ARESLab

Adaptive Regression Splines toolbox for Matlab

ver. 1.4

Gints Jekabsons

Institute of Applied Computer Systems
Riga Technical University
Meza 1/3, LV-1048, Riga, Latvia
E-mail: gints.jekabsons@rtu.lv
URL: http://www.cs.rtu.lv/jekabsons/

Reference manual

May, 2010

CONTENTS

1. INTRODUCTION

What is ARESLab

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

The toolbox allows building models (referred to as ARES models) using different settings, testing them on a separate test set or using k-fold Cross-Validation, using them for prediction, outputting equations for deployment, plotting the models etc. The built models can also be used as metamodels (also known as surrogate models) for design optimization tasks (e.g., see Chen et al. 2006, Kalnins et al. 2008, Kalnins et al. 2009, Jekabsons 2010).

This reference 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. 2 or any later version.

Some parts of `aresbuild` and `createList` functions are derived from ENTOOL toolbox (Merkwirth & Wichard 2003, Norgaard 2000) which also falls under the GPL licence.

For any feedback on the toolbox including bug reports feel free to contact me.

Details

The ARESLab toolbox is written entirely in Matlab. I tried to implement the main functionality of the MARS technique for regression as close to the description in the Friedman's original paper (Friedman 1991) as possible. While implementing the knot placement part (see remarks about `minSpan` and `endSpan` in Section 2), I also took a look at the source code of the R Earth package (Milborrow 2009) and implemented it very similarly to Earth version 2.4-0. The only major difference at the moment I think is that the model building is not accelerated using "Fast MARS" queuing (Friedman 1993) together with the "fast least-squares update technique" (Friedman 1991). This difference however affects more the speed of the algorithm execution rather than the predictive performance of built models.

The absence of "Fast MARS" queuing means that the code might be rather slow for large data sets (however see the function descriptions on how to make it faster by setting more conservative values for algorithm parameters). Note that a much faster version of Multivariate Adaptive Regression Splines is included in VariReg software tool (Jekabsons 2009, available at http://www.cs.rtu.lv/jekabsons/) which also can be put to work from within the Matlab environment (although with much less functionality). Another alternative is to use the Earth package for R which is very sophisticated however lacks the ability to create piecewise-cubic models.

Possible future updates for the toolbox:

- optional complete re-training of an existing model (for slightly changed data);
- setting the upper limit of interactivity for each input variable separately;
- automatic variable scaling;
- modelling for classification problems (although for two classes, one can code the output as 0/1, treat the problem as a regression, and use the current version of ARESLab).

Some further aspects of MARS mentioned in Friedman's papers but not implemented in ARESLab:

- "Fast MARS" queuing;
- automatic handling of missing values;
- automatic handling of categorical input variables (with the current version of ARESLab, the user must create a number of dummy variables in the usual way before building the model).

Citing the ARESLab toolbox

Please give a reference to the webpage in any publication describing research performed using the toolbox e.g., like this:

Jekabsons G., ARESLab: Adaptive Regression Splines toolbox for Matlab, 2010, 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 configuration for ARES model building algorithm for further use with `aresbuild`, `arescv`, or `arescvc` functions;
- `arespredict` – makes predictions using an ARES model;
- `arestest` – tests an ARES model on a test data set;
- `arescv` – tests ARES performance using k-fold Cross-Validation;
- `arescvc` – finds the "best" value for penalty *c* (Generalized Cross-Validation penalty per knot) from a set of candidate values using k-fold Cross-Validation and MSE;
- `aresplot` – plots surface of an ARES model;
- `areseq` – outputs the ARES model in an explicit mathematical form;
- `aresanova` – performs ANOVA decomposition;
- `aresanovareduce` – reduces an ARES model according to ANOVA decomposition.

2.1. Function `aresbuild`

Purpose:

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

Call:

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

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

Input:

`Xtr, Ytr`	: Training data cases (`Xtr`(*i*,:), `Ytr`(*i*)), $i = 1,...,n$. Note that it is recommended to pre-scale `Xtr` values to [0,1] (Friedman 1991) and to standardize `Ytr` values (Milborrow 2009). 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 technique 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 1991).
`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 data case weights; if supplied, the algorithm calculates the sum of squared errors multiplying the squared residuals by the supplied weights. The length of `weights` vector must be the same as the number of data cases (i.e., *n*). The weights must be nonnegative.
`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 e.g., in `arescvc` function.
`verbose`	: Set to `false` for no verbose. (default value = `true`)

Output:

`model`	: The built ARES model – a structure with the following elements:
`coefs`	: Coefficient vector of the regression model (for the intercept term and each basis function).
`knotdims`	: Cell array of indexes of used input variables for each knot in each basis function.
`knotsites`	: Cell array of knot sites for each knot and used input variable in each basis function.
`knotdirs`	: Cell array of directions (-1 or 1) of the hinge functions for each used input variable in each basis function.
`parents`	: Vector of indexes of direct parents for each basis function (0 if there is no direct parent or it is the intercept term).
`trainParams`	: A structure of training parameters for the algorithm (the same as in the input).
`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 GCV is calculated using `trainParams.c` argument (for details on GCV calculation, see Friedman 1991). The value may also be `Inf` if model's effective number of parameters (see Eq. 1) is larger than or equal to *n*.
`t1`	: For piecewise-cubic models only. Matrix of knot sites for the additional side knots on the left of the central knot.
`t2`	: For piecewise-cubic models only. Matrix of knot sites for the additional side knots on the right of the central knot.
`minX`	: Vector of minimums for input variables (used for `t1` and `t2` placements as well as for model plotting).
`maxX`	: Vector of maximums for input variables (used for `t1` and `t2` placements as well as for model plotting).
`endSpan`	: The used value of `endSpan`.
`time`	: Algorithm execution time (in seconds)

Remarks:

The model building 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) the difference between `err` and `newErr` is smaller than `trainParams.threshold`, where `newErr` is calculated by dividing sum of squared residuals by the variance of `Ytr` and `err` is the `newErr` value from the previous iteration;
3) the `newErr` is smaller than `trainParams.threshold`;
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 *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 again 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 2009):

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

where MSE_{train} is Mean Squared Error of the evaluated model in the training data, n is the number of data cases 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 not only the number of model's basis functions but also the number of knots. Also note that in ARESLab in the situation when $enp \geq n$ the GCV value will be equal to `Inf` (the model is considered infinitely bad).

After the pruning, the largest possible final model has $k = \mathrm{int}((n + c/2)/(1 + c/2))$ basis functions for `maxInteractions` > 1 and $k = \mathrm{int}((n + c/3)/(1 + c/3))$ basis functions for `maxInteractions` = 1. In the forward phase the models may also get larger than this however for such models GCV = `Inf` as then $enp \geq n$.

2.2. Function `aresparams`

Purpose:

Creates a structure of ARES configuration parameter values for further use with `aresbuild`, `arescv`, or `arescvc` functions.

Call:

```
trainParams = aresparams(maxFuncs, c, cubic, cubicFastLevel, selfInteractions,
maxInteractions, threshold, prune, useMinSpan, useEndSpan, maxFinalFuncs)
```

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

Input:

For most applications, it can be expected that the most attention should be paid to the following parameters: `maxFuncs`, `c`, `cubic`, `maxInteractions`, and maybe `threshold`.

`maxFuncs` : The maximal number of basis functions included in the model in the forward model building phase (before pruning in the backward phase). Includes the intercept term. (default value = 21). The recommended value for this parameter is two times the number of basis functions in the final model (Friedman 1991). While building a model, the number may also not be reached because the number of coefficients in the model cannot exceed the number of input data cases (or because of some stopping criterion – see remarks for details).

`c` : Generalized Cross-Validation (GCV) penalty per knot. Theory suggests values in the range of about 2 to 4. Larger values will lead to fewer knots being placed (i.e., final models will be simpler). A value of 0 penalizes only terms, not knots (can be useful e.g., with lots of data and low noise). The recommended (and default) value is 3 (Friedman 1991). Note that if `maxInteractions = 1` (additive modelling) then function `aresbuild` will recalculate c so that the actually used value is $2c$ / 3; this is recommended for additive modelling (Friedman 1991).

`cubic` : Whether to use piecewise-cubic (`true`) or piecewise-linear (`false`) type of modelling (Friedman 1991). It is expected that the piecewise-cubic modelling will give higher predictive performance for smoother and less noisy data. (default value = `true`) `cubicFastLevel` : In ARESLab, there are three types (levels) of piecewise-cubic modelling implemented. In level 0 cubic modelling for each candidate model is done in both phases of the technique (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. Levels 0 and 1 may bring extra precision in the modelling process however the results can actually also be worse. It is expected that the two much slower levels will mostly be not worth the waiting.

`selfInteractions` : The maximum degree of self interactions for any input variable. In ARESLab, it can be larger than 1 only for piecewise-linear modelling. Usually the self interactions are never allowed. (default 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. The larger the value of `threshold` the potentially simpler models are generated (see remarks section of `aresparams` and `aresbuild` for details). Default value = 1e-4. For noise-free data the value may be lowered.

`prune` : Whether to perform the model pruning (the backward phase) or not. (default value = `true`)

`useMinSpan` : In order to lower the local variance of the estimates, a minimum span is imposed that makes the technique resistant to runs of positive or negative error values between knots (by jumping over a (`minSpan`) number of data cases each time the next potential knot placement is requested) (Friedman 1991). `useMinSpan` allows to disable (set to 0 or 1) the protection so that all the data cases are considered for knot placement in each dimension (except, see `useEndSpan`). Disabling `minSpan` may enable to create a model which is more responsive to local variations in the data however this can lead to an overfitted model even for noise-free data. Setting the `useMinSpan` to > 1, enables also 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 technique 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 (`endSpan`) to the end of data interval) (Friedman 1991). `useEndSpan` allows to disable (set to 0 or 1) the protection so that all the data cases are considered for knot placement in each dimension (except, see `useMinSpan`). Disabling `endSpan` may enable to create a model which is more responsive to local variations in the data however this can lead to an overfitted model even for noise-free data. Setting the `useMinSpan` to > 1, enables also 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. (default value = `Inf`).

Output:

`trainParams` : A structure of training parameters for `aresbuild` function containing the provided values of the parameters (or default ones, if not provided).

Remarks:

The knot placement in `aresbuild` is implemented very similarly to R Earth package version 2.4-0 (Milborrow 2009) with calculations of `minSpan` and `endSpan` values using formulas given in Eq. 45 and Eq. 43 of the Friedman's original paper (Friedman 1991) 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 (including the intercept term) which is used for generation of new basis functions.

If more speed is required, one can try some of the following options:

1) decreasing `maxFuncs` (less iterations in the forward phase);
2) increasing `cubicFastLevel` or turning the piecewise-cubic modelling completely off (setting the level below 2 makes the procedure considerably slower; however note that if `cubicFastLevel` = 2, turning the piecewise-cubic modelling off will give almost no speed gain);
3) decreasing `selfInteractions` (less candidate models in the forward phase);
4) decreasing `maxInteractions` (less candidate models in the forward phase);
5) increasing `threshold` (may result in less iterations in the forward phase);
6) manually increasing `useMinSpan` and `useEndSpan` (less candidate models in the forward phase).

Note that decreasing the number of iterations or candidate models in the forward phase may also result in underfitted final models.

2.3. Function `arespredict`

Purpose:

Predicts output values for the given query points using an ARES model.

Call:

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

Input:

`model` : ARES model

`Xq` : Inputs of query data points (`Xq`(*i*,:)), *i* = 1,...,*nq*

Output:

`Yq` : Predicted outputs of the query data points (`Yq`(*i*)), *i* = 1,...,*nq*

2.4. Function `arestest`

Purpose:

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

Call:

```
[MSE, RMSE, RRMSE, R2] = arestest(model, Xtst, Ytst)
```

Input:

`model`	: ARES model
`Xtst, Ytst`	: Test data cases (`Xtst`(*i*,:), `Ytst`(*i*)), *i* = 1,...,*ntst*

Output:

`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.

Call:

```
[avgMSE, avgRMSE, avgRRMSE, avgR2, avgTime] = arescv(X, Y, trainParams,
weights, k, shuffle, cvc_cTry, cvc_k, verbose)
```

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

Input:

`X, Y`	: Data cases (`X`(*i*,:), `Y`(*i*)), *i* = 1,...,*n*
`trainParams`	: See function `aresbuild`.
`weights`	: See function `aresbuild`.
`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 data cases before performing Cross-Validation. Note that the random seed value can be controlled externally before calling `arescv`. (default value = `true`)
`cvc_cTry, cvc_k`	: `cTry` and k values for `arescvc` function. Supply these values if you want to perform another Cross-Validation for finding the "best" penalty *c* value in each iteration of the outer Cross-Validation loop of `arescv`. (default values = [], meaning that a fixed *c* is used)
`verbose`	: Set to `false` for no verbose. (default value = `true`)

Output:

`avgMSE`	: Average Mean Squared Error
`avgRMSE`	: Average Root Mean Squared Error
`avgRRMSE`	: Average Relative Root Mean Squared Error
`avgR2`	: Average Coefficient of Determination
`avgTime`	: Average execution time

2.6. Function `arescvc`

Purpose:

Finds the "best" value for penalty c from a set of candidate values using k-fold Cross-Validation and MSE.

Call:

```
cBest = arescv(X, Y, trainParams, cTry, weights, k, shuffle, verbose)
```

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

Input:

`X, Y`	: Data cases (`X`(*i*,:), `Y`(*i*)), $i = 1,...,n$
`trainParams`	: See function `aresbuild`.
`cTry`	: A set of candidate values for *c*. (default = 1:5)
`weights`	: See function `aresbuild`.
`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 data cases before performing Cross-Validation. Note that the random seed value can be controlled externally before calling `arescvc`. (default value = `true`)
`verbose`	: Set to `false` for no verbose. (default value = `true`)

Output:

`cBest`	: The “best” value for penalty *c*.

Remarks:

This function finds the best penalty *c* value using Cross-Validation in a clever way using function `aresbuild` i.e., in each CV 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 surface of an ARES model.

Call:

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

All the arguments, except the first one, of this function 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 in `vals` using `NaN` values. By default the two first variables will be varied and all the other will be fixed at (*max* – *min*) / 2.
`gridSize`	: Grid size. (default value = 50)

2.8. Function `areseq`

Purpose:

Outputs the ARES model in an explicit mathematical form (useful e.g., for deployments of built ARES models in other software).

Call:

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

Input:

`model`	: ARES model
`precision`	: Number of digits in the model coefficients and knot sites.

Output:

`eq`	: A cell array of equations for individual basis functions and the main model. Note that the outputted equations will have piecewise-linear form even if the model itself is really piecewise-cubic. The model coefficients however will be those from the original piecewise-cubic model.

2.9. Function `aresanova`

Purpose:

Performs ANOVA decomposition (see Sections 3.5 and 4.3 of the original paper by Jerome Friedman (Friedman 1991) for details) of the given ARES model and reports the results.

Call:

```
aresanova(model, Xtr, Ytr)
```

Input:

`model`	: ARES model
`Xtr, Ytr`	: Training data cases (`Xtr`(*i*,:), `Ytr`(*i*)), $i = 1,...,n$.

Remarks:

To understand the table outputted by the function, here is n excerpt from the original paper by Jerome Friedman (Friedman 1991) Section 4.3:

> *"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."*

2.10. Function `aresanovareduce`

Purpose:

Deletes all the basis functions from an ARES model 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 be plotted using function aresplot to visualize the contributions.

Call:

```
[modelReduced usedBasis] = aresanovareduce(model, vars, exact)
```

Input:

`model`	: ARES model
`vars`	: A vector of indexes for input variables to stay in the model. The size of the vector should be between 1 and *d*, where *d* is 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:

`modelReduced`	: The reduced model
`usedBasis`	: The list of original indexes for the used basis functions

3. EXAMPLES OF USAGE

3.1. Ten-dimensional function with noise

We start by creating a data set using a ten-dimensional function with Gaussian noise. The data consists of 200 cases randomly uniformly distributed in a ten-dimensional unit hypercube.

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

We define the maximal number of basis functions to be 21 (including the intercept term), and limit maximum interaction level to 2 (only pairwise products of basis functions will be allowed), leaving all the other parameters to their defaults. The model will be of piecewise-cubic type as it is the default.

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

Now the 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 final model. It has 16 basis functions including the intercept term.

```
model =
          coefs: [16x1 double]
       knotdims: {15x1 cell}
      knotsites: {15x1 cell}
       knotdirs: {15x1 cell}
        parents: [15x1 double]
    trainParams: [1x1 struct]
            MSE: 0.3234
            GCV: 0.4960
             t1: [15x10 double]
             t2: [15x10 double]
           minX: [1x10 double]
           maxX: [1x10 double]
        endSpan: 10
```

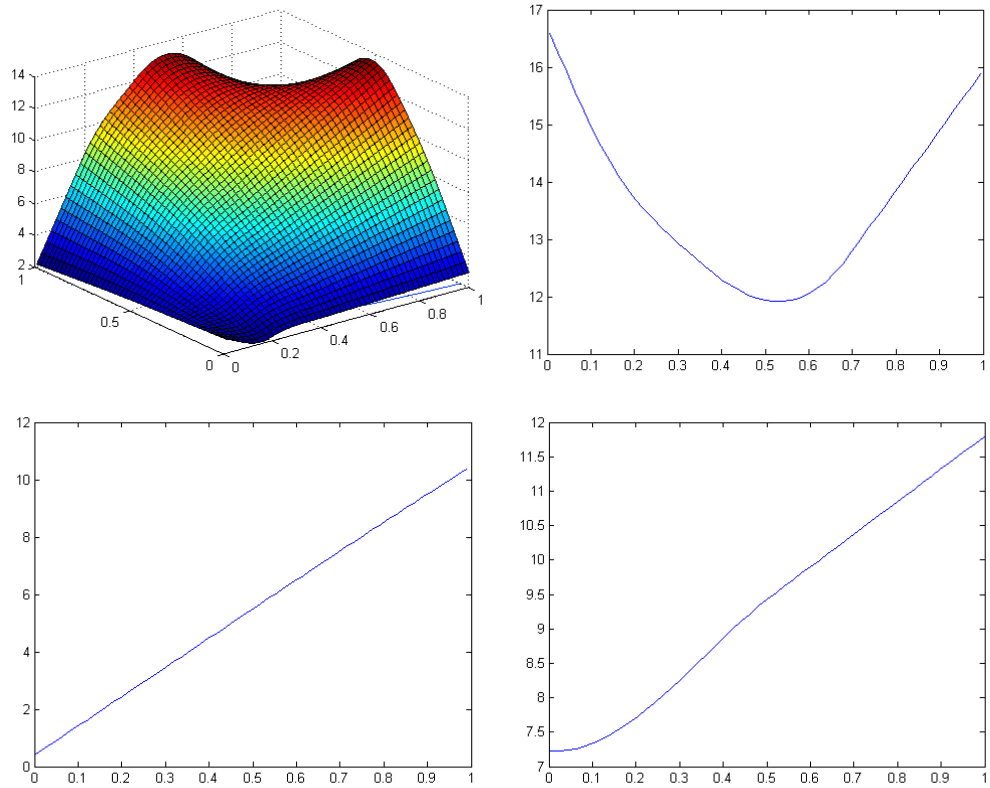
Now we can perform ANOVA decomposition.

```
aresanova(model, X, Y)

Type: piecewise-cubic
GCV: 0.496
Total number of basis functions: 16
Total effective number of parameters: 38.5
ANOVA decomposition:
Func.    STD   GCV         #basis      #params     variable(s)
1        4.675 52.925      2            5.0        1
2        2.621 10.342      2            5.0        2
3        1.281 59.738      2            5.0        3
4        2.956 17.665      2            5.0        4
5        1.445  8.879      1            2.5        5
6        3.732 37.938      5           12.5        1 2
7        0.289  0.617      1            2.5        3 5
```

We can see that the last ANOVA function gives very small contribution and maybe should be deleted (here this ANOVA function corresponds to one basis function which uses input variables number 3 and number 5):

Let's plot pair-wise (for variables x_1 and x_2) and individual (for variables x_3, x_4, and x_5 contributions of variables.

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

for i = 3 : 5
    modelReduced = aresanovareduce(model, i);
    Xtmp = zeros(51,10);
    Xtmp(:,i) = [model.minX(i):((model.maxX(i)-model.minX(i))/50):model.maxX(i)]';
    figure
    plot(Xtmp(:,i), arespredict(modelReduced, Xtmp));
end
```



Now let's evaluate predictive performance of this ARES configuration on the data using 5-fold Cross-Validation.

```
rand('state',0);
[avgMSE, avgRMSE, avgRRMSE, avgR2] = arescv(X, Y, params, [], 5)

avgMSE = 0.4427
avgRMSE = 0.6543
avgRRMSE = 0.1339
avgR2 = 0.9818
```

Now let's try piecewise-linear modelling.

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

```
model =
           coefs: [16x1 double]
        knotdims: {15x1 cell}
       knotsites: {15x1 cell}
        knotdirs: {15x1 cell}
         parents: [15x1 double]
     trainParams: [1x1 struct]
             MSE: 0.3802
             GCV: 0.5830
            minX: [1x10 double]
            maxX: [1x10 double]
         endSpan: 10
```

```
rand('state',0);
[avgMSE, avgRMSE, avgRRMSE, avgR2] = arescv(X, Y, params, [], 5)

avgMSE = 0.5254
avgRMSE = 0.7165
avgRRMSE = 0.1469
avgR2 = 0.9783
```

Finally we output the equation of the piecewise-linear model with all its basis functions.

```
areseq(model, 5);

BF1 = max(0, x4 -0.66938)
BF2 = max(0, 0.66938 -x4)
BF3 = max(0, x2 -0.57231)
BF4 = max(0, 0.57231 -x2)
BF5 = max(0, x1 -0.23961)
BF6 = max(0, 0.23961 -x1)
BF7 = max(0, x5 -0.036179)
BF8 = max(0, 0.553 -x3)
BF9 = BF3 * max(0, x1 -0.5981)
BF10 = BF3 * max(0, 0.5981 -x1)
BF11 = BF4 * max(0, x1 -0.14539)
BF12 = BF4 * max(0, 0.14539 -x1)
BF13 = max(0, x3 -0.16723)
BF14 = BF7 * max(0, x3 -0.8273)
BF15 = BF5 * max(0, x2 -0.25707)
y = 7.6853 +9.8771*BF1 -10.312*BF2 +15.52*BF3 -6.1597*BF4 +15.121*BF5 -20.444*BF6 +4.9892*BF7
+15.757*BF8 -65.272*BF9 -28.664*BF10 -25.491*BF11 +53.099*BF12 +9.7016*BF13 +18.575*BF14 -18.91*BF15
```

3.2. Noise-free two-dimensional function

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

```
clear
[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));
```

Such noise-free functions can be approximated very precisely. We define the maximal number of basis functions to be 121, no penalty for knots, and maximum interaction level equal to 2 (the number of input variables), leaving all the other parameters to their defaults. The model will be of piecewise-cubic type as it is the default.

```
params = aresparams(121, 0, [], [], [], 2);
```

Now the 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 44 basis functions including the intercept term.

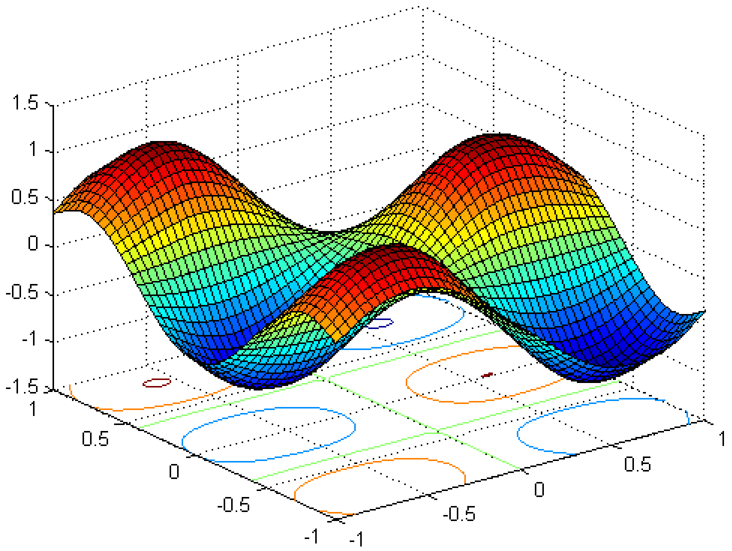
```
model =
          coefs: [44x1 double]
       knotdims: {43x1 cell}
      knotsites: {43x1 cell}
       knotdirs: {43x1 cell}
        parents: [43x1 double]
    trainParams: [1x1 struct]
            MSE: 1.6901e-004
            GCV: 4.1734e-004
             t1: [43x2 double]
             t2: [43x2 double]
           minX: [-1 -1]
           maxX: [1 1]
        endSpan: 8
```

We test the model using test data.

```
[MSE, RMSE, RRMSE, R2] = arestest(model, Xt, Yt)

MSE = 1.9909e-004
RMSE = 0.0141
RRMSE = 0.0244
R2 = 0.9994
```

Plot the surface of the model.

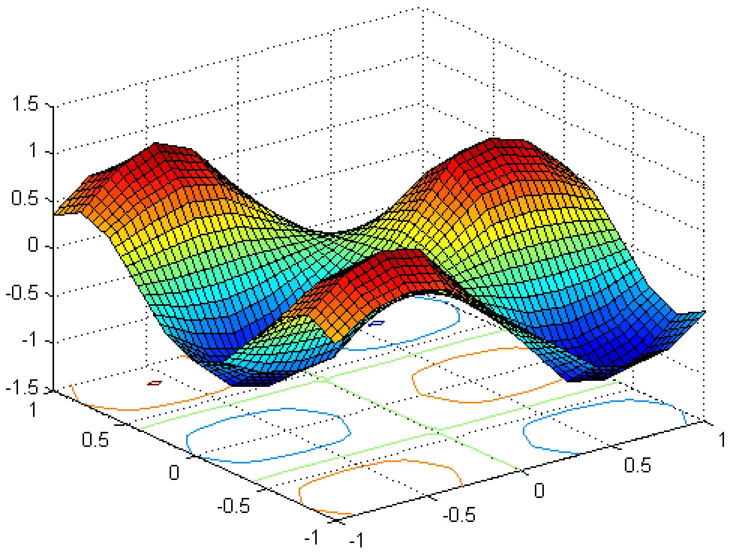
```
aresplot(model);
```



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

```
params = aresparams(100, 0, false, [], [], 2);
model = aresbuild(X, Y, params);
[MSE, RMSE, RRMSE, R2] = arestest(model, Xt, Yt)

MSE = 0.0023
RMSE = 0.0480
RRMSE = 0.0829
R2 = 0.9931
```

```
aresplot(model);
```



4. REFERENCES

1. Chen V.C.P., Tsui K-L., Barton R.R., Meckesheimer M. A review on design, modeling and applications of computer experiments, IIE Transactions, Vol. 38, No. 4, 2006, pp. 273-291.
2. Friedman J.H. Multivariate Adaptive Regression Splines (with discussion), The Annals of Statistics, Vol. 19, No. 1, 1991, pp. 1-141
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. Jekabsons G., VariReg software tool, 2010, available at http://www.cs.rtu.lv/jekabsons/
6. Jekabsons G. Adaptive Basis Function Construction: an approach for adaptive building of sparse polynomial regression models. Machine Learning, Yagang Zhang (ed.), In-Tech, 2010, pp. 127-156.
7. Kalnins K., Jekabsons G., Rikards R. Metamodels for optimisation of post-buckling responses in full-scale composite structures. Proceedings of 8th World Congress on Structural and Multidisciplinary Optimization, WCSMO 2009, Lisbon, Portugal, 2009 (CD edition paper #1659, 9 pages)
8. Kalnins K., Ozolins O., Jekabsons G. Metamodels in design of GFRP composite stiffened deck structure. Proceedings of 7th ASMO-UK/ISSMO International Conference on Engineering Design Optimization, Association for Structural and Multidisciplinary Optimization in the UK (ASMO-UK), ISBN: 978-0853162728, Bath, UK, 2008, pp. 263-273.
9. Merkwirth C. and Wichard J. A Matlab toolbox for ensemble modelling, 2003, available at http://www.j-wichard.de/
10. Milborrow S., Earth: Multivariate Adaptive Regression Spline Models (derived from code by T. Hastie and R. Tibshriani), 2009, R package available at http://cran.r-project.org/src/contrib/Descriptions/earth.html
11. Norgaard M. Neural network based system identification toolbox, ver. 2, Tech. Report. 00-E-891, Department of Automation, Technical University of Denmark, 2000