Implementation of smooth interpolation for optimizations

Joakim Larsson, *Lund University*

Imagine if all the optimization problems could be solved with the same software. Then an engineer trying to optimize a model would be free to work with the actual physical model, without having to dive into how the optimization process works. The aim of this thesis was to expand the scope of the optimization tool CasADi, by implementing support for table-based relationships.

Modelling

When modelling a physical system this is most often done by expressing the physical relationships present in the model as mathematical functions. These functions are often derived from the laws of nature. For instance, if you want to model a bouncy ball when it is dropped from a certain height to the floor and bounces up, the relations governing this movement can be expressed as:

$$E_{pot} = m \cdot g \cdot h$$
$$E_{kin} = v^2/2$$
$$v_{after} = e \cdot v_{before}$$

where E_{pot} is the potential energy of the ball, m is the mass of the ball, g is the gravitational constant ($9.81\ m/s^2$), h is the height the ball is dropped from, E_{kin} is the kinetic energy of the ball, v is the speed of the ball and e is a constant describing the loss in speed when hitting the floor. Since energy needs to be preserved these formulas are sufficient to calculate the behaviour of the ball by hand.

When modelling the same problem in the modelling language Modelica it is written like this:

```
model BouncingBall "The 'classic' bouncing ball model"
  type Height=Real(unit="m");
  type Velocity=Real(unit="m/s");
  parameter Real e=0.8 "Coefficient of restitution";
  parameter Height h0=1.0 "Initial height";
  Height h;
  Velocity v;
initial equation
  h = h0;
equation
  v = der(h);
  der(v) = -9.81;
  when h<0 then
    reinit(v, -e*pre(v));
  end when;
end BouncingBall;
```

i.e. the relations are likewise derived directly from the laws of physics. When simulating a problem the compiler of the Modelica code starts by giving the variables initial values, and based on the values and the equations present calculates what their value should be a short time later. These new values are then used to calculate what the value should be a short time later. Continuing to perform small "time-steps" in this fashion leads to a simulation of the system.

This is a simple example, and a more complex

example would not be calculable by hand, therefore software such as Modelica is necessary. Modelica allows you to simulate models with more complex relationships between the variables. For instance, if the relationship between two variables are expressed via tables, i.e. the value of one variable y is dependent on another variable x but their relationship are only known for certain values x, Modelica still permits simulation.

x	0.0	1.0	2.0	3.0	4.0	5.5	6.0	7.0	8.0
y	0.0	0.4	0.4	0.1	0.1	0.8	0.3	0.7	0.0

Such scenarios can occur when the values are based on empirical data.

When a physical model is to be *optimized* the optimization problem is expressed similarly to the simulation problem, i.e. mostly via mathematical functions. Unfortunately the scope of problems that can be optimized in Modelica is limited. The aforementioned table based relationships cannot be optimized from a Modelica compiler. To solve this problem one needs to interpolate.

Interpolation

In this thesis the software required for interpolating a table as a mathematical function was developed. In order for the interpolation function $f(x)$ to be usable in the optimization chain it had to be continuous even when derived twice, i.e. the order of continuity had to be at least 2, also written C^2.

The problem was solved using b-splines, a function which is defined by the intervals of the table of the independent variable x. B-splines may be used in a linear combination to create a function of an arbitrary order of continuity; see Figure 1. If you derive a function made from a linear combination of b-splines the resultant function is also a linear combination of b-splines.

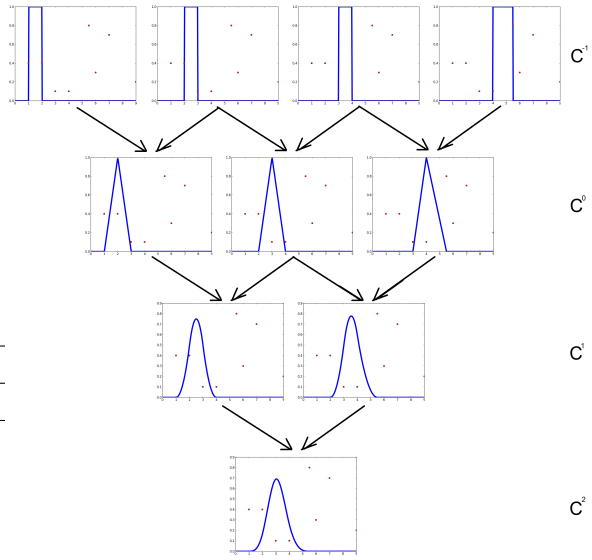

Figure 1: *How b-splines can be constructed*

By creating all necessary b-splines of C^2 continuity and multiplying these b-splines with appropriate coefficients the values may be interpolated; see Figure 2.

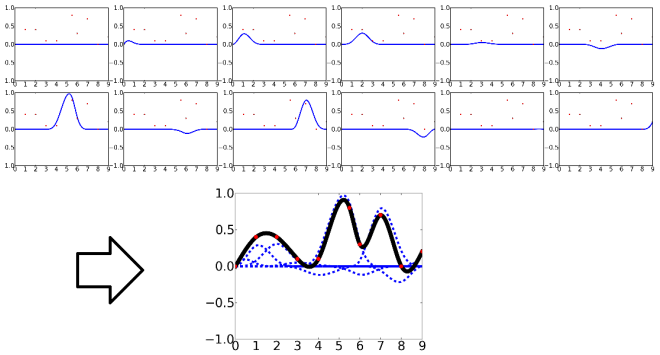

Figure 2: *How b-splines can construct an interpolation function*

A function written in C which from two tables creates an interpolation with which the function and its first, second and third derivative could be evaluated was made in this Thesis. For the cases where the independent variable x had equidistant values the error of the interpolation was smaller than machine-$\epsilon \times |y_{max}|$, where y_{max} was the most extreme value of the dependent variable y's table. It was also integrated to CasADi.

The integration with CasADi is unfortunately prototypical, its key limitation being that the C code has a predefined x and y table to interpolate.