Implementation of smooth interpolation for optimization

Joakim Larsson

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
ISRN LUTFD2/TFRT--5996--SE
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

© 2015 by Joakim Larsson. All rights reserved.
Printed in Sweden by Tryckeriet i E-huset
Lund 2015

Abstract

Models of physical systems are often expressed as a system of mathematical expressions derived from first principles. However some of the relationships present in a model are more conveniently expressed as a table, i.e. for certain values of independent variables the values of dependent variables are known. This imposes a limitation on the optimization software to be used, since optimization software is often dependent on all relationships being expressed as mathematical functions. Many useful optimization methods also require that the functions be twice continuously differentiable.

In this thesis software for interpolating table data relationships between two variables as a twice continuously differentiable mathematical function has been developed. This software has also been prototypically made callable from the automatic differentiation tool CasADi.

CasADi is used in the optimization tool chain in JModelica.org, an open source platform for optimization and simulation. By implementing support for table based relations a larger range of problems may be solved using CasADi.

The software developed for interpolating tables uses cubic b-splines and de Boor evaluation. Using it one may evaluate the interpolant and its derivatives up to the third order. The resultant function is demonstrated to be twice continuously differentiable and to interpolate the value within machine epsilon range of the correct one at the data points, provided the table data points are equidistantly distributed. The oscillations that occur when interpolating non-equidistant table data points are also examined.

Acknowledgements

I would like to extend my gratitude to my supervisors, Toivo Henningsson at Modelon and Fredrik Magnusson at LTH. They came with continuous support and assistance, and their comments were of immense value.

I would also like to thank the non-supervising employees at Modelon, who were always willing to help.

Contents

1 Introduction **5**

2 Background **5**
2.1 Software description 5
2.1.1 Modelica 6
2.1.2 JModelica.org 6
2.1.2.1 Optimization 6
2.1.3 CasADi 7
2.1.3.1 Usage 7
2.1.3.2 Mathematical description 8
2.1.3.3 Code structure 8
2.2 Mathematical background 11
2.2.1 Basis splines 11
2.2.2 Constructing a complete basis 13
2.2.3 Coincident knots 14
2.2.4 The de Boor algorithm 14

3 Implementation **15**
3.1 Calculation of the α vector 15
3.2 Software implementation 20
3.2.1 Description of the functions 21
3.2.2 Creating and calling an interpolation 23
3.2.3 Integration with CasADi 24

4 Results **27**
4.1 Accuracy of the interpolation 29
4.2 Accuracy of the derivative interpolation 32
4.3 Examining basis functions 35
4.3.1 Overall shape 35
4.3.2 Accuracy (equidistant knots) 36
4.3.3 Accuracy (varying knot distance) 40

5 Discussion **42**
5.1 Integration in CasADi 42
5.2 Possible improvements 43

5.2.1 Extension of the number of dependent variables 43
5.2.2 Extension of the number of independent variables . . . 43
5.2.3 Decreasing oscillations 45

1 Introduction

A simulation or optimization of a physical systems is often performed by setting up a system of equations, derived from first principles. Some relations in a physical system are however more easily described by table data. This means that the system contains at least two variables, some of which are dependent upon the other(s), where their relationship is only known for certain values of the independent variable(s).

Since simulation solvers often require that the relationships are expressed entirely via systems of equations this means that support for table based relations are limited. The goal of this thesis was to implement support for models with tables in the optimization and simulation framework *JModelica.org*[5] by creating interpolating functions from a given table. Since the subroutines JModelica.org utilizes for optimization require twice continuous differentiability an additional demand was that the resulting interpolation function be of C^2 continuity.

What has been done in summary is a one dimensional interpolation has been implemented and a prototype integration has been made with *CasADi*[6], a framework which JModelica.org uses when solving optimization problems.

The report is outlined as follows: the background section goes through the software architecture and the mathematical theory on which the implementation relies. The implementation section goes through the approach in implementing the interpolation code and the resultant files. The results section hosts a visual representation of how the function interpolates data tables, as well as an evaluation of how well the code interpolated the table data. In the discussion section future developments/improvements are discussed.

2 Background

2.1 Software description

The architecture of the software used is quite complicated, and for this reason a brief outline without closer explanation of all the terminology used is provided here. This terminology is described in more detail in the subsections.

- JModelica.org is a framework which can simulate models written in Modelica and/or Optimica[4].
- JModelica.org reformulates optimization problems to NLP problems, which are plugged into CasADi via CasADiInterface[15].
- CasADi calculates the appropriate partial derivative functions, and supplies them to an NLP solver.
- The NLP solver returns the solution.

2.1.1 Modelica

Modelica[13] is a language used for the modelling of physical systems. It is object oriented and equation based.

The formulation of some Modelica models utilizes variables whose relationships are expressed by tables, i.e. their relationship is only known at certain values. This is the case in for instance the models of pump characteristics in the Modelica Standard Library, where the relationship between the volume flow of a pump is expressed as a number of data values corresponding to the same number of possible pump heads for quadratic and linear flow[2].

2.1.2 JModelica.org

JModelica.org is an open source framework for working with Modelica models. It interfaces with CasADi. When formulating an optimization problem within the JModelica.org framework one way JModelica.org employs in order to solve the problem is via dynamic optimization of DAEs (Differential Algebraic Equations) using direct collocation with CasADi. JModelica.org also extends to the optimization language Optimica, which is used when solving optimal control problems (OCP, see section 2.1.2.1).

2.1.2.1 Optimization Optimization with JModelica.org involves OCPs. An example of an OCP would be the following problem:

$$\begin{aligned} \text{minimize} \int_0^1 u^4 dt + x(1) \\ \text{s.t.} \dot{x} = -x + u \\ x(0) = 0 \end{aligned}$$

These involve a simulation of a physical system over time where there exist one or several variables whose values when used during the simulation is yet to be decided. The problem is how to chose trajectories for those variables given that the system should fulfil certain criteria and at the same time a cost function is to be minimized.

When using Modelica for optimization the end user provides the equations describing the physical model and the cost function. JModelica.org reformulates the problem to a nonlinear program (NLP).

An NLP is an optimization problem which in CasADi is expressed on the following form

$$\begin{array}{lll} \text{minimize} & f(x), & x \in \mathbb{R}^{n+1} \\ \text{subject to} & \underline{g} \leq g_i(x) \leq \bar{g}, & i = 0, \ldots, n \\ & \underline{x} \leq x \leq \bar{x} & \end{array}$$

JModelica.org is interfaced with CasADi, which in turn is interfaced with a variety of NLP solvers[1], such as IPOPT[17] and WORHP[11]. NLP solvers often depend on Newton-like methods. Since the Newton-like methods for optimization are dependent on the second order derivative[10] these need be provided to the NLP solvers for the problem to be solvable. Furthermore, to have any hope of convergence these must be continuous.

Some of these optimization problems cannot be completely formulated with equations; instead table data will need to be used to represent parts of the model. This means that NLP solvers cannot be utilized when dealing with such problem formulations unless the table data would be transformed into a differentiable function.

2.1.3 CasADi

`CasADi` is a symbolic framework for algorithmic differentiation and numeric optimization. Its structure is a simple algebra system where functions may be differentiated.

Since the final code of this thesis is integrated with CasADi this framework will be explained in more detail.

2.1.3.1 Usage When an ordinary differential equation (ODE) needs to be solved or simulated it is useful to have the ODE be expressible in computer form. It is also useful to be able to solve nonlinear programs (NLPs), described in section (2.1.2.1).

By formulating the ODEs and the NLPs within the CasADi framework it allows CasADi to automatically differentiate the functions, thereby letting the machine handle the complexity involved in calculating partial derivatives. With these partial derivatives calculated CasADi is able to call various NLP solvers.

2.1.3.2 Mathematical description In the CasADi framework one may formulate an expression of variables. CasADi represents these as directed acyclic graphs (DAGs), utilizing the fact that expressions consist of a number of sub-expressions with accompanying mathematical operations between each pair of sub-expressions. For instance, the function $f(x, y) = x^2 \cdot \sin(x+y) - y$ would be internally represented in CasADi as Figure 1 shows.

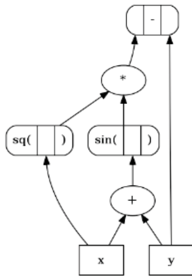

Figure 1: How the function $f(x, y) = x^2 \cdot \sin(x + y) - y$ is represented in the CasADi framework

A mathematical function will thus be expressed as a node tree of basic operations in this manner. Since the derivative of a basic operation is another basic operation the derivatives may be easily calculated within the CasADi framework.

2.1.3.3 Code structure To illustrate CasADi's functionality and syntax below is an example of how one would get the Jacobian of the function $f(x, y) = x^2 \cdot \sin(x + y) - y$ in C++:

```
MX x = MX::sym("x"); //declares x as a variable
```

```
MX y = MX::sym("y"); //declares y as a variable
MX z = x*x*sin(x+y)-y; //creates the expression for the function

vector<MX> inputs; //creates a vector to store the inputs in
inputs.push_back(x);
inputs.push_back(y);

vector<MX> outputs; //creates a vector to store the output(s) in
outputs.push_back(z);

MXFunction f = MXFunction(inputs, outputs); //creates the func-
tion f to evaluate the expression of z
f.init(); //initializes the function by flatteningthe node graph
down to a series of instructions

Function f_der= f.fullJacobian(); //calls the fullJacobian of the
MXFunction class
```

After running the above code `f_der` would contain the Jacobian of `f`.

What happens in order is that the `MX` objects `x` and `y` are set as symbolic variables and the `MX` object `z` is set as the expression corresponding to the function f.

`MX` is a CasADi defined class for handling expressions. Internally it is always structured as a matrix, no matter if the end user treats it as a scalar, as in the example above. There also exist a similar CasADi class `SX` which only handles scalars ([8], chapter 3).

`MX` objects can be used as variables, constants, combinations of `MX` objects or matrices consisting of the above.

The `MX` objects from the example are put into the vectors `inputs` and `outputs` and linked together via the `MXFunction f`.

`Function` ([8], chapter 4) objects are objects defined within the CasADi framework for handling mathematical functions. It is seldom used in its base form, instead one often use its derived subclasses, such as `SXFunction`, `ExternalFunction` or, as in the example, `MXFunction`.

The key aspect of `Function` and its subclasses is that they can be easily evaluated and differentiated.

When calling the constructor of a `Function` object the CasADi framework makes that object call the constructor of another `Function` object, `Func-`

tionInternal. In this internal function class the functionality accessible through the external Function is stored. This is also true for the subclasses, i.e. the creation of an MXFunction leads to the creation of an MXFunction-Internal; the creation of an ExternalFunction leads to the creation of an ExternalFunctionInternal and so on.

Depending on which subclass of Function that is used the arguments for the constructor and which functions that are available varies. An External-Function requires an already compiled function in C to be used as argument in the constructor, which will then be its internal function. An MXFunction takes, as can be seen in the example, two vectors of MX objects as its input and output.

By initializing f as in the example the CasADi framework assumes that the Function won't be further modified. This means that the MXFunction f may be converted to corresponding C code by calling f.generateCode() and that the Jacobian may be calculated by, as the example shows, calling f.fullJacobian().

This ensures that the MXFunction f_der now contains what the Jacobian of f is, which according to theory should be

$$\mathbf{J} = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = (2 \cdot x \cdot \sin(x+y) + x^2 \cdot \cos(x+y), x^2 \cdot \cos(x+y) - 1)$$

In order to examine what the Jacobian actually would be at a certain point, one would need to actually set the variables to the values associated with that point. So if for instance the point $(x, y) = (1.2, 3.4)$ is to be evaluated the following code can be used:

```
f_der.init();
f_der.setInput(1.2,0);
f_der.setInput(3.4,1);
f_der.evaluate();

DMatrix f_der_x = f_der.getOutput(0)[0];
DMatrix f_der_y = f_der.getOutput(0)[1];
```

Since the first variable in the input vector corresponds to x, the zeroth variable in inputs is the one to get the corresponding x value 1.2. For the same reason the first variable in inputs gets the value of 3.4.

Calling `evaluate` makes the CasADi framework calculate what the output is for the current input. This result may then be extracted by calling the `getOutput` function. The way the `fullJacobian` function is formulated in CasADi means that the output will be one output vector with the two partial derivatives stored in its two positions. Hence the expressions used for extracting the two partial derivatives `f_der_x` and `f_der_y`.

2.2 Mathematical background

Most of the following background is a summation of chapter 1 of [12].

2.2.1 Basis splines

When interpolating over one dimensional table data the data is provided by two vectors of length $n+1$, one describing the independent variables x and one describing the corresponding dependent values y.

$$\boldsymbol{x} = \{x_0, x_1, \cdots, x_n\}, \boldsymbol{x} \in \mathbb{R}^{n+1}$$

$$\boldsymbol{y} = \{y_0, y_1, \cdots, y_n\}, \boldsymbol{y} \in \mathbb{R}^{n+1}$$

The x values are also known as the *knots*. They are strictly increasing.

When interpolating it is common to insert additional knots, and since it's beneficial to have the original $n+1$ values to interpolate preserved, the nomenclature for the knots to interpolate once additional have been inserted has in this report been written as λ values.

To handle such an interpolation problem the concept of spline functions may be utilized. Spline functions of a certain degree $k \in \mathbb{N}$ are functions that fulfil the following criteria:

- On each knot interval $[\lambda_i, \lambda_{i+1}]$ the function is given by a polynomial of degree k at most.
- The function has C^{k-1} continuity as long as all the λ values are distinct.

The splines for a given knot sequence span a vector space. The dimension of the vector space can be shown to be $n+k$.

Basis splines, also known as *b-splines*, is a basis for this vector space. Using b-splines it is possible to create a function of any order of continuity provided a vector of knots. This function can be expressed as a linear

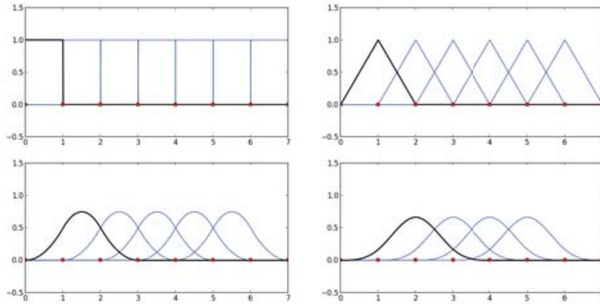

Figure 2: Illustration of b-splines of order 0 through 3; one index marked

combination of the b-splines, see Equation (6). Furthermore the νth derivative, where $\nu \in \mathbb{N}, 0 \leq \nu \leq k$, may also be constructed by a combination of b-splines.

A b-spline of order 0 is a function that is equal to one inside an interval $[\lambda_i, \lambda_{i+1})$ and is otherwise equal to zero.

$$N_{i,1}(x) = \begin{cases} 1 & \text{if x} \in [\lambda_i, \lambda_{i+1}) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

It is discontinuous i.e. C^{-1} continuity ([12], equation 1.24).

To create higher orders of continuity one combines two b-splines of one lower order of continuity,

$$N_{i,k+1}(x) = \frac{x - \lambda_i}{\lambda_{i+k} - \lambda_i} \cdot N_{i,k}(x) + \frac{\lambda_{i+k+1} - x}{\lambda_{i+k+1} - \lambda_{i+1}} \cdot N_{i+1,k}(x) \tag{2}$$

where the order of continuity is $k-1$ and the order of the b-spline is $k+1$ ([12], equation 1.24). To illustrate these b-splines Figure 2 charts the first four b-splines acquired from the case when all the intervals the b-spline is based on have the same length.

Using b-splines of order 0 constructed by Equation (1) one is able to create b-splines of order 1, with C^0 continuity. These can then be used to create b-splines of order 2. Continuing in this fashion one is able to create

polynomial functions of any requested order of continuity. For the remainder of the text the order is assumed to be k and the order of continuity is hence C^{k-1}.

The derivative of a b-spline is calculable by

$$\frac{\partial N_{i,k+1}(x)}{\partial x} = k \cdot \left\{ \frac{N_{i,k}(x)}{\lambda_{i+k} - \lambda_i} - \frac{N_{i+1,k}(x)}{\lambda_{i+k+1} - \lambda_{i+1}} \right\} \tag{3}$$

([12], equation 1.25)

2.2.2 Constructing a complete basis

In order to let the b-splines reach C^{k-1} continuity over the entire range of x the b-splines must form a basis in the vector space of spline functions with the given knots with continuity C^{k-1}. This corresponds to $n+k$ b-splines. $n+1$ knots can be used to construct n b-splines of order 0, since each of these b-splines require one interval as per Equation (1). From this it is possible to create an interpolation function of continuity C^{-1} from the b-splines of order 0.

If C^0 continuity shall be obtained over the range of x using b-splines one must use b-splines of order 1, since their order of continuity is C^0. From Equation (2) one realizes that each b-spline of order 1 depends on two b-splines of order 0. From the b-splines of order 0 obtainable directly from x one can construct $n-1$ b-splines of order 1 according to Equation (2). That is insufficient, since the vector space of splines with C^0 continuity has the dimension $n+1$. Therefore two additional b-splines of order 0 would need to be present to create a complete basis. This corresponds to adding two additional knots. For the same reason each order of continuity require the addition of two knots to the original $n+1$. This may be done by repeating the first and final value k times before and after the initially present knots.

$$\begin{gathered}
\boldsymbol{x} = \{x_0, x_1, \cdots, x_n\} \rightarrow \boldsymbol{\lambda} = \{\lambda_{-k}, \lambda_{-k+1}, \cdots, \lambda_{n+k}\} \\
\text{where} \\
\{x_0, x_1, \cdots, x_n\} = \{\lambda_0, \lambda_1, \cdots, \lambda_n\} \\
\{\lambda_{-k}, \lambda_{-k+1}, \cdots, \lambda_{-1}\} = \{x_0, x_0, \cdots, x_0\} \\
\{\lambda_{n+1}, \lambda_{n+2}, \cdots, \lambda_{n+k}\} = \{x_n, x_n, \cdots, x_n\}
\end{gathered}$$

2.2.3 Coincident knots

The formula for the b-spline (2) would need to modified in order to account for the fact that the difference between two subsequent knots could be equal.

If $\lambda_i = \cdots = \lambda_{i+k} < \lambda_{i+k+1}$

$$N_{i,k+1}(x) = \begin{cases} (\frac{\lambda_{i+k+1}-x}{\lambda_{i+k+1}-\lambda_i})^k & \text{if x} \in [\lambda_i, \lambda_{i+k+1}] \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

If $\lambda_i < \lambda_{i+1} = \cdots = \lambda_{i+k+1}$

$$N_{i,k+1}(x) = \begin{cases} (\frac{x-\lambda_i}{\lambda_{i+k+1}-\lambda_i})^k & \text{if x} \in [\lambda_i, \lambda_{i+k+1}] \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Equation (3) remains unaffected.

2.2.4 The de Boor algorithm

The formula from which one can construct a continuous function based on all the now defined b-splines can be expressed as

$$s(x) = \sum_{i=j-k}^{j} \alpha_i \cdot N_{i,k+1}(x) \tag{6}$$

where $x \in [\lambda_j, \lambda_{j+1})$ and α is a vector calculated from the y values. The α vector Alternatively the *de Boor interpolation*

$$s(x) = \alpha_j^{[k]}(x) = \begin{cases} \alpha_j & \text{if i} = 0 \\ \frac{(x-\lambda_j)\cdot\alpha_j^{[i-1]}(x)+(\lambda_{j+k+1-i}-x)\cdot\alpha_{j-1}^{[i-1]}(x)}{\lambda_{j+k+1-i}-\lambda_j} & \text{if i} > 0 \end{cases} \tag{7}$$

([12], equation 1.38) where $x \in [\lambda_j, \lambda_{j+1})$ and the α vector is unchanged from Equation (6) may be used. It returns the same result as Equation (6). Once the subsequent α values have been calculated the derivatives of the interpolation can be calculated easily; if $x \in [\lambda_j, \lambda_{j+1})$, the ν:th derivative of the interpolation is:

$$s^{(\nu)}(x) = \prod_{i=1}^{\nu}(k+1+i) \cdot \sum_{i=j-k+\nu}^{j} c_i^{(\nu)} \cdot N_{i,k+1-\nu}(x) \tag{8}$$

with

$$c_j^{(i)} = \begin{cases} \alpha_j & \text{if i} = 0 \\ \frac{\alpha_j^{(i-1)} - \alpha_{j-1}^{(i-1)}}{\lambda_{j+k+1-i} - \lambda_j} & \text{if i} > 0 \end{cases} \tag{9}$$

This equation can be derived from Equation (3).

It should be noted that in order for the de Boor algorithm to interpolate the value correctly at the end point λ_n it is $\alpha_{j=n-1}^{[k]}(\lambda_n)$, not $\alpha_{j=n}^{[k]}(\lambda_n)$, that must be evaluated.

3 Implementation

The implementation assumes C^2 continuity, i.e. $k = 3$. It assumes that the x and y vectors were externally supplied.

The interpolation at a certain point in the span of the knots may, according to Equation (7), be calculated as the sum of the interpolation of the individual b-splines multiplied by the values of a vector α. According to Equation (8) the same is true for the calculation of the derivative of the interpolation, though the values to multiply by differs. With this in mind, the first step of the implementation consists of calculating the aforementioned α vector. The second step is to calculate the vectors containing the values for calculating the derivatives. These can be extracted from Equation (8) and (9), and shall henceforth be referred to as α', α'' and α'''.

After these vectors had been calculated the end user would be able to access the value/derivative of the function by letting the function interpolate with Equation (7).

3.1 Calculation of the α vector

In order to interpolate the table data the interpolation function must pass through the corresponding y values when interpolating at the pre set x values. With $k = 3$ two additional conditions must be present in order to determine the α vector. In this interpolation the conditions chosen were *natural boundary conditions*, i.e. the second derivative at the first and last knots were set to zero.

Natural boundary conditions are hardly the only alternative when interpolating with b-splines, for instance *periodic* and *Hermite* conditions are other alternatives often used ([16], page 112). Natural boundary conditions

were chosen because they make the calculation of the α vector relatively fast and from the resultant interpolation one may extrapolate to points outside of the span of x.

In order to calculate the α vector a system of equations was set up as a matrix. Each row of the matrix corresponded to (6) where each element was the interpolation of a certain b-spline at one of the x_i values. The natural boundary conditions were set up as the second and second to last row, for reasons that will soon become obvious.

$$\mathbf{A} = \begin{pmatrix} N_{-k,4}(x_0) & N_{-k+1,4}(x_0) & \cdots & N_{n-2,4}(x_0) & N_{n-1,4}(x_0) \\ \frac{\partial^2 N_{-k,4}(x_0)}{\partial x^2} & \frac{\partial^2 N_{-k+1,4}(x_0)}{\partial x^2} & \cdots & \frac{\partial^2 N_{n-2,4}(x_0)}{\partial x^2} & \frac{\partial^2 N_{n-1,4}(x_0)}{\partial x^2} \\ N_{-k,4}(x_1) & N_{-k+1,4}(x_1) & \cdots & N_{n-2,4}(x_1) & N_{n-1,4}(x_1) \\ N_{-k,4}(x_2) & N_{-k+1,4}(x_2) & \cdots & N_{n-2,4}(x_2) & N_{n-1,4}(x_2) \\ N_{-k,4}(x_3) & N_{-k+1,4}(x_3) & \cdots & N_{n-2,4}(x_3) & N_{n-1,4}(x_3) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ N_{-k,4}(x_{n-2}) & N_{-k+1,4}(x_{n-2}) & \cdots & N_{n-2,4}(x_{n-2}) & N_{n-1,4}(x_{n-2}) \\ N_{-k,4}(x_{n-1}) & N_{-k+1,4}(x_{n-1}) & \cdots & N_{n-2,4}(x_{n-1}) & N_{n-1,4}(x_{n-1}) \\ \frac{\partial^2 N_{-k,4}(x_n)}{\partial x^2} & \frac{\partial^2 N_{-k+1,4}(x_n)}{\partial x^2} & \cdots & \frac{\partial^2 N_{n-2,4}(x_n)}{\partial x^2} & \frac{\partial^2 N_{n-1,4}(x_n)}{\partial x^2} \\ N_{-k,4}(x_n) & N_{-k+1,4}(x_n) & \cdots & N_{n-2,4}(x_n) & N_{n-1,4}(x_n) \end{pmatrix}$$

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_{-k} & \alpha_{-k+1} & \alpha_{-k+2} & \alpha_{-k+3} & \cdots & \alpha_{n-3} & \alpha_{n-2} & \alpha_{n-1} \end{pmatrix}^T$$

$$\mathbf{d} = \begin{pmatrix} y_0 & 0 & y_1 & y_2 & \cdots & y_{n-1} & 0 & y_n \end{pmatrix}^T$$

$$\mathbf{A}\boldsymbol{\alpha} = \mathbf{d}$$

According to Equation (1) the knot x_i is only contained in one b-spline of order 0, $N_{i,1}(x)$. The only b-splines of order 3 that contains this b-spline of order 0 has a non zero value in the matrix. These are, using (2) repeatedly, $N_{i,4}(x_i), N_{i-1,4}(x_i), N_{i-2,4}(x_i)$ and $N_{i-3,4}(x_i)$. $N_{i,4}(x_i)$ will however always be zero, since by Equation (2) $N_{i,4}(x_i) = \frac{x_i-\lambda_i}{\lambda_{i+l}-\lambda_i} \cdot N_{i,3}(x_i) + 0 = \frac{0}{\lambda_{i+l}-\lambda_i} \cdot N_{i,3}(x) = 0$.

The rows containing the second order derivatives are calculable by (3). At $x = x_i$ only those derivatives that contain $N_{i,1}$ are non zero in the matrix. Those b-splines are $N_{i-3,4}, N_{i-2,4}, N_{i-1,4}$ according to Equation (2). For $i = 0$ and $i = n$ this means that only the first three respectively the last three values of their respective rows are non zero.

Additionally the first and last row of the matrix was known beforehand. When the boundary knots x_0 and x_n are interpreted only the first respectively the last b-spline are non zero. This is due to Equation (4) and (5) returning $(\frac{1}{1})^3 = 1$ at these points. For the second and the second to last b-spline

Equation (2) eventually turns out to contain nothing but zeros, leading to another simplification.

This leads to a tridiagonal matrix.

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \frac{\partial^2 N_{-3,4}(x_0)}{\partial x^2} & \frac{\partial^2 N_{-2,4}(x_0)}{\partial x^2} & \frac{\partial^2 N_{-1,4}(x_0)}{\partial x^2} & 0 & \cdots & 0 & 0 \\ 0 & N_{-2,4}(x_1) & N_{-1,4}(x_1) & N_{0,4}(x_1) & \cdots & 0 & 0 \\ 0 & 0 & N_{-1,4}(x_2) & N_{0,4}(x_2) & \cdots & 0 & 0 \\ 0 & 0 & 0 & N_{0,4}(x_3) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & N_{n-2,4}(x_{n-1}) & 0 \\ 0 & 0 & 0 & & \cdots & \frac{\partial^2 N_{n-2,4}(x_n)}{\partial x^2} & \frac{\partial^2 N_{n-1,4}(x_n)}{\partial x^2} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

A tridiagonal matrix of size $(n+3) \times (n+3)$ can be solved in $\mathcal{O}(n+3)$ time and space, using the *tridiagonal matrix algorithm*[3]. The algorithm in essence consists of a normal Gaussian elimination, but due to the structure of the matrix the construction of the upper triangular form require only $n+2$ row subtractions. The resultant matrix only has elements on the main and superdiagonal, and hence also only require $n+2$ row subtractions. This is a marked improvement in comparison to the time required to solve such a problem with Gaussian elimination which is of $\mathcal{O}(n^3)$. Furthermore, since the resulting matrix is tridiagonal it is sufficient to store these diagonals as vectors, making the matrix something that is never actually constructed in the program.

The question of whether such an interpolation is always solvable thus arises. It is a well known fact that such is the case, as showed in for instance [16]. For completeness, a proof will however be presented here.

If it is not solvable that would mean one of two things: either it is not sufficient to demand that the combined piecewise polynomials that interpolates the table data are of C^2 continuity and that the second order derivative is zero at the end points (meaning more than one solution), or those demands are insufficient(meaning no solution). If the former, there would be at least two possible α vectors that solves a given system.

If one thus considers the case where the prescribed values at each knot is zero there has to exist an α vector other than the one that only consists of zeros (the homogeneous solution) that still interpolates the values.

If one describes the polynomial within $[\lambda_i, \lambda_{i+1})$ as

$$f(x) = a_i \cdot (x - \lambda_i)^3 + b_i \cdot (x - \lambda_i)^2 + c_i \cdot (x - \lambda_i) + d_i \tag{10}$$

it must thus be true that for $i = 0$ is $f(\lambda_0) = f''(\lambda_i) = f(\lambda_1) = 0$, or in other words that

$$f(\lambda_i) = a_i \cdot (\lambda_i - \lambda_i)^3 + b_i \cdot (\lambda_i - \lambda_i)^2 + c_i \cdot (\lambda_i - \lambda_i) + d_i = 0 \tag{11}$$

$$f(\lambda_{i+1}) = a_i \cdot (\lambda_{i+1} - \lambda_i)^3 + b_i \cdot (\lambda_{i+1} - \lambda_i)^2 + c_i \cdot (\lambda_{i+1} - \lambda_i) + d_i = 0 \tag{12}$$

$$f''(\lambda_0) = 6 \cdot a_0 \cdot (\lambda_0 - \lambda_0) + 2 \cdot 1 \cdot b_0 = 0 \tag{13}$$

The thing to consider is now how the values of the first and second derivative at one knot affects the the values of the first and second derivative at another knot. Put in a matrix form the relationship may be written as:

$$\begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix} \cdot \begin{bmatrix} f'(\lambda_i) \\ f''(\lambda_i) \end{bmatrix} = \begin{bmatrix} f'(\lambda_{i+1}) \\ f''(\lambda_{i+1}) \end{bmatrix} \tag{14}$$

The values of the $f'(x)$ and $f''(x)$ can be easily calculable from Equation (10), and inserted into Equation (14).

$$(10),(14) \Rightarrow \begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix} \cdot \begin{bmatrix} 3 \cdot a_i \cdot (\lambda_i - \lambda_i)^2 + 2 \cdot b_i \cdot (\lambda_i - \lambda_i) + c_i \\ 6 \cdot a_i \cdot (\lambda_i - \lambda_i) + 2 \cdot b_i \end{bmatrix} =$$

$$\begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix} \cdot \begin{bmatrix} c_i \\ 2 \cdot b_i \end{bmatrix} = \begin{bmatrix} 3 \cdot a_i \cdot (\lambda_{i+1} - \lambda_i)^2 + 2 \cdot b_i \cdot (\lambda_{i+1} - \lambda_i) + c_i \\ 6 \cdot a_i \cdot (\lambda_{i+1} - \lambda_i) + 2 \cdot b_i \end{bmatrix} \tag{15}$$

Using that $f(\lambda_i) = f(\lambda_{i+1}) = 0$ Equation (11) and (12) yields

$$(11) \Rightarrow a_i \cdot (\lambda_i - \lambda_i)^3 + b_i \cdot (\lambda_i - \lambda_i)^2 + c_i \cdot (\lambda_i - \lambda_i) + d_i = a_i \cdot 0 + b_i \cdot 0 + c_i \cdot 0 + d_i = d_i = 0 \tag{16}$$

$$(11),(12) \Rightarrow a_i \cdot (\lambda_{i+1} - \lambda_i)^3 + b_i \cdot (\lambda_{i+1} - \lambda_i)^2 + c_i \cdot (\lambda_{i+1} - \lambda_i) = 0 \Leftrightarrow$$

$$c_i = -a_i \cdot (\lambda_{i+1} - \lambda_i)^2 - b_i \cdot (\lambda_{1+1} - \lambda_i) \tag{17}$$

Having thus gotten expressions for c_i and d_i these can be inserted into Equation (15):

$(15),(16),(17) \Leftrightarrow$

$$\begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix} \cdot \begin{bmatrix} -a_i \cdot (\lambda_{i+1} - \lambda_i)^2 - b_i \cdot (\lambda_{i+1} - \lambda_i) \\ 2 \cdot b_i \end{bmatrix} =$$

$$\begin{bmatrix} 3 \cdot a_i \cdot (\lambda_{i+1} - \lambda_i)^2 + 2 \cdot b_i \cdot (\lambda_{i+1} - \lambda_i) - a_i \cdot (\lambda_{i+1} - \lambda_i)^2 - b_i \cdot (\lambda_{i+1} - \lambda_i) \\ 6 \cdot a_i \cdot (\lambda_{i+1} - \lambda_i) + 2 \cdot b_i \end{bmatrix} =$$

$$\begin{bmatrix} 2 \cdot a_i \cdot (\lambda_{i+1} - \lambda_i)^2 + b_i \cdot (\lambda_{i+1} - \lambda_i) \\ 6 \cdot a_i \cdot (\lambda_{i+1} - \lambda_i) + 2 \cdot b \end{bmatrix}$$

From these numbers the matrix henceforth known as M can be calculated:

$$\begin{bmatrix} -2 & -\frac{\lambda_{i+1} - \lambda_i}{2} \\ -\frac{6}{\lambda_{i+1} - \lambda_i} & -2 \end{bmatrix} \cdot \begin{bmatrix} f'(\lambda_i) \\ f''(\lambda_i) \end{bmatrix} = \begin{bmatrix} f'(\lambda_{i+1}) \\ f''(\lambda_{i+1}) \end{bmatrix}$$

As can be seen all of the values of M are strictly negative. For the specific case when $i = 0$ it is true according to Equation (13) that a value of the left-hand side vector is known:

$$\begin{bmatrix} f'(\lambda_0) \\ f''(\lambda_0) \end{bmatrix} = \begin{bmatrix} f'(\lambda_0) \\ 0 \end{bmatrix}$$

If one assumes that $f'(\lambda_0) = 0$ as well then M is multiplied by a vector consisting of zeroes, meaning the resultant vector will also only consist of zeroes. This would mean that $f'(\lambda_1) = f''(\lambda_1) = 0$. If one then repeats the calculation the result would be that $f'(\lambda_2) = f''(\lambda_2) = 0$, and with the same reasoning the first and second derivative of all the knots would be equal to zero. This corresponds to having the coefficients of all the polynomials be equal to zero, which is precisely the homogeneous solution.

The only way to have a non homogeneous solution is thus if $f'(\lambda_0) \neq 0$. Since M is strictly negative this would mean that the sign of $f'(\lambda_1)$ and $f''(\lambda_1)$ would be opposite of $f'(\lambda_0)$. The key aspect is that both would have the same sign. For this reason the calculation of $f'(\lambda_2)$ and $f''(\lambda_2)$ would be a strictly negative matrix times a vector where all coordinates are of the same sign, meaning the sign of $f'(\lambda_2)$ and $f''(\lambda_2)$ would be equal. Applying the same calculation for each interval leads to the sign of the first and second derivative of each individual knot to be equal. The key aspect is that it will never be equal to zero, meaning the natural boundary conditions cannot be fulfilled.

Therefore only the homogeneous solution exists, and if a solution exist it

is unique.

To prove that a solution always exist: assume that there exists an x and y vector for which no solution can be found. This would correspond to the matrix constructed from Gaussian elimination of the tridiagonal matrix at some point presenting an equation which is unsolvable, i.e. saying that the α vector times zero equals a value that is not equal to zero. If that scenario would occur, consider the problem of interpolating the same x vector with the y vector only consisting of zeroes. In that scenario the same equation would be that the α vector times zero equals zero. This has an infinite amount of solutions, a situation which has already been proven to never occur. Therefore one solution must always exist, and this solution is unique.

3.2 Software implementation

The code used for interpolation was first written i Python, then converted to C and finally integrated with CasADi, which is written in C++. This was done due to the relative ease with which prototyping could be performed in Python.

The changes experienced when going from Python to C was mainly that the length of the vectors and the addresses of the variables were something the developer had to keep track of personally. Additionally some functionality was part of the standard library in Python, which had to be implemented when working in C.

In CasADi there is support for taking a completed CasADi `Function` and use the command `generateCode` to generate corresponding C code. A C function generated this way has a signature which makes it trivial to integrate with CasADi via CasADi's standard function `ExternalFunction`.

To integrate the completed interpolation code a trivial CasADi function was created and the corresponding C code was generated to get an implementation template. In this template, called `bSplineExt.c`, the functionality was gradually replaced with the functionality of the interpolation code. Afterwards `ExternalFunction` was used with `bSplineExt.c` as the internal function.

The C code by itself creates a `struct` called `bSpline` where the data is stored.

```
struct bSpline{
d *xV;
```

```
d *yV;
d *alpha;
int deg;
int numDataPoints;
int xLength;
int alphaLength;
int xStart;
d xMin;
d xMax;
};
```

The `bSpline struct` contains the `x` and `y` vectors as `xV` and `xV`. The α, α', α'' and α''' vectors used when interpolating the derivative of zeroth to third degree is stored as `alpha`. `deg` corresponds to k, `numDataPoints` to n, `xLength` the total length of the x vector when boundary points have been added, `alphaLength` corresponds to the length of α, `xStart` is the index of the first value of `xV` which was not copied when adding additional boundary knots. `xMin` and `xMax` are the smallest respectively largest x values present.

3.2.1 Description of the functions

The code contains the following key functions

- `void solveTriDiagonalMatrix(int length, d a[], d b[], d c[], d y[], d dest[])`
- `void addEndKnots(d xV[], struct bSpline* B)`
- `void setUpBSpline(d* xV, d* yV, int length, int deg, struct bSpline* B)`
- `d evalDeBoor(d t, int order, d c[], struct bSpline* B)`
- `void convertAlpha(d c[], int cLength, int der, d* dest, struct bSpline* B)`
- `void calcCoeff(struct bSpline* B)`
- `d __call__(d t, int der)`
- `void setAlpha(struct bSpline* B)`

- `int init(int *n_in, int *n_out)`
- `int eval(const d* const* arg, d* const* res, int* iii, d* w)`
- `int evaluateWrap(const d** arg, d** res)`
- `void evaluate(const d* x0, const d* x1, d* r0)`
- `int getSparsity(int i, int *nrow, int *ncol, int **colind, int **row)`

Apart from these there are a few assistant functions whose purpose don't need any further explanation.

In the code the key phrase "`d`" has been redefined as standing for the floating point number type `double`.

`solveTriDiagonalMatrix` solves the equation $Mx = y$, assuming M is tridiagonal. It takes the subdiagonal `a`, the main diagonal `b`, the superdiagonal `c`, the length of the main diagonal `length`, the right hand side y and returns implicitly the solution with the argument `dest`.

`addEndKnots` inserts the additional knots required at both ends of the x values. It takes the values to be used as the knots `xV`, and a `struct bSpline` from which the `deg` value is used.

`setUpBSline` is a function which, from a given set of x- and y-values and a requested degree creates all the values of `struct bSpline`, save for the α values. The inputs are the x-values `xV`, the y-values `yV`, their common `length`, the degree `deg` and the `struct bSpline` where all the values will be stored.

`evalDeBoor` calculates the de Boor evaluation at a certain point. The inputs are the requested points `t`, the `order` of the b-spline interpolation, the vector `c` used as the α vector, and a `struct bSpline` which contains the knots and the degree.

`convertAlpha` converts a provided α vector to the corresponding α', α'' or α''' vector. It takes the vector that is assumed to be the α vector as `c`, its length `cLength`, the requested order of derivative `der`, the implicit resulting vector `dest` and a `struct bSpline` which contains the knots and degree required to convert the α vector.

`calcCoeff` calculates the α vector assuming natural boundary conditions and third degree b-spline interpolation. All information is provided via a `struct bSpline`, which contains all information contained in a `struct bSpline` save for its α values.

`__call__` returns the result of interpolating at a certain point. `t` is provided as the point in question, with `der` as the requested derivative order, which may be zero.

`setAlpha` is an assistant function that sets a `struct bSpline`'s derivative α vectors to what they correspond to, provided that its α vector is already calculated.

`init` is the function that creates the interpolation. Since it was automatically generated its input parameters corresponds to CasADi's requirements on the `init` function of an `ExternalFunctionInternal`. CasADi requires that a function called `init` be present, and that it should implicitly return the number of inputs and outputs. These input parameters `n_in` and `n_out` are given the values 2 and 1 respectively, corresponding to the inputs and outputs of `__call__`. Since `init` is called in the `ExternalFunctionInternal`'s constructor it is this function that creates the interpolation.

`eval`, `evaluate` and `evaluateWrap` are the functions with which CasADi integrates when calling the interpolation function. They were all automatically generated. `eval` was changed to call `__call__`, hence ensuring that the interpolation actually occurs when calling `bSplineExt.c`.

`getSparsity` returns the sparsity of the Jacobian of the function. This function was automatically generated and untouched after its creation. Since the interpolation contains no structural zeroes `getSparsity` will always return the value 1.

3.2.2 Creating and calling an interpolation

Of the functions present in `bSplineExt.c` only a few will ever be called once the interpolation has been created from `init`.

`eval`, `evaluate` and `evaluateWrap` are autogenerated wrapping functions which ensure that CasADi integrates to the function. It takes two inputs corresponding to the point to interpolate and the desired derivative.

These call the `__call__` function which takes the present `struct bSpline` and calls `evalDeBoor`, provided the supplied point and requested derivative is within the range of the x-values. It also supplies the appropriate α vector from the `struct bSpline` to `evalDeBoor`.

`evalDeBoor` performs a de Boor evaluation with a given point, α vector and order on the b-spline interpolation, using Equation (7). All the other variables are supplied through the `struct bSpline`.

The rest of the functions are used when creating the `struct bSpline`, and

are hence never called from the CasADi environment, once the interpolation function has been created.

`init` takes two vectors of data points, the requested degree and a number indicating the amount of data points. These are supplied to `setUpBSpline` where the global `struct bSpline` is supplied with all values save for the α vectors. `setUpBSpline` also calls `addEndKnots`, which provides the additional boundary knots. Since this interpolation has been constructed assuming that $k = 3$ only that degree is currently accepted.

`calcCoeff` calculates the values of the sub-, super- and main diagonals of the tridiagonal matrix (3.1), using `evalDeBoor`. Since the matrix' values mostly consist of the evaluation of particular b-splines `evalDeBoor` is supplied with α vectors which are unit vectors of the appropriate index. For the second order derivative the unit vector passed through to `convertAlpha` before `evalDeBoor` is used with the resulting α'' vector.

`convertAlpha` takes a vector corresponding to the assumed α vector and an integer `der` corresponding to the requested order of derivative, and returns the α vector used to interpolate the `der`:th degree derivative. This is done utilizing the fact that Equation (8) will still be a de Boor interpolation after Equation (9) has been used.

After `calcCoeff` has created the three diagonal vectors it calls `solveTriDiagonalMatrix`, which solves the system and returns the α vector to be used for the interpolation.

After that `setAlpha` is called from `init`. This function takes the calculated α vector and, calling `convertAlpha` repeatedly, creates all the interpolation vectors required for the degree of the `struct bSpline`. This is done by creating each derivative α vector one by one; α is used to create α', α' is used to create α'' and α'' is used to create α'''.

3.2.3 Integration with CasADi

The first step in integrating the C-code with CasADi is to load it as the internal function of a CasADi `Function`. This is most easily done with the CasADi `ExternalFunction`. Therefore the first step is to create a compiled version of the C-code using GCC.

```
system("gcc -shared -fPIC bSplineExt.c -o inner.so");
ExternalFunction f = ExternalFunction("inner.so");
```

It should here be noted that the constructor of an `ExternalFunction` does not allow for supplying additional arguments. In this case it corresponds to it being impossible to supply the actual table data to `bSplineExt.c` from the CasADi environment. For this reason the way the code is structured is that the table data is hard-coded into `bSplineExt.c`'s `init` function. Anyone wishing to have a different table data would therefore need to edit `bSplineExt.c`

Since `bSplineExt.c` encodes a function which takes two inputs and returns one output value this is the signature the `Function f` adapts. The two inputs correspond to the desired point to be interpolated at and the order of derivative searched for, respectively.

However, the standard way to extract the Jacobian of a `Function` within the CasADi environment is to call the function `fullJacobian` applied to the function in question, i.e. `f.fullJacobian()`. This means that the `Function` f will need to be used as the basis of a new `Function` in order to keep the signature, while f itself will never be directly called.

The approach for getting a `Function` of the correct signature was to extract the four different interpolation functions present in f and link them together with the CasADi function `setFullJacobian`. This was done by creating four `MX` objects and four `MXFunction` objects. More details below.

Four `MX` objects were locked as the values 0, 1, 2 respectively 3.

```
MX m0= MX(0);
MX m1= MX(1);
MX m2= MX(2);
MX m3= MX(3);
```

After that they were inserted into vectors of `MX` objects together with the input variable `x`:

```
vector<MX> in0, in1, in2, in3;
in0.push_back(x);
in0.push_back(m0);

in1.push_back(x);
in1.push_back(m1);

in2.push_back(x);
in2.push_back(m2);

in3.push_back(x);
in3.push_back(m3);
```

After that the results received by calling the `Function` f with `in0`, `in1`, `in2` respectively `in3` function was stored into additional vectors of `MX` objects:

```
vector<MX> out0, out1, out2, out3;

out0.push_back(f(in0)[0]);
out1.push_back(f(in1)[0]);
out2.push_back(f(in2)[0]);
out3.push_back(f(in3)[0]); //Since the result will be returned as
a list the first entry must be extracted from each call, hence [0]
```

`out0`, `out1`, `out2` and `out3` now contain the result of taking the 0th, 1st, 2nd and 3rd order derivative of the interpolation function f, respectively. With `x` as input the four requested `Functions` can now be created and linked together.

```
MXFunction d0func = MXFunction(x,out0);
MXFunction d1func = MXFunction(x,out1);
MXFunction d2func = MXFunction(x,out2);
MXFunction d3func = MXFunction(x,out3);

d0func.init();
d1func.init();
d2func.init();
d3func.init();

d2func.setFullJacobian(d3func);
d1func.setFullJacobian(d2func);
d0func.setFullJacobian(d1func);
```

`d0func` now contains the entirety of the interpolation, with its derivatives accessible by the `Function fullJacobian`.

4 Results

An important note about the results is that the graphs have been made in Python, due to there existing the library `matplotlib`[14] with which plots of the result can easily be made. To highlight that the two implementations of the algorithm return the same result Figure 3 contains two plots; one corresponds to the interpolation of Table 2 in Python and one corresponds to the interpolation in CasADi, with the result exported and plotted in Python. The largest difference between these plots are the order of 10^{-13}. To illustrate this difference it was charted in Figure 4.

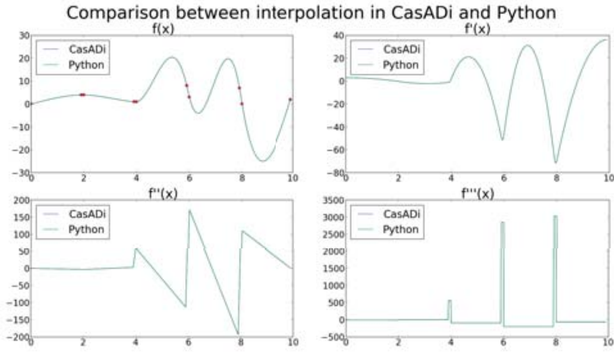

Figure 3: Comparison between interpolation in CasADi and Python

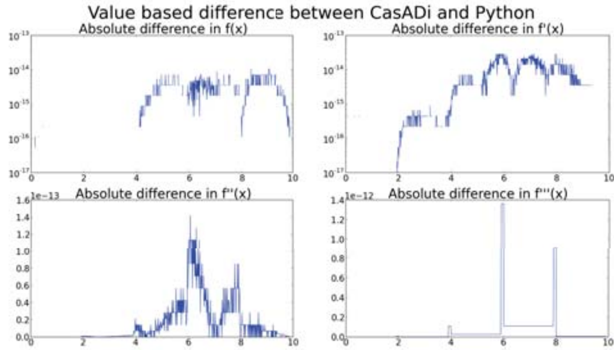

Figure 4: Comparison between interpolation in CasADi and Python

4.1 Accuracy of the interpolation

In order to verify the accuracy several interpolation tables were created. These tables can be seen in Table 1, 2 and 3.

x	0	1	2
y	0	1	2

Table 1: A small interpolation table

x	0.0	1.9	2.0	3.9	4.0	5.9	6.0	7.9	8.0	9.9
y	0.0	4.0	4.0	1.0	1.0	8.0	3.0	7.0	0.0	2.0

Table 2: A medium interpolation table

x	0	1	2	3	4	5	6	7	8	9	20	21	22	23	24	25	26	27	28	29	30
y	0	-1	2	-1	4	-5	6	-7	8	-9	20	-21	22	-23	24	-25	26	-27	28	-29	30

Table 3: A large interpolation table

In Figure 5, 6 and 7 the interpolations of Table 1, 2 respectively 3 can be seen, along with their derivatives. They also contain the various derivatives calculated by finite differences. As can be seen the resulting graph appears to be of C^2 continuity, given that only the third derivative is decidedly discontinuous. The first and second order derivative should be C^1 and C^0 respectively, and by their appearance such appears to be the case. The curve also passes through the prescribed y values at the prescribed knots, in accordance to the demands of the task.

As can be seen particularly in Figure 7 a larger distance between two concurrent knots may cause the interpolation to reach values that are particularly large or small compared to the values of two concurrent knots with a smaller distance.

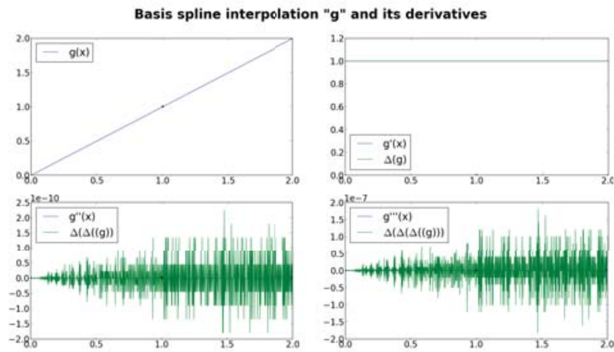

Figure 5: Interpolation of a small table

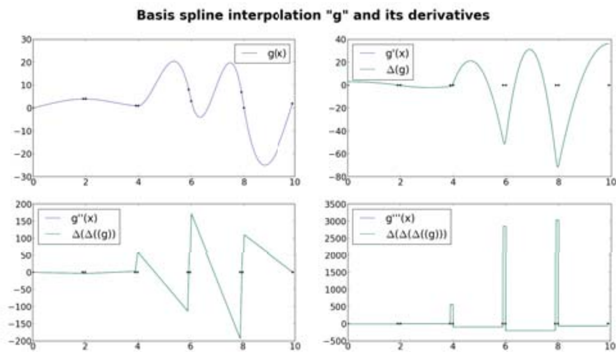

Figure 6: Interpolation of a medium table

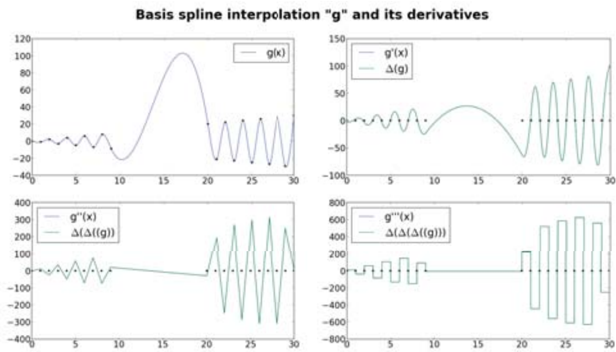

Figure 7: Interpolation of a large table

4.2 Accuracy of the derivative interpolation

In the aforementioned plots the derivatives are plotted in two ways: firstly by calling the `bSpline`'s derivative function and secondly by taking the values of the first graph and by calculating finite difference approximations of the derivatives.

The difference between the result when calculating the derivatives in such ways can be viewed in Figure 8, 9 and 10.

As can be seen particularly clearly in Figure 9 and 10 the difference between the two approaches become particularly large in the direct vicinity of the knots. This difference can be attributed to the fact that when the derivatives are calculated by finite differences the values used are taken from the initial graph of the function itself. This means that the points that are in the direct vicinity of the knots will by necessity use values from different b-splines when calculating their derivatives. Hence it is only natural that a derivative that should have a much sharper curvature will, when calculating with finite differences, get smoother. This is illustrated by zooming in around these knots, as in Figure 11.

It can also be observed that the first derivative of all examined interpolation functions save for the one in Figure 8 got a larger difference than for the second and third derivative, if one ignores the points in direct vicinity of the knots. Further investigations showed that increasing the amount of values charted made this difference decrease by a tenfold. It is therefore likely that the difference can be more attributed to inaccuracy in the derivatives calculated by finite differences.

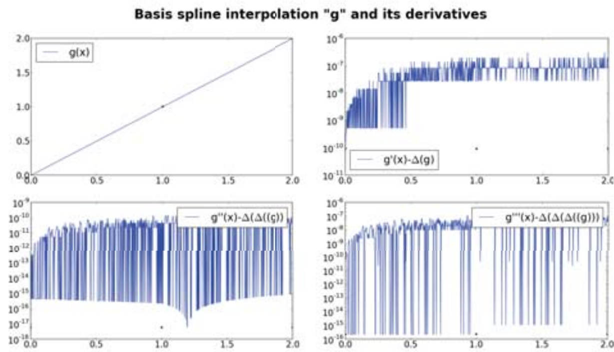

Figure 8: Interpolation compared to finite differences for a small table

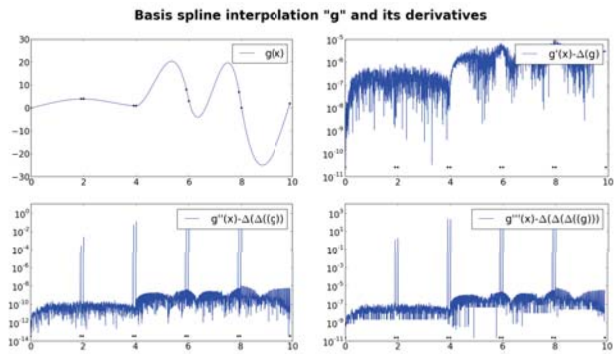

Figure 9: Interpolation compared to finite differences for a medium table

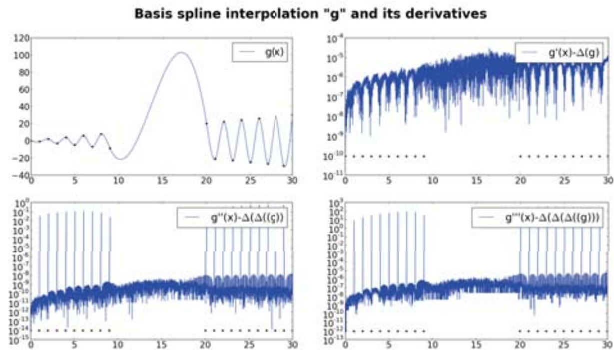

Figure 10: Interpolation compared to finite differences for a large table

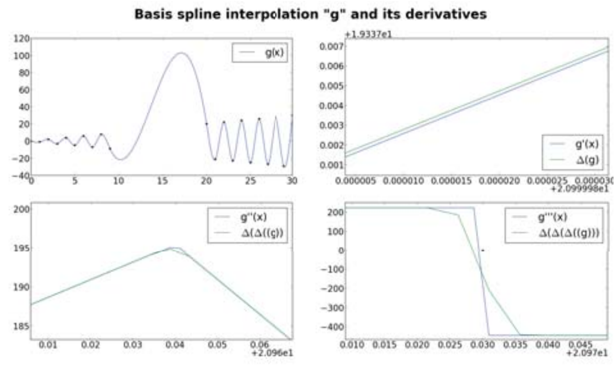

Figure 11: Interpolation of a large table (detail)

4.3 Examining basis functions

Since the interpolation can be said to be a linear combination of the basis functions it is worthwhile to consider what the error is when only the basis functions themselves are examined. Any flaws discovered with that interpolation will be linearly increased with a greater outstanding y value at the corresponding knot. This corresponds to interpolating a table where the y vector consists of zeroes apart from one non-zero value. An interpretation with a y vector with additional non-zero values should have flaws that are of the same nature as the one with one outstanding value, only additionally located at these other non-zero values.

4.3.1 Overall shape

The first thing to examine was the overall shape of the resulting interpolating function. This was done in two experiments.

First an x vector with the values $x_i = i, i = 0 \dots 10$ and y vector with only $y_5 = 1$ was interpolated. Afterwards a knot was inserted halfway between each knot of the x vector with the corresponding y values being set to zero. This knot-insertion was repeated 10 times.

The second experiment was repeating the first experiment with the x vector starting as the x vector of Table 2.

The result of the interpolation was charted together with the absolute error in every knot. The error in some cases was equal to zero, and in those cases the value charted was set to a value greater than zero but lower than the highest non-zero error. This was to ensure that the key features of the graphs would be visible. Some key graphs from the first experiment can seen in Figure 16 and 17; some key graphs from the second can be seen in Figure 12,13,14 and 15. Note that in the lower graph of these figures the x axis does not represent the position of the knots, instead the absolute error is plotted as a function of the knot number.

The most noticeable feature of Figure 12 is that two bulges with values much greater than the non-zero value appears in the neighbourhood of the knot with non-zero value. This should be seen in comparison to the interpolation of the equidistantly distributed knots, where the bulge is centred on the non-zero knot and not greater in value than 1. This follows quite naturally from the fact that in the non-equidistantly distributed interpolation the distance over which the interpolation function must go from one to zero

is much smaller, meaning that the derivative over this interval much be must greater. In order to maintain C^2 continuity it is only natural that the function would need to create two bulges in the intervals around this knot. In Figure 13, 14 and 15 additional bulges are clearly visible to the left of the non zero value. This is in contrast to Figure 12 where those were barely visible. This is due to the additional knots to the immediate right of the non zero value not allowing for the large bulge that formerly was located there. It's continuity preserving functionality must therefore be expressed by enlarging the bulges in the intervals of greater distance.

By comparing the two different experiments one may observe that no matter how the knots are distributed the error is polynomially dependent upon the distance in knot number to the the knot with the non zero value.

4.3.2 Accuracy (equidistant knots)

The second thing to examine was how accurately the resulting function interpolates the table data. Since only the actual knots have a definite correct value to interpret, these are the points that was examined for a table data where one value was non-zero, see Figure 16 and17.

As can be seen, with increasing distance to the non-zero knot the error in interpolation decreases exponentially. The maximum error was of the order of 10^{-15} with irregular knots and 10^{-16} with equidistant knots. Assuming equidistant knots the maximum error one value could transfer to another knot would therefore be of the order of 10^{-16}. Given that the error for one non-zero value is proportional to the non-zero value itself, it would mean that the maximum error in any point of an equidistant interpolation would be

$||error||_\infty \leq ||y||_\infty \cdot \epsilon_{\texttt{machine}}$

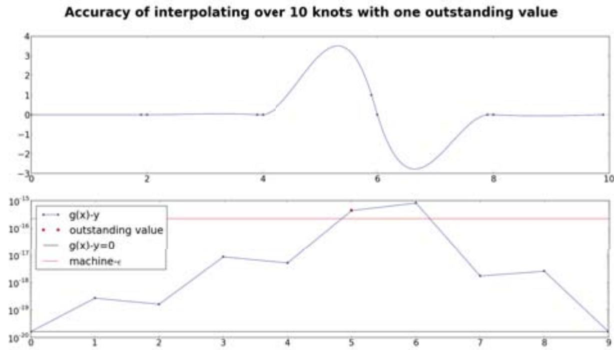

Figure 12: Interpolation of an outstanding value with 10 knots (upper graph); The error at each knot (lower graph)

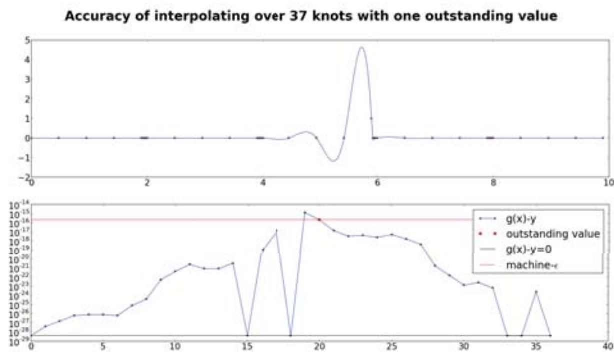

Figure 13: Interpolation of an outstanding value with 37 knots (upper graph); The error at each knot (lower graph)

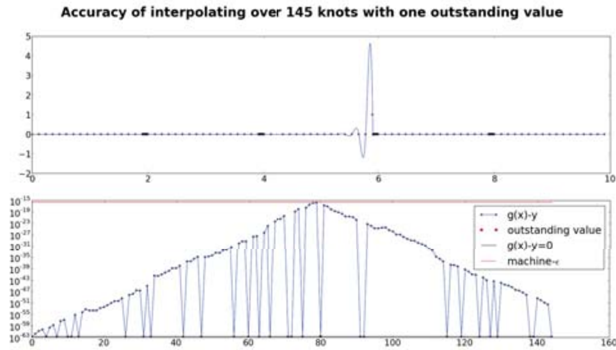

Figure 14: Interpolation of an outstanding value with 145 knots (upper graph); The error at each knot (lower graph)

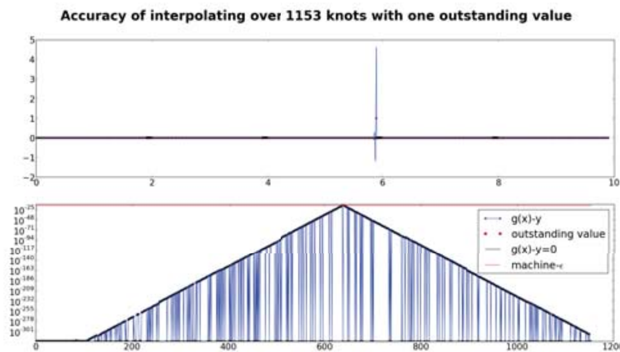

Figure 15: Interpolation of an outstanding value with 1153 knots (upper graph); The error at each knot (lower graph)

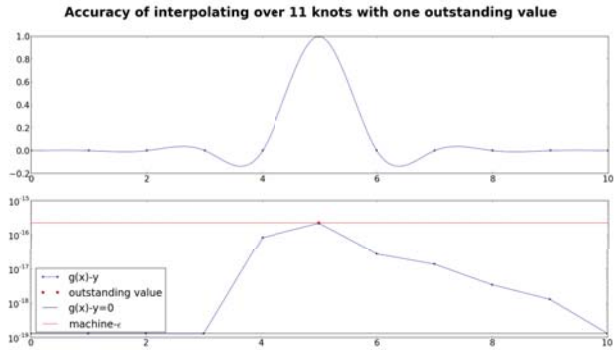

Figure 16: Interpolation of an outstanding value with 11 knots (upper graph); The error at each knot (lower graph)

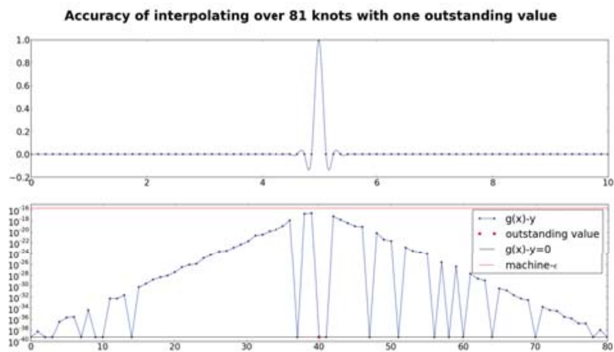

Figure 17: Interpolation of an outstanding value with 81 knots (upper graph); The error at each knot (lower graph)

4.3.3 Accuracy (varying knot distance)

Since C^2 continuity must be maintained even when the individual distances between concurrent knots varies greatly, and as discovered in Figure 12 these can cause additional bulges, it is worthwhile to consider their effect.

In order to test the effect of a greatly varying step length the x-values was created by alternating between a step length with a high order of magnitude and a step length of a low order of magnitude. In order to let there be a certain variance these step lengths were multiplied by the absolute value of a number generated from the normal distribution. If any feature could be gathered from having a completely regular irregularity (as in the x vector of Table 2, where the step length alternated between 0.1 and 1.9) this feature would be eliminated. In order to have some y values these were randomly generated from a normal distribution and multiplied by 10^9.

The test was performed by generating 100 tables for each step-length-ratio examined. The maximum error in each test was stored and charted. From section (4.3.2) the maximum error at a point of equidistant knots was proportional to $\max(|y|)$. To eliminate that factor the maximum error was divided by that number before being charted. The result can be seen in Figure 18, the average proportional error can be seen in Figure 19.

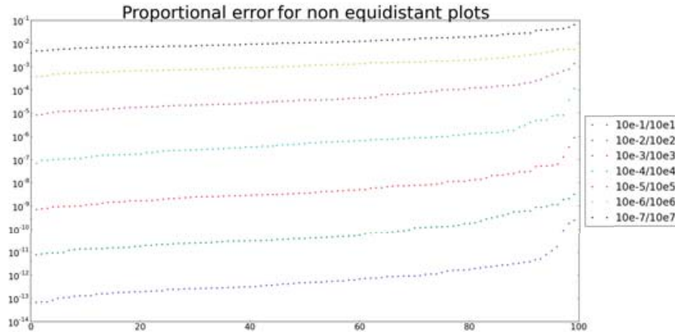

Figure 18: The error for interpolations when the step length is varying greatly

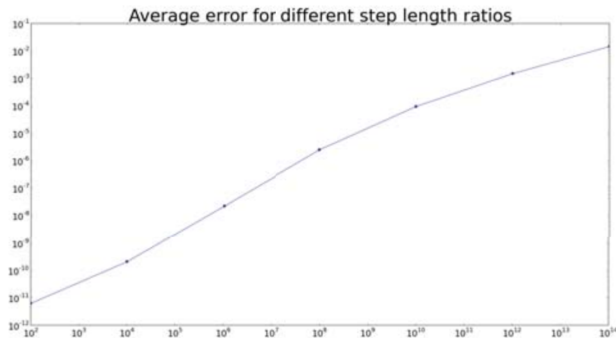

Figure 19: The average error for each step length ratio

From Figure 19 it appears that the total error of the interpolation is proportional to the maximum ratio between the step lengths as well.

5 Discussion

The thesis covered a variety of fields, and so does the discussion. In section 5.1 ways to make the code be better integrated in CasADi are discussed. In section 5.2 ways to implement further improvements to the code are discussed.

5.1 Integration in CasADi

From an end user's perspective an obvious improvement would be the ability to supply a table to `bSplineExt.c`. In the current implementation the end user needs to edit `bSplineExt.c` every time another table is to be interpolated; the ideal set up would be if the table data could be supplied through the constructor of `ExternalFunction` in `CasADi`. `CasADi` is very much a project under constant development, and the lack of a user specified field in the constructor of the `ExternalFunction` is an issue that is currently acknowledged[7].

Another problem with the current implementation is that any point may be interpolated, even those outside the span of x. The current solution is to return zero if such a point is to be interpolated. One future development would be to either let the function return an `Error` if the point is outside of the span or extrapolating from the current data. Since the end points have the second order derivative as 0 as per the natural boundary conditions and the first derivative at the same points are calculable a linear extrapolation would be possible. As a linear function is of C^∞ continuity the complete function would still remain of C^2 continuity.

Given that the motivation for this thesis was to be able to calculate the solution to optimization problems even when some of the relationships are expressed by tables, one future development would be to integrate this code into the optimization tool chain. The steps required in order to do so would be to rewrite `bSplineExt.c` as a stand-alone function, not following the signatures required by CasADi's `ExternalFunction` by necessity. Support for having a function in Modelica externally defined as a C file already exists, and defining the derivative of a variable in a Modelica model is standard operating procedure in Modelica. This means that the information contained within `bSplineExt.c` may be extracted for the relationships Modelica demands. As soon as CasADi's `ExternalFunction` has support for putting in additional

arguments in its constructor the plan would be to develop `CasADiInterface` to be able to accept a function with the interpolation code and a table as inputs in `ExternalFunction` thereby ensuring that the interpolation code could be used from a CasADi environment. It should also be noticed that the latest version (2.5) of CasADi was not integrated in JModelica.org; the last version still integrated (2.1) used different syntax. The choice to focus on a CasADi version not integrated was made to future proof the code.

5.2 Possible improvements

5.2.1 Extension of the number of dependent variables

If one wishes to have multiple variables dependent on one variable the current solution would be to call `bSplineExt` several times with different y vectors for each dependent variable. A future development would be to allow for the y vector to be a vector of rows, signifying multiple variables dependent on x. Since the formulas for the b-splines (Equation (1) and (2) are independent of the y-values, the only difference would be that the α vector would be a vector of rows as well, and the calculation of it and the corresponding α', α'' and α''' vectors would need to account for this. In principle this means that the calculations applied to one value of the y vector would need to be repeated for all the values of the row that took that value's place.

5.2.2 Extension of the number of independent variables

The principles used when interpolating with b-splines in one dimension may also be used to interpolate in several dimensions. For the two dimensional case this would correspond to two vectors x and y:

$$\boldsymbol{x} = \{x_0, x_1, \cdots, x_n\}, \boldsymbol{x} \in \mathbb{R}^{n+1}$$
$$\boldsymbol{y} = \{y_0, y_1, \cdots, y_m\}, \boldsymbol{y} \in \mathbb{R}^{m+1}$$

and a table Z:

$$\boldsymbol{Z} = \begin{pmatrix} z_{0,0} & z_{0,1} & \cdots & z_{0,n} \\ z_{1,0} & z_{1,1} & \cdots & z_{1,n} \\ \vdots & \vdots & \vdots & \vdots \\ z_{m,0} & z_{m,1} & \cdots & z_{m,n} \end{pmatrix}, \boldsymbol{Z} \in \mathbb{R}^{(n+1)\times(m+1)}$$

where an interpolative function $f(x, y)$

$$\begin{cases} f \in \mathbb{R}[x_0, x_n] \times \mathbb{R}[y_0, y_m] \Rightarrow \mathbb{R}, \\ f(x_i, y_j) = z_{j,i} \end{cases}$$

where

$$\begin{cases} i, j \in \mathbb{N}, \\ 0 \leq i \leq n, \\ 0 \leq j \leq m \end{cases}$$

is the requested function.

In order to construct f one would use *bivariate splines*([12], chapter 2). These are constructed by creating b-splines along the x- and y-axis of the interpolation, and creating the complete interpolation by multiplying b-splines of the x-axis with b-splines of the y-axis, i.e.

$$f(x, y) = \sum_{i=-k}^{n} \sum_{j=-k}^{m} c_{i,j} N_{i,k+1}(x) M_{j,k+1}(y) \tag{18}$$

where $N_{i,k+1}(x)$ is the b-spline interpolation along the x axis, $M_{j,k+1}(y)$ is the b-spline interpolation along the y axis, $c_{i,j}$ are values for ensuring that the interpolation passes through the values of Z at the knots and k is the degree. Compare this formula with Equation (6).

This interpolation formula and all its partial derivatives up to the k:th is continuous ([12], page 24), i.e.
$\frac{\partial^{i+j} f(x,y)}{\partial x^i y^j} \in C[x_0, x_n] \times [y_0, y_n], 0 \leq i < k, 0 \leq j < k$

The corresponding task would then be to calculate the matrix C which contains all the $c_{i,j}, 0 \leq i \leq n, 0 \leq j \leq m$ values.

The approach would be to treat the columns of Z as the y values in the one dimensional case and use the x values as the x values. This would correspond to solving a one dimensional problem $m + 1$ times. The resultant α vectors would be stored as rows of a matrix A.

The second step would be to treat the columns of A as the y values in the one dimensional case and the y values as the x values of the one dimensional case. This would correspond to solving a one dimensional problem $n + 1$ times. This would mean that if the resultant matrix is interpolated along the y axis at the knots of the y vector the resultant vector would correspond

to one of the rows of A, i.e. the α vector for that row. At that point the problem would be an ordinary one dimensional interpolation, and hence return a correct result. The resultant matrix would therefore be the C matrix.

Using the same principle it would be possible to extend the number of independent variables to any number the user would require.

5.2.3 Decreasing oscillations

As was seen in section (4.3.1) and in Figure 13 in the current implementation it was not uncommon for the interpolation to cause oscillations. Depending on what the table data represents this might be a less than ideal interpolative function. To counteract this one might use *minimum curvature splines* ([9], chapter 6). The principle is to select fewer spline knot locations than the initial data, and determine the spline coefficients that makes the sum of the least square errors at each knot be minimized, i.e. if $f(x)$ is the b-spline interpolation with an α vector then the task would be to minimize

$$g(c_{i,j}) = \sum_{k=1}^{l}[f(x_k, y_k) - z_k]^2 \tag{19}$$

where l is the amount of knots left after eliminating knots from the initial data, and (x_k, y_k) corresponds to the coordinates of the remaining knots. To stop the oscillations one imposes monotonicity on the individual intervals, i.e.

$$\left[\frac{\partial f}{\partial x}\right] \geq 0 \text{ if } y_{k+1} - y_k \geq 0$$
$$\left[\frac{\partial f}{\partial x}\right] \leq 0 \text{ if } y_{k+1} - y_k \leq 0$$

and the same conditions for the y axis. With monotonicity imposed the oscillations should be eliminated, as long as a solution can be found. The final problem to solve turns out to be to minimize the "curvature" of the interpolation, given by the problem

$$\min g(c_{i,j}) = \sum_{k=1}^{L}[\frac{\partial^2 f}{\partial x^2}(x_k, y_k)]^2 + [\frac{\partial^2 f}{\partial y^2}(x_k, y_k)]^2$$
$$\text{s.t.} z_k - \epsilon \leq f(x_k, y_k) \leq z_k + \epsilon$$

where ϵ is a data precision.

References

References

[1] Casadi's official website. https://github.com/casadi/casadi/wiki. Accessed: 2015-10-05.

[2] Modelica Standard Library. http://modelica.github.io/Modelica/. Accessed: 2015-10-12.

[3] Tridiagonal matrix algorithm. https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm. Accessed: 2015-10-05.

[4] Johan Åkesson, K-E Årzén, Magnus Gäfvert, Tove Bergdahl, and Hubertus Tummescheit. Modeling and optimization with optimica and jmodelica. org—languages and tools for solving large-scale dynamic optimization problems. *Computers & Chemical Engineering*, 34(11):1737–1749, 2010.

[5] Johan Åkesson, Karl-Erik Årzén, Magnus Gäfvert, Tove Bergdahl, and Hubertus Tummescheit. Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problems. *Computers and Chemical Engineering*, 34(11):1737–1749, November 2010.

[6] Joel Andersson. *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, October 2013.

[7] Joel Andersson. User data in external function interface #1417. https://github.com/casadi/casadi/issues/1417, 2015. Accessed: 2015-10-05.

[8] Joel Andersson, Joris Gillis, and Moritz Diehl. *User Documentation for CasADi v2.3.0-51.6c493af*. CasADi.org, 2015.

[9] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*, volume 19. Siam, 2010.

[10] Lars-Christer Böiers. *Mathematical Methods of Optimization.* Studentlitteratur AB, 2010.

[11] Christof Büskens and Dennis Wassel. The esa nlp solver worhp. In *Modeling and Optimization in Space Engineering*, pages 85–110. Springer, 2013.

[12] Paul Dierckx. *Curve and Surface Fitting with Splines (Numerical Mathematics and Scientific Computation).* Clarendon Press, 6 1995.

[13] Hilding Elmqvist. *A Structured Model Language for Large Continuous Systems.* PhD thesis, Department of Automatic Control, Lund University, Sweden, May 1978.

[14] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[15] Björn Lennernäs. A casadi based toolchain for jmodelica.org, 2013. Master's thesis.

[16] Tom Lyche and Knut Morken. Spline methods draft. *Department of Informatics, University of Oslo, URL= http://heim. ifi. uio. no/knutm/komp04. pdf*, 2004.

[17] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER´S THESIS

Date of issue
November 2015

Document Number
ISRN LUTFD2/TFRT--5996--SE

Author(s)
Joakim Larsson

Supervisor
Toivo Henningsson, Modelon
Fredrik Magnusson, Dept. of Automatic Control, Lund University, Sweden
Pontus Giselsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle

Implementation of smooth interpolation for optimization

Abstract

Models of physical systems are often expressed as a system of mathematical expressions derived from first principles. However some of the relationships present in a model are more conveniently expressed as a table, i.e. for certain values of independent variables the values of dependent variables are known. This imposes a limitation on the optimization software to be used, since optimization software is often dependent on all relationships being expressed as mathematical functions. Many useful optimization methods also require that the functions be twice continuously differentiable.

In this thesis software for interpolating table data relationships between two variables as a twice continuously differentiable mathematical function has been developed. This software has also been prototypically made callable from the automatic differentiation tool CasADi.

CasADi is used in the optimization tool chain in JModelica.org, an open source platform for optimization and simulation. By implementing support for table based relations a larger range of problems may be solved using CasADi.

The software developed for interpolating tables uses cubic b-splines and de Boor evaluation. Using it one may evaluate the interpolant and its derivatives up to the third order. The resultant function is demonstrated to be twice continuously differentiable and to interpolate the value within machine epsilon range of the correct one at the data points, provided the table data points are equidistantly distributed. The oscillations that occur when interpolating non-equidistant table data points are also examined.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title	*ISBN*
0280-5316	

Language	*Number of pages*	*Recipient's notes*
English	1-48	
Security classification		

http://www.control.lth.se/publications/