Fuzzy Modeling with Adaptive Simulated Annealing

Hime Aguiar e Oliveira Junior (hime AT engineer DOT com)

July/2004

Abstract

A new method for data-based fuzzy system modeling is presented. The approach uses Takagi-Sugeno models and Adaptive Simulated Annealing (ASA) to achieve its goal .

The problem to solve is well defined - given a training set containing a finite number of input-output pairs, construct a fuzzy system that approximates the behavior of the real system that originated that set , within a pre-established precision . Such an approximation must have generalization ability to be useful in the real world, considering the finiteness of the training set and other constraints .

Introduction

Black box , realistic fuzzy system modeling typically gives rise to global nonlinear optimization problems. By "black box" , I mean modeling only from observed data, not taking into account any prior knowledge about the system under study. So, supposing the system with n inputs and one output – the so called MISO (Multiple Input Single Output) system – all we have is a training set , formed by points in (n+1)-dimensional Euclidean space . The first n components of each (n+1)-tuple are the inputs to the given system and the last coordinate is the corresponding response .

Generally, the aim is to develop a mathematical model that behaves like the real system within a certain error .In our case, that error is the function to minimize – it depends on the training set elements and model parameters in a strongly nonlinear manner. Besides, the resulting functions present high-dimensional domains and the COD (Curse Of Dimensionality) comes into action .Restricting our attention to fuzzy models, the usual choice is to employ Mamdani or Takagi-Sugeno systems, since there are theoretical results assuring their approximating properties under very reasonable conditions.

In this paper, I'll describe results obtained from TS models , but the same arguments apply in the Mamdani case, after some little effort – the global optimization algorithm used in this work (ASA) just receives a certain number of parameters, process them and gives them back in an interactive way, until the final convergence.

Affine Takagi-Sugeno fuzzy systems

Their structure consists of rules in the following form:

$$R_i : IF\ x\ IS\ A_i\ THEN\ y_i = a_0^i + \sum_{k=1}^{n} a_k^i x_k\ ,$$

where

$- x = (x_1, x_2, ..., x_n) \in D \subset R^n$ *represents the input*

$- A_i$ *is a fuzzy set with membership function defined on D*

$- y_i \in R$ *is the scalar output corresponding to rule i*

$- a_k^i \in R$ *are the parameters associated with rule i*

$- i \in \{1, ..., NR\}$

$- NR =$ *Total number of rules*

The global output of such a MISO system is given by

$$y(x) = \frac{\sum_{k=1}^{NR} \mu_{A_k}(x) y_k}{\sum_{k=1}^{NR} \mu_{A_k}(x)}$$

where, as above, $\mu_{A_k}(.)$ *is the membership function associated with the* $i-th$ *rule.*

Sometimes, it can be easier to represent the antecedent part as

$IF\ x_1\ IS\ A_{i1}\ AND\ x_2\ IS\ A_{i2}\ AND.........\ AND\ x_n\ IS\ A_{in}$
, for example, where A_{ik} *has a scalar membership function.*

In this case, the multidimensional function can be computed as

$$\mu_{A_i}(x) = \mu_{A_i}(x_1, x_2, ..., x_n) = \bigwedge_{k=1}^{n} \mu_{A_{ik}}(x_k)$$

Of course, we can use other t-norms than the minimum operator, such as the product one – we'll do that in the sequel.

The problem to be solved

Given "p" (n+1)-tuples representing the actual behavior of a MISO system, build a TS fuzzy system that, when submitted to the same inputs (first n coordinates) produces an approximation of the true output (last coordinate).

The "p" tuples constitute the TRAINING SET and can be arranged in tabular form:

$i_{11}\ i_{12}\ i_{13}.....i_{1n}\quad o_1$
$i_{21}\ i_{22}\ i_{23}....i_{2n}\quad o_2$
– – – – – – – – – – –
. .
. .
. .
– – – – – – – – – – –
$i_{p1}\ i_{p2}\ i_{p3}\ i_{pn}\quad o_p$

It's important to use TEST and VALIDATION SETS to measure the quality of the final model (generalization ability, etc.) . Those sets look exactly like training sets, but aren't used in the fitting process itself .

Adaptive Simulated Annealing

In the implementation of the training phase I used ASA (Adaptive Simulated Annealing) , a very efficient and flexible stochastic global optimization method that simplified the whole construction as we'll see later .As the name says, it is based on the Simulated Annealing method, created in a simpler form (Monte Carlo importance-sampling technique) by N. Metropolis and others .

Along the decades, the method evolved and nowadays we have several good reasons to use it as a global optimization tool:

- There are theoretical results assuring its convergence to the global minimum (or maximum) of a given function under very realistic conditions.
- The function to minimize don't need to be differentiable or even continuous.
- There are many good implementations of "flavors" of SA – among them we can find the excellent ASA code (www.ingber.com) .

So , ASA can be described as a statistical method that is used to globally minimize (or maximize) general multidimensional numerical functions defined on hyper-rectangles (Cartesian product of real intervals).It's also possible to do constrained and integer optimization with ASA , due to its flexible structure and large number of configuration parameters. For additional details on the internal structure of ASA , please refer to [2] and additional documentation in www.ingber.com .

Description of the adopted approach – Part ONE

Initially, we solve the approximation problem by means of the following steps:

Step1 – Normalize the training set by scaling the input and output variables and transforming the original intervals into [0,1].

Suppose the training set has p elements and is given by:

$$((i_{11,}i_{12,}...i_{1n,}),o_{1,})$$
$$((i_{21,}i_{22,}...i_{2n,}),o_{2,})$$
$$.................$$
$$.................$$
$$..................$$
$$((i_{p1,}i_{p2}...i_{pn,}),o_{p,})$$

We get a new (normalized) training set by computing the linear transformations

$$i_{kl}^{'} = \frac{i_{kl} - \min(i_{1l}, i_{2l}, i_{3l}, ..., i_{pl})}{\max(i_{1l}, i_{2l}, i_{3l}, ..., i_{pl}) - \min(i_{1l}, i_{2l}, i_{3l}, ..., i_{pl})}$$

and

$$o_l^{'} = \frac{o_l - \min(o_1, o_2, o_3, ..., o_p)}{\max(o_1, o_2, o_3, ..., o_p) - \min(o_1, o_2, o_3, ..., o_p)}$$

From this point on we work with the new training set ,that represents a mapping from [0,1]x[0,1]x...x[0,1] into [0,1] .
It's important to save some parameters from the original TS so that we cam de-normalize the final results , using the inverse transformation.

Step 2 – At this point , we have a new function to fit by means of a Takagi-Sugeno fuzzy system - both have as their domain the n-dimensional unitary cube and their images will be subsets of [0,1]. Our choice - in this first method - was to define five input fuzzy terms over the universe of discourse [0,1] and evaluate each dimension separately . After that , the activation degree is found by composing the several membership values.
The functions were named VERY LOW (VL) , LOW (L) , ZERO (Z) ,HIGH (H) and VERY HIGH (VH) and their definitions are shown below

$$\begin{aligned} VL(x) &= 1 \quad , x < 0 \\ &= 1\text{-}3x \text{ , x in } [0,1/3] \\ &= 0 \quad , x > 1/3 \end{aligned}$$

$$L(x) = \begin{cases} 3x & , x \text{ in } [0,1/3) \\ 3(1-2x) & , x \text{ in } [1/3,1/2) \\ 0 & , x > 1/2 \end{cases}$$

$$Z(x) = \begin{cases} 2(3x-1) & , x \text{ in } [1/3,1/2) \\ 2(2-3x) & , x \text{ in } [1/2,2/3) \\ 0 & , \text{otherwise} \end{cases}$$

$$H(x) = \begin{cases} 3(2x-1) & , x \text{ in } [1/2,2/3) \\ 3(1-x) & , x \text{ in } [2/3,1] \\ 0 & , \text{otherwise} \end{cases}$$

$$VH(x) = \begin{cases} 0 & , x < 2/3 \\ 3x-2 & , x \text{ in } [2/3,1] \\ 1 & , x > 1 \end{cases}$$

Each fuzzy rule will have the following general form

$$R_i : IF\ x_1\ IS\ T_{i_1}\ AND \cdots AND\ x_n\ IS\ T_{i_n}\ THEN\ y_i = a_{i0} + \sum_{j=1}^{n} a_{ij} x_j ,$$

($y_i \in [0,1]$)

At first , and to investigate the efficiency of the ASA method in this kind of task, our choice was to allow linguistic variables to assume each of the 5 terms above. By doing so , we arrived at the following table relating the dimension of the input space and the number of parameters and rules to find in each case

Dimension	Number of rules	Number of parameters
1	5	10
2	25	75
3	125	500
n	5^n	$(n+1)5^n$

For input dimensions greater than 3 the COD shows up and even a powerful algorithm needs an alternative approach to finish the minimization task in an acceptable period of time , taking into account the computational power available at present . I'll describe such algorithm in the next section.

Step 3 – Now , it's time to synthesize the error function that's going to guide the fitting process and will be used by the ASA algorithm as its cost function. We'll adopt the so called batch training method, in which all deviations (relative to the training set) are taken into account simultaneously , opposite to the incremental learning approach.

That said, we define the expression of the error function as

$$\sum_{k=1}^{p} E_k^2 \quad , \quad p = \text{cardinality of training set}$$

$$E_k = y_k^{app} - o_k \quad , \qquad y_k^{app} = \frac{\sum_{i=1}^{R} AD(i)(x) * y_i(x)}{\sum_{i=1}^{R} AD(i)(x)} \qquad (x \in R^n)$$

$R = \text{Number of rules}$

$o_k = \text{Output part of the } k-\text{tuple of the TS}$

$AD(i)(x) = \text{Activation degree of rule } i \text{ at point } x \in R^n, \text{defined as}$
$AD(i)(x) = \prod_{l=1}^{n} \mu_{T_{il}}(x_l)$,
$y_i(x) = \text{Output of rule } i \text{ corresponding to input } x$
defined as $\quad y_i(x) = a_{i0} + \sum_{k=1}^{n} a_{ik} x_k$,

$x = (x_1, x_2, ..., x_n) \in R^n$

a_{ij} are the free parameters that the global minimization algotithm will use to fit the original function.

Step 4 – Having defined the cost function, all we have to do is to start the training phase by activating the ASA algorithm, that will guide the fitting process .

We show an example of application in the figure below, including the curve of training error against the number of function evaluations

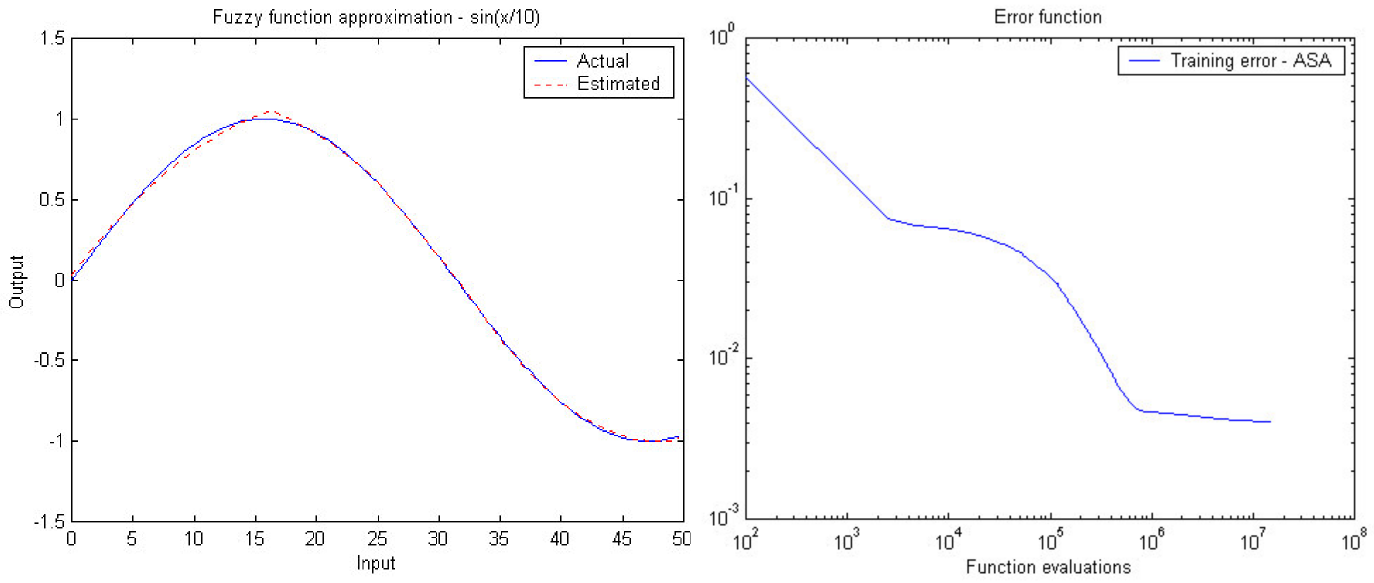


FIGURE 1

Description of the adopted approach – Part TWO

The previous method works well for lower dimensional input spaces, but for higher number of dimensions it was necessary to build a less sensitive algorithm .

We now describe a different and more adequate method for application in higher dimensional systems :

Step 1 – Identical to the previous one – the training set is normalized.

Step 2 – The new training set is submitted to a clustering process that produces a set of cluster centers. Each center will define a multidimensional membership function given by

$$\mu_{x_c}(x) = \frac{2}{(1 + e^{10*NC*d(x,x_c)})},$$

where NC is the total number of cluster centers and d(.,.) is the Euclidean distance .

that will be used to synthesize the fuzzy rules like

$$R_i \; : \; \text{IF } x \text{ IS } A_i \text{ THEN } y_i = a_{i0} + \sum_{k=1}^{n} a_{ik} x_k$$

where A_i is the fuzzy set whose membership function is defined above.

We observe that each cluster center will originate one fuzzy rule and the number of clusters is defined in advance. In the implementation, we have used the Kohonen SOM (Self Organizing Map) to realize the clustering (or vector quantization , if you prefer) , but any similar mechanism will do the intended task, such as FCM, Gustafson-Kessel fuzzy clustering algorithm, etc. . Our aim is to get a finite (and as small as possible) set of points that can "represent" the given training set. It's also possible to improve the approximation accuracy (and we'll do that) by tuning the location of cluster centers.

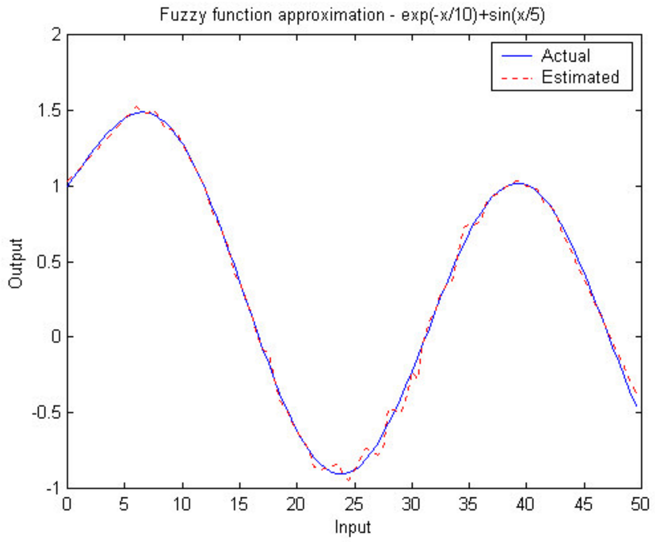
Step 3 – The expression of the error function is the same as before, with a different activation degree

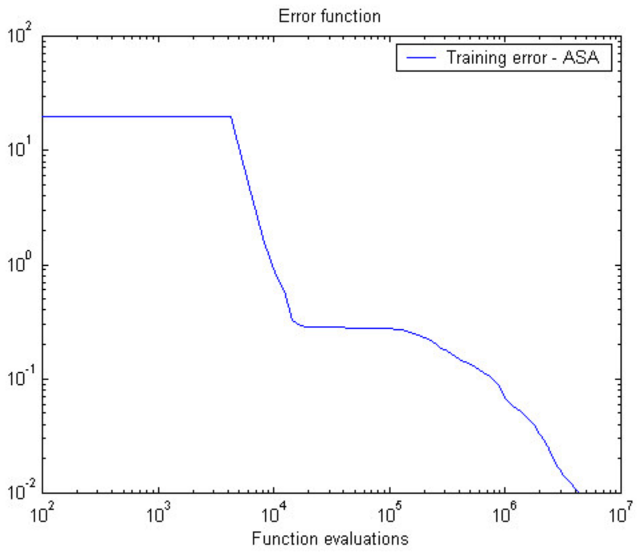
$$AD(i)(x) = \mu_{A_i}(x) = \frac{2}{1 + \exp(10 * NC * d(x, x_c))}$$

An advantage of this type of membership function is that it's defined over the whole input domain and its evaluation is very simple.

Step 4 – Same as in part one – the ASA algorithm is started with the error function as its cost function. This time the number of free parameters is R(2n+1), where R is the number of rules (the same as of cluster centers) and n is the dimension of the input space – to each rule corresponds 1 cluster center (n parameters) and n+1 consequent coefficients. For example, if we choose R=16 and the input space has dimension 3, the number of parameters to find is 16(2x3+1)=112 . In the first method, we would have 500 parameters to fit, that is certainly a harder task to achieve.

Below we can find some results of the second approach and the corresponding training error curves





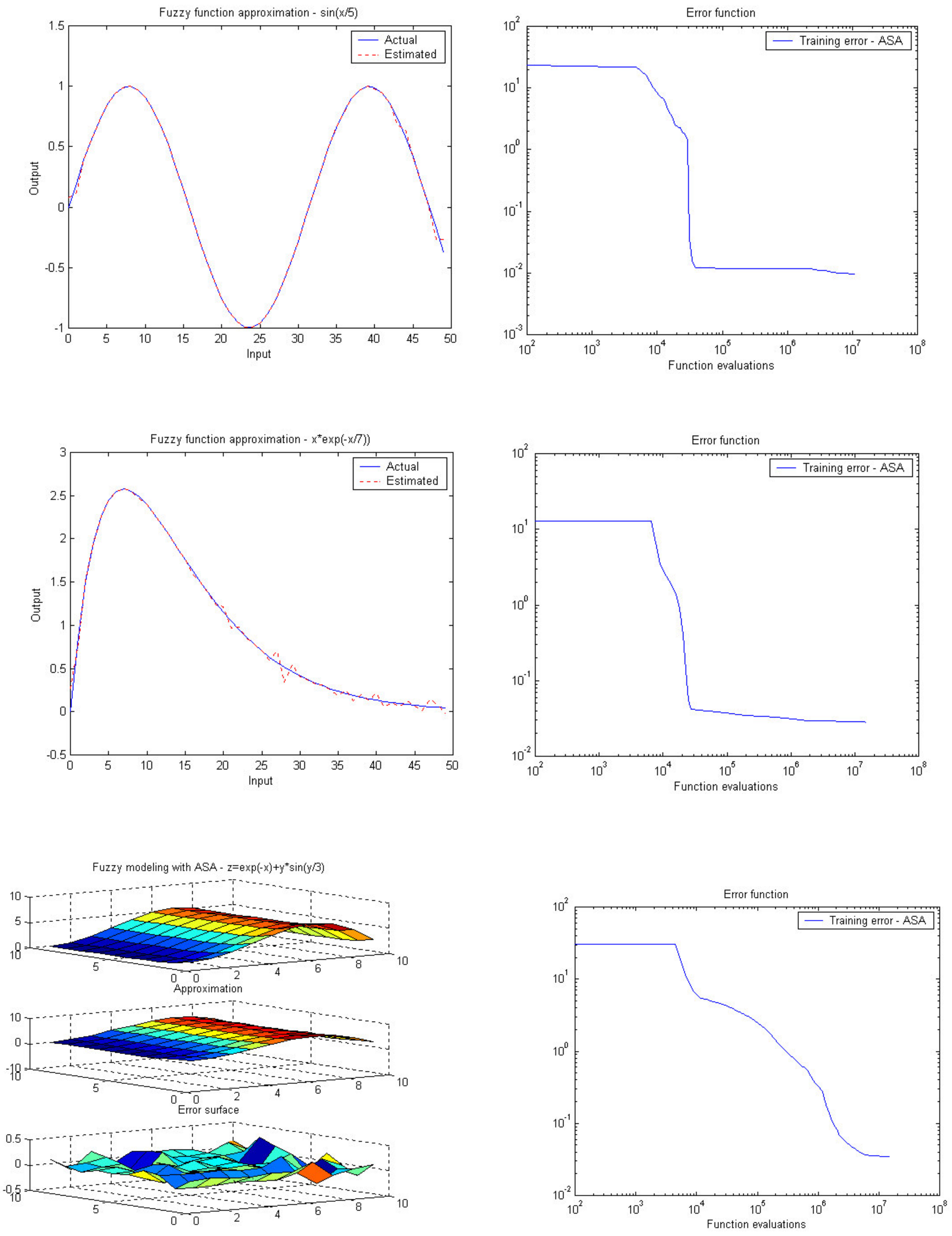

Fuzzy function approximation - sin(x/5)
Actual
Estimated
Output
Input
Error function
Training error - ASA
Function evaluations
Fuzzy function approximation - x*exp(-x/7))
Actual
Estimated
Output
Input
Error function
Training error - ASA
Function evaluations
Fuzzy modeling with ASA - z=exp(-x)+y*sin(y/3)
Approximation
Error surface
Error function
Training error - ASA
Function evaluations

FIGURE 2

Implementation details

The implementation was done by means of a few modules whose task is to compute the error function, serialize and de-serialize the parameter vector presented to the (Fuzzy) ASA main routine, generate code corresponding to the final Takagi-Sugeno fuzzy system and create some auxiliary files related to the fitting process.

I've used the C language (gcc compiler/LINUX operating system).

I'll show just some code related to the second (clustering) approach, although both of them were coded and tested. Note that , in this case , the coefficients present in the consequent part of the rules AND the cluster centers' coordinates are adjusted during the training.

As I mentioned before, the metric used in the multidimensional membership functions was coded as

```
static double METRIC(int NOOFCLUSTERS,double DISTANCE)
{
      double aux;

      aux = 2/(1+exp(10*NOOFCLUSTERS*DISTANCE)) ;

      return aux ;
}
```

And a sample , automatically generated module implementing a trained fuzzy system is

```
#include <math.h>
#include "neurofuzzyASAkoh.h"

#include "metrica.h"

#define NOOFRULES 24
#define NOOFDIMENSIONS 1
#define LIMINFY -0.905447
#define LIMSUPY 1.482035

double LIMINFX[NOOFDIMENSIONS] = {
0.000000 ,
};
double LIMSUPX[NOOFDIMENSIONS] = {
49.000000 ,
};

int indice;
double Distancia(double *,double *,int);
double resultado;
double *ponteCEN[NOOFRULES];

double CENTROS[NOOFRULES][NOOFDIMENSIONS] = {
{  0.104563 , },
{  0.166978 , },
{  0.369484 , },
{  0.501125 , },
{  0.621556 , },
{  0.692647 , },
{  0.159181 , },
{  0.316846 , },
{  0.457436 , },
{  0.585923 , },
{  0.699786 , },
{  0.777816 , },
{  0.211800 , },
{  0.294220 , },
{  0.416952 , },
{  0.544356 , },
{  0.688868 , },
{  0.839139 , },
{  0.287393 , },
{  0.364804 , },
{  0.495139 , },
{  0.638977 , },
{  0.841994 , },
{  0.893483 , },

            };

double CoefTSK[NOOFRULES][NOOFDIMENSIONS+1] = {
{ 0.809942 ,1.698251 } ,
{ 0.971276 ,-0.186623 } ,
{ 0.810681 ,-1.776344 } ,
{ -2.174582 ,4.332631 } ,
{ 1.978076 ,-2.766474 } ,
{ -2.854453 ,4.999999 } ,
{ 0.782019 ,1.573650 } ,
{ 2.072977 ,-4.999995 } ,
{ -0.084302 ,0.216232 } ,
{ 0.813728 ,-1.092641 } ,
{ 1.032576 ,-0.462558 } ,
{ -0.026735 ,1.052491 } ,
{ 1.525753 ,-2.991132 } ,
{ 1.430221 ,-3.022035 } ,
{ 1.172736 ,-2.581313 } ,
{ 0.500654 ,-0.811309 } ,
{ 1.775324 ,-1.869671 } ,
{ 0.373963 ,0.537639 } ,
{ 1.597618 ,-3.322790 } ,
{ 0.603078 ,-0.595101 } ,
{ 2.413492 ,-4.898921 } ,
{ -0.728927 ,1.803645 } ,
{ 2.014128 ,-1.538158 } ,
{ 3.681011 ,-3.421391 } ,

};
double FuzzySystem( double *X ) {

double  GRAUATIVACAO , SAIDA ;double SAIDA_GLOBAL = 0 ; double
SOMAGRAUS = 0 ;
int j,k;
double XNorm[NOOFDIMENSIONS];double dista;
```

```
for (j=0;j<NOOFRULES;j++)
      ponteCEN[j] = CENTROS[j];

      for ( j=0 ; j < NOOFDIMENSIONS ; j++)
            XNorm[j] = (X[j] - LIMINFX[j])/(LIMSUPX[j] - LIMINFX[j]);

 for (j=0;j < NOOFRULES ; j++)
{
      dista = Distancia(XNorm,ponteCEN[j],NOOFDIMENSIONS);
        GRAUATIVACAO = METRICA(NOOFRULES,dista) ;

      SAIDA = CoefTSK[j][0] ;

for ( k=1 ; k < NOOFDIMENSIONS + 1 ; k++ )
       SAIDA += CoefTSK[j][k] * XNorm[k-1] ;

       SAIDA_GLOBAL += SAIDA*GRAUATIVACAO ;
       SOMAGRAUS += GRAUATIVACAO ;
              }   //// for j

             return LIMINFY + (SAIDA_GLOBAL/SOMAGRAUS)*(LIMSUPY-
LIMINFY) ;

      }
```

To evaluate the quality of the generated fuzzy systems, we can use something like this (example in dimension 1)

```
#include <stdio.h>
#include <math.h>

extern double FuzzySystem(double *);

#define NOITERA   50

main(int argc,char **argv)
{
double i;

static double baba[1] ;

double ERROTOTAL =0 , avali;
double ERROREL = 0 ;
double valfunc ;
int j;
FILE *fp = fopen("GRAFCLU1.DAT","w");

for (i=0;i<NOITERA;i+=.5 )
                { baba[0]=i;
                  avali = FuzzySystem(baba);
                  valfunc = exp(-i/10) + sin(i/5) ;

                 printf("\n%f => %f (should be %f)", i,avali
,valfunc);
```

```
              fprintf(fp,"%f %f %f\n",i,valfunc,avali);

                   ERROTOTAL += fabs(avali-valfunc);

               if (valfunc != 0)
                   ERROREL += fabs(avali-valfunc)/fabs(valfunc);
                else if (avali != 0)
                     ERROREL += fabs(avali-valfunc)/fabs(avali);

                 }

  printf("\nMean absolute error = %f\n",ERROTOTAL/(NOITERA*2));

  fclose(fp);


}
```

The coding task was not very hard, taking into account the availability of the ASA code that has done the difficult part. It's worth to note that I've used what I called Fuzzy ASA (see [5]) in all the examples, but the original ASA code could be used as well (it would be a good exercise to compare the two ways).

Conclusions

A new method for data based fuzzy modeling was presented. That approach can be used for function approximation, in general, and specific tasks such as fuzzy controller design and synthesis can be realized in a relatively uniform manner.

Although the focus was placed over Takagi-Sugeno fuzzy systems, the reasoning can be easily applied to Mamdani systems as well, taking into account that the underlying problem can be placed as the global minimization of a numeric error function. In particular, the present method is an alternative to the neuro-fuzzy modeling approach that, in the end, faces a global optimization problem.

Bibliography

1 – H. Hellendorm , D. Driankov (Eds) – Fuzzy Model Identification-Selected Approaches - Springer-Verlag – 1997

2 – Lester Ingber – Adaptive Simulated Annealing (ASA): Lessons Learned – www.ingber.com

3 – R. Babuska , H.B.Verbruggen - Constructing Fuzzy Models by Product Space Clustering - Chapter in [1] above.

4 – T. Kohonen - Self-Organizing Maps - Springer-Verlag – 2001

5 – Hime Aguiar e O. Jr. - Fuzzy Control of Stochastic Global Optimization Algorithms and Very Fast Simulated Reannealing – www.optimization-online.org