Implementation of a Modelica Library for Smooth Spline Approximation

Jörg Ungethüm Dirk Hülsebusch
German Aerospace Center, Institute of Vehicle Concepts
Pfaffenwaldring 38-40, 70569 Stuttgart
joerg.ungethuem@dlr.de dirk.huelsebusch@dlr.de

Abstract

In system modeling, table data interpolation is frequently used for e.g. characteristic maps or as replacement of a complicated system. In many cases, the available data are noisy and of limited accuracy. It turns out, that data approximation is advantageous against data interpolation in these cases. A Modelica library for 1-D and 2-D spline approximation on basis of external functions is presented in this paper.

Keywords: interpolation, approximation, polynomial splines, data tables

1 Introduction

The Modelica standard library provides table objects for 1-D and 2-D data interpolation. However, there are several shortcomings of these implementations:

- Table data with of more than 2 dimensions is not supported (extension up to 4 dimensions is presented in [1])
- Only linear and 3rd order spline interpolation but not data approximation is supported
- Step functions cannot be modeled by standard tables as grids are required to be strictly increasing
- 2-D table data must be given on a rectangular grid. As measurement data is commonly available only as scattered data, the actual interpolation table must be generated by some kind of interpolation which introduces new inaccuracy.
- In case of inputs out of the table data range, data is extrapolated using the nearest 2 data points. In many real world problems this type of data extrapolation is not suitable.

2 Approximation vs. Interpolation

In system modeling, table data are often used for measured data or as a replacement for a complex external calculation (e.g. media properties). In many cases the table data are prone to random and discretization errors. In Figure 1 a simple example of 1-D measured data is given. The discretization of the measurement device is clearly seen in the sampled data ($\Delta U = 0.01V$). Linear interpolation of these data leads to a partially stepwise constant curve, which obviously indicates zero gradients. Assuming a similar curve as a table based function definition this might introduce numerical problems. Unfortunately also cubic spline interpolation is not appropriate as the gradient of the spline is partially opposite to the gradient of the linear interpolation. It should be noted, that the spline interpolation introduces numerous local extrema which are not present in the original data. Again, this might lead to numerical trouble.

To overcome these difficulties, function approximation is occasionally used (e.g. in most media models). Unfortunately, finding suitable trial functions is

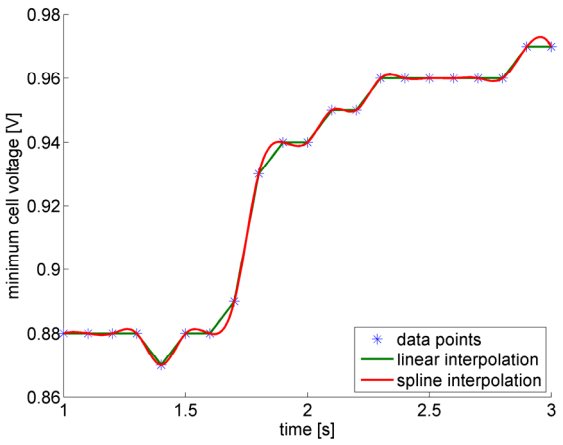

Figure 1 Linear and cubic spline interpolation of sample measured data points

not trivial at all. Only in very simple cases polynomials might be used.

Another approach is to use polynomial splines for piecewise approximation. That is, the knots of the generated splines are not fixed at the data points but selected by the algorithm. By means of the smoothing parameter s the smoothness of the approximation is controlled. Setting s to zero, an interpolating spline is constructed. Moving over to greater values of s, the closeness is relaxed for the benefit of smoothness. The smoother the approximation is the lesser knots are needed. Cubic spline approximation in 1D using different values for s is shown in Figure 2.

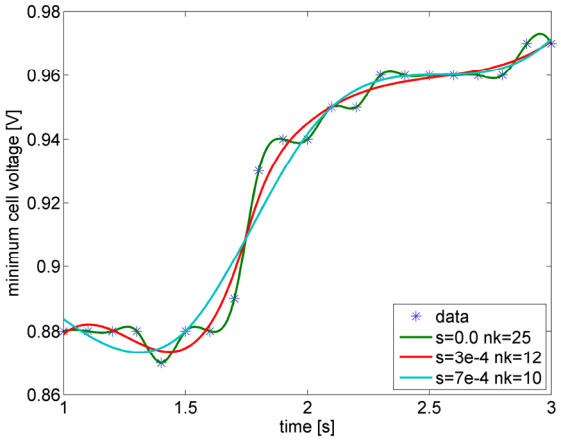

Figure 2 Cubic spline approximation of sample measured data points

3 Modelica implementation

The core routines for curve and surface fitting by spline approximation are public available in the library DIERCKX at Netlib [2]. Splines of order 1 to 5 are supported. DIERCKX routines are written in standard FORTRAN 77. To make them usable in the Modelica environment, only an interface had to be implemented. The DIERCKX library was translated into Ansi-C by f2c (also available from Netlib) to simplify mixed language programming.

The Modelica implementation uses the ExternalObject approach. Thus, splines are objects with constructor and destructor functions called automatically exactly once before the first respectively last use of the object. The internal representation of the splines as Ansi-C structures is completely hidden for the user. The spline generation involving dynamic memory allocation is done only during object initialization. Actual data interpolation is fast and does not involve memory allocation. As a nice feature, DIERCKX contains routines for derivative calculation. The first order derivatives are also available as Modelica functions. The polynomial order of the splines can be chosen between 1 and 5 whereas even numbers are strongly discouraged. Each data point might be weighted individually. The total number of spline objects is not limited nor is the size of data. For 1-D curve data fitting of periodic curves is supported. The data for surface fitting might be supplied on a rectangular grid in the same format that is used in the standard interpolation tables. However, also scattered input data are supported. In common, number and location of knots are chosen by the algorithm but this might be overridden by a user-specified knot selection.

4 Interfaces

The interface is designed with ease of use in mind. Thus, for any inputs reasonable defaults are supplied.

The basic usage of 1-D curve is:

```
/* spline initialization */
Import C=ApproxSpline.Curve1d;
C.Type curve=C.Type(data=data, s=0.01);
…
equation
/* spline data evaluation at x → y*/
y = C.eval(curve,x);
```

Any input arguments of the initialization routine except of the table data are optional. The inputs are:

data[:,2-3]	Data to be approximated as row wise triples: x,y,w (w is optional weight)
s = 0	Approximation smoothness
k = 3	Polynomial spline order (1..5)
periodic = false	Generate periodic spline if true
x_lim[2] = {min(data[:,1]), max(data[:,1])}	Lower and upper limits of the generated spline curve
t = fill(0,0)	if non-zero length, t is interpreted as user-specified knot selection

The order of the input data array does not matter as it is internally sorted. However, abscissa elements must be mutually unequal. The approximation smoothness has to be greater or equal zero. The generated spline approximation p(x) is found by minimizing the discontinuity of the k^{th} derivation whereas the condition

$$\sum_{i=1}^{n} w_i \left(y_i - p(x_i)\right)^2 \le s$$

is still fulfilled.

Per default, the limits of a non-periodic spline curve are set according to the minimum and maximum data value. However, the limits might be modified by parameter x_{lim}. This might be used e.g. for extrapolation purpose. In any case, outside of the limits the spline evaluation will return the boundary coordinate.

$$p(x) = p(x_{\max}) \quad | \, x \ge x_{\max}$$
$$p(x) = p(x_{\min}) \quad | \, x \le x_{\min}$$

In case of periodic splines, the period is given by

$$\Delta = \max(x) - \min(x)$$

The y-coordinate of x_{max} as well as any given limits are ignored in this case.

If automatic knot selection is not sufficient, user might specify knots as parameter array.

For convenience, a block interface is provided. The block dialog is shown in Figure 3.

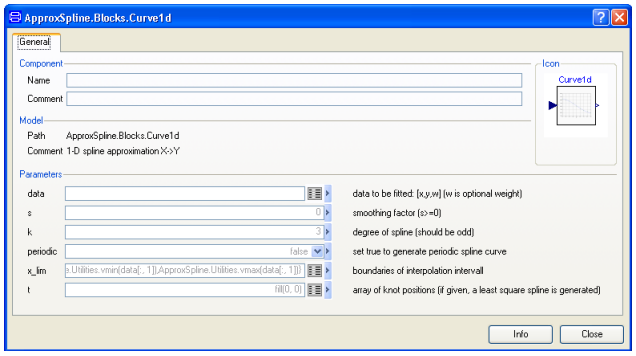

Figure 3 Parameter dialog of 1-D curve block

The interface of the 2-D surface is very similar to the 1-D case.

```
/* spline initialization */
Import S=ApproxSpline.Surf2d;

S.Type surf=S.Type(data=data, s=0.01)
…
equation
/* spline data evaluation [x,y] → y*/
y = S.eval(surf,x,y);
```

Any input arguments of the initialization routine except of the table data are optional. The inputs are:

rectangular=false	true if input data is on rectangular grid, otherwise assume scattered data
data[:,:]	Data to be approximated (either scattered or on rectangular grid)
s = 0	Approximation smoothness
kx = 3	Polynomial order in X-dir (1..5)
ky = 3	Polynomial order in Y-dir (1..5)
x_lim[2] = {min(x), max(x)}	Lower and upper limits in X-dir of the generated spline surface
y_lim[2] = {min(y), max(y)}	Lower and upper limits in Y-dir of the generated spline surface
tx = fill(0,0)	if non-zero length, tx is interpreted as user-specified X-dir knot selection
ty = fill(0,0)	if non-zero length, tx is interpreted as user-specified Y-dir knot selection

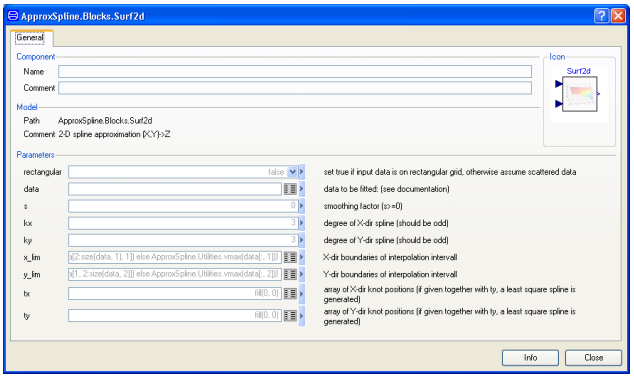

Figure 4 Parameter dialog of 2-D surface block

The input data might be provided on a rectangular grid or as scattered data. In the first case, the input format is similar to the standard Modelica interpolation table. In case of scattered input data, a simple table with 3 or 4 columns has to be provided as shown in Figure 5 whereas the 4th column is the optional weight. For rectangular data it is not possible to provide individual data point weights.

0	y_1	y_2	…	y_{ny}
x_1	$z_{1,1}$	$z_{1,2}$	…	$z_{1,ny}$
x_2	$z_{2,1}$	$z_{2,2}$	…	$z_{2,ny}$
⋮	⋮	⋮	⋱	⋮
x_{nx}	$z_{nx,1}$	$z_{nx,2}$	$z_{1,1}$	$z_{nx,ny}$

x	y	z	w
x_1	y_1	z_1	w_1
x_2	y_2	z_2	w_2
⋮	⋮	⋮	⋮
x_n	y_n	z_n	w_n

Figure 5 Input data schemes: rectangular grid (left) , scattered data (right)

5 Examples

As a typical example, a table based model of a turbo compressor is used. The model of the turbo compressor is built up from the equations:

The dimensionless isentropic compression work:

$$Y = \int_{p_1}^{p_2} v dp = RT_1 \frac{\kappa}{\kappa - 1}\left(\left(\frac{p_2}{p_1}\right)^{\frac{\kappa-1}{\kappa}} - 1\right)$$

By the use of dimensionless characteristic numbers the model becomes independent of variable inlet temperature and pressure. The most common characteristic numbers for turbo compressors are the head coefficient ψ and the flow coefficient φ

$$\psi = \frac{Y}{u^2/2} \qquad \varphi = \frac{\dot{V}}{\frac{\pi}{4}D^2 u}$$

whereas u is the wheel tip speed, D is the wheel diameter and $\dot{V}$ is the gas volume flow rate. The last characteristic number used is the tip speed Mach number, which is the ratio of tip speed and the sonic speed of gas.

$$Ma_u = \frac{u}{a}$$

The core of the compressor model are two 2-D lookup tables which maps head coefficient and tip speed Mach number to flow coefficient and inner efficiency.

$$\varphi = table_\varphi(\psi, Ma_u)$$
$$\eta_i = table_\eta(\psi, Ma_u)$$

In case of connecting both fluid ports with control volumes and the rotational flange with a rotating mass, the table inputs can be calculated directly from dynamic states. The mass flow rate is subsequently calculated from the flow coefficient.

$$\dot{V} = \varphi \max(u, 0)\frac{\pi D^2}{4}$$

$$\dot{m} = \frac{\dot{V}}{\rho_1}$$

The shaft power might be computed by means of the isentropic and the mechanic efficiency from mass flow rate and isentropic compression work. However, to avoid a singularity at zero speed, it is advantageous to calculate the shaft torque without introducing the mechanic power.

$$\tau = \varphi \frac{\pi}{8}\frac{Y}{\eta_i}\rho_1 i_G D^3$$

The model has been implemented both with standard Modelica table interpolation as well as with approximating spline approximation. Table data were taken from a commercial automotive air compressor. There are 56 data points available. From these scattered data, the gridded data as needed for Modelica standard tables is generated by interpolation. As an example, the interpolated data using a 20x20 grid is plotted in Figure 6. The corresponding 3rd order approximating spline surface is shown in Figure 7.

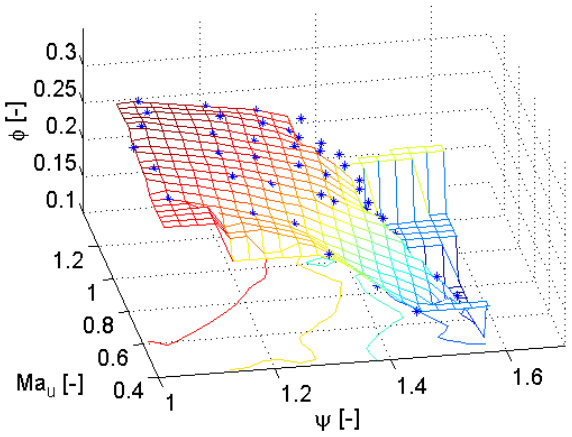

Figure 6 Flow coefficient as function of tip speed Mach number and head coefficient (mesh by interpolation, original data points as stars)

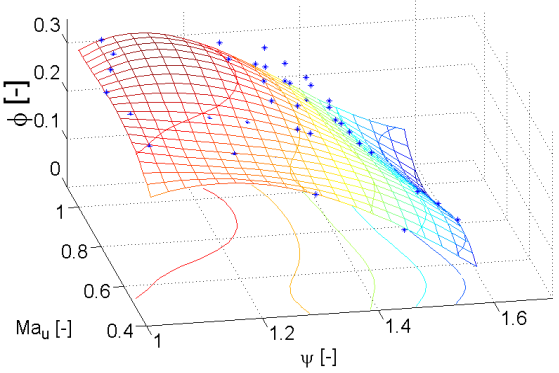

Figure 7 Flow coefficient as function of tip speed Mach number and head coefficient (mesh by spline approximation, original data points as stars)

The compressor model has been included into a simple test model which is shown in Figure 8. Both fluid ports of the compressor model are connected to control volumes. The left control volume is fed through a pipe from a boundary model. The right control volume blows off through an ideal nozzle. The compressor shaft is connected to a rotating mass which is driven by a speed-controlled torque. Starting with zero compressor speed, the torque accelerates the

shaft up to the given rotating speed set value. Simultaneously the volume flow rate through the compressor and the pressure in the right control volume rise. The test model was run with both compressor implementations. The comparison of flow coefficient and inner efficiency which are calculated by table lookup respectively spline approximation are shown in Figure 9 and Figure 10. Obviously the spline approximation gives a smoother curve which in turn leads to slightly better simulation performance (Table 1)[1].

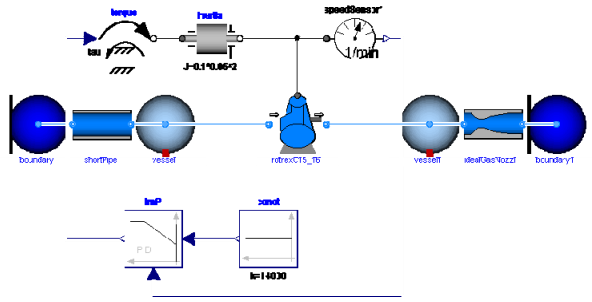

Figure 8 Compressor test model setup

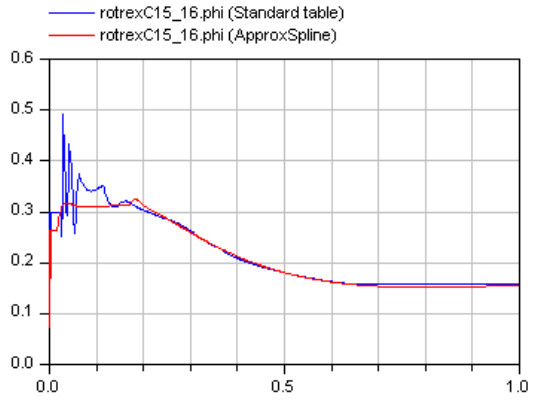

Figure 9 Comparison of flow coefficient

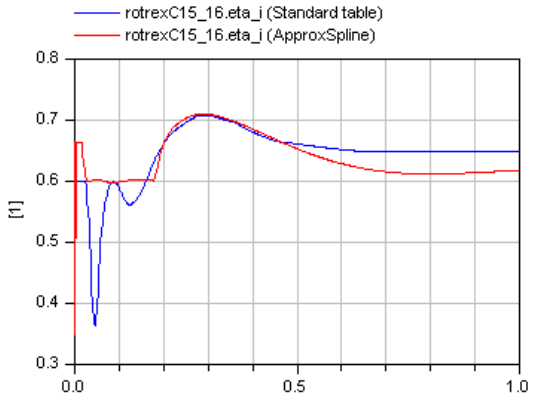

Figure 10 Comparison of inner efficiency

[1] Any performance comparison was done on a double PC, dual processor, dual core AMD Opteron, 2.21 GHz, 2.75 GB RAM running Windows XP and Dymola 7.0

Table 1 Comparison of test model performance

	Standard	ApproxSpline	Diff
CPU-time [s]	0.094	0.079	-16%
F-evaluations	1399	885	-37%
H-evaluations	836	713	-15%
Jacobian-eval.	108	80	-26%

To make the equation system more difficult to solve, another pipe is introduced in the model between the compressor and the right control volume (Figure 11). In this modified model the pressure ratio of the compressor cannot be calculated from system states any more. As the pressure drop of the pipe depends on its mass flow rate, the pressure ratio of compressor in turn depends on the mass flow rate. As a result, a nonlinear equation system of dimension 4 after reduction which includes the table lookup procedure is generated. The simulation of the model using Modelica standard tables fails after approx. 0.1s simulation time due to a Newton solver failure. Simulation of the model using approximating splines works fine. To get the Model with the standard tables working, a modification of the table data is necessary. If an input signal of standard tables is outside of the defined interval, the corresponding value is determined by extrapolation through the last or first two points of the table. In several cases, this leads to nonsensical results, e.g. efficiency less than zero. By adding another 2 rows and columns to the table data, the extrapolation can be forced to return the last or first point of the table. Doing this makes the modified test model solvable even with Modelica standard tables. The comparison of the performance is shown in Table 2.

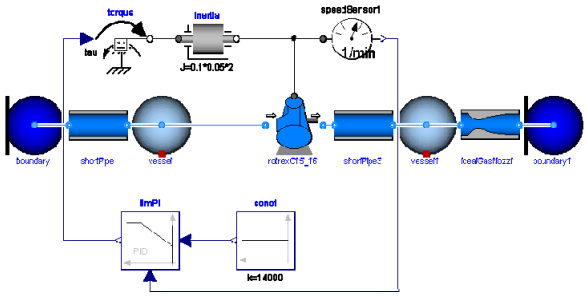

Figure 11 Modified compressor test model setup

Table 2 Comparison of modified test model performance

	Standard	ApproxSpline	Diff
CPU-time [s]	0.297	0.266	-10%
F-evaluations	1128	904	-20%
H-evaluations	786	714	-9%
Jacobian-eval.	87	82	-6%

An interesting point is to study the sensitivity against random noise of table data. As mentioned before, table data are frequently obtained by measurements which are in most cases not smooth. To simulate this, a random number is added to each table data point:

$$\eta_i^* = \eta_i + \left(\max(\eta_i) - \min(\eta_i)\right)\mathrm{rnd}\left([-0.5...0.5]\right)$$

$$\varphi^* = \varphi + \left(\max(\varphi) - \min(\varphi)\right)\mathrm{rnd}\left([-0.5...0.5]\right)$$

The comparison of the models using these noisy table data is shown in Table 3. The shape of the spline surface only slightly changes in comparison to that generated from smooth data points (Figure 12). As a result, the simulation performance of the test model is nearly the same as if smooth data points were used (Table 3). In contrast the computational burden of the model using noisy data with standard Modelica table interpolation is much higher as if smooth data points were used. Thus it turns out, that spline approximation is advantageous especially if the table data is noisy.

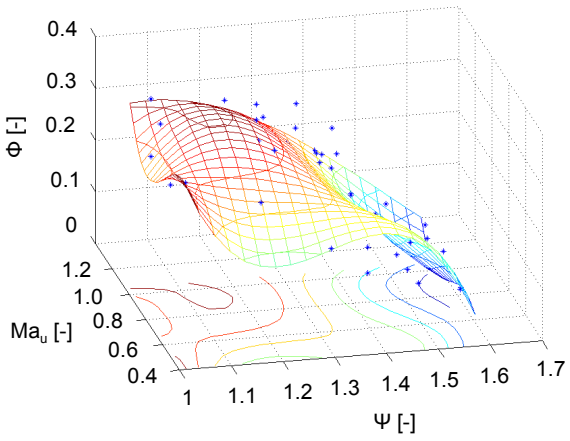

Figure 12 Flow coefficient as function of tip speed Mach number and head coefficient (mesh by spline approximation, noisy data points as stars)

Table 3 Comparison of test model performance with noisy table data

	Standard	ApproxSpline	Diff
CPU-time [s]	0.156	0.078	-50%
F-evaluations	2315	974	-58%
H-evaluations	1032	731	-29%
Jacobian-eval.	210	84	-60%

Another issue with measured table data is limited accuracy, e.g. because of the discretization error of an A/D-converter. To show this effect, the table data points are rounded to one digit (Figure 13). For the standard table model, the performance is clearly worse than with the original data. It is assumed, that the partially zero gradient of the data points is responsible for this behaviour. The approximating spline model is only marginal declined compared to the model that uses the original data (Table 4).

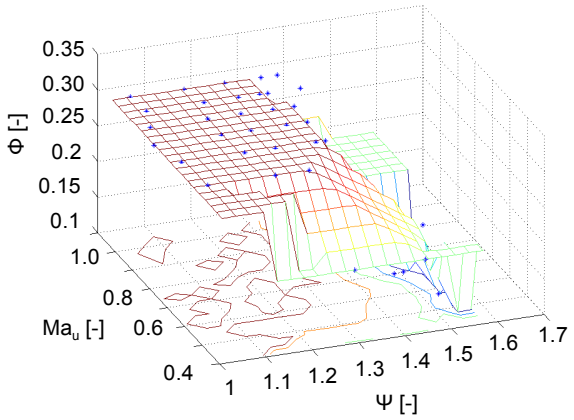

Figure 13 Flow coefficient as function of tip speed Mach number and head coefficient (mesh by interpolation, rounded data points as stars)

Table 4 Comparison of test model performance with rounded table data

	Standard	ApproxSpline	Diff
CPU-time [s]	0.109	0.078	-28%
F-evaluations	1520	924	-39%
H-evaluations	867	714	-18%
Jacobian-eval.	129	83	-36%

6 Conclusion

A new Modelica library for data approximation with polynomial splines in 1 and 2 dimensions is presented. Better performance compared to standard Modelica table interpolation was found in some cases, e.g. noisy or piecewise constant table data. The user interface is partially similar to the standard tables, so migration from standard interpolation tables to approximating splines should be of modest effort. For convenience, table data might be given as scattered data, so no external tool is necessary to generate regular gridded data points.

The library will be made freely available.

7 Acknowledgment

Many thanks to Prof. Paul Dierckx the author of the DIERCKX FORTAN code.

References

[1] T. Hirsch und M. Eck, “4-Dimensional Table Interpolation with Modelica,” *Proceedings of 6th International Modelica Conference 2008*, Bielefeld: 2008.

[2] P. Dierckx, *Curve and Surface Fitting with Splines*, Mcgraw Hill Book Co, 1995.