ARESLab

Adaptive Regression Splines toolbox for Matlab/Octave

ver. 1.10.1

Gints Jekabsons

E-mail: gints.jekabsons@rtu.lv
http://www.cs.rtu.lv/jekabsons/

User's manual

October, 2015

CONTENTS

1. INTRODUCTION

What is ARESLab

ARESLab is a Matlab/Octave toolbox for building piecewise-linear and piecewise-cubic regression models using the Multivariate Adaptive Regression Splines method (also known as MARS). (The term "MARS" is a registered trademark and thus not used in the name of the toolbox.) The author of the MARS method is Jerome Friedman (Friedman, 1991a; Friedman, 1993).

With this toolbox you can build MARS models (hereafter referred to as ARES models) for single-response and multi-response data, test them on separate test sets or using Cross-Validation, use the models for prediction, print their equations, perform ANOVA decomposition, assess input variable importance, as well as plot the models.

The user's manual provides overview of the functions available in the ARESLab.

ARESLab can be downloaded at http://www.cs.rtu.lv/jekabsons/.

The toolbox code is licensed under the GNU GPL ver. 3 or any later version. Some parts of `aresbuild` and `createList` functions were initially derived from ENTOOL toolbox (Merkwirth & Wichard, 2003) which also falls under the GPL licence.

Details

The ARESLab toolbox is written entirely in Matlab/Octave. The MARS method is implemented according to the Friedman's original papers (Friedman, 1991a; Friedman, 1993). While implementing the knot placement part (see remarks about `useMinSpan` and `useEndSpan` in Section 2.2), I also took a look at the source code of the R Earth package (Milborrow, 2015) and implemented it very similarly to Earth version 4.4.3.

One major difference is that the model building is not accelerated using the "fast least-squares update technique" (Friedman, 1991a). This difference however affects only the speed of the algorithm execution, not predictive performance of the built models.

The absence of the acceleration means that the code might be slow for large data sets (however, see description of `aresparams` on how to make the process faster by using the "Fast MARS" algorithm and/or setting more conservative values for algorithm parameters). An alternative is to use the open source R Earth package which is faster and in some aspects more sophisticated, however currently lacks the ability to create piecewise-cubic models. Yet another open source alternative is py-earth for Python (Rudy, 2015).

ARESLab does not automatically handle missing data or categorical input variables with more than two categories. Such categorical variables must be replaced with dummy binary variables before using ARESLab.

Feedback

For any feedback on the toolbox including bug reports feel free to contact me via the email address given on the title page of this user's manual.

Citing the ARESLab toolbox

Jekabsons G., ARESLab: Adaptive Regression Splines toolbox for Matlab/Octave, 2015, available at http://www.cs.rtu.lv/jekabsons/

2. AVAILABLE FUNCTIONS

ARESLab toolbox provides the following list of functions:

- `aresbuild` – builds an ARES model;
- `aresparams` – creates a structure of ARES configuration parameters for further use with `aresbuild`, `arescv`, and `arescvc` functions;
- `arespredict` – makes predictions using ARES model;
- `arestest` – tests ARES model on a test data set;
- `arescv` – tests ARES performance using Cross-Validation; has additional built-in capabilities for finding the "best" number of basis functions for an ARES model;
- `arescvc` – finds the "best" value for penalty *c* of the Generalized Cross-Validation criterion from a set of candidate values using Cross-Validation;
- `aresplot` – plots ARES model;
- `areseq` – prints ARES model;
- `aresanova` – performs ANOVA decomposition and variable importance assessment;
- `aresanovareduce` – reduces ARES model according to ANOVA decomposition;
- `aresdel` – deletes basis functions from ARES model.

2.1. Function `aresbuild`

Purpose:

Builds a regression model using the Multivariate Adaptive Regression Splines method.

Call:

```
[model, time, resultsEval] = aresbuild(Xtr, Ytr, trainParams, weights, keepX, modelOld, dataEval, verbose)
```

All the input arguments, except the first two, are optional. Empty values are also accepted (the corresponding default values will be used).

Input:

`Xtr, Ytr` : `Xtr` is a matrix with rows corresponding to observations, and columns corresponding to input variables. `Ytr` is either a column vector of response values or, for multi-response data, a matrix with columns corresponding to response variables. The structure of the output of this function changes depending on whether `Ytr` is a vector or a matrix (see below).

All values must be numeric. Categorical variables with more than one category must be replaced with dummy binary variables before using `aresbuild` (or any other ARESLab function).

For multi-response data, each model will have the same set of basis functions but different coefficients. The models are built and pruned as usual but with the GCVs (for minimization) and RSSs (for threshold checking) summed across all responses. Since all the models are optimized simultaneously, the results for each model won't be as good as building the models independently. However, the combined model may be better in other senses, depending on what you are trying to achieve. For example, it could be useful to select the set of basis functions that is best across all responses (Milborrow, 2015).

It is recommended to pre-scale `Xtr` values to [0,1] (Friedman, 1991a). This is because widely different locations and scales for the input variables can cause instabilities that could affect the quality of the final model. The MARS method is (except for numerics) invariant to the locations and scales of the input variables. It is therefore reasonable to perform a transformation that causes resulting locations and scales to be most favourable from the point of view of numeric stability (Friedman, 1991a).
For multi-response modelling, it is recommended to pre-scale `Ytr` values so that each response variable gets the appropriate weight during model building. A variable with higher variance will influence the results more than a variable with lower variance (Milborrow, 2015).

`trainParams` : A structure of training parameters for the algorithm. If not provided, default values will be used (see function `aresparams` for details).

`weights` : A vector of weights for observations. If supplied, the algorithm calculates the sum of squared errors multiplying the squared residuals by the supplied weights. The length of the vector must be the same as the number of observations in `Xtr` and `Ytr`. The weights must be nonnegative.

`keepX` : Set this to true to retain `model.X` matrix (see description of `model.X`). For multi-response modelling, the matrix will be replicated for each model. (default value = `false`)

`modelOld` : If here an already built ARES model is provided, no forward phase will be done. Instead this model will be taken directly to the backward phase and pruned. This is useful for fast selection of the "best" penalty `trainParams.c` value using Cross-Validation in `arescvc` function and for fast tuning of other parameters of backward phase (`cubic`, `maxFinalFuncs`).

`dataEval` : A structure containing fields `X`, `Y`, and, optionally, `weights`. Used from function `arescv` to get evaluations for all candidate models in the pruning phase.

`verbose` : Whether to output additional information to console (default value = `true`).

Output:

`model` : A single ARES model for single-response `Ytr` or a cell array of ARES models for multi-response `Ytr`. A structure defining one model has the following fields:

- `coefs` : Coefficients vector of the regression model (for the intercept term and each basis function). Because of the coefficient for the intercept term, this vector is one row longer than the others.
- `knotdims` : Cell array of indices of used input variables for knots in each basis function.
- `knotsites` : Cell array of knot sites for each knot and used input variable in each basis function. `knotdims` and `knotsites` together contain all the information for locating the knots.
- `knotdirs` : Cell array of directions (-1 or 1) of the hinge functions for each used input variable in each basis function.
- `parents` : Vector of indices of direct parents for each basis function (0 if there is no direct parent).
- `trainParams` : A structure of training parameters for the algorithm. The values are updated if any values are chosen automatically. Except `useMinSpan`, because in automatic mode it is calculated for each parent basis function separately.

`MSE` : Mean Squared Error of the model in the training data set.

`GCV` : Generalized Cross-Validation (GCV) of the model in the training data set. The value may also be `Inf` if model's effective number of parameters (see Eq. 1) is larger than or equal to the number of observations in `Xtr`,`Ytr`.

`t1` : For piecewise-cubic models only. Matrix of sites for the additional side knots on the left of the central knot.

`t2` : For piecewise-cubic models only. Matrix of sites for the additional side knots on the right of the central knot.

`minX` : Vector of minimums for input variables.

`maxX` : Vector of maximums for input variables.

`X` : Matrix of values of basis functions applied to `Xtr`. The number of columns in `X` is equal to the number of rows in `coefs`, i.e., the first column is for the intercept (all ones) and all the other columns correspond to their basis functions. Each row corresponds to a row in `Xtr`. Multiplying `X` by `coefs` gives ARES prediction for `Ytr`. This variable is available only if argument `keepX` is set to `true`.

`time` : Algorithm execution time (in seconds).

`resultsEval` : Available only if `dataEval` is not empty. The structure has the following fields:

`R2GCV` : R^2_{GCV} value for each candidate model in the pruning phase.

`R2oof` : Out-of-fold R^2 for each candidate model in the pruning phase.

Remarks:

The algorithm builds a model in two phases: forward selection and backward deletion. In the forward phase, the algorithm starts with a model consisting of just the intercept term and iteratively adds reflected pairs of basis functions giving the largest reduction of training error. The forward phase is executed until one of the following conditions is met:

1) reached maximum number of basis functions (`trainParams.maxFuncs`);
2) adding a new basis function changes R^2 by less than `trainParams.threshold`;
3) reached a R^2 of 1 – `trainParams.threshold` or more;
4) the number of model's coefficients (i.e., the number of all the basis functions including the intercept term) in the next iteration is expected to be equal to or larger than the number of data observations *n*.

At the end of the forward phase we have a large model which typically overfits the data, and so a backward deletion phase is engaged. In the backward phase, the model is simplified by deleting one least important basis function (according to GCV) at a time until the model has only the intercept term. At the end of the backward phase, from those "best" models of each size one model of lowest GCV value is selected and outputted as the final one.

GCV for a model is calculated as follows (Hastie et al., 2009; Milborrow, 2015):

$$GCV = MSE_{train} \Big/ \left(1 - \frac{enp}{n}\right)^2, \tag{1}$$

where MSE_{train} is Mean Squared Error of the model in the training data, n is the number of observations in the training data, and enp is the effective number of parameters:

$$enp = k + c \times (k-1)/2, \tag{2}$$

where k is the number of basis functions in the model (including the intercept term) and c is `trainParams.c`. Note that $(k-1)/2$ is the number of hinge function knots, so the formula penalizes the model not only for its number of basis functions but also for its number of knots. Also note that in ARESLab in the situation when $enp \geq n$ the GCV value is equal to `Inf` (the model is considered infinitely bad).

2.2. Function `aresparams`

Purpose:

Creates configuration for building ARES models. The structure is for further use with `aresbuild`, `arescv`, and `arescvc` functions.

Call:

```
trainParams = aresparams(maxFuncs, c, cubic, cubicFastLevel, selfInteractions, maxInteractions, threshold, prune, fastK, fastBeta, fastH, useMinSpan, useEndSpan, maxFinalFuncs, endSpanAdjust, newVarPenalty)
```

All the input arguments of this function are optional. Empty values are also accepted (the corresponding default values will be used).

Parameters `c`, `prune`, and `maxFinalFuncs` are used in the backward pruning phase. Parameter `cubic` can be used in both phases depending on `cubicFastLevel`. All other parameters are used in the forward phase only.

For most applications, it can be expected that the most attention should be paid to the following parameters: `maxFuncs`, `c`, `cubic`, `maxInteractions`, and in some cases `maxFinalFuncs`. If Fast MARS algorithm is to be used, parameters `fastK`, `fastBeta`, and `fastH` should be looked at. It is quite possible that the default values for `maxFuncs` and `maxInteractions` will be far from optimal for your data.

Input:

`maxFuncs` : The maximum number of basis functions included in model in the forward building phase (before pruning in the backward phase). Includes the intercept term. The recommended value for this parameter is about two times the expected number of basis functions in the final model (Friedman, 1991a). Note that the algorithm may also not reach this number. This can happen when the number of coefficients in the model exceeds the number of observations in data or because of the `threshold` parameter. The default value for this parameter is -1 in which case maxFuncs is calculated automatically using formula $\min(200, \max(20, 2d)) + 1$, where d is the number of input variables (Milborrow, 2015). This is fairly arbitrary but can be useful for first experiments.

To enforce an upper bound on the final model size, use `maxFinalFuncs` instead. This is because the forward phase can see only one basis function ahead while the backward pruning phase can choose any of the built basis functions to include in the final model.

`c` : Generalized Cross-Validation (GCV) penalty per knot. Larger values for `c` will lead to fewer knots (i.e., the final model will have fewer basis functions). A value of 0 penalizes only terms, not knots (can be useful, e.g., with lots of data, low noise, and highly structured underlying function of the data). Generally, the choice of the value for c should greatly depend on size of the dataset, how structured is the underlying function, and how high is the noise level, and mildly depend on the thoroughness of the optimization procedure, i.e., on the parameters `maxFuncs`, `maxInteractions`, and `useMinSpan` (Friedman, 1991a). Simulation studies suggest values for `c` in the range of about 2 to 4 (Friedman, 1991a). The default value for this parameter is -1 in which case `c` is chosen automatically using the following rule: if `maxInteractions` = 1 (additive modelling) `c` = 2, otherwise `c` = 3. These are the values recommended by Friedman (Friedman, 1991a).

`cubic` : Whether to use piecewise-cubic (`true`) or piecewise-linear (`false`) type of modelling. In general, it is expected that the piecewise-cubic modelling will give better predictive performance for smoother and less noisy data. (default value = `true`)

`cubicFastLevel` : `aresbuild` implements three levels of piecewise-cubic modelling. In level 0, cubic modelling for each candidate model is done in both phases of the method (slow). In level 1, cubic modelling is done only in the backward phase (much faster). In level 2, cubic modelling is done after both phases, only for the final model (fastest). The default and recommended level is 2 (and it corresponds to the recommendations in Friedman's paper (Friedman, 1991a)). Levels 0 and 1 may bring extra accuracy in the situations when the underlying function of the data has sharp thresholds that require knot placements different that those required for piecewise-linear modelling.

`selfInteractions` : This is experimental feature. The maximum degree of self interactions for any input variable. It can be larger than 1 only for piecewise-linear modelling. The default, and recommended, value = 1, no self interactions.

`maxInteractions` : The maximum degree of interactions between input variables. Set to 1 for additive modelling (i.e., no interaction terms). For maximal interactivity between the variables, set the parameter to $d \times$`selfInteractions`, where d is the number of input variables – this way the modelling procedure will have the most freedom building a complex model. Typically only a low degree of interaction is allowed, but higher degrees can be used when the data warrants it. (default value = 1)

`threshold` : One of the stopping criteria for the forward phase (see remarks section of function `aresbuild` for details). Default value = 1e-4. For noise-free data, the value may be lowered but setting it to 0 can cause numerical issues and instability.

`prune` : Whether to perform model pruning (the backward phase). (default value = `true`)

`fastK` : Parameter (integer) for Fast MARS algorithm (Friedman, 1993, Section 3.0). Maximum number of parent basis functions considered at each step of the forward phase. Typical values for `fastK` are 20, 10, 5 (default value = `Inf`, i.e., no Fast MARS). With lower `fastK` values model building is faster at the expense of some accuracy. Good starting values for fast exploratory work are `fastK` = 20, `fastBeta` = 1, `fastH` = 5 (Friedman, 1993). Friedman in his paper concluded that, while changing the values of `fastK` and `fastH` can have big effect on training computation times, predictive performance is largely unaffected over a wide range of their values (Friedman, 1993).

`fastBeta` : Artificial ageing factor for Fast MARS algorithm (Friedman, 1993, Section 3.1). Typical value for `fastBeta` is 1 (default value = 0, i.e., no artificial ageing). The parameter is ignored if `fastK` = `Inf`.

`fastH` : Parameter (integer) for Fast MARS algorithm (Friedman, 1993, Section 4.0). Number of iterations till next full optimization over all input variables for each parent basis function. Higher values make the search faster. Typical values for `fastH` are 1, 5, 10 (default value = 1, i.e., full optimization in every iteration). Computational reduction associated with increasing `fastH` is most pronounced for data sets with many input variables and when large `fastK` is used. There seems to be little gain in increasing `fastH` beyond 5 (Friedman, 1993). The parameter is ignored if `fastK` = `Inf`.

`useMinSpan` : In order to lower the local variance of the estimates, a minimum span is imposed that makes the method resistant to runs of positive or negative error values between knots (by jumping over a `minSpan` number of observations each time the next potential knot placement is requested) (Friedman, 1991a). `useMinSpan` allows to disable (set to 0 or 1) the protection so that all *x* values are considered for knot placement in each dimension (except, see `useEndSpan`). Disabling `minSpan` may allow creating a model which is more responsive to local variations in the data however this can also lead to overfitting. Setting the `useMinSpan` to > 1, enables to manually tune the value. (default and recommended value = -1 which corresponds to the automatic mode)

`useEndSpan` : In order to lower the local variance of the estimates near the ends of data intervals, a minimum span is imposed that makes the method resistant to runs of positive or negative error values between extreme knot locations and the corresponding ends of data intervals (by not allowing to place a knot too near to the end of data interval) (Friedman, 1991a). `useEndSpan` allows to disable (set to 0) the protection so that all the observations are considered for knot placement in each dimension (except, see `useMinSpan`). Disabling `endSpan` may allow creating a model which is more responsive to local variations in the data however this can also lead to a model that is overfitted near the edges of the data. Setting the `useMinSpan` to > 1, enables to manually tune the value. (default and recommended value = -1 which corresponds to the automatic mode)

`maxFinalFuncs` : Maximum number of basis functions (including the intercept term) in the pruned model. Use this (rather than the `maxFuncs` parameter) to enforce an upper bound on the final model size (that is less than `maxFuncs`). (default value = `Inf`)

`endSpanAdjust` : For basis functions with variable interactions, `endSpan` gets multiplied by this value. This reduces probability of getting overfitted interaction terms supported by just a few observations on the boundaries of data intervals. Still, at least one knot will always be allowed in the middle, even if `endSpanAdjust` would prohibit it. Useful values range from 1 to 10. (default value = 1, i.e., no adjustment)

`newVarPenalty` : Penalty for adding a new variable to a model in the forward phase. This is the gamma parameter of Eq. 74 in the original paper (Friedman, 1991a). The higher is the penalty, the more reluctant will be the forward phase to add a new variable to the model – it will rather try to use variables already in the model. This can be useful when some of the variables are highly collinear. As a result, the final model may be easier to interpret although usually the built models also will have worse predictive performance. Useful non-zero values typically range from 0.01 to 0.2 (Milborrow, 2015). (default value = 0, i.e., no penalty)

Output:

`trainParams` : A structure of parameters for further use with `aresbuild`, `arescv`, and `arescvc` functions containing the provided values (or default ones, if not provided).

Remarks:

The knot placement in `aresbuild` is implemented very similarly to R Earth package version 4.4.3 (Milborrow, 2015). `minSpan` and `endSpan` are calculated using formulas given in Eq. 45 and Eq. 43 of the Friedman's original paper (Friedman, 1991a) with *alpha* = 0.05. Note that for a fixed

dimensionality of the data, the `endSpan` value always stays the same but the value of the `minSpan` is recalculated for each individual parent basis function that is used for generation of new basis functions. The knots are placed symmetrically so that there are approximately equal number of skipped observations at each end of data intervals.

If more speed is required, try using the Fast MARS algorithm by setting `fastK` parameter to something other than `Inf`. Good starting values for fast exploratory work are `fastK` = 20, `fastBeta` = 1, `fastH` = 5 (Friedman, 1993). For more information, see descriptions of the mentioned parameters and the Friedman's paper.

Alternatively, for more speed you can try some of the following options (if they are adequate for your situation):

1) decreasing `maxFuncs` (less iterations in the forward phase);
2) setting `cubicFastLevel` = 2 (only for piecewise-cubic modelling);
3) decreasing `selfInteractions` (less candidate models in the forward phase);
4) decreasing `maxInteractions` (less candidate models in the forward phase);
5) increasing `threshold` (less iterations in the forward phase);
6) manually increasing `useMinSpan` and/or `useEndSpan` (less candidate models in the forward phase).

Note that decreasing the number of iterations or candidate models may also result in worse models.

2.3. Function `arespredict`

Purpose:

Predicts response values for the given query points using ARES model.

Call:

```
Yq = arespredict(model, Xq)
```

Input:

`model` : ARES model or a cell array of ARES models (for multi-response modelling).

`Xq` : A matrix of query data points.

Output:

`Yq` : Either a column vector of predicted response values or, for multi-response modelling, a matrix with columns corresponding to response variables.

2.4. Function `arestest`

Purpose:

Tests ARES model on a test data set (`Xtst`, `Ytst`).

Call:

```
results = arestest(model, Xtst, Ytst, weights)
```

Input:

`model` : ARES model or a cell array of ARES models (for multi-response modelling).

`Xtst, Ytst` : `Xtst` is a matrix with rows corresponding to testing observations, and columns corresponding to input variables. `Ytst` is either a column vector of

response values or, for multi-response data, a matrix with columns corresponding to response variables.

`weights` : Optional. A vector of weights for observations. See description for function `aresbuild`.

Output:

`results` : A structure of different error measures calculated on the test data set. For multi-response data, all error measures are given for each model separately in a row vector. The structure has the following fields:

- `MAE` : Mean Absolute Error.
- `MSE` : Mean Squared Error.
- `RMSE` : Root Mean Squared Error.
- `RRMSE` : Relative Root Mean Squared Error.
- `R2` : Coefficient of Determination.

2.5. Function `arescv`

Purpose:

Tests ARES performance using *k*-fold Cross-Validation.

The function has additional built-in capabilities for finding the "best" number of basis functions for the final ARES model (`maxFinalFuncs` for function `aresparams`).

Call:

```
[resultsTotal, resultsFolds, resultsPruning] = arescv(X, Y, trainParams, k, shuffle, nCross, weights, testWithWeights, evalPruning, verbose)
```

All the input arguments, except the first two, are optional. Empty values are also accepted (the corresponding default values will be used).

Note that, if argument `shuffle` is set to `true`, this function employs random number generator for which you can set seed before calling the function.

Input:

`X, Y` : Observations. See description for function `aresbuild`.

`trainParams` : A structure of training parameters. If not provided, default values will be used (see function `aresparams` for details).

`k` : Value of *k* for *k*-fold Cross-Validation. The typical values are 5 or 10. For Leave-One-Out Cross-Validation set *k* equal to *n*. (default value = 10)

`shuffle` : Whether to shuffle the order of observations before performing Cross-Validation (default value = `true`).

`nCross` : How many times to repeat Cross-Validation with different data partitioning. This can be used to get more stable results. Default value = 1, i.e., no repetition. Useless if `shuffle` = `false`.

`weights` : A vector of weights for observations. See description for function `aresbuild`.

`testWithWeights` : Set to `true` to use `weights` vector for both, training and testing. Set to `false` to use it for training only. This argument has any effect only when `weights` vector is provided. (default value = `true`)

`evalPruning` : Whether to evaluate all the candidate models of the pruning phase. If set to `true`, the output argument `resultsPruning` contains the results. See example of usage in the user's manual. (default value = `false`)

`verbose` : Whether to output additional information to console (default value = `true`).

Output:

`resultsTotal` : A structure of Cross-Validation results. The results are averaged across Cross-Validation folds and, in case of multi-response data, also across multiple models.

`resultsFolds` : A structure of vectors or matrices (in case of multi-response data) of results for each Cross-Validation fold. Columns correspond to Cross-Validation folds. Rows correspond to models.

Both structures have the following fields:

`MAE` : Mean Absolute Error.

`MSE` : Mean Squared Error.

`RMSE` : Root Mean Squared Error.

`RRMSE` : Relative Root Mean Squared Error. Not reported for Leave-One-Out Cross-Validation.

`R2` : Coefficient of Determination. Not reported for Leave-One-Out Cross-Validation.

`nBasis` : Number of basis functions in model (including the intercept term).

`nVars` : Number of input variables included in model.

`maxDeg` : Highest degree of variable interactions in model.

`resultsPruning` : Available only if `evalPruning` = `true`. See example of usage in the user's manual. The structure has the following fields:

`R2GCV` : A matrix of R^2_{GCV} values for models of each size in each Cross-Validation fold. The number of rows is equal to `k`×`nCross`. Column index corresponds to the number of basis functions in a model.

`meanR2GCV` : A vector of mean R^2_{GCV} values for each model size across all Cross-Validation folds.

`nBasisR2GCV` : The number of basis functions (including the intercept term) for which the mean R^2_{GCV} is maximum.

`R2oof` : A matrix of out-of-fold R^2 values for models of each size in each Cross-Validation fold. The number of rows for this matrix is equal to `k`×`nCross`. Column index corresponds to the number of basis functions in a model.

`meanR2oof` : A vector of mean out-of-fold R^2 values for each model size across all Cross-Validation folds.

`nBasisR2oof` : The number of basis functions (including the intercept term) for which the mean out-of-fold R^2 is maximum.

2.6. Function `arescvc`

Purpose:

Finds the "best" value for penalty *c* of the Generalized Cross-Validation criterion from a set of candidate values using Cross-Validation assuming that all the other parameters for function `aresparams` would stay fixed.

Call:

```
[cBest, results] = arescvc(X, Y, trainParams, cTry, k, shuffle, nCross,
weights, testWithWeights, verbose)
```

All the input arguments, except the first three, are optional. Empty values are also accepted (the corresponding default values will be used).

Note that, if argument `shuffle` is set to `true`, this function employs random number generator for which you can set seed before calling the function.

Input:

`X, Y`	: Observations. See description for function `aresbuild`.
`trainParams`	: A structure of training parameters. If not provided, default values will be used (see function `aresparams` for details).
`cTry`	: A set of candidate values for *c* (default value = `1:5`).
`k`	: Value of *k* for *k*-fold Cross-Validation. The typical values are 5 or 10. For Leave-One-Out Cross-Validation set *k* equal to *n*. (default value = 10)
`shuffle`	: Whether to shuffle the order of the observations before performing Cross-Validation (default value = `true`).
`nCross`	: How many times to repeat Cross-Validation with different data partitioning. This can be used to get more stable results. Default value = 1, i.e., no repetition. Useless if `shuffle` = `false`.
`weights`	: A vector of weights for observations. See description for function `aresbuild`.
`testWithWeights`	: Set to `true` to use `weights` vector for both, training and testing. Set to `false` to use it for training only. This argument has any effect only when `weights` vector is provided. (default value = `true`)
`verbose`	: Whether to output additional information to console (default value = `true`).

Output:

`cBest`	: The best found value for penalty *c*.
`results`	: A matrix with two columns. First column contains all values from `cTry`. Second column contains the calculated MSE values (averaged across all Cross-Validation folds) for the corresponding `cTry` values.

Remarks:

This function finds the "best" penalty *c* value in a clever way of using function `aresbuild`. In each Cross-Validation iteration, the forward phase in `aresbuild` is done only once while the backward phase is done separately for each `cTry` value. The results will be the same as if each time a full model building process would be performed because in the forward phase the GCV criterion is not used.

2.7. Function `aresplot`

Purpose:

Plots ARES model. For datasets with one input variable, plots the function together with its knot locations. For datasets with more than one input variable, plots the surface.

The function works with single-response models only.

Call:

```
aresplot(model, minX, maxX, vals, gridSize)
```

All the input arguments, except the first one, are optional. Empty values are also accepted (the corresponding default values will be used).

Input:

`model`	: ARES model.

`minX, maxX` : User-defined minimum and maximum values for each input variable (this is the same type of data as in `model.minX` and `model.maxX`). If not supplied, the `model.minX` and `model.maxX` values will be used.

`vals` : Only used when the number of input variables is larger than 2. This is a vector of fixed values for all the input variables except the two varied in the plot. The two varied variables are identified using `NaN` values. By default the two first variables will be varied and all the other will be fixed at (min + max) / 2.

`gridSize` : Grid size for the surface. The default value is 100 for datasets with one input variable and 50 for datasets with two input variables.

2.8. Function `areseq`

Purpose:

Prints ARES model.

The function works with single-response models only.

Call:

```
eq = areseq(model, precision, varNames, hideCubicSmoothing)
```

All the input arguments, except the first one, are optional.

Piecewise-cubic spline equations are very complex as they involve smoothing. For easier interpretation use piecewise-linear models or set `hideCubicSmoothing` to `true`.

Input:

`model` : ARES model.

`precision` : Number of digits in the model coefficients and knot sites. (default value = 15)

`varNames` : A cell array of variable names to show instead of the generic ones. This is used for piecewise-linear models only.

`hideCubicSmoothing` : This is for piecewise-cubic models only. Whether to hide piecewise-cubic spline smoothing. It's easier to understand the equations if smoothing is hidden. The model will look like piecewise-linear but the coefficients will be for the actual piecewise-cubic model. (default value = `true`)

Output:

`eq` : A cell array of strings containing equations for individual basis functions and the main model.

2.9. Function `aresanova`

Purpose:

Performs ANOVA decomposition and variable importance assessment of the given ARES model and reports the results. For details, see remarks below as well as Sections 3.5 and 4.3 in (Friedman, 1991a) and Sections 2.4, 4.1, and 4.4 in (Friedman, 1991b).

The function works with single-response models only.

Call:

```
varImportance = aresanova(model, Xtr, Ytr, weights)
```

All the input arguments, except the last one, are required.

Input:

`model`	: ARES model.
`Xtr, Ytr`	: Training data observations.
`weights`	: A vector of weights for observations. The same weights that were used when the model was built.

Output:

`varImportance`	: Relative variable importances. Scaled so that the relative importance of the most important variable has a value of 100.

Remarks:

The function prints two tables. The first table is ANOVA decomposition of the ARES model. The second table reports relative variable importances. Data in the second table is also available through output argument `varImportance`.

To understand the first table, below is an excerpt from the original paper by Jerome Friedman (Friedman, 1991a) Section 4.3. Note that in the excerpt, starting from the mentioning of the fourth column, all the column numbers should be increased by one. This is because `aresanova` adds an additional column reporting GCV estimate of the Coefficient of Determination R^2 as suggested in (Friedman, 1991b). It estimates the proportion of variance explained when all the basis functions comprising the ANOVA function are excluded from the model. By comparing it to the GCV estimate of R^2 for the full model, one can see the amount of reduction the exclusion brings.

> *"The ANOVA decomposition is summarized by one row for each ANOVA function. The columns represent summary quantities for each one. The first column lists the function number. The second gives the standard deviation of the function. This gives one indication of its (relative) importance to the overall model and can be interpreted in a manner similar to a standardized regression coefficient in a linear model. The third column provides another indication of the importance of the corresponding ANOVA function, by listing the GCV, score for a model with all of the basis functions corresponding to that particular ANOVA function removed. This can be used to judge whether this ANOVA function is making an important contribution to the model, or whether it just slightly helps to improve the global GCV score. The fourth column gives the number of basis functions comprising the ANOVA function while the fifth column provides an estimate of the additional number of linear degrees-of-freedom used by including it. The last column gives the particular predictor variables associated with the ANOVA function."*

Contents of the second table are calculated according to (Friedman, 1991b) Section 4.4:

> "*The relative importance of a variable is defined as the square root of the GCV of the model with all basis functions involving that variable removed, minus square root of the GCV score of the corresponding full model, scaled so that the relative importance of the most important variable (using this definition) has a value of 100.*"

Note that the variance of the variable importance estimates can be high – different realizations of the data can give different estimates. Another approach, independent from ARESLab, would be to employ Random Forests or Bagging with regression trees or model trees, for example using the M5PrimeLab toolbox for Matlab/Octave (available from the author of ARESLab) or using TreeBagger class from Matlab's own Statistics and Machine Learning Toolbox.

2.10. Function `aresanovareduce`

Purpose:

Deletes all the basis functions from ARES model (without recalculating model's coefficients and relocating additional knots of piecewise-cubic models) in which at least one used variable is not in the given list of allowed variables. This can be used to perform ANOVA decomposition as well

as for investigation of individual and joint contributions of variables in the model, i.e., the reduced model can then be plotted to visualize the contributions.

The function works with single-response models only.

Call:

```
[model, usedBasis] = aresanovareduce(model, varsToStay, exact)
```

Input:

`model`	: ARES model.
`varsToStay`	: A vector of indices for input variables to stay in the model. The size of the vector should be between one and the total number of input variables.
`exact`	: Set this to true to get a model with only those basis functions where the exact combination of variables is present (default value = `false`). This is used from function `aresanova`.

Output:

`model`	: Reduced ARES model.
`usedBasis`	: List of original indexes for basis functions still in use.

2.11. Function `aresdel`

Purpose:

Deletes basis functions from ARES model, recalculates model's coefficients and relocates additional knots for piecewise-cubic models (as opposed to `aresanovareduce` which does not recalculate anything).

Call:

```
model = aresdel(model, funcsToDel, Xtr, Ytr, weights)
```

Input:

`model`	: ARES model or a cell array of ARES models (for multi-response modelling).
`funcsToDel`	: A vector of indices for basis functions to be deleted. Intercept term is not indexed, i.e., the ordering is the same as in `model.knotdims`, `model.knotsites`, and `model.knotdirs`.
`Xtr, Ytr`	: Training data observations. The same data that was used when the model was built.
`weights`	: A vector of weights for observations. The same weights that were used when the model was built.

Output:

`model`	: Reduced ARES model.

3. EXAMPLES OF USAGE

3.1. Ten-dimensional function with noise

We start by creating a dataset using a ten-dimensional function with added Gaussian noise. The data consists of 200 observations randomly uniformly distributed in a ten-dimensional unit hypercube. The function actually uses only the first five variables.

```
X = rand(200,10);
Y = 10*sin(pi*X(:,1).*X(:,2)) + 20*(X(:,3)-0.5).^2 + ...
    10*X(:,4) + 5*X(:,5) + 0.5*randn(200,1);
```

We set the maximum number of basis functions to 21 (including the intercept term), choose piecewise-cubic modelling, and limit maximum interaction level to 2 (only pairwise products of basis functions will be allowed), leaving all the other parameters to their defaults.

Note that for most applications, it can be expected that the most attention should be paid to the following parameters: `maxFuncs`, `c`, `cubic`, `maxInteractions`, and in some cases `maxFinalFuncs`. If Fast MARS algorithm is to be used, parameters `fastK`, `fastBeta`, and `fastH` should be looked at. It is quite possible that the default values for `maxFuncs` and `maxInteractions` will be far from optimal for your data.

```
params = aresparams(21, [], true, [], [], 2);
```

ARES model is built by calling `aresbuild`. As the model building process ends, we can examine the data structure of the final model. It has 18 basis functions including the intercept term.

```
model = aresbuild(X, Y, params)

model =
            MSE: 0.2741
            GCV: 0.4476
          coefs: [18x1 double]
       knotdims: {17x1 cell}
      knotsites: {17x1 cell}
       knotdirs: {17x1 cell}
        parents: [17x1 double]
    trainParams: [1x1 struct]
             t1: [17x10 double]
             t2: [17x10 double]
           minX: [1x10 double]
           maxX: [1x10 double]
```

From the returned structure `model` we can find out what is its Mean Squared Error (`model.MSE`) and GCV (`model.GCV`) in the training data set. Also we can see what were the training parameters for the method when the model was built (`model.trainParams`). And if we want to extract knot locations, we can use `model.knotdims` and `model.knotsites` for that.

Now let's perform ANOVA decomposition and assess variable importances.

```
aresanova(model, X, Y);

Type: piecewise-cubic
GCV: 0.447615
R2GCV: 0.983249
Total number of basis functions: 18
Total effective number of parameters: 43.5
ANOVA decomposition:
Function   STD          GCV        R2GCV     #basis     #params     variable(s)
1         3.135        3.408       0.8725          2         5.0     1
2         3.750        3.350       0.8746          2         5.0     2
3         1.269        2.655       0.9007          2         5.0     3
4         2.989       12.954       0.5152          2         5.0     4
5         1.524        1.350       0.9495          2         5.0     5
6         2.558        2.528       0.9054          5        12.5     1 2
7         0.285        0.446       0.9833          2         5.0     1 5
Relative variable importance:
Variable      Importance
1                 74.975
2                 71.245
3                 32.772
4                100.000
5                 34.466
6                  0.000
7                  0.000
8                  0.000
9                  0.000
10                 0.000
```

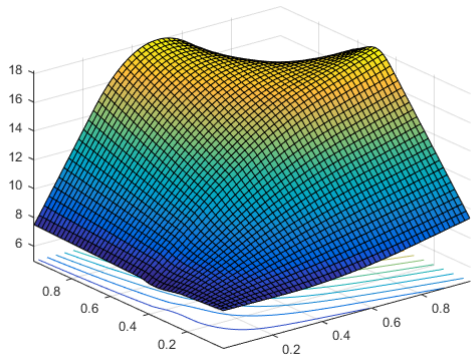
From the first table, we can see that the last ANOVA function (the one comprised of the 1st and the 5th input variable) gives relatively small contribution and maybe its basis functions should be deleted. From the second table, we can see that the 4th variable has the highest relative importance while the 3rd and 5th have the lowest (ignoring the last five variables not included in the model).

Let's say we want to delete the two basis functions comprising the 7th ANOVA function. Deletion of basis functions can be done using function `aresdel`. The function requires that we supply indices of the basis functions to delete. We can find those indices using `model.knotdims`. In our case they happen to be number 16 and number 17.

```
modelSmaller = aresdel(model, [16 17], X, Y);
```

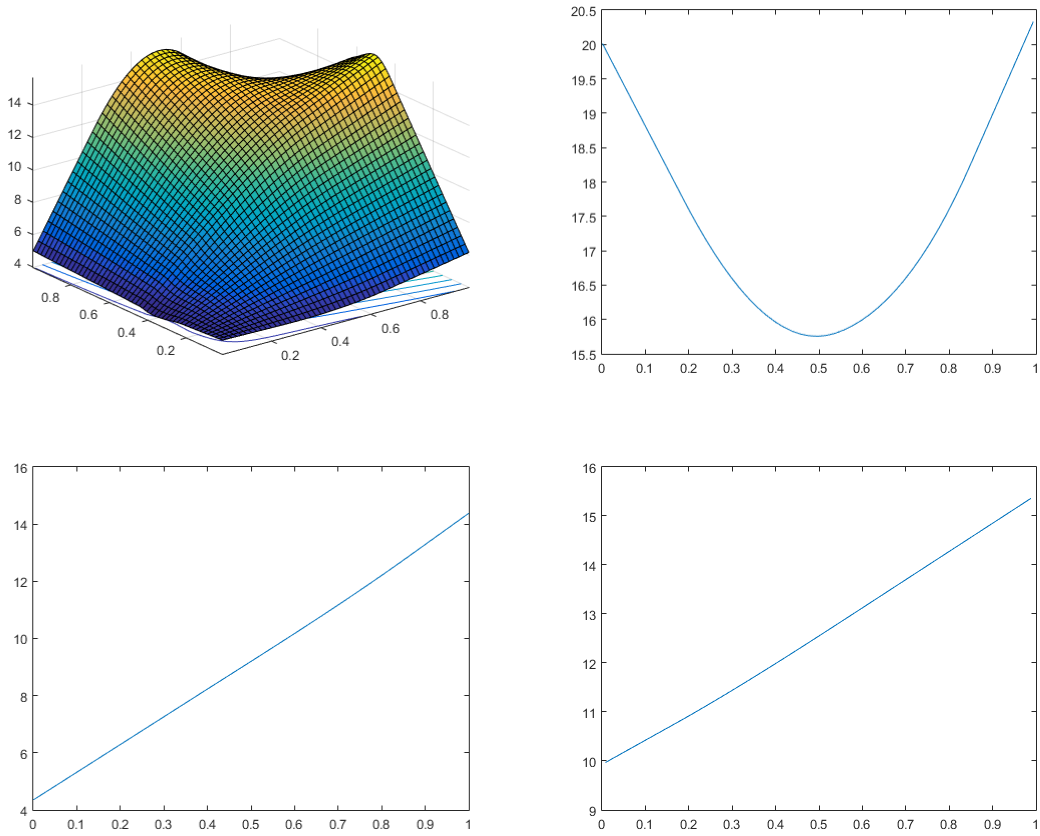
Function `aresplot` is used to plot ARES models. Let's make a plot using the first two variables, while values for all the other variables will be fixed at the middle of their ranges.

```
vals = [NaN NaN (modelSmaller.minX(3:end) + modelSmaller.maxX(3:end)) ./ 2];
aresplot(modelSmaller, [], [], vals);
```



We can also plot ANOVA functions if they comprise one or two input variables. This allows us to visualize the contributions of the ANOVA functions. Let's plot pair-wise (for variables x_1 and x_2; three ANOVA functions) and individual (for variables x_3, x_4, and x_5; one ANOVA function each) contributions of variables.

```
modelReduced = aresanovareduce(model, [1 2]);
aresplot(modelReduced)

for i = 3 : 5
    Xtmp = zeros(101,length(model.minX));
    Xtmp(:,i) = [model.minX(i):((model.maxX(i)-model.minX(i))/100):model.maxX(i)]';
    figure
    modelReduced = aresanovareduce(model, i);
    plot(Xtmp(:,i), arespredict(modelReduced, Xtmp));
end
```



To evaluate performance of this ARES configuration on the data using 10-fold Cross-Validation, use `arescv`. Note that for more stable results one should consider repeating Cross-Validation several times (see description of the argument `nCross` for function `arescv`).

```
rng(1);
results = arescv(X, Y, params)

results =
       MAE: 0.5089
       MSE: 0.4263
      RMSE: 0.6470
     RRMSE: 0.1304
        R2: 0.9826
    nBasis: 17.1000
     nVars: 5
    maxDeg: 2
```

To print the equation of the model with all its basis functions, use `areseq`. Note that by default the piecewise-cubic spline smoothing is hidden so that the output is more readable. As a result, the printed model looks like a piecewise-linear model but the coefficients are for our actual piecewise-cubic model (for piecewise-linear coefficients see the next model below). To see all the cubic smoothing calculations, set the `hideCubicSmoothing` argument to `false`.

```
areseq(model, 5);

BF1 = max(0, x4 -0.73434)
BF2 = max(0, 0.73434 -x4)
BF3 = max(0, x1 -0.48679)
BF4 = max(0, 0.48679 -x1)
BF5 = max(0, x2 -0.4357)
BF6 = max(0, 0.4357 -x2)
BF7 = max(0, x5 -0.31343)
BF8 = max(0, 0.31343 -x5)
BF9 = max(0, x3 -0.3774)
BF10 = BF3 * max(0, x2 -0.48969)
BF11 = BF3 * max(0, 0.48969 -x2)
BF12 = BF4 * max(0, x2 -0.42825)
BF13 = BF4 * max(0, 0.42825 -x2)
BF14 = max(0, 0.70405 -x3)
BF15 = BF3 * max(0, x2 -0.72244)
BF16 = BF7 * max(0, x1 -0.86869)
BF17 = BF7 * max(0, 0.86869 -x1)
y = 11.477 +10.938*BF1 -9.7047*BF2 +7.3369*BF3 -14.105*BF4 +11.409*BF5 -16.066*BF6
+5.7534*BF7 -4.935*BF8 +14.366*BF9 -35.948*BF10 -9.5483*BF11 -22.149*BF12 +34.821*BF13
+12.142*BF14 -45.775*BF15 -32.841*BF16 -1.9156*BF17
WARNING: Piecewise-cubic spline smoothing hidden.
```

Let's try piecewise-linear modelling.

```
params = aresparams(21, [], false, [], [], 2);
model = aresbuild(X, Y, params)

model =
            MSE: 0.2996
            GCV: 0.4893
          coefs: [18x1 double]
       knotdims: {17x1 cell}
      knotsites: {17x1 cell}
       knotdirs: {17x1 cell}
        parents: [17x1 double]
    trainParams: [1x1 struct]
           minX: [1x10 double]
           maxX: [1x10 double]

rng(1);
results = arescv(X, Y, params)

results =
       MAE: 0.5763
       MSE: 0.5338
      RMSE: 0.7242
     RRMSE: 0.1459
        R2: 0.9782
    nBasis: 17.1000
     nVars: 5
    maxDeg: 2
```

Finally, we print the equation of the model with all its basis functions.

```
areseq(model, 5);

BF1 = max(0, x4 -0.73434)
BF2 = max(0, 0.73434 -x4)
BF3 = max(0, x1 -0.48679)
```

```
    BF4 = max(0, 0.48679 -x1)
    BF5 = max(0, x2 -0.4357)
    BF6 = max(0, 0.4357 -x2)
    BF7 = max(0, x5 -0.31343)
    BF8 = max(0, 0.31343 -x5)
    BF9 = max(0, x3 -0.3774)
    BF10 = BF3 * max(0, x2 -0.48969)
    BF11 = BF3 * max(0, 0.48969 -x2)
    BF12 = BF4 * max(0, x2 -0.42825)
    BF13 = BF4 * max(0, 0.42825 -x2)
    BF14 = max(0, 0.70405 -x3)
    BF15 = BF3 * max(0, x2 -0.72244)
    BF16 = BF7 * max(0, x1 -0.86869)
    BF17 = BF7 * max(0, 0.86869 -x1)
    y  =  11.664  +10.758*BF1  -9.708*BF2  +8.5785*BF3  -13.385*BF4  +9.838*BF5  -15.279*BF6
+6.2628*BF7  -4.3835*BF8  +13.418*BF9  -32.827*BF10  -13.016*BF11  -16.532*BF12  +32.427*BF13
+11.22*BF14  -45.029*BF15  -46.087*BF16  -2.493*BF17
```

3.2. Two-dimensional function without noise

We start by creating training and test data using a two-dimensional noise-free function. The training data consists of 121 observations distributed in a regular 11×11 grid. The test data has 10000 observations distributed randomly.

```
clear X;
[tmpX1,tmpX2] = meshgrid(-1:0.2:1, -1:0.2:1);
X(:,1) = reshape(tmpX1, numel(tmpX1), 1);
X(:,2) = reshape(tmpX2, numel(tmpX2), 1);
clear tmpX1; clear tmpX2;
Y = sin(0.83*pi*X(:,1)) .* cos(1.25*pi*X(:,2));
Xt = rand(10000,2);
Yt = sin(0.83*pi*Xt(:,1)) .* cos(1.25*pi*Xt(:,2));
```

There is no noise and the data is plenty – we can hope for a very accurate model. We set the maximum number of basis functions to 101 (including the intercept term), no penalty for knots, piecewise-cubic modelling, and maximum interaction level equal to 2 (the number of input variables), leaving all the other parameters to their defaults.

```
params = aresparams(101, 0, true, [], [], 2);
```

ARES model is built by calling `aresbuild`.

```
model = aresbuild(X, Y, params)
```

As the model building process ends, we can examine the data structure of the new model. The model has 48 basis functions including the intercept term.

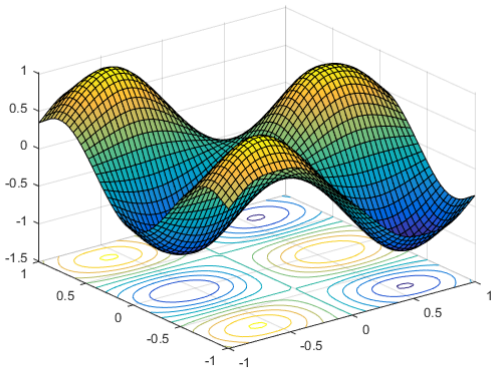
```
model =
            MSE: 1.5734e-04
            GCV: 4.3227e-04
          coefs: [48x1 double]
       knotdims: {47x1 cell}
      knotsites: {47x1 cell}
       knotdirs: {47x1 cell}
        parents: [47x1 double]
    trainParams: [1x1 struct]
             t1: [47x2 double]
             t2: [47x2 double]
           minX: [-1 -1]
           maxX: [1 1]
```

We test the model using the test data.

```
results = arestest(model, Xt, Yt)

results =
      MAE: 0.0115
      MSE: 2.0000e-04
     RMSE: 0.0141
    RRMSE: 0.0253
       R2: 0.9994
```

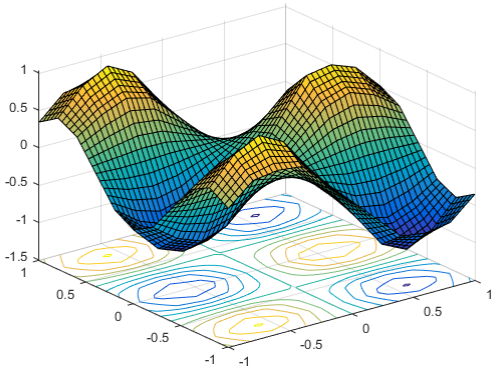
Plot the surface of the model.

```
aresplot(model);
```



Let's try doing the same but instead of piecewise-cubic modelling we will use piecewise-linear.

```
params = aresparams(101, 0, false, [], [], 2);
model = aresbuild(X, Y, params);
results = arestest(model, Xt, Yt)

results =
      MAE: 0.0409
      MSE: 0.0023
     RMSE: 0.0482
    RRMSE: 0.0862
       R2: 0.9926

aresplot(model);
```



3.3. Using Cross-Validation to select the number of basis functions

One of the ways to select the "best" number of basis functions for ARES models (i.e., value for argument `maxFinalFuncs` for function `aresparams`) or to confirm that the GCV criterion is indeed making good choices, is to evaluate all the candidate models in the pruning phase using Cross-Validation and compare which one would be chosen by GCV and which one by Cross-Validation.

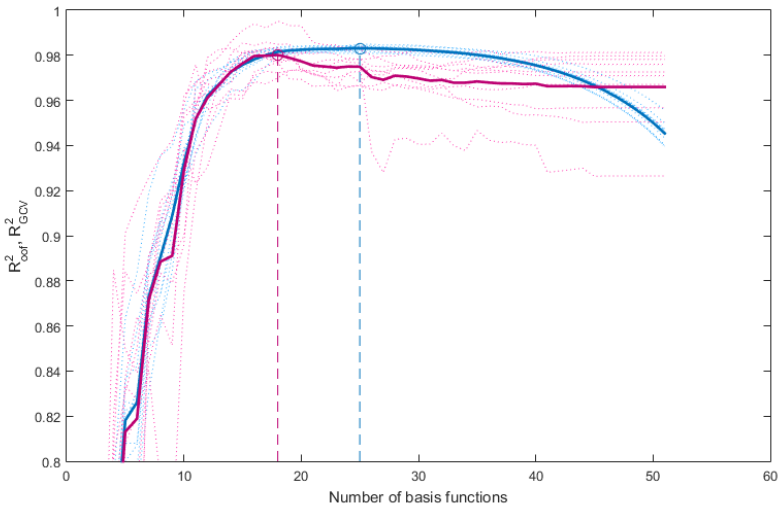
This can be done using function `arescv` by setting its argument `evalPruning` to `true`. In each Cross-Validation iteration, a new ARES model is built and pruned as usual using the GCV in the in-fold (training) data, but additionally in the pruning phase for all candidate models we also calculate out-of-fold error so that model of each size has now both, R^2 (Coefficient of Determination) estimated using GCV, as well as R^2 estimated using out-of-fold data.

Let's try this with the data from Section 3.1. We will use 10-fold Cross-Validation. Note that for more stable results one should consider repeating Cross-Validation several times (this can be set up using the argument `nCross` of function `arescv`).

```
params = aresparams(51, [], true, [], [], 2);

rng(1);
[~, ~, resultsPruning] = arescv(X, Y, params, 10, [], [], [], [], true);
```

Here's a code for plotting the results:

```
figure;
hold on;
for i = 1 : size(resultsPruning.R2GCV,1)
    plot(resultsPruning.R2GCV(i,:), ':', 'Color', [0.2590 0.7060 1]);
    plot(resultsPruning.R2oof(i,:), ':', 'Color', [1 0.2590 0.7060]);
end
plot(resultsPruning.meanR2GCV, 'Color', [0 0.4470 0.7410], 'LineWidth', 2);
plot(resultsPruning.meanR2oof, 'Color', [0.7410 0 0.4470], 'LineWidth', 2);

ylim = get(gca, 'ylim');
posY = resultsPruning.meanR2GCV(resultsPruning.nBasisR2GCV);
plot([resultsPruning.nBasisR2GCV resultsPruning.nBasisR2GCV], [ylim(1) posY], '--', 'Color', [0 0.4470 0.7410]);
plot(resultsPruning.nBasisR2GCV, posY, 'o', 'MarkerSize', 8, 'Color', [0 0.4470 0.7410]);
posY = resultsPruning.meanR2oof(resultsPruning.nBasisR2oof);
plot([resultsPruning.nBasisR2oof resultsPruning.nBasisR2oof], [ylim(1) posY], '--', 'Color', [0.7410 0 0.4470]);
plot(resultsPruning.nBasisR2oof, posY, 'o', 'MarkerSize', 8, 'Color', [0.7410 0 0.4470]);

xlabel('Number of basis functions');
ylabel('R^{2}_{oof}, R^{2}_{GCV}');
hold off;
```



The blue dotted lines show the R^2_{GCV} (R^2 estimated using GCV) for models of each fold. The blue solid line is the mean R^2_{GCV} for each model size (i.e., the average of the blue dotted lines). The pink dotted lines show the out-of-fold R^2 for models of each fold. The pink solid line is the mean out-of-fold R^2 for each model size (i.e., the average of the pink dotted lines).

The two vertical dashed lines are at the maximum of the two solid lines, i.e., they show the optimum number of terms estimated by GCV (blue) and Cross-Validation (pink). Ideally, the two vertical lines would coincide. In practice, the they are usually close but not identical. In our case the two lines are at 18 (for R^2_{oof}) and 25 (for R^2_{GCV}). This information can be used to set the number of basis functions for the final ARES model (argument `maxFinalFuncs` for function `aresparams`).

Such statistics can also be generated for different values of the GCV penalty per knot (argument `c` for function `aresparams`) to see how the parameter influences the selection of the final model. Just call `arescv` once for each considered value of `c` and compare the graphs.

4. REFERENCES

1. Friedman J.H. Multivariate Adaptive Regression Splines (with discussion), The Annals of Statistics, Vol. 19, No. 1, 1991a, 141 p.
2. Friedman J.H. Estimating functions of mixed ordinal and categorical variables using adaptive splines, Technical Report No. 108, Laboratory for Computational Statistics, Department of Statistics, Stanford University, 1991b, 49 p.
3. Friedman J.H. Fast MARS, Department of Statistics, Stanford University, Tech. Report LCS110, 1993
4. Hastie T., Tibshirani R., Friedman J. The elements of statistical learning: Data mining, inference and prediction, 2nd edition, Springer, 2009
5. Merkwirth C. and Wichard J. A Matlab toolbox for ensemble modelling, 2003, available at http://www.j-wichard.de
6. Milborrow S., Earth: Multivariate Adaptive Regression Spline Models (derived from code by T. Hastie and R. Tibshriani), 2015, R package available at http://cran.r-project.org/src/contrib/Descriptions/earth.html
7. Rudy J., py-earth: A Python implementation of Jerome Friedman's Multivariate Adaptive Regression Splines, 2015, available at https://github.com/jcrudy/py-earth