Again To Terminate

Figure 3: Illustration of the **inputpol** function: *click*, *marks* and *drag*-mode.

bayes_class	implements the bayes classification in multi-dimensions.
bayes_class_2d	implements the bayes classification in 2 dimensions.
class_statistics	estimates the statistics of the given class number
dumb_class_2d	implements a demo classification in 2 dimensions.
nneigh_class	implements the nearest neighbour classification in multi-dimensions
nneigh_class_2d	implements the nearest neighbour classification in 2 dimensions
polynom_class	training of polynomial classifier and classification.
polynom_regress	evaluate polynomial (vectorial) regression function
randnormal	generation of arbitrary dimensional normally distributed random data
regress_matrix	calculates polynomial regression matrix for vectorial function y=f(x).
visclass	visualization of classification regions in multi dimensions
visclass_2d	visualization of classification regions in 2 dimensions

Additionally the toolbox provides some basic mathematical and data-manipulation routines, which are mainly auxiliary functions for the presented ones.

2.3 Demos

In this section we describe in detail the provided demos. These can be either invoked by pressing the *Demos* Button in the Scilab window and choosing *Presto-Box Pattern Recognition Demos* or by directly calling **patrec_demos**. Currently only few demos are available and not all contents of the toolbox are covered, but it gives a first impression of the functionality and extent.

Polygon Generation - Interactive

The demo called **inputpol** demonstrates the corresponding function and the different modes of interactive input of a polygon, cf. Figure 3. Basically the possibilities are *clicking* of the vertices which are immediately connected. Alternatively the connection in drawing can be turned off and the points are only indicated by *marks*, e.g. input of single data points can be realized with this. The last possibility is input by *dragging*, where each hit pixel is taken as a polygon vertex.

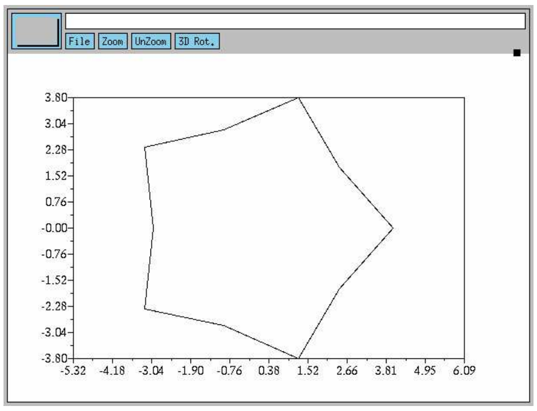
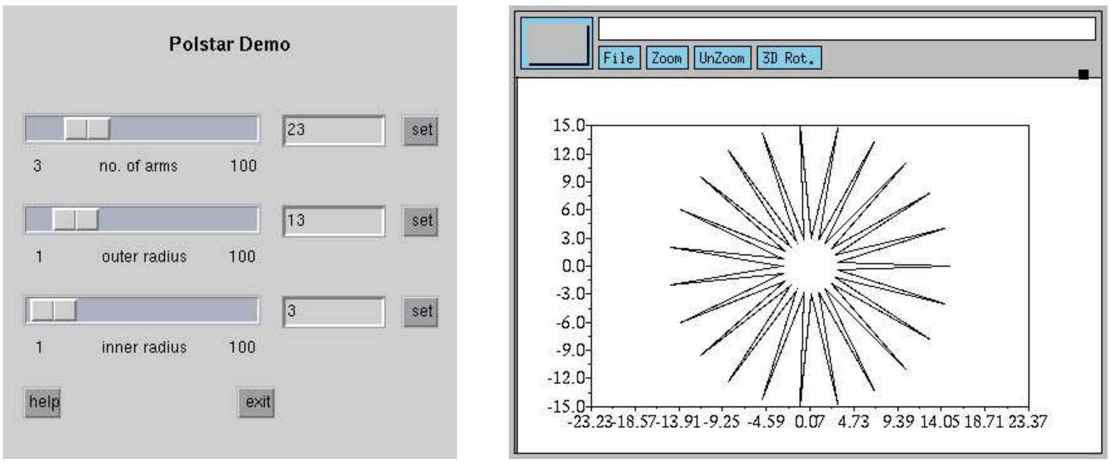


Figure 4: Illustration of the **plotpol** function.



3 no. of arms 100: 23

1 outer radius 100: 13

1 inner radius 100: 3

Figure 5: Illustration of **polstar_demo**.

Polygon Visualization - Single Polygon

This is not a real interactive demo but merely a call of the polygon visualization routine **plotpol** with some default polygon, see Figure 4. The polygon is closed by connecting the first and last vertex and coordinate axes are added.

Polygon Generation - Polygon Star

The function **polstar_demo** is the first to demonstrate the interactive capabilities of Scilab. It demonstrates the result of the function **polstar**. This is a function generating a default polygon representing a star with variable number of arms and variable inner and outer radius. The demo consists of two windows, one for the interactive controls and one for the graphical output. The demo allows to change the star's parameters by using sliders or by explicitly entering the values in corresponding text-fields. In the latter case the change has to be confirmed by pushing the corresponding *set*-button. Figure 5 displays the windows.

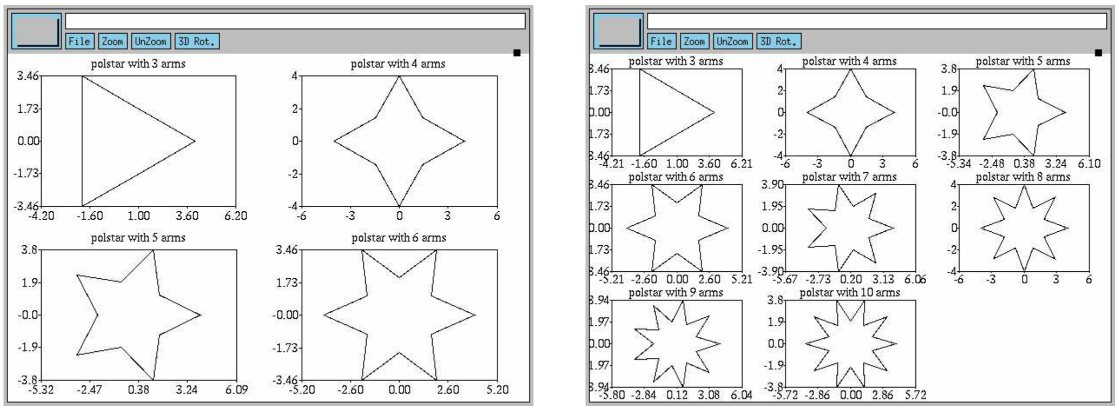


Figure 6: Illustration of the **plotpolseq** function.

Polygon Visualization - Multiple Polygons

The demo **plotpolseq** demonstrates the visualization of multiple polygons within one graphical window, each polygon headed by a separate title. The number of polygons can be chosen, the appropriate command is displayed and executed in the Scilab-command window and the result is plotted. Examples can be found in Figure 6.

Fourier-coefficients - Approximation Sequence

The demo **plotFourierseq** is not a real interactive demo but a demonstration of the use of the corresponding function. The command

plotFourierseq(polstar(5,4,3),1,41,5)

is displayed in the Scilab command window and produces the output in Figure 7. It performs a Fourier analysis of the input-polygon given by the first argument and generates a series of Fourier approximations using an increasing number of Fourier-coefficients specified by the remaining function arguments. In the example the approximation sequence uses 1 to 41 coefficients increasing by 5 in each approximation.

Fourier-coefficients - Interactive Exploration

The function **Fc_demo** again is a real interactive demo for exploration and modification of Fourier-coefficients of complex contours. The demo consists of at least 2 windows, one for the interactive controls, the other for the graphical output of the Fourier reconstruction of the current Fourier-coefficients. By choosing the option "showspectrum" an additional window is generated which plots the current Fourier-spectrum, i.e. the absolute values of the complex Fourier-coefficients, cf. Figure 8.

The Fourier-coefficients **c_i** of a complex contour can be selected by choosing the index **i** with the upper slider, the absolute value and the angle of the chosen **c_i** are then displayed and are editable with the lower two sliders. Beside the sliders the values can be entered explicitly. After pressing the *set* button, the value will be set. If some

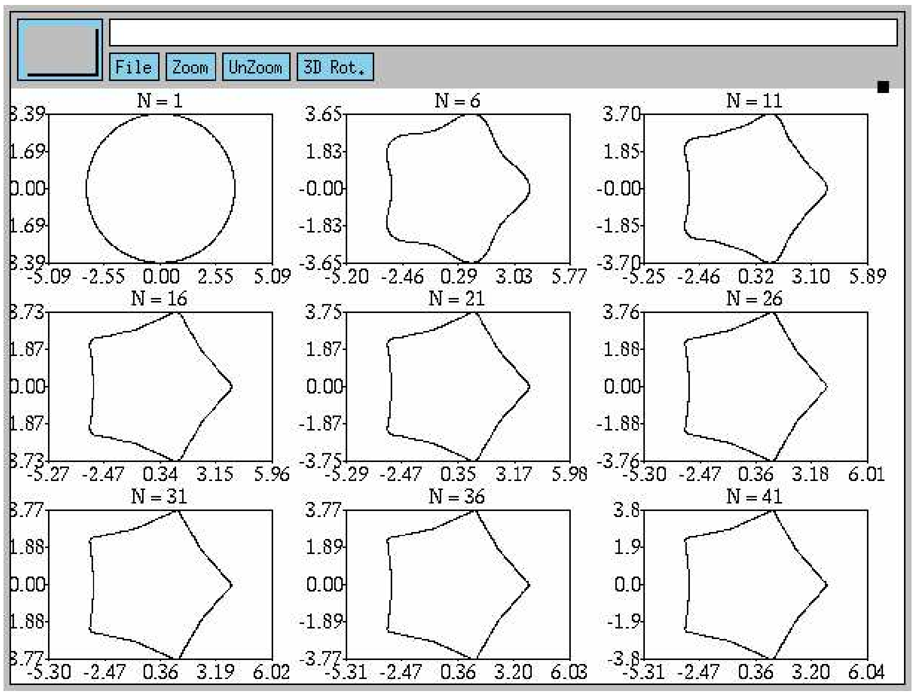
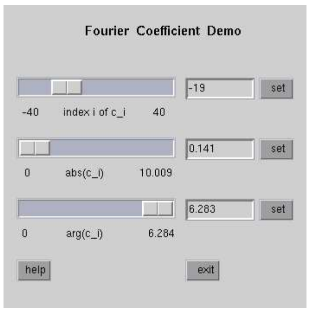


Figure 7: Illustration of the **plotFourierseq** function.



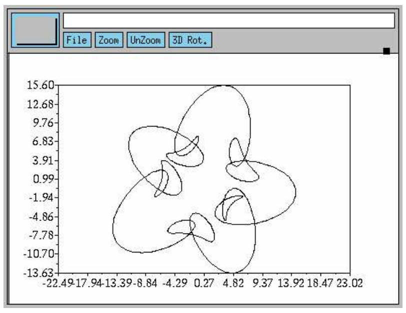
Fourier Coefficient Demo

-19 set
-40 index i of c_i 40

0.141 set
0 abs(c_i) 10.009

6.283 set
0 arg(c_i) 6.284

help exit



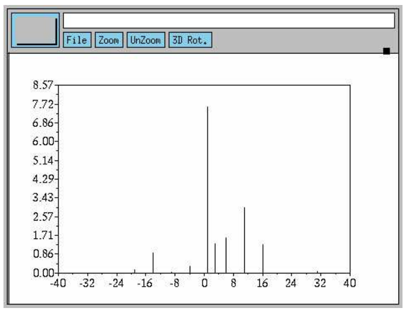


Figure 8: Illustration of different windows of **Fc_demo**.

Fourier-coefficient is modified, a Fourier-synthesis is calculated and the resulting contour is displayed.

By calling **Fc_demo** from the command line, an arbitrary polygon can be passed as argument, which then can be modified.

Some insights are easily possible with this demo, e.g. the interpretation of **c_0** as the line center-of-gravity. Modifying this value directly results in the corresponding shift of the object, but does not change the contour. The rotational symmetry degree s can nicely be found in the spectrum by the repeating regions of length $s-1$ with vanishing coefficients. Changing one coefficient in these 0-regions immediately decreases the initial rotational symmetry degree, changing nonzero coefficients maintains the rotational symmetry. The (quadratic) decay in the power-spectrum with increasing coefficient-index also is directly visible.

Polygon Classification

This demonstration illustrates the complete pattern recognition chain as illustrated in Figure 1. The objects to be classified are hand-drawn contours. Various feature-extraction methods can be combined, most of them are pose-invariant, such that rotations or translations of the input polygon do not change the classification result. Classification is performed in a very simple way by computing the feature-vector distances between the object to 4 reference objects L, T, F and E and choosing the object with the smallest distance.

The demo makes use of 3 windows as depicted in Figure 9. The first windows is the window with the interactive controls. Here the features to be used for classification can be selected by the checkboxes. The polygon to be classified can be entered after pressing the button *new polygon*, this opens a second window. The classification of the last entered polygon is performed by the *classify polygon* button.

After classification the reference polygons are plotted in a third window in the order of increasing distance to the newly generated polygon. The (squared) distance of the feature-vectors is given above each reference polygon. The example demonstrates that the hand drawn letter T is correctly and clearly classified as T by using the Fourier-descriptors, because they are invariant with respect to rotation, scaling and translation.

Aspects that can be understood by this demo are:

1. Area and boundary are not scale-invariant and not suited for the given reference set, as T and L have identical values.

2. Compactness is a scale-invariant feature however not suited as T and L have identical values.

3. Fourier-coefficients are not scale- and rotation-invariant, however very discriminative.

4. Fourier-descriptors are fully similarity invariant and very discriminative.

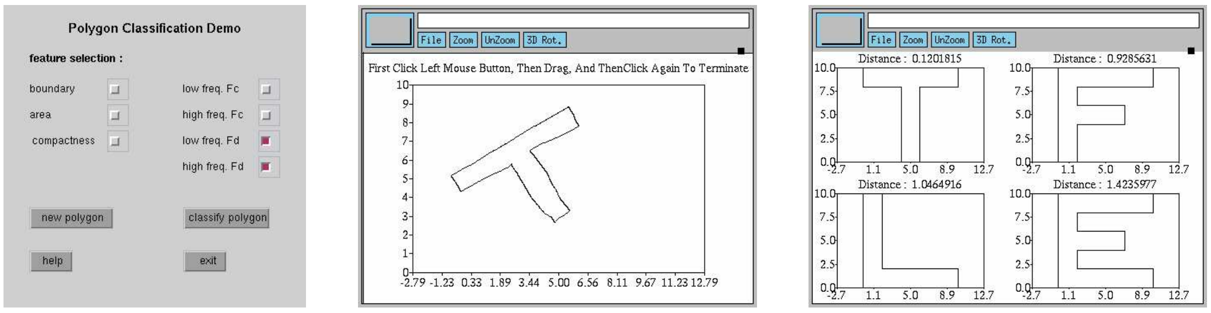


Figure 9: Illustration of the different windows of **polclass_demo**.

5. The Fourier-coefficients and therefore the Fourier-descriptors are depending on the orientation of the polygon. If a polygon is drawn with positive orientation (=counterclockwise), the comparison with the reference polygons will be informative. However, if a polygon is negatively oriented, the difference to the (positively oriented) reference polygons will not yield useful results.

Classifier Comparison

The demo **plotclassseq** is missing real interactive components, but demonstrates the use of the corresponding command. It visualizes the classification regions of different classifiers on given training data. By this the differences between the classifiers can be illustrated. Figure 10 gives the default result. Given some normally distributed two-class data in 2 dimensions plotted in the upper left corner, a series of classifiers is trained on the data. The illustrated classifiers are the bayes-classifier, nearest-neighbour classifier and polynomial classifiers of increasing degree 1, 2, 5 and 8.

In case of normal distributed data and plenty of training points the Bayes classifier definitely is a good choice, as can be seen in the nice class boundaries. The nearest neighbour classifier seems to overfit too much to the data. The linear classifier is too simple to capture the correct boundary, the quadratic classifier seems to do a good job. One argument for this might be that it is of 2nd order as the Bayesian class boundaries are. Polynomial classifiers of degrees 5 and 8 seem increasingly overfitted and not suited for the given problem. By varying the input data however other distributions can be constructed, where these higher order classifiers are better suited than the Bayes classifier or the lower polynomial degrees.

Classifier Visualization

The last demo **visclass('gui')** illustrates the use of the function for interactive setting of visualization options for a given classifier. Therefore the first choice is deciding which classifier should be visualized. The demo opens 2 windows, again one with the interactive controls and the second for the graphical output, which can be switched to 3d or 2d mode, see Figure 11. The graphics contains the classification results on an arbitrary axis-parallel

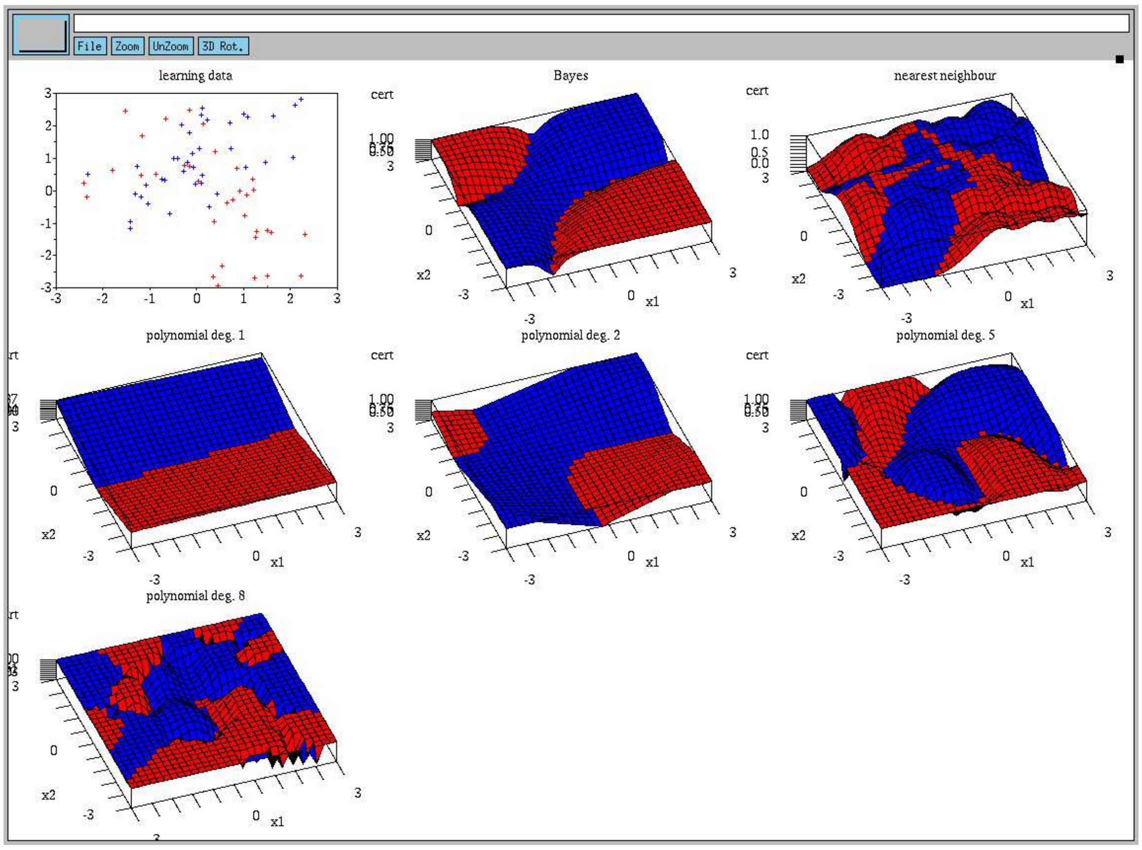
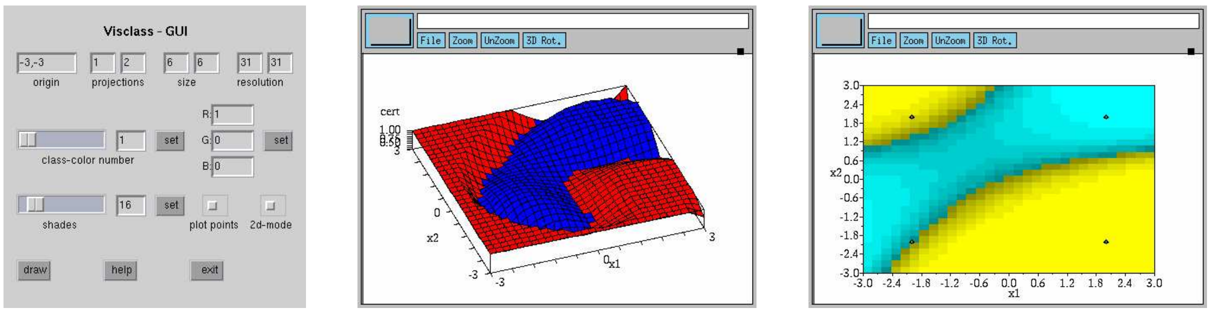


Figure 10: Illustration of **plotclassseq**.



origin -3,-3
projections 1 2
size 6 6
resolution 31 31
class-color number 1 set
R: 1
G: 0
B: 0
set
shades 16 set
plot points
2d-mode

Figure 11: Illustration of the different windows of **visclass-demo**.

rectangle in arbitrary n-dimensional feature-space. The rectangle is specified by an n-dimensional vector "origin" the 2 "projection"-directions (integers between 1 and n) and the "size" of the rectangle in these both directions starting from the origin. "resolution" defines the direction-wise number of test-points extracted from the rectangle.

As drawing is quite time consuming a redraw is only invoked after pressing *draw*. This will perform a new classification of the test-points defined by the specified grid.

"2d-mode" results in a color-shaded rectangle: the classes are represented by different colors, the certainties by different shading-intensities. In non-"2d-mode" the results are displayed by a colored function-graph in 3d: the classes are again represented by different colors, the certainties as the height of the function-graph.

The class-colors can be edited by choosing a "class-color number" either by slider or the corresponding edit-field. After pressing the *set* button the current color will be displayed by its rgb-components and can be edited. After pressing the color-*set* button, the chosen color is saved in the colormap to be used for the next drawing. The number of "shades" can be adjusted by a corresponding edit-field and slider. If the checkbox "plot points" is set and a list of points is passed by the argument list, these points are displayed additionally in the 2d-mode, e.g. the right plot in Figure 11.

2.4 Support and Development

We finish the general part for "first reading" with this section where we comment on support and development aspects.

Additional Hints on Usage

If the user wants to access this toolbox from various operating systems or from different Scilab versions, then before each start of Scilab, the binaries of the *Presto-Box* functions have to be regenerated. A call of `builder.sce` is *not sufficient* as this only rebuilds binaries for sources that are newer than the corresponding binaries. Solution: delete all `*.bin`, `names` and `lib` files in the subdirectories of `macros`, then execute `builder.sce` as described in the installation section. Another solution is to generate one separate presto-box directory per platform and Scilab-version.

Even if the user has generated a `.scilab` file for loading *Presto-Box* at Scilab startup, some operating-system/Scilab-versions do not read the file properly. In this case execution of the startup-file can be forced by command line options: `scilab -f .scilab`

Contact

The toolbox is in continuous further development. The basic source for new versions is the site [3]. A copy of the latest version will also be available at the "contributions" section of the Scilab home page

`http://scilabsoft.inria.fr/`.

We maintain a buglist on the Presto-Box website which currently only contains few known bugs. Users are requested to kindly help us by providing feedback with possible bugs and errors occurred during usage. Please specify your operating system and Scilab-version in bug-reports for making them reproducible.

We welcome such bug-reports but also further comments, contributions or questions which can be submitted to

`presto-box@informatik.uni-freiburg.de`

Future Extensions

Development of the toolbox will kept organized at the university of Freiburg, where it is maintained in a cvs-repository. Beside fixing known bugs and realizing minor improvements, future releases will develop towards these directions:

- interactive demo for the 2-dimensional translation invariant transformations CT
- implementation of demonstration for the Karhunen-Loeve transformation
- inclusion/adaption of the routines which evolved from the course during winter 02/03 and 03/04, e.g. Mahalanobis classification, model-selection routines.
- inclusion of an image-processing toolbox and default image data
- boundary extraction of images (matrix yields a polygon.)
- implementation of perceptron training and classification
- implementation of support vector machine training and classification
- implementation of neural net training and classification

3 Function List

This section presents the complete list of functions in the toolbox and detailed descriptions. These descriptions are parts of the help-texts which are obtained by **help command_name**. Details on using the demos can be found in Section 2.3.

3.1 Overview

CT	general translation-invariant CT-transformation of vectors
CT_DI	general 2d translation-invariant CT_DI-transformation of matrices
CT_SZ	general 2d-translation-invariant CT_SZ-transformation of matrices
CT_ZS	general 2d-translation-invariant CT_ZS-transformation of matrices
Fc_demo	a GUI for demonstration of Fourier-coefficients of a polygon
Fsynthesis	Fourier-synthesis of a polygon using its Fourier-coefficients

MT	1d-translation-invariant CT-transformation using max and min.
MT_DI	2d-translation-invariant CT_DI-Transformation using max and min
MT_SZ	2d-translation-invariant CT_SZ-transformation using max and min.
MT_ZS	2d-translation-invariant CT_ZS-transformation using max and min.
QT	1d-translation-invariant CT-transformation using + and squared-difference
QT_DI	2d-translation-invariant CT_DI-transformation using + and squared-difference
QT_SZ	2d-translation-invariant CT_SZ-transformation using + and squared-difference
QT_ZS	2d-translation-invariant CT_ZS-transformation using + and squared-difference
RT	1d-translation-invariant CT-transformation using + and difference
RT_DI	2d-translation-invariant CT_DI-transformation using + and difference
RT_SZ	2d-translation-invariant CT_SZ-transformation using + and difference
RT_ZS	2d-translation-invariant CT_ZS-transformation using + and difference
bayes_class	implements the bayes classification in multi-dimensions.
bayes_class_2d	implements the bayes classification in 2 dimensions.
binom_coeff	calculation of binomial-coefficient.
class_statistics	estimates the statistics of the given class number
coarsenpol	coarsens (removes points of) a polygon with respect to a threshold angle
computeFc	Fourier-coefficients of closed polygons in the complex plane
computeFd	Fourier-descriptors of closed polygons in the complex plane
cpolygon	converts a real-valued 2*n matrix to a complex-valued vector
cyclmat	implementation of cyclic translation matrix of dimension n*n
cyclrot	implements cyclic rotation of a matrix
cycltrans	implementation of cyclic translation for matrices.
detect_symmetry	detection of rotational symmetry degree of a complex polygon
dumb_class_2d	implements a demo classification in 2 dimensions.
fcycltrans	fast implementation of cyclic translation for matrices.
getmatrix	gets the data stored in an object figure (as a matrix)
inputpol	generates a polygon interactively (by mouse clicks or dragging)
monomvec	calculates the power-substitution-vector, the vector of monomials
new_window	gets the number of the next possible new graphics window
nneigh_class	implements the nearest neighbour classification in multi-dimensions
nneigh_class_2d	implements the nearest neighbour classification in 2 dimensions
patrec_demos	a GUI showing a list of all the Presto-Box demos
plotFourierseq	plots Fourier-sequence of a polygon for given range and step values
plotFs	plotting of the Fourier-spectrum (mainly used in Fc_demo)
plotclassseq	plots the effect of different classifiers on given learning data
plotpol	plots a polygon represented by a 2*n real matrix
plotpolseq	plots the given list of polygons in sequence
polarea	area of a real polygon
polboundary	length of boundary of a real polygon
polclass_demo	a GUI for interactive polygon classification
polcog	center of gravity of a real polygon
pollinecog	line center of gravity of a real polygon

polstar	generation of a star-shaped origin-centered polygon.
polstar_demo	a GUI for interactive polygon generation
polygon	converts a complex-valued vector to a real-valued matrix
polynom_class	training of polynomial classifier and classification.
polynom_regress	evaluate polynomial (vectorial) regression function
pw_absdiff	elementwise absolute difference.
pw_add	elementwise addition.
pw_diffsquare	elementwise square of difference.
pw_max	elementwise maximum.
pw_min	elementwise minimum.
randnormal	generation of arbitrary dimensional normally distributed random data
regress_matrix	calculates polynomial regression matrix for vectorial function y=f(x).
repmat	constructs a large matrix of n*m blocks each consisting of "mat"
setmatrix	stores the data (an integer matrix) in an object figure.
simtrans	performs simple similarity-transformation of a polygon.
visclass	visualization of classification regions in multi dimensions
visclass_2d	visualization of classification regions in 2 dimensions
w_DI	performs w_DI-transformation of a matrix

3.2 Detailed Descriptions

CT - general translation-invariant CT-transformation of vectors

CALLING SEQUENCE
res = CT(f1, f2, x)

PARAMETERS

f1,f2: commutative functions with calling syntax r = f1(a,b) where a, b & r are all equally sized matrices, r containing the results of elementwise dyadic operations.

x: vector/matrix each column of which will be processed by the CT.

res: vector/matrix of the columnwise results of the transformation of x.

DESCRIPTION
CT performs CT transformation of the given column-vector x using the commutative-functions f1 and f2 (if x is a matrix then each of its column-vectors will be processed simultaneously).

CT_DI - general 2d translation-invariant CT_DI-transformation of matrices

CALLING SEQUENCE
res = CT_DI(f1, f2, x)

PARAMETERS

f1,f2: commutative functions with calling syntax r = f1(a,b) where a, b & r are equally sized matrices, r containing the results of elementwise dyadic operations.

x: matrix to be transformed.

res: matrix of the CT_DI transformation of x.

DESCRIPTION

CT_DI (is a matrix transformation) performs CT_DI transformation, i.e. first w_DI transformation then CT_ZS transformation, using the commutative functions f1 and f2 on the given matrix x.

CT_SZ - general 2d-translation-invariant CT_SZ-transformation of matrices

CALLING SEQUENCE

res = CT_SZ(f1, f2, x)

PARAMETERS

f1,f2: commutative functions with calling syntax r = f1(a,b) where a, b & r are equally sized matrices, r containing the results of elementwise dyadic operations.

x: matrix to be transformed by CT_SZ.

res: matrix of CT_SZ transformation (of matrix x).

DESCRIPTION

CT_SZ (is a matrix transformation) performs CT_SZ tranformation i.e. first row by row CT-transformation then columnwise CT-transformation, using the commutative-functions f1 and f2 on the given matrix x.

CT_ZS - general 2d-translation-invariant CT_ZS-transformation of matrices

CALLING SEQUENCE

res = CT_ZS(f1, f2, x)

PARAMETERS

f1,f2: commutative functions with calling syntax r = f1(a,b) where a, b & r are equally sized matrices, r containing the results of elementwise dyadic operations.

x: matrix to be processed by the CT_ZS.

res: matrix with the result of the transformation of x.

DESCRIPTION

CT_ZS (is a matrix transformation) performs CT_ZS transformation i.e. first columns are CT-processed and then rows) using the commutative functions f1 & f2 on the given matrix x.

Fc_demo - a GUI for demonstration of Fourier-coefficients of a polygon

CALLING SEQUENCE

Fc_demo() - it can take varargin, as follows: Fc_demo(['setnumber', val], ['setpolygon', polygon], ['showspectrum'])

PARAMETERS

val: highest index of Fourier-coefficient to be used, default value is 20

polygon: 2*n matrix with x and y coordinates of the polygon for which the Fourier-coefficients are to be demonstrated. the default value is polstar(5,4,3).

DESCRIPTION

Fc_demo allows interactive investigation of Fourier-coefficients (Fc) of complex contours. The Fourier-coefficients c_i of a complex contour can be selected by choosing the index i with the upper slider, the absolute value and the angle of the chosen c_i are then displayed and are editable by the lower two sliders. Beside the sliders the values can be entered explicitly. After pressing the "set" button, the value will be set. When some some Fc is modified, a Fourier-synthesis is calculated and the resulting contour is displayed. If the demo is started with the option "showspectrum", the power-spectrum of the current contour is plotted additionally.

Fsynthesis - Fourier-synthesis of a polygon using its Fourier-coefficients

CALLING SEQUENCE

res = Fsynthesis(Fc)

PARAMETERS

Fc: complex Fourier-coefficients, can be obtained by computeFc function.

res: A vector of 1000 complex values representing the approximate polygon.

DESCRIPTION

Fsynthesis synthesises the input complex vector Fc (generally obtained from "computeFc") into 1000 points in the complex plane and outputs the resulting complex vector. It can be drawn by "plotpol" function once if it is converted to a real matrix by using "polygon" function.

MT - 1d-translation-invariant CT-transformation using max and min.

CALLING SEQUENCE

res = MT(x)

PARAMETERS

x: vector/matrix each column of which will be processed by the MT.

res: vector/matrix of the columnwise results of the transformation of x.

DESCRIPTION

MT performs CT transformation using the commutative functions pairwise- maximum and pairwise-minimum on the given column vector x, if x is a matrix then each of its column vectors will be processed simultaneously.

MT_DI - 2d-translation-invariant CT_DI-Transformation using max and min

CALLING SEQUENCE

res = MT_DI(x)

PARAMETERS

x: matrix to be processed by the MT_DI.

res: matrix of the MT_DI transformation of x.

DESCRIPTION

MT_DI (is a matrix transformation) performs CT_DI transformation using the commutative functions pairwise-maximum and pairwise-minimum on the given matrix x, i.e. first w_DI transformation then MT_ZS transformation.

MT_SZ - 2d-translation-invariant CT_SZ-transformation using max and min.

CALLING SEQUENCE

res = MT_SZ(x)

PARAMETERS

x: matrix to be processed by the MT_SZ.

res: matrix of the MT_SZ transformation of x.

DESCRIPTION

MT_SZ (is a matrix transformation) performs CT_SZ transformation i.e. first row by row then columnwise, using the commutative functions pairwise-maximum and pairwise-minimum, on the given matrix x.

MT_ZS - 2d-translation-invariant CT_ZS-transformation using max and min.

CALLING SEQUENCE

res = MT_ZS(x)

PARAMETERS

x: matrix to be processed by the MT_ZS.

res: matrix of the MT_ZS transformation of x.

DESCRIPTION

MT_ZS (is a matrix transformation) performs CT_ZS transformation i.e. first columns are processed and then rows) using the commutative functions pairwise-maximum and pairwise-minimum, on the given matrix x.

QT - 1d-translation-invariant CT-transformation using + and squared-difference

CALLING SEQUENCE

res = QT(x)

PARAMETERS

x: vector/matrix each column of which will be processed by the QT.

res: vector/matrix of the columnwise results of the transformation of x.

DESCRIPTION

QT performs CT transformation using the commutative functions pairwise-addition and squared-difference on the given column vector x, if x is a matrix then each of its column vectors will be processed simultaneously.

QT_DI - 2d-translation-invariant CT_DI-transformation using + and squared-difference

CALLING SEQUENCE

res = QT_DI(x)

PARAMETERS

x: a matrix to be processed by the QT_DI.

res: matrix of the results of the QT_DI transformation of x.

DESCRIPTION

QT_DI (is a matrix transformation) performs CT_DI transformation using the commutative functions pairwise-addition and squared-difference on the given matrix x, i.e. first w_DI transformation then QT_ZS transformation.

QT_SZ - 2d-translation-invariant CT_SZ-transformation using + and squared-difference

CALLING SEQUENCE

res = QT_SZ(x)

PARAMETERS

x: matrix to be processed by the QT_SZ.

res: matrix of the result of the QT_SZ transformation of x.

DESCRIPTION

QT_SZ (is a matrix transformation) performs CT_SZ transformation i.e. first row by row then columnwise, using the commutative functions: pairwise- addition and squared-difference on the given matrix x.

QT_ZS - 2d-translation-invariant CT_ZS-transformation using + and squared-difference

CALLING SEQUENCE

res = QT_ZS(x)

PARAMETERS

x: a matrix to be processed by the QT_ZS.

res: matrix of the result of the QT_ZS transformation of x.

DESCRIPTION

QT_ZS (is a matrix transformation) performs CT_ZS transformation (i.e. first columns are processed and then rows) using the commutative functions pairwise-addition and squared-difference on the given matrix x.

RT - 1d-translation-invariant CT-transformation using + and difference

CALLING SEQUENCE

res = RT(x)

PARAMETERS

x: vector/matrix each column of which will be processed by the RT.

res: vector/matrix of the columnwise results of the transformation of x.

DESCRIPTION

RT performs CT transformation using the commutative functions pairwise-addition and absolute-difference on the given column vector x, if x is a matrix then each of its column vectors will be processed simultaneously.

RT_DI - 2d-translation-invariant CT_DI-transformation using + and difference

CALLING SEQUENCE

res = RT_DI(x)

PARAMETERS

x: a matrix to be processed by the RT_DI.

res: matrix of the result of the RT_DI transformation of x.

DESCRIPTION

RT_DI (is a matrix transformation) performs CT_DI transformation using the commutative functions pairwise-addition and absolute-difference on the given matrix x, i.e. first w_DI transformation then ZS transformation.

RT_SZ - 2d-translation-invariant CT_SZ-transformation using + and difference

CALLING SEQUENCE

res = RT_SZ(x)

PARAMETERS

x: a matrix to be processed by the RT_SZ.

res: matrix of the result of the RT_SZ transformation of x.

DESCRIPTION

RT_SZ (is a matrix transformation) performs CT_SZ tranformation i.e. first row by row then columnwise, using the commutative functions pairwise- addition and absolute-difference on the given matrix x.

RT_ZS - 2d-translation-invariant CT_ZS-transformation using + and difference

CALLING SEQUENCE

res = RT_ZS(x)

PARAMETERS

x: a matrix to be processed by the RT_ZS.

res: matrix of the result of the RT_ZS transformation of x.

DESCRIPTION

RT_ZS (is a matrix transformation) performs CT_ZS transformation i.e. first columns are processed and then rows) using the commutative functions pairwise-addition and absolute-difference on the given matrix x.

bayes_class - implements the bayes classification in multi-dimensions.

CALLING SEQUENCE

[classes,cert] = bayes_class(x, [p1,m1,K1,p2,m2,K2,...]).

PARAMETERS

x: a n1*n2 matrix with n2 columnwise data points to be classified

p1,p2,...: prior probabilities of the classes (sum == 1).

m1,m2,...: mean vectors of the classes (length n1)

K1,K2,...: covariance matrices of the distributions (of size n1*n1)

classes: vector of estimated class memberships (length n2).

cert: vector (length n2) of certainty measures for the classifications, in this case the a-posteriori probabilities given by the input distributions.

DESCRIPTION

bayes_class implements bayes classification of n1-dimensional vectors in case of a multiclass problem with the class conditional probability distributions assumed to be the normal distributions defined by the statistical parameters p1,m1,K1,p2,m2,K2,...

bayes_class_2d - implements the bayes classification in 2 dimensions.

CALLING SEQUENCE

[classes,cert] = bayes_class_2d(x, p1,m1,K1,p2,m2,K2).

PARAMETERS

x: a 2*n matrix with columnwise n data points to be classified

p1,p2: prior probability of the 2 classes (sum == 1).

m1,m2: mean vector of the 2 classes.(of length 2)

K1,K2: covariance matrices of the 2 distributions (of size 2*2).

classes: vector of estimated class memberships(1 or 2) of length n.

cert: vector (length n) of certainty measures for the classifications, in this case the a-posteriori probabilities.

DESCRIPTION

bayes_class_2d implements bayes classification of 2-dimensional vectors in case of a 2-class problem with the class conditional probability distributions assumed to be the normal distributions defined by the statistical parameters p1, m1, K1, p2, m2, K2.

binom_coeff - calculation of binomial-coefficient.

CALLING SEQUENCE

res =binom_coeff(n,k)

PARAMETERS

n: a positive integer

k: a positive integer in the range from 0 to n

res: binomial coefficient

DESCRIPTION

binom_coeff calculates the binomial-coefficient "k out of n" where n is a positive integer and k should be in the range [0,...,n].

class_statistics - estimates the statistics of the given class number

CALLING SEQUENCE

[p,m,k] = class_statistics(learn_data [,labels, class_no])

PARAMETERS

learn_data: a matrix of columnwise learning data points

labels: a vector of length equal to the number of points, default labels are ones.

class_no: number of the class for which the statistics should be generated, default is 1.

p: prior probability of the given (or else default) class number

m: mean vector of the given (or else default) class number

k: covariance matrix of the given (or else default) class number

DESCRIPTION

class_statistics is a function for generating class statistics i.e. converting the learning data and labels for classification into the statistics for e.g. normal distribution estimation.

coarsenpol - coarsens (removes points of) a polygon with respect to a threshold angle

CALLING SEQUENCE
res = coarsenpol(p,t)

PARAMETERS

p: a 2*n matrix representing the vertices of the polygon as its columns

t: threshold angle

res: coarsened polygon resulting from p

DESCRIPTION
coarsenpol removes all the vertices of the given polygon p whose internal angles are more than π-t and outputs the resulting polygon.

computeFc - Fourier-coefficients of closed polygons in the complex plane

CALLING SEQUENCE
res = computeFc(n, cpol)

PARAMETERS

n: maximum index of Fourier-coefficients to be calculated, integer $\geq$1

cpol: vector of complex values representing the polygon.

res: vector of (complex valued) Fourier-coefficients c_i (i ranging from -n, ..., 0, ..., +n)

DESCRIPTION
computeFc takes a complex vector representing a closed polygon and calculates the Fourier-coefficients (in the complex plane). The output can be used in "Fsynthesis", "computeFd" functions.

computeFd - Fourier-descriptors of closed polygons in the complex plane

CALLING SEQUENCE
res = computeFd(q, s, Fc)

PARAMETERS

q: Index of the Fourier-coefficient which will be used for normalization.

s: rotation symmetry of the polygon represented by the Fourier-coefficients,(i.e. distance between indices of non-vanishing coefficients).

Fc: vector of (complex valued) Fourier-coefficients.

DESCRIPTION
computeFd takes Fourier-coefficients of a closed polygon and calculates the similarity invariant complex Fourier-descriptors.

cpolygon - converts a real-valued 2*n matrix to a complex-valued vector

CALLING SEQUENCE
res = cpolygon(pol)

PARAMETERS

pol: a 2*n matrix of real values representing the vertices of a polygon res: a vector (of length n) of complex values

DESCRIPTION

cpolygon converts a 2*n matrix of real values to a complex vector of length n.

cyclmat - implementation of cyclic translation matrix of dimension n*n

CALLING SEQUENCE

res = cyclmat(n)

PARAMETERS

n: dimension of the desired translation matrix (an integer ≥1)

res: cyclic translation matrix, e.g. if n = 4 then res' * [1 2 3 4]' = [2 3 4 1]'

DESCRIPTION

implementation of cyclic translation matrix of dimension n*n. Any vector can be translated cyclically by simple multiplication with the transpose of this resulting matrix.

cyclrot - implements cyclic rotation of a matrix

CALLING SEQUENCE

res = cyclrot(mat, phi)

PARAMETERS

mat: matrix to be rotated

phi: rotation angle, a real value

res: rotated matrix

DESCRIPTION

cyclrot implements the cyclic rotation of the given matrix "mat" around its center using nearest neighbour interpolation.

cycltrans - implementation of cyclic translation for matrices.

CALLING SEQUENCE

res = cycltrans(mat, m, n)

PARAMETERS

mat: arbitrary typed matrix to be translated.

m: shift in y-Direction = first matrix-dimension, integer value.

n: shift in x-Direction = second matrix-dimension integer value.

res: cyclic translated matrix.

DESCRIPTION

cycltrans implements cyclic translation of matrices with help of (expensive) matrix multiplications. e.g. cycltrans([1 2 3; 4 5 6; 7 8 9], 1,1)= [5 6 4; 8 9 7 ; 2 3 1]

detect_symmetry - detection of rotational symmetry degree of a complex polygon

CALLING SEQUENCE

res = detect_symmetry(Fc)

PARAMETERS

Fc: vector of Fourier-coefficients obtained from a polygon.

res: (largest) symmetry degree of the polygon represented by Fc

DESCRIPTION

detect_symmetry detects rotational symmetry degree s of a polygon represented by its Fourier-coefficients (Fc). This is the maximum number for which a rotation of $2\pi/s$ around the polygon's center exactly reproduces the polygon. If the polygon is not symmetric then the output will be 1.

dumb_class_2d - implements a demo classification in 2 dimensions.

CALLING SEQUENCE

[cl,cert] = dumb_class_2d(v)

PARAMETERS

v: a 2*n - matrix consisting of n points to be classified

cl: n-vector of estimated class memberships

cert: n-vector of certainty measures for the classifications

DESCRIPTION

dumb_class_2d implements a demo classification of 2-dimensional vectors: labels 1,2,3 are assigned to x-coordinates in the intervalls $-\text{inf} < x < 0$, $0 \leq x < 1$, $1 \leq x < +\text{inf}$. "cert" is set to the y value of the points. Purpose of routine: demonstration of use of the function visclass: visclass(dumb_class) produces graphical output of classification regions.

fcycltrans - fast implementation of cyclic translation for matrices.

CALLING SEQUENCE

res = fcycltrans(mat, m, n)

PARAMETERS

mat: arbitrary typed matrix to be translated.

m: shift in y-Direction = first matrix-dimension, integer value.

n: shift in x-Direction = second matrix-dimension integer value.

res: cyclic translated matrix.

DESCRIPTION

fcycltrans implements fast cyclic translation of matrices. e.g. fcycltrans([1 2 3; 4 5 6; 7 8 9], 1,1) = [5 6 4; 8 9 7 ; 2 3 1]

getmatrix - gets the data stored in an object figure (as a matrix)

CALLING SEQUENCE

res = getmatrix(f, s)

PARAMETERS

f: object figure number

s: string argument specifying the name with which the data(as a matrix) is stored in the figure

res: the matrix stored in the "object figure".

DESCRIPTION

getmatrix gets the data stored (as a matrix) in an "object figure". This function works for huge data, and outputs a matrix but not a list in contrast to the function "get".

inputpol - generates a polygon interactively (by mouse clicks or dragging)

CALLING SEQUENCE

res = inputpol(['mode', Mode], ['size', Size], ['noclose'], ['no-nw'])

PARAMETERS

Mode: sets the mode of mouse drawing ('click', 'drag', 'marks'); default: 'click'

Size: a 1*4 matrix which sets the region of drawing; default :[-5,-5,10,5];

'noclose': by default the graphics-window will be closed as soon as the drawing is finished. This parameter should be given in order to keep the window opened.

'no-nw': This parameter should be given in order to draw in an already existing window. Default is that a new graphics-window will be popped up.

res: a 2*n matrix with indices of the resulting polygon as its elements

DESCRIPTION

inputpol allows the user to draw a polygon by clicking/dragging the mouse(left mouse button should be pressed for drawing) in the window which is popped up at the function-call. If the user gives 'no-nw' then the currently-active window will be set for drawing. The window will be closed after drawing, if the user doesn't give 'no-close' argument. With giving 'marks', the lines of the polygon will not be drawn but marks put at the vertices instead.

monomvec - calculates the power-substitution-vector, the vector of monomials

CALLING SEQUENCE

res = monomvec(p, x)

PARAMETERS

p: desired polynomial degree

x: input vector (x1, ..., xn) of arbitrary length n (if matrix, each column is processed)

res: power substitution vector of x

DESCRIPTION

monomvec Calculates the power-substitution-vector i.e vector of monomials of a given vector. If p is the desired polynomial degree and x =(x1...xn) is the given vector, then the power-substitution-vector of x is (1, x1, ... , xn, x1^2, ...x1x2, ..., xn^2,... xn^p). This is the lexicographical ordered set of monomials of x of degree $\leq$ p. If X is a matrix, result will be a matrix with columnwise results.

new_window - gets the number of the next possible new graphics window

CALLING SEQUENCE

res = new_window()

PARAMETERS

res: is the number of the possible new window i.e. one number higher than the highest number in all of the existing windows.

DESCRIPTION

new_window gets the value (number) of the next possible new graphics window

nneigh_class - implements the nearest neighbour classification in multi-dimensions

CALLING SEQUENCE

[classes,cert] = nneigh_class(x, xlearn, labels).

PARAMETERS

x: a n1*n2 matrix representing n2 data points to be classified as its columns.

xlearn: n1*m - matrix consisting of learning points

labels: vector of integer class-labels (length m)

classes: vector of estimated class memberships (length n2).

cert: vector of certainty measures for the classifications (length n2).

DESCRIPTION

nneigh_class implementation of nearest neighbour classification in arbitrary dimensions and arbitrary classes. As heuristic certainty measure simly exp(-sqr(d)) is used, where d is the distance of a classified point to its nearest neighbour in the set of learning points.

nneigh_class_2d - implements the nearest neighbour classification in 2 dimensions

CALLING SEQUENCE

[classes,cert] = nneigh_class_2d(x, xlearn, labels).

PARAMETERS

x: a 2*n matrix n 2-dimensional data points to be classified

xlearn: 2* m - matrix consisting of learning points

labels: m-vector of integer class-labels of learning points

classes: n-vector of estimated class memberships, integers.

cert: n-vector of "certainty" measures for the classifications, in this case e^(-dmin^2) (only heuristic, no theoretic foundation for "certainty").

DESCRIPTION

nneigh_class_2d implements nearest neighbour classification of two-dimensional vectors and arbitrary many classes based on the learning data and labels. The classification regions can be visualized by visclass_2d(nneigh_class_2d, xlearn, labels).

patrec_demos - a GUI showing a list of all the Presto-Box demos

CALLING SEQUENCE

patrec_demos()

DESCRIPTION

patrec_demos displays a gui object which contains a list of all the demos and graphic-functions of Presto-Box. Users can also view these demos with the name "Presto-Box: Pattern-Recognition-Demos" after pressing the "Demos" button in the scilab toolbar.

plotFourierseq - plots Fourier-sequence of a polygon for given range and step values

CALLING SEQUENCE

plotFourierseq(pol, n_start, n_end, n_step)

PARAMETERS

pol: a 2*n polygon for which the Fourier-sequence is to be calculated

n_start: start of the range of index of coefficients

n_end: end of the range of index of coefficients

n_step: step of the range of index of coefficients

DESCRIPTION

plotFourierseq generates a list (sequence) of polygons in the given range of the index of coefficients from a given polygon, then it plots the list of polygons using the plotpolseq function. Users can find a sample demo in the list displayed by patrec_demos or in by choosing the Presto-Box demos shown after pressing the "Demos" button in the scilab toolbar.

plotFs - plotting of the Fourier-spectrum (mainly used in Fc_demo)

CALLING SEQUENCE

plotFs(Fc)

PARAMETERS

Fc: a vector of complex values representing the Fourier-coefficients.

DESCRIPTION

plotFs plots the Fourier-spectrum (power-spectrum), i.e. the absolute values of the Fourier-coefficients. This function is mainly used in "Fc_demo" when the demo is started with "show spectrum" option.

plotclassseq - plots the effect of different classifiers on given learning data

CALLING SEQUENCE

plotclassseq(learn_data, labels)

PARAMETERS

learn_data: matrix of columnwise learning vectors

labels: vector with integer-label for each learning vector

DESCRIPTION

plotclassseq plots the classification results from different classifiers defined like bayes_class, nneigh_class polynom_class in a sequence.

plotpol - plots a polygon represented by a 2*n real matrix

CALLING SEQUENCE

plotpol(pol)

PARAMETERS

pol: a 2*n matrix representing the polygon

DESCRIPTION

plotpol plots a polygon represented by a 2*n matrix

plotpolseq - plots the given list of polygons in sequence

CALLING SEQUENCE

plotpolseq(pol1, str1 [, pol2, str2, ...])

PARAMETERS

pol1,pol2,...: 2*n matrices representing the polygons, this can be repeated any number of times but should be followed by its title (as a string argument str) every time.

str1,str2,...: a string argument representing the title of the polygon preceding this parameter.

DESCRIPTION

This function plots a sequence of polygons into different subplots of one graphical window. Each polygon is labelled with an individual title above the plot.

polarea - area of a real polygon

CALLING SEQUENCE

res = polarea(p)

PARAMETERS

p: a 2*n real matrix representing the polygon for which the area is to be calculated.

res: area of the input polygon

DESCRIPTION

polarea calculates the area of the polygon represented by a 2*n real valued matrix.

polboundary - length of boundary of a real polygon

CALLING SEQUENCE

res = polboundary(p)

PARAMETERS

p: a 2*n real matrix representing the polygon for which the length of its boundary is to be calculated.

res: length of boundary of the input polygon

DESCRIPTION

polboundary calculates the length of boundary of the polygon represented by a 2*n real valued matrix.

polclass_demo - a GUI for interactive polygon classification

CALLING SEQUENCE

polclass_demo([[”setpoly”, pol]...] [, ”high_index”, hfc] [, ”low_index”, lfc] [, ”pfeatures”])

PARAMETERS

”setpoly”: optional string parameter, for setting a reference polygon to compare with the polygon to be classified. A 2*n matrix ”pol” should follow this argument. the set of these two can be repeated any number of times. default values are given by a list of polygons representing the letters E, F, L, T.

”high_index”: optional string parameter for setting the highest index of high frequency Fourier-coefficients This argument should be followed by an integer ”hfc”.

”low_index”: optional string parameter, for setting the highest index of Fourier-coefficients to be regarded as ”low-frequency Fc”. This argument should be followed by an integer lfc.

”pfeatures”: optional string parameter, if the user wants to print the feature vectors to the scilab window.

DESCRIPTION

polclass_demo takes the polygons given by the user with the argument ”setpoly”, otherwise the default values E, F, L, T. Then it displays a GUI showing the features to be selected and a ”new polygon” and ”classify polygon” button. First the user has to click ”new polygon” button for drawing a polygon interactively. then he has to make sure that at least one feature is selected in the list, then has to click the ”classify polygon” button to see the classification of the input polygon. more detailed information is available in the help button in the GUI object of this demo. The demo is also available in the patrec_demos list or by choosing the Presto-Box demos after pressing the ”Demos” button in the scilab toolbar.

polcog - center of gravity of a real polygon

CALLING SEQUENCE

res = polcog(p)

PARAMETERS

p: a 2*n real matrix representing the polygon for which the center of gravity is to be calculated.

res: center of gravity of the input polygon

DESCRIPTION

polcog calculates the center of gravity of the polygon represented by a 2*n real valued matrix. This is the ”physical” center of gravity if area of the polygon is filled uniformly with mass.

pollinecog - line center of gravity of a real polygon

CALLING SEQUENCE

res = pollinecog(p)

PARAMETERS

p: a 2*n real matrix representing the polygon for which the line center of gravity is to be calculated.

res: line center of gravity of the input polygon

DESCRIPTION

pollinecog calculates the line center of gravity of the polygon represented by a 2*n real valued matrix. This is the ”physical” center of gravity if mass is distributed uniformly along the boundary line of the polygon.

polstar - generation of a star-shaped origin-centered polygon.

CALLING SEQUENCE
res = polstar(n, R, r)

PARAMETERS

n: number of arms

R: outer radius

r: inner radius

res: matrix with the corners/edges of the origin centered star of n arms

DESCRIPTION
polstar generates a default polygon. Result is a star with origin as the center, n as the number of arms, R as the outer radius and r as the inner radius.

polstar_demo - a GUI for interactive polygon generation

CALLING SEQUENCE
polstar_demo()

DESCRIPTION
polstar_demo demonstrates the function polstar. first it displays a GUI showing the slider, edit field, push button for each "number of arms", "outer radius","inner radius". The user then can modify these values by dragging the slider or by typing some value in the edit field and pressing the "set" button. More detailed information is available by the "help" button in the GUI object of this demo. The demo is also available in the patrec_demos list or by choosing the Presto-Box demos after pressing the "Demos" button in the scilab toolbar.

polygon - converts a complex-valued vector to a real-valued matrix

CALLING SEQUENCE
res = polygon(cpol)

PARAMETERS

cpol: a vector(of length n) of complex values

res: a 2*n matrix of real values

DESCRIPTION
cpolygon converts a complex vector of length n to a 2*n matrix of real values.

polynom_class - training of polynomial classifier and classification.

CALLING SEQUENCE
[classes, cert] = polynom_class(X, p, Xlearn, labels)

PARAMETERS

X: arbitrary sized matrix of columnwise points which have to be classified.

p: polynomial degree to be used (positive integer)

Xlearn: matrix of columnwise learning vectors, dimension of the vectors has to coincide with columns of X

labels: vector of integer-labels (starting from 1) of the learning vectors (of same length as number of Xlearn-examples) classes: vector of estimated class-numbers for each vector in X.

cert: vector of "certainties" for each class decision. Values between 0 and 1, simply the maximum of the winning decision function.

DESCRIPTION

polynom_class first performs polynomial regression on the learning data, (uses regress_matrix and polynom_regress functions for this purpose) then it uses this regression-function for classification.

polynom_regress - evaluate polynomial (vectorial) regression function

CALLING SEQUENCE

res = polynom_regress(p, X, A)

PARAMETERS

p: desired polynomial degree of regression (also implicitly contained in matrix A)

X: matrix of columnwise points in which the regression function will be evaluated.

A: Regression matrix defining the regression function obtained e.g. from regress_matrix.

res: matrix with columnwise results from the regression function evaluated in the columns of X

DESCRIPTION

polynom_regress perform evaluation of a trained polynomial (vectorial) regression function by res = transpose(A)*monomvec(p,X). The output of this data is e.g. used in polynom_class function for classification.

pw_absdiff - elementwise absolute difference.

CALLING SEQUENCE

res = pw_absdiff(x, y)

PARAMETERS

x: first vector/matrix for computation

y: second vector/matrix for computation, same size as x

res: elementwise absolute difference same size as x,y

DESCRIPTION

pw_absdiff dummy function needed for CT-transformation performing elementwise absolute difference.

pw_add - elementwise addition.

CALLING SEQUENCE

res = pw_add(x, y)

PARAMETERS

x: first vector/matrix for computation

y: second vector/matrix for computation, same size as x

res: elementwise addition same size as x,y

DESCRIPTION

pw_add dummy function needed for CT-transformation performing elementwise addition.

pw_diffsquare - elementwise square of difference.

CALLING SEQUENCE

res = pw_diffsquare(x, y)

PARAMETERS

x: first vector/ matrix for computation

y: second vector/ matrix for computation, same size as x

res: elementwise square of difference same size as x,y

DESCRIPTION

pw_diffsquare dummy function needed for CT-transformation performing elementwise square of difference.

pw_max - elementwise maximum.

CALLING SEQUENCE

res = pw_max(x, y)

PARAMETERS

x: first vector/matrix for computation

y: second vector/matrix for computation, same size as x

res: elementwise maximum same size as x,y

DESCRIPTION

pw_max dummy function needed for CT-transformation performing elementwise maximum.

pw_min - elementwise minimum.

CALLING SEQUENCE

res = pw_min(x, y)

PARAMETERS

x: first vector/matrix for computation

y: second vector/matrix for computation, same size as x

res: elementwise minimum same size as x,y

DESCRIPTION

pw_min dummy function needed for CT-transformation performing elementwise minimum.

randnormal - generation of arbitrary dimensional normally distributed random data

CALLING SEQUENCE

res = randnormal(m, K [,n])

PARAMETERS

m: mean vector of length L.

K: covariance matrix(an L x L square matrix).

n: number of the learning data points to be generated, default value is 1000.

res: learning data points with length of the mean vector as its dimension.

DESCRIPTION

randnormal generates arbitrary dimensional normally distributed random data. The result can be used as the Xlearn data in the functions nneigh_class, polynom_class etc.

regress_matrix - calculates polynomial regression matrix for vectorial function y=f(x).

CALLING SEQUENCE

res = regress_matrix(p, Xlearn, Y)

PARAMETERS

p: desired polynomial degree of regression

Xlearn: matrix with columnwise observation vectors x_i

Y: matrix with columnwise target vectors y_i

res: regression matrix defining regression function.

DESCRIPTION

regress_matrix calculates the polynomial regression-matrix given some learning data. This matrix can then be used for evaluation of the regression function in new points by polynom_regress

repmat - constructs a large matrix of n*m blocks each consisting of "mat"

CALLING SEQUENCE

res = repmat(mat, n, m)

PARAMETERS

mat: a matrix of dimension x*y where x & y are ≥1

n: desired number of copies of mat in y- direction

m: desired number of copies of mat in x- direction

res: block-matrix consisting of n*m copies of "mat".

DESCRIPTION

repmat: repeating of matrix "mat" (to n rows and m columns) i.e constructing a large matrix of n*m blocks each consisting of "mat".

setmatrix - stores the data (an integer matrix) in an object figure.

CALLING SEQUENCE

setmatrix(f, s, mat)

PARAMETERS

f: object figure number

s: string argument specifying the name with which the data (as a matrix) is to be stored in the "object figure"

mat: data to be stored in the "object figure" (an integer matrix)

DESCRIPTION

setmatrix sets the data (an integer matrix) in an "object figure". This function works even for huge data in contrast to the "set" function.

simtrans - performs simple similarity-transformation of a polygon.

CALLING SEQUENCE

res = simtrans(p, r, a, tv)

PARAMETERS

p: is a 2*n matrix with indices of polygon to be translated.

r: scale-factor (r≥0).

a: angle of rotation.

tv: translation vector(of length 2).

res: a 2*n matrix with points of the transformed polygon.

DESCRIPTION

simtrans is a routine performing a similarity-transformation of a polygon i.e. rotation, scaling and translation simultaneously.

visclass - visualization of classification regions in multi dimensions

CALLING SEQUENCE

visclass(["origin", ori], ["projections", proj1, proj2], ["size", Size], ["resolution", resol], ["gui"], ["class_color", cl_no, RGB_Vect], ["2d", ["plot_points", points, labels], ["shades", num_shades]], "classifier", classif, class_data)

PARAMETERS

"origin": this string should be followed by a vector(ori) representing the origin of the plot, default value is [-3,-3].

"projections": this string should be followed by two distinct positive integers > 0 & ≤ length(ori), default values are 1 & 2.

"size": this string should be followed by a vector of length 2 for setting the size of the range of x and y coordinates for plotting, default value is [6,6].

"resolution": this string should be followed by a vector of length 2 for setting resolution of the range of x and y coordinates for plotting, the default value is [31,31].

"gui": this should be given if the user wants to modify some of the parameters of this function interactively, (a GUI will be popped up for this).

"class_color": this string should be followed by the class number (an integer) & a vector [R G B] of length 3 for setting class_color R,G,B. Each should be between 0 & 1.

"2d": since the default plotting type is "3d", the user should use this parameter if he wants to see the visualization in 2d-graphics.

"plot_points": plots some points with different shades in the plotting region, this string should be followed by a (2 x n) matrix of the points to be plotted and a vector of their class labels, this is an optional argument, it should be given only when "2d" option is selected.

"shades": this string should be followed by a positive integer <100, which sets the number of shades of 2d plotting. should be given only when "2d" option is selected, the default value is 16.

"classifier": should be at the end of all the arguments. this string should be followed by any of the classifiers like: dumb_class_2d, bayes_class, nneigh_class, polynom_class & the corresponding classifier specific data.

"class_data": classifier specific parameters ordered as in the individual argument syntax. For example p, m, k for bayes_class, learn_data & labels for nneigh_class.

DESCRIPTION

visclass visualizes classification of a given classifier in multi dimensions. If called with the "gui" option, interactive setting of various display-parameters is possible.

visclass_2d - visualization of classification regions in 2 dimensions

CALLING SEQUENCE

visclass_2d(classifier,varargin)

PARAMETERS

classifier: Name of the classifier like bayes_class, nneigh_class, polynom_class etc. default value is: dumb_class_2d

varargin: Classification data of the classifier, for example p, m, k for bayes_class, learn_data & labels for nneigh_class etc.

DESCRIPTION

visclass_2d visualizes the regions of classification of various classifiers with 2-dimensional data.

w_DI - performs w_DI-transformation of a matrix

CALLING SEQUENCE

res = w_DI(x)

PARAMETERS

x: matrix to be transformed

res: transformed matrix

DESCRIPTION

w_DI transforms a given matrix x such that the diagonal elements will come into the first column of the resulting matrix, i.e. it translates the i-th row cyclically such that the element in the i-th column comes into the first. This function is necessary for the translation-invariant transformations of the class CT_DI.

References

[1] H. Burkhardt and B. Haasdonk. Mustererkennung WS 02/03, ein multimedialer Grundlagenkurs im Hauptstudium Informatik, CD 1 + 2 (TSCC-Video). Computer Science Department, University of Freiburg, Germany, 2003.

[2] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification.* Wiley Interscience, 2nd edition, 2001.

[3] B. Haasdonk and B.R. Poluru. Presto-Box - Pattern REcognition Scilab TOolBOX. `http://lmb.informatik.uni-freiburg.de/lmbsoft/presto-box`, 2002.

[4] S. Theodoridis and K. Koutroumbas. *Pattern Recognition.* Academic Press, London, 1999.