Kernel-based Orthogonal Projections to Latent Structures (K-OPLS)

Max Bylesjö and Judy Fonville and Mattias Rantalainen

August 24, 2010

Abstract

This document describes features of the `kopls` package for MATLAB. The presented package provides an open-source, platform-independent implementation of the Kernel-based Orthogonal Projections to Latent Structures (K-OPLS) method; a kernel-based classification and regression method. In relation to other kernel-based methods, K-OPLS offers unique properties facilitating separate modeling of predictive variation and structured noise in the feature space. While providing prediction results similar to other kernel-based methods, K-OPLS features enhanced interpretational capabilities; allowing detection of unanticipated systematic variation in the data such as instrumental drift, batch variability or unexpected biological variation. To optimize kernel parameter for Gaussian kernels, a simulated annealing approach with nested-cross validation has been implemented for automated optimal model building.

Contents

1 Features

The package features the following main functionality:

1. Estimation (training) of K-OPLS models.
2. Prediction of new data using the estimated K-OPLS model in the preceding step.
3. Cross-validation functionality to estimate the generalization error of a K-OPLS model. This is intended to guide the selection of the number of Y-predictive components A and the number of Y-orthogonal components A_o. The supported implementations are:
   - n-fold cross-validation.
   - Monte Carlo Cross-Validation (MCCV)
   - Monte Carlo Class-balanced Cross-Validation (for discriminant analysis cases).
4. Kernel functions, including the polynomial and Gaussian kernel functions.
5. Automated optimization of the kernel parameter with either:
   - Gridsearch based on user-defined settings
   - Simulated annealing scheme
6. Model statistics:
   - The explained variation of X (R^2_X).
   - The explained variation of Y (R^2_Y).
   - Prediction statistics over cross-validation for regression tasks (Q^2_Y, which is inversely proportional to the generalization error).
   - Prediction statistics over cross-validation for classification tasks (sensitivity, specificity and ROC measures).
7. Plot functions for visualization:
   - Scatter plot matrices for model score components.
   - Model statistics and diagnostics plots.
   - Real-time optimisation with simulated annealing.

2 Requirements

A functional installation of MATLAB 7.0 or later is required.

To get full functionality for the `koplsPlotScores()` function, the MATLAB statistics toolbox is required (see documentation on this function for further details).

3 Installation

Unzip the MATLAB source files (`*.m`) into any directory of choice. Inside MATLAB, set the path to include the extracted files.

4 Getting started

The package includes a demonstration of the functionality available in the package, which is intended to be a starting point for future use. The demonstration is based on a simulated data set, represented by 1000 spectral variables from two different classes and is available in a supplied workspace. The demonstration essentially consists three main steps.

The first step is to demonstrate how K-OPLS handles the model evaluation (using cross-validation), model building and subsequent classification of external data from a non-linear data set.

The second step is to demonstrate how K-OPLS works in the presence of response-independent (Y-orthogonal) variation, using the same data set but with a strong systematic class-specific disturbance added.

The third step is to demonstrate how simulated annealing and grid search can be used to optimise the kernel parameter, and real-time visualisation of the optimisation for both regression and discriminant analysis is visible.
To run the demonstration:

1. Make sure that all included `kopls*.m` files are in the current path
2. Run the demo by typing `koplsDemo`.

5 Availability

The K-OPLS package for MATLAB is freely available for download at `http://kopls.sourceforge.net/`.

6 Specification of functions

6.1 koplsBasicClassify

```
function [predClass]=koplsBasicClassify(data,k)


Classification function that assesses class belonging of a predicted
response in 'data' based on a fixed threshold 'k'.

** INPUT
```

```
data = matrix containing the predicted response matrix Y,
  where columns denote classes and rows observations.
k = threshold value used to assign class categories.

** OUTPUT
predClass = the predicted class(es) of 'data' given 'k'.
```

6.2 koplsCV

```
function [modelMain]=koplsCV(K,Y,A,oax,nrcv,cvType,preProcK,preProcY,
cvFrac,modelType,verbose)


Function for performing K-OPLS cross-validation for a set of
Y-orthogonal components. The function returns a number of diagnostic
parameters which can be used to determine the optimal number
of model components.

** INPUT
K = The kernel matrix (un-centered); see 'koplsKernel()' for details.
Y = The response matrix (un-centered/scaled). Could be binary (for
  discriminant analysis) or real-valued.
A = The number of Y-predictive components (integer).
oax = The number of Y-orthogonal components (integer).
nrcv = Number of cross-validation rounds (integer).
cvType = Type of cross-validation. Either 'nfold' for n-fold
  cross-validation, 'mccv' for Monte Carlo CV or 'mccvb' for
  Monte Carlo class-balanced CV. See also 'koplsCrossValSet()' for
  details.
preProcK = Pre-processing settings for the kernel matrix.
  Either 'mc' for mean-centering or 'no' for no pre-processing.
preProcY = Pre-processing parameter for Y. Either 'mc' for
  mean-centering, 'uv' for mc + scaling to unit-variance,
  'pa' for mc + Pareto-scaling or 'no' for no scaling.
cvFrac = Fraction of observations in the training set during
  cross-validation. Only applicable for 'mccv' or 'mccvb'
  cross-validation (see 'cvType').
modelType = 'da' for discriminant analysis, 're' for regression.
  If 'da', sensitivity and specificity will be calculated.
verbose = If zero, no output will be displayed, otherwise some
  output will be displayed regarding the cross-validation progress
   (default).
```

```
** OUTPUT
modelMain = Object with 'A' predictive components and 'oax'
  Y-orthogonal components. Contains the following entries:
 cv = Cross-validation results:
     Q2Yhat = Total Q-square result for all Y-orthogonal components.
 Q2YhatVars = Q-square result per Y-variable for all Y-orthogonal
      components.
 Yhat = All predicted Y values as a concatenated matrix.
 Tcv = Predictive score vector T for all cross-validation rounds.
 cvTrainIndex = Indices for the training set observations during
      the cross-validation rounds.
 cvTestIndex = Indices for the test set observations during the
      cross-validation rounds.
 da = Cross-validation results specifically for discriminant
      analysis (DA) cases:
   predClass = Predicted class list per class and Y-orthogonal
      components (integer values).
 trueClass = Predicted class list per class and Y-orthogonal
      components (integer values).
 sensSpec = Sensitivity and specificity values per class and
      Y-orthogonal components (integer values).
 confusionMatrix = Confusion matrix during cross-validation
      rounds.
 nclasses = Number of classes in model.
 decisionRule = Decision rule used: 'max' or 'fixed'.
 args = Arguments to the function:
     A = Number of Y-predictive components.
 oax = Number of Y-orthogonal components.
```

6.3 koplsCVopt

```
function [modelMain]=koplsCVopt(X,Y,A,oax,modelType,optargin)
```

```
Function for performing K-OPLS cross-validation for a set of
Y-orthogonal components. The function returns a number of diagnostic
parameters which can be used to determine the optimal number
of model components. Optimisation of the kernel parameter is possible
with grid search within defined limits and simulated annealing.

Use:
[modelMain]=koplsCVopt(X,Y,A,oax,modelType,{optional,input})
```

```
** INPUT
X = The measurement matrix.
Y = The response matrix. Could be binary (for
  discriminant analysis) or real-valued.
A = The number of Y-predictive components (integer).
oax = The number of Y-orthogonal components (integer).
modelType = 're' for regression, 'da' and 'daAUC' for discriminant
  analysis,If 'da' or 'daAUC', sensitivity and specificity will be
  calculated together with the area under the ROC curve. For simulated
  annealing, optimisation is done for area under ROC curve in 'daAUC'
  (only for two-class problems), and for 'da' the mean sensitivity is
  optimised.

optargin = cell with optional settings:
  Possible Input parameters for K-OPLS model, pair-wise input in
  optargin:
  kernelType   = kernel type, e.g. 'g' for Gaussian or 'p' for polynomial.
  preProcK     = Pre-processing settings for the kernel matrix.
                 Either 'mc' for mean-centering (default) or 'no' for no
                 pre-processing.
  preProcY     = Pre-processing parameter for Y. Either 'mc' for
                 mean-centering (default), 'uv' for mc + scaling to unit-
                 variance, 'pa' for mc + Pareto-scaling or 'no' for no
                 scaling.
  opt          = optional settings to do kernel parameter optimisation,
                 'no' for no optimisation, uses kernelParams, or 'SA' for
                 simulated annealing (default for Gaussian kernel), 'GS'
                 for gridsearch between values as input in kernelParams
                 (default for polynomial kernel)
  kernelParams = settings for the kernel parameter, leave empty for
                 optimisation with SA (default), give three values
                 [start end nr_steps] for gridsearch, between which the
                 gridsearch will be performed and the number of steps;
                 note the kernel parameter can not be zero, and polynomial
                 parameters can only be integer.
  nrcvinner    = Number of inner cross-validation rounds (integer,
                 default = 10).
  nrcvouter    = Number of outer cross-validation rounds (integer,
                 default = 20).
  cvType       = Type of cross-validation. Either 'nfold' for n-fold
                 cross-validation, 'mccv' for Monte Carlo CV or 'mccvb'
                 for Monte Carlo class-balanced CV; default 'mccv' for
                 're' and 'mccvb' for 'da' and 'daAUC' modelType. See
                 also 'koplsCrossValSet()' for details.
  cvFrac       = Fraction of observations (default = 0.75) in the
                 training set during cross-validation. Only applicable
                 for 'mccv' or 'mccvb' cross-validation (see 'cvType').
  verbose      = If zero, no output will be displayed (default), if one
                 some output will be displayed regarding the
```

```
                cross-validation progress. If used with SA optimisation,
                this will also display SA results plot and temperature
                updates.
  Possible Input parameters for Simulated Annealing, pair-wise input in
  optargin:
  StX       = Starting position for algorithm search - can be initiated
              automatically or from a number of preset positions.
  T_0       = Starting temperature; default = 0.1
  epsilon   = Termination criterion, default 0.01.
  Neps      = Number of subsequent optimised points evaluated for
              convergence criterion, default 2.
  v_0       = Starting step vector size, will be adjusted throughout
              optimisation but should preferable be able to cover a large
              range of the possible optimalkernel parameter values;
              default is StX.
  Ns        = Number of points before reducing vector length, default 5.
  Nt        = Number of vector reductions before temperature update,
              default 5.
  rT        = Exponential cooling of temperature, default 0.1.
  nrreps    = Number of runs, default 1.


** OUTPUT
modelMain = Object with 'A' predictive components and 'oax'
  Y-orthogonal components. Contains the following entries:
 koplsModel = Model calculated with all data and using single (i.e. not-
    nested) cross-validation optimisation of the kernel parameter.
 kernelParamslist = (best if optimised) kernel parameter for different
    CV rounds.
 KParamfinal = Final kernel parameter used, based on full data set.
 cv = Cross-validation results:
   Q2Yhat = Total Q-square result for all Y-orthogonal components.
   Q2YhatVars = Q-square result per Y-variable for all Y-orthogonal
      components.
   Yhat = All predicted Y values as a concatenated matrix.
   Tcv = Predictive score vector T for all cross-validation rounds.
   cvTrainIndex = Indices for the training set observations during
      the cross-validation rounds.
   cvTestIndex = Indices for the test set observations during the
      cross-validation rounds.
 da = Cross-validation results specifically for discriminant
      analysis (DA) cases:
   predClass = Predicted class list (integer values).
   trueClass = True class list (integer values).
   sensSpec = Sensitivity and specificity values per class and
      Y-orthogonal components (integer values).
   confusionMatrix = Confusion matrix during cross-validation
      rounds.
   nclasses = Number of classes in model.
   decisionRule = Decision rule used: 'max' or 'fixed'.
```

```
   ROC = structure containing the AUC (area under ROC curve) and the two
     vectors based on sensitivity (sens) and 1-specificity (spec) used
     for calculation, based on the training data.
 args = Arguments to the function:
   A = Number of Y-predictive components.
   oax = Number of Y-orthogonal components.

 Additional SA output:
 SAsettings = Settings for simulated annealing.

 Additional GS output:
 GSresults = optimised predictive performance values for the gridsearched
 values for the different cross-validation rounds.
 GSsettings = Grid that was searched.
```

6.4 koplsCenterKTeTe

```
function [KteTe]=koplsCenterKTeTe(KteTe,KteTr,KtrTr)


Centering function for the test kernel, which is constructed
from the test matrix Xte as KteTe = <phi(Xte), phi(Xte)>.
Requires additional (un-centered) kernels KteTr and KteTr to
estimate mean values (see 'koplsKernel()' for details on
constructing a kernel matrix).

** INPUT
KteTe = Test kernel matrix; KteTe = <phi(Xte), phi(Xte)>.
KteTr = Test/training kernel matrix;
  KteTr = <phi(Xte), phi(Xtr)>.
KtrTr = Training kernel matrix; KtrTr = <phi(Xtr), phi(Xtr)>.

** OUTPUT
KteTe = The centered test kernel matrix.
```

6.5 koplsCenterKTeTr

```
function [KteTr]=koplsCenterKTeTr(KteTr,KtrTr)


Centering function for the hybrid test/training kernel, which
is constructed from the test matrix Xte and the training matrix
Xtr as KteTr = <phi(Xte), phi(Xtr)>. Requires additional
(un-centered) training kernel to estimate mean values
(see 'koplsKernel()' for details on constructing a kernel matrix).

** INPUT
KteTr = Hybrid test/training kernel matrix;
  KteTr = <phi(Xte), phi(Xtr)>.
KtrTr = Training kernel matrix; Ktrain = <phi(Xtr), phi(Xtr)>.

** OUTPUT
KteTr = The centered kernel matrix.
```

6.6 koplsCenterKTrTr

```
function [K]=koplsCenterKTrTr(K)


Centering function for the training kernel, which is constructed
from the training matrix Xtr as K = <phi(Xtr), phi(Xtr)>
(see 'koplsKernel()' for details on constructing a kernel matrix).

** INPUT
K = Training kernel matrix; K = <phi(Xtr), phi(Xtr)>.

** OUTPUT
K = The centered kernel matrix.
```

6.7 koplsConfusionMatrix

```
function [A]=koplsConfusionMatrix(true,pred)


Calculates a confusion matrix from classification results.

** INPUT
true = true class belonging.
pred = predicted class assignment.

** OUTPUT
A = Confusion matrix.
```

6.8 koplsCrossValSet

```
function [cvSet]=koplsCrossValSet(K,Y,modelFrac,type,nfold,nfoldRound)


Generates sets of training/test observations useful for cross-
validation (CV). How the sets are generated is determined by the
'type' parameter, which can be either 'nfold' for n-fold cross-
validation, 'mccv' for Monte Carlo CV, 'mccvb' for Monte Carlo
class-balanced CV.

** INPUT
K = Kernel matrix.
Y = Response matrix.
type = Type of cross-validation:
'nfold' for n-fold, 'mccv' for Monte Carlo CV, 'mccvb' for
Monte Carlo class-balanced CV.
 nfold = Number of total nfold rounds (if type='nfold').
 nfoldRound = Current nfold round (if type='nfold').

** OUTPUT
 Object 'cvSet' with the following entries:
  type = Cross-validation type.
  nfold = Number of nfold rounds.
  nfoldRound = The current nfold round.
  KTrTr = Kernel training matrix; KTrTr = <phi(Xtr),phi(Xtr)>.
```

```
  KTeTr = Kernel test/training matrix;
      KTeTr = <phi(Xte),phi(Xtr)>.}
  KTeTe = Kernel test matrix; KTeTe = <phi(Xte),phi(Xte)>.
  yTraining = Y training set.
  yTest = Y test set.
  trainingIndex = Indices of training set observations.
  testIndex = Indices of test set observations.
```

6.9 koplsDummy

```
function [dummy, labels_sorted]=koplsDummy(class, numClasses);


Converts integer vector to binary matrix (dummy matrix).

** INPUT
class = vector with class belongings (integer).
numClasses = pre-defines the number of classes in the output
(if undefined, the number of unique entries in 'class' will be
  used).

** OUTPUT
dummy = A matrix with rows corresponding to observations and
  columns to classes. Each element in matrix is either one
  (observation belongs to class) or zero (observation does not
  belong to class).
labels_sorted = The class labels that are found in class in
sorted order.
```

6.10 koplsKernel

```
function [K]=koplsKernel(X1,X2,Ktype,params)


Constructs a kernel matrix K = <phi(X1), phi(X2)>.
The kernel function k() determines how the data is transformed and
```

```
is passed as the separate parameter 'Ktype' to the function.
Currently 'Ktype' can be either 'g' (Gaussian) or 'p' (polynomial).

** INPUT
X1 = the first X matrix (non-centered). This is the left side in
  the expression K = <phi(X1), phi(X2)>.
X2 = the second X matrix (non-centered). This is the right side
  in the expression K = <phi(X1), phi(X2)>.
 If X2 = [] (empty set), then only X1 will be used for
 the calculations. This way, only (n^2 - n)/2 instead of n^2
 calculations have to be performed, which is typically much
 faster. Only applicable for pure training or testing kernels.
Ktype = the type of kernel used. Supported entries are:
  - 'g': Gaussian kernel.
  - 'p': Polynomial kernel.
params = A vector with parameter for the kernel function.
  (Currently, all supported kernel functions use a scalar value
   so the vector property of the parameters is for future
   compability).

** OUTPUT
K = The kernel matrix, transformed by the kernel function
  specified by 'Ktype'.
```

6.11 koplsMaxClassify

```
function [predClass]=koplsMaxClassify(data)


Classification function that assesses class belonging of 'data'
based on the maximum value.

** INPUT
data = matrix containing the predicted response matrix Y,
  where columns denote classes and rows observations.

** OUTPUT
predClass = the predicted class(es) of 'data'.
```

6.12 koplsModel

```
function [model]=koplsModel(K,Y,A,nox,preProcK,preProcY)


Function for training a K-OPLS model. The function constructs a
predictive regression model for predicting the values of 'Y by
using the information in 'K'. The explained variation is separated
into predictive components (dimensionality is determined by the
parameter 'A') and 'Y'-orthogonal components (dimensionality
determined by the parameter 'nox').

** INPUT
K = Kernel matrix (un-centered); K = <phi(Xtr),phi(Xtr)>.
Y = Response matrix (un-centered/scaled).
A = Number of predictive components.
nox = Number of Y-orthogonal components.
preProcK = Pre-processing parameters for the 'K' matrix:
  'mc' for mean-centering, 'no' for no centering.
preProcY = Pre-processing parameters for the 'Y' matrix:
  'mc' for mean-centering, 'uv' for mc + scaling to unit variance,
  'pa' for mc + Pareto, 'no' for no scaling.

** OUTPUT
model = Object with the following entries:
  Cp = Y loading matrix.
  Sp = Sigma matrix, containing singular values from Y'*K*Y used
      for scaling.
  Sps = Sp^(-1/2).
  Up = Y score matrix.
  Tp = Predictive score matrix for all Y-orthogonal components.
  T = Predictive score matrix for the final model.
  co = Y-orthogonal loading vectors.
  so = Eigenvalues from estimation of Y-orthogonal loading vectors.
  To = Y-orthogonal score matrix.
  toNorm = Norm of the Y-orthogonal score matrix prior to scaling.
  Bt = T-U regression coefficients for predictions.
  A = Number of predictive components.
  nox = Number of Y-orthogonal components.
  K = The kernel matrix.
  EEprime = The deflated kernel matrix for residual statistics.
  sstot_K = Total sums of squares in 'K'.
  R2X = Cumulative explained variation for all model components.
  R2XO = Cumulative explained variation for Y-orthogonal
      model components.
  R2XC = Explained variation for predictive model components after
      addition of Y-orthogonal model components.
  sstot_Y = Total sums of squares in Y.
```

```
  R2Y = Explained variation of Y.
  preProc = Pre-processing parameters:
      K = Pre-processing setting for K = 'preProcK'.
  Y = Pre-processing setting for Y = 'preProcY'.
  paramsY = Scaling parameters for Y.
```

6.13 koplsModelInternal

```
function outval=koplsModelInternal(Xtr,Ytr,A,oax,nrcv,cvType,preProcK,
preProcY,cvFrac,modelType,kernelType,Xnew)


Function to make a K-OPLS model for nested cross validation purposes,
with returning output (1-Q2) for regression and  (1-sensitivity) for
discriminant analysis target function, or (1-area under ROC curve) for
'daAUC' discriminant analysis. These outputs are used as minimisation
criteria in grid search and simulated annealing optimisation of the
kernel parameter setting, see also 'koplsCVopt'.

Use:
[outval]=koplsModelInternal(Xtr,Ytr,A,oax,nrcv,cvType,preProcK,preProcY,
                            cvFrac,modelType,kernelType,Xnew)

** INPUT
Xtr = The kernel X training set.
Ytr = The response matrix Y training set.
A = The number of Y-predictive components (integer).
oax = The number of Y-orthogonal components (integer).
nrcv = Number of cross-validation rounds (integer).
cvType = Type of cross-validation. Either 'nfold' for n-fold
  cross-validation, 'mccv' for Monte Carlo CV or 'mccvb' for
  Monte Carlo class-balanced CV.
preProcK = Pre-processing settings for the kernel matrix.
  Either 'mc' for mean-centering or 'no' for no pre-processing.
preProcY = Pre-processing parameter for Y. Either 'mc' for
  mean-centering, 'uv' for mc + scaling to unit-variance,
  'pa' for mc + Pareto-scaling or 'no' for no scaling.
cvFrac = Fraction of observations in the training set during
  cross-validation. Only applicable for 'mccv' or 'mccvb'
  cross-validation (see 'cvType').
modelType = 're' for regression, 'da' and 'daAUC' for discriminant
  analysis, if 'da', the mean sensitivity is optimised, for 'daAUC' the
  area under the receiver operating characteristic curve is optimised
```

```
  (only for two-class problems).
kernelType = kernel type, e.g. 'g' for Gaussian or 'p' for polynomial
  When using 'p' make sure the kernel parameter is not too big
  to prevent the polynomial from exploding, resulting in SVD errors.
Xnew = the kernel parameter used for evaluation.

** OUTPUT
outval
  for discriminant analysis ('da')    = 1- sensitiviy
  for discriminant analysis ('daAUC') = 1-area under ROC curve.
  for regression analysis ('re')      = 1-Q^2
   These values are calculated from a kernel OPLS model using
   cross-validation without kernel optimisation.
```

6.14 koplsPlotCVDiagnostics

```
function []=koplsPlotCVDiagnostics(modelFull)


Plots diagnostic parameters from K-OPLS cross-validation.
This includes:
- R2X = Cumulative explained variation for all model components.
- R2Xortho = Cumulative explained variation for all Y-orthogonal
   model components.
- R2Xcorr = Explained variation for predictive model components
   after addition of Y-orthogonal model components.
- Q2Y = Total Q-square result (1 - pred. residual / original var.)
  for all Y-orthogonal components.

For further information regarding the definitation and calculation
of these quantities, see e.g the appendix of:
 * Trygg J and Wold S. J Chemometrics 2003; 17:53-64.

** INPUT
model = a model constructed using 'koplsModel()'.
```

6.15 koplsPlotModelDiagnostics

```
function []=koplsPlotModelDiagnostics(model)


Plots diagnostic parameters from K-OPLS cross-validation.
This includes:
- R2X = Cumulative explained variation for all model components.
- R2XO = Cumulative explained variation for all Y-orthogonal model
  components.
- R2XC = Explained variation for predictive model components after
      addition of Y-orthogonal model components.

For further information regarding the definitation and calculation
of these quantities, see e.g appendix of:
 * Trygg J and Wold S. J Chemometrics 2003; 17:53-64.

** INPUT
model = a model constructed using 'koplsModel()'.
```

6.16 koplsPlotOptResults

```
function [hh]=koplsPlotOptResults(model,optmethod,modelType)


Function for plotting the results of the nested cross-validation for
optimisation of the kerenel parameter. Plotted are Q2Yhat and the R
values as a function of the number of orthogonal components for the final
model after optimisation. In the right-hand subplot, the kernel parameter
optimisation for the different CV rounds. Note that only temporary best
values are plotted, not the whole of the searched space for simulated
annealing. For simulated annealing, a bar plot is created demonstrating
the time the individual CV rounds took.

** INPUT
model = the returned cross-validated model (gridsearch or simulated
  annealing)
optmethod = 'SA' or 'GS', method used for optimisation
modeltype = 're' for regression, 'da' for discriminant analysis optimised
  to (mean sensitivity), 'daAUC' for optimised area under receiver
```

```
  operating characteristic curve.

** OUTPUT
[hh] = handle for produced figure
```

6.17 koplsPlotScores

```
function []=koplsPlotScores(model,x,xsub,y,ysub)


Produces score plots from K-OPLS models. If model components are
unspecified, all possible combinations are displayed as a scatter
plot matrix (default). Otherwise, two selected components will be
shown using a traditional 2D scatter plot.

NB: The diagonal of the scatter plot matrix requires functionality
from the 'MATLAB statistics toolbox' to produce variable densities.
In the absence of the toolbox, the diagonal will consist of blank
plots.

** INPUT
model = the K-OPLS model generated using 'koplsModel()'.
x = the vector index for the x axis.
xsub = the vector identifier {'p', 'o'} for the x axis.
y = the vector index for the y axis.
ysub = the vector identifier {'p', 'o'} for the y axis
```

6.18 koplsPlotSensSpec

```
function [res]=koplsPlotSensSpec(modelFull)

Plots sensitivity and specificity results from cross-validation
in a bar plot. The produced bars are shown separately for each
class including overall sensitivity and specificity results.

** INPUT
modelFull = the K-OPLS cross-validation results from 'koplsCV()'
```

```
** OUTPUT
res = The resulting sensitivity and specificity measures.
```

6.19 koplsPredict

```
function [modelp]=koplsPredict(KteTr,Ktest,Ktrain,model,nox,rescaleY)


Performs prediction of new samples from an existing K-OPLS model
(see koplsModel()' to calculate K-OPLS models).
The function projects the Y-predictive and Y-orthogonal scores
components to predict a value of the response matrix Y.
The dimensionality of the parameters is determined from the
specified model.

** INPUT
KteTr = The hybrid test/training kernel matrix;
  KteTr = <phi(Xte),phi(Xtr)>.
Ktest = The pure test kernel matrix;
  Ktest = <phi(Xte),phi(Xte)>.
Ktrain = The training kernel matrix (same as used in
  model training); Ktrain = <phi(Xtr),phi(Xtr)>.
model = K-OPLS model object.
nox = Number of Y-orthogonal components. If not specified, the
  number used during model training will be employed.
rescaleY = Boolean parameter. If true, predicted values of the
  response (Yhat) is rescaled according to the pre-processing
  settings of the model. If false, Yhat is not rescaled (default).

** OUTPUT:
modelp = Object with the following entries:
  Tp = Predicted predictive score matrix for all generations
      0:'nox' of Y-orthogonal vectors.
  T = Predictive score matrix for the final model with 'nox'
      Y-orthogonal vectors.
  to = Predicted Y-orthogonal score vectors.
  EEprime = Calculated residuals for the test kernel 'Ktest',
      useful e.g. for residual statistics.
  Yhat = Predicted values of the response matrix.
```

6.20 koplsReDummy

```
function [classVect]=koplsReDummy(Y)


Reconstructs a (integer) class vector from a binary (dummy) matrix.

** INPUT
Y = Dummy matrix. See 'koplsDummy()' for details.

** OUTPUT
classVect = The reconstructed integer class vector.
```

6.21 koplsRescale

```
function [scaleS]=koplsRescale(scaleS,varargin)


Scales a matrix based on pre-defined parameters from a scaling
object.

** INPUT
scaleS = An object containing scaling parameters
  (see 'koplsScale()').
varargin = If defined, this matrix will be scaled and returned.
Otherwise the original data set in the scaleS object will be
  scaled and returned.

** OUTPUT
scaleS = An object containing the following entries:
  centerType = 'mc' (mean-centering) or 'no' (no centering).
  scaleType = 'uv' (unit variance), 'pa' (pareto) or 'no'
      (no scaling).
  meanV = vector with mean values for all columns in X.
  stdV = vector with standard deviations for all columns in X.
  X = Scaled version of 'varargin', if defined, otherwise,
      scaled version of scaleS.X from input. Scaling is done
```

```
      according to 'centerType' and 'scaleType'.
```

6.22 koplsRoc

```
function [ROC]=koplsRoc(data,trueclass)


Calculates the approximate area under the receiver operating
characteristic curve to assess the classification performance.

** INPUT
data = matrix containing the predicted response matrix Y,
  where columns denote classes and rows observations.
trueclass = true class designation.

** OUTPUT
ROC = the sensitivities and 1-specificities for different thresholds
      as well as the area under curve
```

6.23 koplsSA

```
function [Optset,settings]=koplsSA(Xtr,Ytr,A,oax,modelType,optargin)


Function to perform simulated annealing for optimisation of the kernel
parameter. The error is mimimised, with the error as defined in
koplsModelInternal, (1-Q2) mimimisation for regression and minimisation
of (1-sensitivity) for discriminant analysis target function, or (1-area
under ROC curve) for 'daAUC' discriminant analysis. The correct
data set should be loaded with Xtr as X training set and Ytr for the
Y training set, these data are used to perform a cross-validation
on to determine the optimal kernel parameter setting.

Use:
[Optset,settings]=koplsSA(Xtr,Ytr,A,oax,modelType,{optional, input})
```

```
** INPUT:
Required input parameters:
Xtr = Training X data.
Ytr = Training Y data.
A = The number of Y-predictive components (integer).
oax = The number of Y-orthogonal components (integer).
modelType = 're' for regression, 'da' and 'daAUC' for discriminant
  analysis, if 'da', the mean sensitivity is optimised, for 'daAUC' the
  area under the receiver operating characteristic curve is optimised
  (only for two-class problems).

Possible input parameters for Simulated Annealing, pair-wise input in
optargin:
kernelType = 'g' for gaussian (default).
preProcK = Pre-processing settings of the X data.
  Either 'mc' for mean-centering (default) or 'no' for no pre-processing.
preProcY = Pre-processing settings for Y data. Either 'mc' for
  mean-centering (default), 'uv' for mc + scaling to unit-variance,
  'pa' for mc + Pareto-scaling or 'no' for no scaling.
StX = Starting position for algorithm search - can be initiated
  automatically or from a number of preset positions.
T_0 = Starting temperature; default = 0.1.
epsilon = Termination criterion; default 0.01.
Neps = Number of subsequent optimal points evaluated for convergence
  criterion; default 2.
v_0 = Starting step vector size, will be adjusted throughout optimisation
  but should preferable be able to cover a large range of the possible
  optimal kernel parameter values; default is StX.
Ns = Number of points before reducing vector length; default 5.
Nt = Number of vector reductions before temperature update; default 5.
rT = Exponential cooling of temperature; default 0.1.
nrreps = Number of runs; default 1.
verbose = 1 for plotting and displaying temperature updates and results,
  0 for not (default).

Possible input PLS model parameters, pair-wise input in optargin:
nrcvinner = Number of cross-validation rounds (integer); default 10.
cvType = Type of cross-validation. Either 'nfold' for n-fold
  cross-validation, 'mccv' for Monte Carlo CV or 'mccvb' for Monte Carlo
  class-balanced CV; default 'mccv' for 're' and 'mccvb' for 'da' and
  'daAUC' modelType.
cvFrac = Fraction of observations in the trainingset during
  crossvalidation. Only applicable to 'mccv', 'mccvb' crossvalidation;
  default 0.75.

** OUTPUT:
Optset = Optimal settings.
settings = Settings and outcomes for simulated annealing.

Reference: Corona et al; 'minimizing multimodal functions of continuous
```

```
variables with the "simulated annealing" algorithm', ACM transactions on
Mathematical Software, vol. 13, no.3, september 1987, pages 262-280.
```

6.24 koplsScale

```
function [scaleS]=koplsScale(X,centerType,scaleType)


Function for mean-centering and scaling of a matrix.

** INPUT
X = X matrix (to be mean-centered/scaled).
centerType = 'mc' for mean-centering, 'no' for no centering.
scaleType = 'uv' for unit variance scaling, 'pa' for Pareto
  scaling, 'no' for no scaling.

** OUTPUT:
scaleS = An object with the following properties:
  centerType = 'mc' or 'no'.
  scaleType = 'uv', 'pa' or 'no'.
  meanV = vector with mean values for all columns in X.
  stdV = vector with standard deviations for all columns in X.
  X = Original input matrix X, scaled according to 'centerType'
      and 'scaleType'.
```

6.25 koplsScaleApply

```
function [scaleSA]=koplsScaleApply(X,scaleS)


Applies scaling from external scaling objects (see
'koplsScale()') on a matrix X. Returns the scaled matrix,
including scaling settings.

** INPUT
X = X matrix (to be scaled).
```

```
scaleS = An object containing scaling parameters
  (see 'koplsScale()').

** OUTPUT
scaleSA = An object with the following properties:
  centerType = 'mc' or 'no'.
  scaleType = 'uv', 'pa' or 'no'.
  meanV = vector with mean values for all columns in X.
  stdV = vector with standard deviations for all columns in X.
  X = Scaled version of 'X', scaled according to 'centerType'
      and 'scaleType'.
```

6.26 koplsSensSpec

```
function [sensvec, specvec, classvec, tot_sens,meanSens,
meanSpec]=koplsSensSpec(v, m)


Calculates sensitivity and specificity in a class-wise fashion.

** INPUT
v = row vector of true class assignments (template).
m = matrix (or row vector) of class assignments to be compared.

** OUTPUT
sensvec = sensitivity for each class.
specvec = specificity for each class.
classvec = the class identifier corresponding to each column in
  sensvec and specvec.
tot_sens = total sensitivity.
meanSens = mean sensitivity over all classes.
meanSpec = mean specificity over all classes.
```

6.27 koplsDemo

```
This script contains a demonstration of the functionality in the
'kopls' package using a simulated data set. The data set is
represented by 1000 spectral variables from two different classes
and is available in the 'koplsDemo.mat' workspace. The
demonstration essentially consists of two main steps.

The first step is to demonstrate how K-OPLS handles the
model evaluation (using cross-validation), model building and
subsequent classification of external data from a non-linear data
set. The second step is to demonstrate how K-OPLS works in the
presence of response-independent (Y-orthogonal) variation, using
the same data set but with a strong systematic class-specific
disturbance added.

** THE 'koplsDemo.mat' WORKSPACE
The koplsDemo.mat workspace contains the following objects:
  Xtr = The training data matrix, with 400 observations and
      1000 spectral variables.
  Xte = The test data matrix, with 400 observations and
      1000 spectral variables.
  Xtro = Same data as 'Xtr', but with class-specific systematic
      noise added.
  Xteo = Same data as 'Xte', but with class-specific systematic
      noise added.
  Ytr = A binary matrix of class assignments for the training data.
  Yte = A binary matrix of class assignments for the test data.
  class1 = A vector of indicies of the samples in class #1.
  class2 = A vector of indicies of the samples in class #2.


** INSTRUCTIONS
1) Make sure that all 'kopls*.m' files are in the current path.
2) Run 'koplsDemo'.
```

References

[1] Rantalainen M, Bylesjö M, Cloarec O, Nicholson JK, Holmes E and Trygg J. **Kernel-based orthogonal projections to latent structures (K-OPLS)**. *J Chemometrics* 2007; 21:376-385.

[2] Bylesjö M, Rantalainen M, Nicholson JK, Holmes E and Trygg J. **K-OPLS package: kernel-based orthogonal projections to latent structures**

for prediction and interpretation in feature space. *BMC Bioinformatics* 2008; 19;9:106.

[3] Corona A, Marchesi M, Martinin C and Ridella S. **minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm**. *ACM transactions on Mathematical Software* 1987;vol.13 no 3:262-280.