Comput Econ (2007) 30:153–169
DOI 10.1007/s10614-007-9092-4

Multidimensional Spline Interpolation: Theory and Applications

Christian Habermann · Fabian Kindermann

Received: 7 April 2006 / Accepted: 15 May 2007 / Published online: 11 July 2007

Abstract Computing numerical solutions of household's optimization, one often faces the problem of interpolating functions. As linear interpolation is not very good in fitting functions, various alternatives like polynomial interpolation, Chebyshev polynomials or splines were introduced. Cubic splines are much more flexible than polynomials, since the former are only twice continuously differentiable on the interpolation interval. In this paper, we present a fast algorithm for cubic spline interpolation, which is based on the precondition of equidistant interpolation nodes. Our approach is faster and easier to implement than the often applied B-Spline approach. Furthermore, we will show how to loosen the precondition of equidistant points with strictly monotone, continuous one-to-one mappings. Finally, we present a straightforward generalization to multidimensional cubic spline interpolation.

Keywords Interpolation · Multidimensional cubic splines

JEL Classification C63 · C65

1 Motivation

In computational economics, one always faces the problem of approximation. Solving large scale OLG models, for example, one quickly is confronted with the problem that there is no analytical solution to household's optimization. Hence, computational economists use numerical solutions. A popular approach is a discretization of the state space of households together with a backward optimization as described e.g. in İmrohoroğlu et al. (1995, 90ff.) or Nishiyama and Smetters (2005, 1111ff.).

C. Habermann (✉) · F. Kindermann
Department of Economics, University of Würzburg, Sanderring 2, Würzburg, 97070 Germany
e-mail: christian.habermann@uni-wuerzburg.de

Beneath the need of several optimization algorithms, programmers often face the problem of interpolation. The most popular interpolation method surely is linear interpolation. In addition to the fact that it doesn't need much complex programming, linear interpolation can easily be generalized to the multidimensional case. However, it is a disadvantage that one needs a lot of data (i.e. interpolation nodes) to obtain good approximation results, which increases the calculation time.

Therefore, various interpolation algorithms, which are much better in fitting functions than linear interpolation, were introduced. Heer and Maussner (2005, 431ff.), for example, suggest Chebyshev polynomials to solve the interpolation problem. B-Splines, like described in Miranda and Fackler (2002), Judd (1998) or de Boor (1978), are also a very useful alternative, especially if one needs some more flexible functions than polynomials.

In this paper, we will give an algorithm for cubic spline interpolation which is a special case of the B-Spline approach. In the next section, after presenting some theory on splines, we will introduce a fast algorithm which is easy to compute in any programming language. Section 3 shows, how spline interpolation can easily be generalized to the multidimensional case. Section 4 gives applications of spline interpolation comparing it to the linear interpolation and standard B-Splines. Section 5 concludes.

2 A Fast Solution to One-Dimensional Cubic Splines

2.1 Preliminaries

A grid $\Delta_n(a, b) = \{x_0, \dots, x_n\}$, $n \in \mathbb{N}$, on the interval $[a, b]$ is a set of points $x_i \in [a, b]$, $i = 0, \dots, n$, with $a = x_0 < x_1 < \cdots < x_n = b$. In the following we will call x_i nodes. By $\mathcal{S}_{k,l}(\Delta_n(a, b)), l, k \in \mathbb{N}_0, l < k$, we denote the function space of all one-dimensional, real-valued $C^l([a, b])$ functions on the bounded interval $[a, b]$ that are piecewise kth degree polynomial on every interval $[x_{i-1}, x_i], i = 1, \dots, n$. A spline of degree k and smoothness l with $n + 1$ nodes in $\Delta_n(a, b)$ is a function $s_n^{k,l} \in \mathcal{S}_{k,l}(\Delta_n(a, b))$. A spline of degree zero is a step function, a spline of degree one is a continuous, piecewise linear function, a spline of degree two and smoothness one is a once continuously differentiable, piecewise second-order polynomial function, etc.

If $\Delta_n(a, b) = \{x_0, \dots, x_n\}$ and the interpolation data $y_i \in \mathbb{R}, i = 0, \dots, n$, is given, a spline $s_n^{k,l} \in \mathcal{S}_{k,l}(\Delta_n(a, b))$ interpolates the data $\{y_i\}_{i=0,\dots,n}$, if it satisfies the $n + 1$ interpolation conditions

$$s_n^{k,l}(x_i) = y_i \quad \text{for } i = 0, \dots, n. \tag{2.1}$$

In the following, we will concentrate on splines of order three and smoothness two, so-called cubic splines. Therefore, for the sake of simplicity, we denote $\mathcal{S}_3(\Delta) := \mathcal{S}_{3,2}(\Delta_n(a, b))$ the function space of all degree three and smoothness two splines on the grid $\Delta_n(a, b)$ and $s \in \mathcal{S}_3(\Delta)$ a degree three and smoothness two spline.

There are several ways to represent such a spline. Since a cubic spline is piecewise third-order polynomial, an intuitive explicit form of a spline function is

$$s(x) = a_i + b_i x + c_i x^2 + d_i x^3 \quad \text{if } x \in [x_{i-1}, x_i] \quad \text{for } i = 1, \dots, n$$

with some adequate coefficients $a_i, b_i, c_i, d_i \in \mathbb{R}$. Given the interpolation data y_i, $i = 0, \dots, n$, we can determine the coefficients a_i, b_i, c_i, d_i for an interpolating spline. The computation of these coefficients is e.g. described in Judd (1998, 225ff.). However, this intuitive approach has several disadvantages. First, in order to evaluate the spline function several times, one has to calculate $4n$ coefficients and store them permanently. Second, and even worse, in order to compute the coefficients one has to solve a linear equation system of $4n$ equations in $4n$ unknowns which may be costly in terms of time. In order to relax these space and time constraints, basis functions of the function space $\mathcal{S}_{k,l}(\Delta_n(a,b))$ were proposed. The most popular basis functions are the B-Spline functions given by the recursive formula

$$\begin{aligned} B_j^{k,\Delta}(x) &= \frac{x - x_{j-1-k}}{x_{j-1} - x_{j-1-k}} B_{j-1}^{k-1,\Delta}(x) \\ &\quad + \frac{x_j - x}{x_j - x_{j-k}} B_j^{k-1,\Delta}(x) \\ \text{with} \quad B_j^{0,\Delta} &= \begin{cases} 1 & \text{if } x_{j-1} \le x < x_j \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

for $j = 1, \dots, n+k$. The $n+k$ B-Spline basis functions $B_j^{k,\Delta}$ form a basis of the function space $\mathcal{S}_{k,k-1}(\Delta_n(a,b))$. A spline $s_n^{k,k-1}$ can therefore be represented by a linear combination of these B-Spline basis functions, i.e.

$$s_n^{k,k-1}(x) = \sum_{j=1}^{n+k} c_j B_j^{k,\Delta}(x)$$

with some suitable coefficients c_j. Note that this modification allows to reduce the linear equation system defining the spline coefficients to $n+k$ equations. For some more information on B-Splines see Miranda and Fackler (2002, 115ff.) or de Boor (1978).

2.2 Interpolation with Equidistant Nodes

If Δ is a set of equidistant nodes, we can apply a fast and easy to implement algorithm for one-dimensional cubic spline interpolation. The approach requires only $n+3$ spline coefficients and does not need a sophisticated linear equation system solver. Consequently, this interpolation method could be easily implemented by programmers in any language.

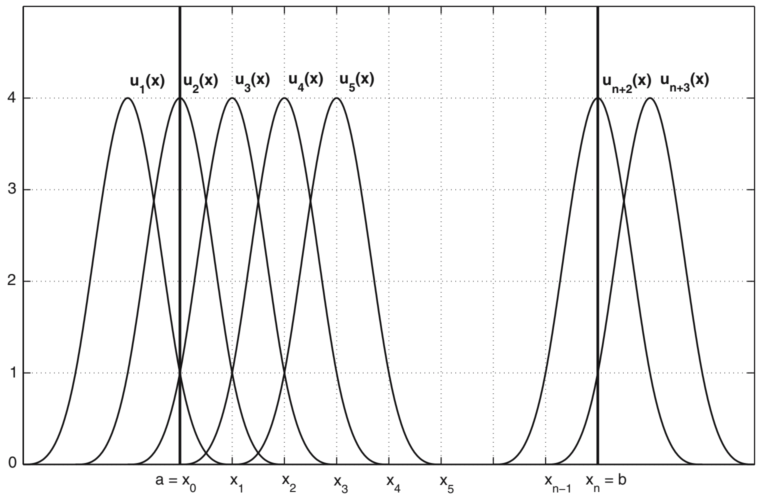

Fig. 1 Base of $\mathcal{S}_3(\Delta)$. This figure shows the basis functions $u_1, u_2, \ldots, u_{n+3}$ of the function space of splines $\mathcal{S}_3(\Delta)$ on the interval $[a, b]$ with equidistant nodes (see (2.2) and (2.3))

Given a set $\Delta = \{x_0, x_1, \ldots, x_n\}$ of equidistant nodes, i.e.

$$x_i = a + ih, \qquad h = \frac{b-a}{n}, \quad i = 0, \ldots, n, \tag{2.2}$$

and the associated interpolation data $y_0, y_1, \ldots, y_n \in \mathbb{R}$, we define $\mathcal{S}_3(\Delta)$ as in Sect. 2.1 and choose the functions

$$u_k(x) = \Phi\left(\frac{x-a}{h} - (k-2)\right), \quad k = 1, \ldots, n+3 \tag{2.3}$$

with

$$\Phi(t) = \begin{cases} (2-|t|)^3, & 1 \le |t| \le 2 \\ 4 - 6|t|^2 + 3|t|^3, & |t| \le 1 \\ 0, & \text{elsewhere,} \end{cases} \tag{2.4}$$

see Fig. 1. $U = \{u_1, u_2, \ldots, u_{n+3}\}$ is a basis of the $n+3$-dimensional space $\mathcal{S}_3(\Delta)$, i.e. every $s \in \mathcal{S}_3(\Delta)$ can be written as

$$s(x) = \sum_{k=1}^{n+3} c_k u_k(x) \tag{2.5}$$

with some coefficients c_k. Actually, the functions u_k are an explicit form of the $B_k^{3,\Delta}$ B-spline basis functions. They can be obtained by employing equidistant nodes like in (2.2) and some straight forward computation.

Due to the bounded support $\operatorname{supp}(u_k) = [x_{k-4}, x_k] \cap [a, b]$, u_k vanishes outside the bounded interval $[x_{k-4}, x_k] \cap [a, b]$ and Eq. (2.5) changes to

$$s(x) = \sum_{k=l}^{m} c_k u_k(x), \qquad l = \left\lfloor \frac{x-a}{h} \right\rfloor + 1, \quad m = \min(l+3, n+3), \tag{2.6}$$

where $\lfloor\cdot\rfloor$ denotes the floor function.

With $\mathcal{S}_3(\Delta)$ being an $n+3$-dimensional space, we need $n+3$ interpolation conditions in order to determine the interpolating spline function uniquely. However $s(x_i) = y_i, i = 0, \ldots, n$, only specifies $n+1$ conditions. Therefore, we add two conditions which define the second-order derivatives of the spline function at the boundaries a and b to be equal to some exogenous constants[1]

$$s''(a) = \alpha, \quad s''(b) = \beta.$$

There are several possibilities to choose the constants α and β. Setting $\alpha = \beta = 0$ leads to a *natural spline*. Natural splines solve the problem

$$\min_{g \in \mathcal{M}} \int_a^b g''(x)^2 \, dx,$$

where $\mathcal{M} \subset C^2[a, b]$ is the set of all twice continuously differentiable functions on $[a, b]$ interpolating the data $y_0, \ldots, y_n$, i.e. the natural spline is the function which minimizes the total curvature over the interval $[a, b]$. Alternatively, one could think of several possibilities to approximate the second-order derivatives at a and b using the given interpolation data in order to get a better approximation in between the interpolation nodes.

In the following we assume that α and β are specified. Then the $n+3$ interpolation conditions and the fact that u_k vanishes outside the bounded interval $[x_{k-4}, x_k] \cap [a, b]$ define the linear equation system

$$\begin{aligned}
s''(x_0) &= \sum_{i=1}^{3} c_i u_i''(x_0) = \alpha, \\
s(x_i) &= \sum_{k=l}^{m} c_k u_k(x_i) = y_i, \qquad l = \left\lfloor \frac{x_i - a}{h} \right\rfloor + 1, \\
&\quad m = \min(l+3, n+3), \quad i = 0, \ldots, n, \\
s''(x_n) &= \sum_{i=n+1}^{n+3} c_i u_i''(x_n) = \beta
\end{aligned}$$

[1] There are several ways to specify those two missing condition. Behforooz and Papamichael (1979) analyze a whole class of end conditions and their implication on error bounds.

which is equivalent to

$$\begin{pmatrix} u_1''(x_0) & u_2''(x_0) & u_3''(x_0) & 0 & \dots & \dots & 0 \\ u_1(x_0) & u_2(x_0) & u_3(x_0) & 0 & \dots & \dots & 0 \\ 0 & u_2(x_1) & u_3(x_1) & u_4(x_1) & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & u_n(x_{n-1}) & u_{n+1}(x_{n-1}) & u_{n+2}(x_{n-1}) & 0 \\ 0 & \dots & \dots & 0 & u_{n+1}(x_n) & u_{n+2}(x_n) & u_{n+3}(x_n) \\ 0 & \dots & \dots & 0 & u_{n+1}''(x_n) & u_{n+2}''(x_n) & u_{n+3}''(x_n) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n+1} \\ c_{n+2} \\ c_{n+3} \end{pmatrix} = \begin{pmatrix} \alpha \\ y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \\ \beta \end{pmatrix}. \quad (2.7)$$

Employing equidistant interpolation nodes like in (2.2), (2.7) turns into

$$\begin{pmatrix} 1 & -2 & 1 & 0 & \dots\dots\dots & 0 \\ 1 & 4 & 1 & 0 & \dots\dots\dots & 0 \\ 0 & 1 & 4 & 1 & 0 \dots\dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 \dots\dots & 0 & 1 & 4 & 1 & 0 \\ 0 \dots\dots\dots & & 0 & 1 & 4 & 1 \\ 0 \dots\dots\dots & & 0 & 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ \vdots \\ \vdots \\ c_{n+2} \\ c_{n+3} \end{pmatrix} = \begin{pmatrix} \alpha h^2/6 \\ y_0 \\ y_1 \\ \vdots \\ y_{n-1} \\ y_n \\ \beta h^2/6 \end{pmatrix}. \quad (2.8)$$

The linear equation system (2.8) can be solved in three steps:

1. From the first and the last two lines we obtain

$$c_2 = \frac{1}{6}\left(y_0 - \frac{\alpha h^2}{6}\right), \quad c_{n+2} = \frac{1}{6}\left(y_n - \frac{\beta h^2}{6}\right).$$

2. Next, we have to compute the solution of the system

$$\begin{pmatrix} 4 & 1 & 0 & \dots\dots\dots\dots & & 0 \\ 1 & 4 & 1 & 0 & \dots\dots\dots & 0 \\ 0 & 1 & 4 & 1 & 0 \dots\dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 \dots\dots & 0 & 1 & 4 & 1 & 0 \\ 0 \dots\dots\dots & & 0 & 1 & 4 & 1 \\ 0 \dots\dots\dots\dots & & & 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} c_3 \\ c_4 \\ \vdots \\ \vdots \\ \vdots \\ c_n \\ c_{n+1} \end{pmatrix} = \begin{pmatrix} y_1 - c_2 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-3} \\ y_{n-2} \\ y_{n-1} - c_{n+2} \end{pmatrix}.$$

As the matrix is a symmetric and tridiagonal matrix, the computation of $(c_3, \dots, c_{n+1})$ should be no problem, see Judd (1998, 61ff.).

3. The first and the last coefficient are then derived from

$$c_1 = \frac{\alpha h^2}{6} + 2c_2 - c_3, \quad c_{n+3} = \frac{\beta h^2}{6} + 2c_{n+2} - c_{n+1}.$$

Given the coefficients $c = (c_1, \ldots, c_{n+3})^T$ of the interpolating spline function s in (2.5) we can compute the value of the interpolating spline at any $x \in [a, b]$.

2.3 Re-scaling of the Nodes

The last section developed a fast algorithm for the computation of splines with equidistant sets of interpolation nodes. With a non-equidistant grid we can use the same algorithm if the following condition is satisfied:

Let the set of nodes Δ and the interpolation data $y_0, \ldots, y_n$ be given. If the x_i are not evenly spaced but determined by a strictly monotone, continuous one-to-one mapping gr: $[0, n] \to [x_0, x_n]$ such that $x_i = \mathrm{gr}(i),\ i = 0 \ldots, n$, we can proceed as follows:

1. Interpolate the data $y_0, \ldots, y_n$ with the algorithm given in Sect. 2.2 and obtain a spline s with coefficients $c_k, k = 1, \ldots, n+3$.
2. In order to evaluate the interpolating function $s^*(x)$ at any $x \in [a, b]$ first compute $z = \mathrm{gr}^{-1}(x) \in [0, n]$ and evaluate $s(z)$, where $a = 0, b = n, h = 1$ (cf. Sect. 2.2). $s^*(x) = s(z)$ then is the value of the function interpolating the data y_i. Notice that s^* satisfies the interpolation conditions $s^*(x_i) = s(\mathrm{gr}^{-1}(x_i)) = s(i) = y_i,\ i = 0, \ldots, n$.

This approach defines a re-scaling of the x-coordinate. As an example, Fig. 2 shows the re-scaling with the original nodes x_i satisfying $x_i = \mathrm{gr}(i)$,

$$\mathrm{gr}(i) = a \cdot ((1+g)^i - 1), \qquad i = 0, \ldots, 10, \quad a \approx 0.078, \quad g = 0.3. \tag{2.9}$$

The interpolation data y_i is computed from $y_i = \sqrt{x}_i$.[2] As one can see, the new scaling approach could allow a better approximation as steeper parts (especially around 0) are flattened.

3 Extension to Multidimensional Spline Interpolation

For a better understanding of multidimensional spline interpolation, we will first introduce the procedure in two dimensions. After knowing this algorithm, a generalization to n-dimensional interpolation will be no problem.

[2] Notice that we didn't chose $\mathrm{gr}(i) = i^2$ on purpose. This would result in a straight line. Since in reality one doesn't know the exact function the interpolation data is computed from, an exponentially growing grid like in (2.9) is useful as long as the function we want to interpolate has its steepest parts at the left endpoint of the interpolation interval. It is easy to fit this grid into an interval and, in addition, the growth of the grid can be directly controlled by the parameter g. However, there are various other grid making functions that may be useful with other interpolation data.

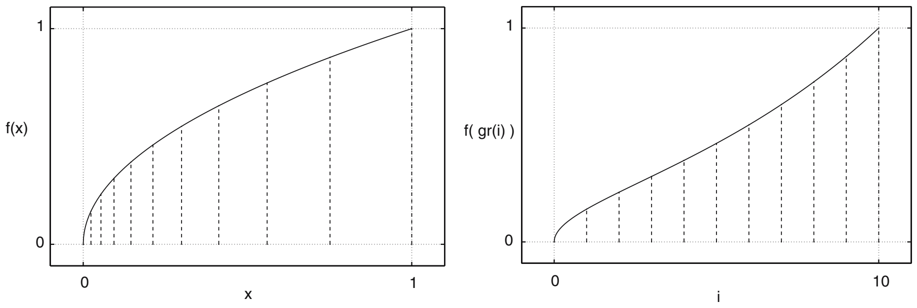

Fig. 2 Re-scaling approach

3.1 Bicubic Spline Interpolation

Given the equidistant grids $\Delta_1 = \{x_0, \dots, x_{n_1}\}$, $\Delta_2 = \{z_0, \dots, z_{n_2}\}$ with

$$x_{i_1} = a_1 + i_1 h_1, \quad h_1 = \frac{b_1 - a_1}{n_1}, \quad i_1 = 0, \dots, n_1$$
$$z_{i_2} = a_2 + i_2 h_2, \quad h_2 = \frac{b_2 - a_2}{n_2}, \quad i_2 = 0, \dots, n_2$$

on the intervals $[a_1, b_1]$ and $[a_2, b_2]$ and the interpolation data $y_{i_1 i_2}$, we are looking for a bicubic spline $s \in \mathcal{S}_3(\Delta_1) \times \mathcal{S}_3(\Delta_2) =: \mathcal{S}_{3,3}$ interpolating the data $y_{i_1 i_2}$. $\mathcal{S}_{3,3}$ is the tensor product of $\mathcal{S}_3(\Delta_1)$ and $\mathcal{S}_3(\Delta_2)$, i.e. if $\{u_1, \dots, u_{n_1+3}\}$ is a base of $\mathcal{S}_3(\Delta_1)$ and $\{v_1, \dots, v_{n_2+3}\}$ is a base of $\mathcal{S}_3(\Delta_2)$,

$$\left\{u_{i_1} v_{i_2} \mid i_1 \in \{1, \dots, n_1 + 3\}, i_2 \in \{1, \dots, n_2 + 3\}\right\}$$

is a base of $\mathcal{S}_{3,3}$. Hence, the dimension of $\mathcal{S}_{3,3}$ is $(n_1 + 3) \cdot (n_2 + 3)$ and any spline $s \in \mathcal{S}_{3,3}$ can be written as

$$s(x, z) = \sum_{i_1=1}^{n_1+3} \sum_{i_2=1}^{n_2+3} c_{i_1 i_2} u_{i_1}(x) v_{i_2}(z), \quad c_{i_1 i_2} \in \mathbb{R}.$$

We can compute $c_{i_1 i_2}$ by the following algorithm:

1. For $q = 0, \dots, n_2$ compute coefficients $c^*_{i_1 q}$ of the splines

$$s_{q,1}(x) = \sum_{i_1=1}^{n_1+3} c^*_{i_1 q} u_{i_1}(x) \quad \in \mathcal{S}_3(\Delta_1)$$

due to the interpolation conditions $s_{q,1}(x_{i_1}) = y_{i_1 q}, \ i_1 = 0, \dots, n_1$.

2. Now solve the spline interpolation problem $s_{i_1,2} \in \mathcal{S}_3(\Delta_2)$,

$$s_{i_1,2}(z) = \sum_{i_2=1}^{n_2+3} c_{i_1 i_2} v_{i_2}(z), \quad i_1 = 1, \dots, n_1 + 3,$$

   with respect to $s_{i_1,2}(z_q) = c^*_{i_1 q},\ q = 0, \dots, n_2$.
3. The bicubic spline interpolating $y_{i_1 i_2}$ is then given by

$$s(x, z) = \sum_{i_1=1}^{n_1+3} \sum_{i_2=1}^{n_2+3} c_{i_1 i_2} u_{i_1}(x) v_{i_2}(z),$$

$$u_{i_1}(x) = \Phi\left(\frac{x - a_1}{h_1} + 2 - i_1\right), \quad v_{i_2}(z) = \Phi\left(\frac{z - a_2}{h_2} + 2 - i_2\right).$$

Notice that the re-scaling approach can also be applied in the multidimensional case.

3.2 A Generalization to Multidimensional Problems

Knowing the bicubic spline interpolation, the generalization to a d-dimensional problem is straightforward. Given d different grids $\Delta_1, \Delta_2, \dots, \Delta_d$ and the interpolation data $y_{i_1 i_2 \dots i_d}$, the following two-step algorithm is applied for multidimensional spline interpolation:

1. If n_i is the number of nodes in Δ_i compute the splines $s_{q,1} \in \mathcal{S}_3(\Delta_1) \times \cdots \times S_3(\Delta_{d-1})$ for $q = 0, \dots, n_d$ with the interpolation data $y_{i_1 i_2 \dots i_{d-1} q}$ and obtain the coefficients $c^*_{i_1 i_2 \dots i_{d-1} q}$.
2. Interpolate for $i_1 = 1, \dots, n_1$, $i_2 = 1, \dots, n_2$, ..., $i_{d-1} = 1, \dots, n_{d-1}$ the data $c^*_{i_1 i_2 \dots i_{d-1} q}$ with nodes $x_q \in \Delta_d$. The coefficients $c_{i_1 i_2 \dots i_d}$ obtained from this interpolation are the coefficients of our interpolating spline function.

Therefore we can write the interpolating spline function at $x = (x^1, \dots, x^d) \in \mathbb{R}^d$ as

$$s(x) = \sum_{i_1=1}^{n_1+3} \cdots \sum_{i_d=1}^{n_d+3} c_{i_1 i_2 \dots i_d} \cdot \prod_{j=1}^{d} u^j_{i_j}\left(x^j\right)$$

with $u^j_{i_j}(x^j) = \Phi\left(\frac{x^j - a_j}{h_j} + 2 - i_j\right)$. Notice that the dimension of this problem is $\prod_{j=1}^{d}(n_j + 3)$.

4 Applications

In this chapter, we present an example for the use of the re-scaling approach and compare in terms of accuracy and speed the linear interpolation, the third degree B-spline

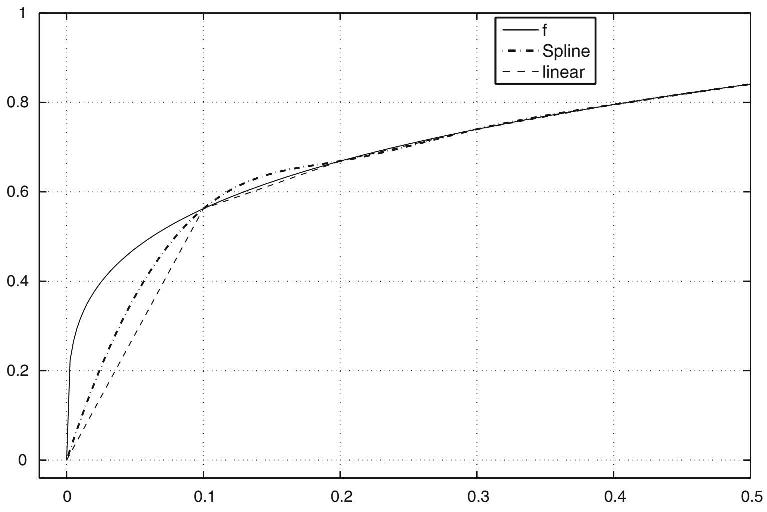

Fig. 3 Interpolation of $x^{0.25}$

interpolation and the fast spline algorithm described above. For the computation of B-splines we use the CompEcon Toolbox by Miranda and Fackler (2002). Besides many other programs, they provide some powerful and efficient algorithms for multi-dimensional interpolation.

We provide results for four different tests. In the first section, we demonstrate graphically the influence of the new scaling approach on the interpolation of some particular type of function. In Sect. 4.2 we apply the three different algorithms to an artificial interpolation problem. In the third section, we'll solve a two-dimensional household maximization problem in a partial equilibrium setting and in the last section we compare the solutions of a multidimensional OLG model, solved with different interpolation methods. The results in section two and three are computed with MATLAB whereas the OLG model is programmed in FORTRAN.

4.1 The Influence of the Re-Scaling Approach

For some utility functions used in economics (e.g. the CRRA-utility function) marginal utility goes to infinity as the amount of the valued item approaches zero. Interpolating a function with these properties, one always faces the problem of bad accuracy in a neighborhood around zero. In the following we will show how one could improve accuracy in this region without losing accuracy in the rest of the interpolation interval.

A group of functions which satisfy the condition mentioned above is $f_m(x) = x^m$, $0 < m < 1$. As cubic splines are curved functions in opposite to piecewise linear ones, it is easier for them to fit a curved function like x^m. Figure 3 demonstrates this issue showing the interpolating spline and piecewise linear function of $x^{0.25}$ on the interval $[0, 0.5]$ with $n = 5$ and equidistant interpolation nodes.[3]

[3] In this figure, we didn't show the results from a B-spline approach, since the result is very close to the one computed with our fast algorithm.

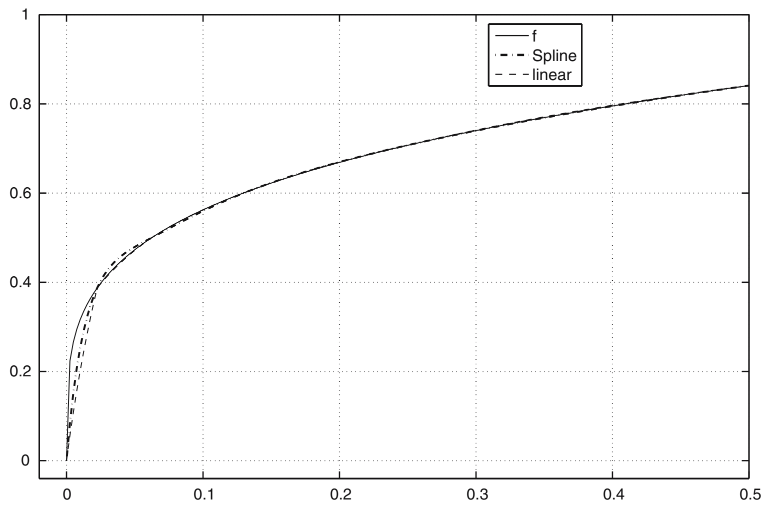

Fig. 4 Interpolation of $x^{0.25}$ with re-scaling

However, this figure also shows the problem of low accuracy in the region around zero. This problem can be reduced by re-scaling as explained in Sect. 2.3. Figure 4 displays the interpolating functions of $x^{0.25}$ after re-scaling the cubic spline as well as the linear interpolating function with the function (2.9) and parameters $a = 0.0279$ and $g = 0.8$.

Obviously, the accuracy in between the interpolation nodes in Fig. 4 rises since there are lots of points in the steep area around zero. Re-scaling, therefore, is a good solution to the problem described above. However, one should be aware that the re-scaling approach can only be applied if one can find the "problematic" parts of the function underlying the interpolation data.

4.2 Interpolation of Given Functions

Next we discuss a stylized interpolation problem. Consider the function

$$f_d(x) = \sum_{i=1}^{d} x_i^{\frac{1}{2i}}, \quad x = (x_1, \ldots, x_d) \in \mathbb{R}^d.$$

Tables 1–3 compare for $d = 1, 2, 3$ the results of different interpolation approaches for different interpolation nodes $n + 1$ on the interval $[0, 1]$. After the calculation of coefficients, the interpolating functions are evaluated at m evenly spaced points on the interpolation interval.[4] The error reported in the tables is the maximum absolute difference of the interpolating function and the original function f. The columns

[4] In the multidimensional cases we use n interpolation nodes and evaluate on an evenly spaced grid of m points in every dimension.

Table 1 Interpolation of one-dimensional function

n	Linear	Linear (g)	B-spline	Fast	Fast (g)
Time expired (s)					
10	0.7030	0.8708	5.4179	2.4650	2.6377
20	0.7075	0.8704	9.0490	2.4611	2.6197
50	0.7082	0.8701	4.0806	2.4589	2.6336
100	0.7082	0.8718	4.1297	2.4748	2.6300
500	0.7061	0.8703	4.2499	2.4692	2.6239
1000	0.7086	0.8718	4.3679	2.4528	2.6230
Error					
10	0.0791	0.0127	0.0488	0.0569	0.0046
20	0.0559	0.0046	0.0327	0.0402	0.0020
50	0.0354	0.0011	0.0200	0.0255	$8.0 \cdot 10^{-4}$
100	0.0250	$6.46 \cdot 10^{-4}$	0.0140	0.0180	$4.71 \cdot 10^{-4}$
500	0.0112	$1.22 \cdot 10^{-5}$	$1.48 \cdot 10^{18}$	0.0080	$8.0 \cdot 10^{-7}$
1000	0.0079	$3.09 \cdot 10^{-6}$	$9.02 \cdot 10^{51}$	0.0057	$3.67 \cdot 10^{-10}$

$m = 1,000,000,\ g = \frac{10}{n}$

Table 2 Interpolation of two-dimensional function

n	Linear	Linear (g)	B-spline	Fast	Fast (g)
Time expired (s)					
10	0.4619	0.5496	2.0214	1.7825	1.9023
20	0.4635	0.5453	2.2075	1.8096	1.8923
50	0.4626	0.5464	2.6144	1.8104	1.8924
100	0.4638	0.5447	3.3266	1.8247	1.9127
Error					
10	0.3447	0.0161	0.2597	0.2839	0.0048
20	0.2793	0.0059	0.2039	0.2311	$5.4557 \cdot 10^{-5}$
50	0.2113	0.0012	0.1498	0.1767	$2.54 \cdot 10^{-6}$
100	0.1729	$3.47 \cdot 10^{-4}$	0.0956	0.1334	$1.20 \cdot 10^{-7}$

$m = 500,\ g = \frac{10}{n}$

"Linear" and "Linear (g)" show results for a linear interpolation without and with re-scaling. The column "B-spline" gives error and expired time for interpolation with B-splines from the CompEcon toolbox. "Fast" and "Fast (g)" present the results from the algorithm discussed above without and with re-scaling.

The results in these tables are not very surprising. As one would expect, linear interpolation is the worst in terms of accuracy and the best in terms of speed. As B-splines and the splines obtained from our fast algorithm are both cubic splines, they give similar results in terms of accuracy. However, especially in the one-dimensional

Table 3 Interpolation of three-dimensional function

n	Linear	Linear (g)	B-spline	Fast	Fast (g)
Time expired (s)					
10	0.5070	0.5675	2.7033	2.6205	2.6746
20	0.5102	0.5727	5.9913	2.6470	2.7107
Error					
10	0.7252	0.0177	0.4754	0.5665	$4.57 \cdot 10^{-4}$
20	0.5116	0.0064	0.1845	0.3192	$5.52 \cdot 10^{-5}$

$m = 50,\ g = \frac{10}{n}$

case, B-splines do slightly better than the splines computed from the fast algorithm. The big error in the column B-splines at $n = 500$ and $n = 1,000$ reflects that the B-spline interpolation matrix is not very well conditioned for a high number of interpolation points. Both the time needed for the fast algorithm as well as for linear interpolation are nearly independent of the number of grid points. For linear interpolation this issue should be clear. For the fast spline algorithm this is mainly due to the definition in (2.6), i.e. the computation of the spline function only needs to evaluate the basis function at a maximum of four points, independent of the number of grid points. Finally, we can see that re-scaling reduces the speed of the algorithm. Of course, this is due to the computation of $z = \text{gr}^{-1}(x)$ (see Sect. 2.3). However, re-scaling provides a much higher accuracy.

4.3 Agent's Optimization

This section presents results from an agent's optimization problem. The agent lives for $J = 3$ periods, while he retires at age $j_R = 3$, i.e. he will not work anymore and receive a pension.

The agent maximizes

$$V_j(a_j, ep_j) = \max_{c_j, \ell_j} \left\{ u(c_j, \ell_j)^{1-\frac{1}{\gamma}} + \beta V_{j+1}(a_{j+1}, ep_{j+1})^{1-\frac{1}{\gamma}} \right\}^{\frac{1}{1-\frac{1}{\gamma}}},$$

where c_j and $\ell_j \in [0, 1]$ denote consumption and leisure at age j, β is the time discount factor, γ is the intertemporal elasticity of substitution between consumption and leisure in different ages and $V_{J+1} = 0$. Agent's instantaneous utility function is given by

$$u(c_j, \ell_j) = \left\{ c_j^{1-\frac{1}{\rho}} + \alpha \cdot \ell_j^{1-\frac{1}{\rho}} \right\}^{\frac{1}{1-\frac{1}{\rho}}},$$

where ρ denotes the elasticity of substitution between consumption and leisure and the leisure preference parameter α is assumed to be age independent.[5] The budget

[5] The utility function used is a transformation of the standard CES utility function. This transformation guarantees that the value function V_j has a lower bound of 0.

constraint is defined as follows:

$$a_{j+1} = a_j(1+r) + w_j(1-\tau_j) + p_j - c_j,$$

with $a_1 = a_{J+1} = 0, a_j \geq 0$ for all j. During employment, agent earns labor income of $w_j = (1-\ell_j) \cdot w \cdot e_j$, where e_j denotes his efficiency in age j. After retirement, the agent will choose $\ell_j = 1$ but receive a pension p_j which depends on the earning points he accumulated during the working periods. The pension is given by $p_j = ep_{j_R} \cdot \text{APA}$, where APA is the exogenously given actual pension amount. In order to receive a pension, one has to pay some pension contribution $\tau_j w_j$ depending on labor income. During the working period, agent can accumulate earning points due to the following constraint:

$$ep_j = ep_{j-1} + \min\left[\frac{w_j}{\bar{w}}\ ,\ 2.0\right],$$

where average income $\bar{w}$ is exogenously given. Therefore, the amount of accumulated earning points in one year is limited to 2.0.

The agent's optimization problem is solved backwards by a discretization of the two-dimensional household state space $A = \{a^1, \ldots, a^n\}$, $EP = \{ep^1, \ldots, ep^n\}$. We optimize with the Nelder–Mead simplex method, see Press et al. (2002, 408ff.). For the interpolation of the future value function we use our three different approaches, linear interpolation, B-spline and the fast spline algorithm. The results are shown in Tables 4–6.

n denotes the number of grid points used in both dimensions, i.e. in total there are n^2 interpolation nodes. The total computation time in seconds needed for the solution of the model can be seen from the column "Time". In order to give some error measure for the different approaches, we calculate a high accuracy solution of the model, i.e.

Table 4 Linear interpolation

n	A	C	L	Time	n	A	C	L	Time
5	0.04436	0.04642	0.00232	1.46971	5	0.01592	0.01844	0.01125	1.32653
8	0.01454	0.01322	0.00613	3.57532	8	0.00573	0.00902	0.00054	3.07855
10	0.00689	0.00804	0.00478	5.38342	10	0.01369	0.00935	0.00592	4.62272
20	0.00681	0.00766	0.00176	20.64993	20	0.00255	0.00150	0.00130	15.96909

On the left with equidistant nodes and on the right with re-scaling, $g = 0.5$ in asset direction

Table 5 B-spline interpolation

n	A	C	L	Time
5	0.00767	0.00801	0.00056	2.45424
8	0.00065	0.00102	0.00111	6.09872
10	0.00043	0.00059	0.00051	9.54312
20	0.00008	0.00010	0.00008	35.71164

Table 6 Fast spline interpolation

n	A	C	L	Time	n	A	C	L	Time
5	0.00885	0.00916	0.00051	1.61290	5	0.00302	0.01166	0.00468	1.51354
8	0.00157	0.00182	0.00131	3.83078	8	0.00099	0.00275	0.00101	3.50598
10	0.00018	0.00075	0.00094	6.03053	10	0.00058	0.00133	0.00025	5.32241
20	0.00063	0.00070	0.00011	22.83018	20	0.00006	0.00037	0.00006	19.82591

On the left with equidistant nodes and on the right with re-scaling, $g = 0.5$ in asset direction

a high number of data points combined with a high accuracy level in the optimization step. We consider this solution as "perfect solution". The errors reported in the columns A, C and L are the maximum absolute differences between the values computed from the model and the "perfect solution" in household's assets, consumption and leisure over the different ages, respectively.

The results in these tables are very similar to the ones in the test above. Splines are better in terms of accuracy but worse in terms of time. However, in this case, the fast algorithm is not much slower than the linear interpolation approach. This may be due to the fact that we use an optimization algorithm, where the different interpolating functions don't necessarily have to be called for the same number of times.

Finally, the B-spline approximation and the fast spline algorithm give similar solutions in terms of accuracy.

4.4 A Multidimensional OLG Model

As a last step, we will take a look at the use of spline interpolation in a multidimensional OLG general equilibrium model. Household's maximization problem in this model is very similar to the one stated in the previous section. We extend the problem of Sect. 4.3 by idiosyncratic wage as well as lifetime uncertainty and set $J = 16$, $j_R = 9$. In order to isolate risk aversion from intertemporal substitution, we follow the approach of Epstein and Zin (1991) and formulate the maximization problem of a representative consumer at age j and state z recursively as

$$V_j(z) = \max_{c_j, \ell_j} \left\{ u(c_j, \ell_j)^{1-\frac{1}{\gamma}} + \beta s_{j+1} \left[\sum_{e_{j+1}} \pi(e_{j+1}|e_j) V_{j+1}(z')^{1-\eta} \right]^{\frac{1-\frac{1}{\gamma}}{1-\eta}} \right\}^{\frac{1}{1-\frac{1}{\gamma}}}$$

with parameters, variables and utility function $u(\cdot)$ like in the previous section and the degree of relative risk aversion η. s_{j+1} denotes the survival probability for a household from period j to period $j + 1$. Productivity e_j at each age j is uncertain and depends on the productivity in the previous period. Consequently, $\pi(e_{j+1}|e_j)$

denotes the probability to experience productivity e_{j+1} in the next period if the current productivity is e_j. The budget constraint is extended to

$$a_{j+1} = a_j(1+r) + w_j(1-\tau_j) + p_j - T(y_j) - (1+\tau^c)c_j + b_j$$

with a progressive income tax function $T(\cdot)$, which depends on the total taxable income y_j of the household, a consumption tax rate τ^c and unintended bequests b_j, which are left by households that don't survive to the next period, as we abstract from annuity markets. The pension system is the same as in the previous section. However, the pension contribution τ_j, the actual pension amount APA and the average earnings $\bar{w}$ are now determined endogenously.

In order to compute a general equilibrium, we add a perfectly competitive production sector populated by a large number of firms, the sum of which we normalize to unity. Aggregate output Y is produced using a Cobb–Douglas production function. The government in each period issues new debt and collects income and consumption taxes in order to finance general government expenditure G as well as the interest payments on its debt. The pension system adjusts the APA and the pension contribution rate τ_j in order to yield a standard pension (i.e. where $ep_j = j_R - 1$) which amounts to 60% of average earnings and to balance the pension system's budget in every period. For further information on the model, its calibration and the results we refer to Fehr and Habermann (2005).

In the following table, we present results from both a linear interpolation approach and the fast spline method with re-scaling. We compute the household's decision as in the previous section but with a line search optimization tool, see Press et al. (2002, 412ff.). In the first column we report the number of grid points used in every dimension of the state space. The column "Time" reports the time in seconds needed to compute a complete steady state. For the last three columns we approach similar to the previous section. For comparison we compute a high accuracy case ("perfect solution"), where the steady state is calculated with splines which use 100 points in every direction. In Table 7 we report the absolute value of the relative difference between this "perfect solution" and the other solutions (in per thousand of the "perfect solution" value) in aggregate capital, aggregate labor supply and household's ex-ante utility at age 1, respectively.

As we would expect, linear interpolation even with re-scaling provides a much lower accuracy than spline interpolation. Furthermore, the fast spline interpolation

Table 7 An OLG simulation model

	Linear				Splines			
n	Time	Δ K	Δ L	Δ U	Time	Δ K	Δ L	Δ U
20	28	6.748	1.063	1.204	43	2.970	0.207	0.091
40	111	0.894	0.233	0.283	128	0.490	0.000	0.009
80	458	0.577	0.078	0.155	703	0.000	0.000	0.009
100	721	0.570	0.068	0.128	1117	0.000	0.000	0.000

algorithm is not that much slower than a linear interpolation approach, especially if we consider the high accuracy of the fast spline interpolation method with re-scaling.

5 Conclusion

The present paper presents an algorithm to cubic spline interpolation as an alternative to linear interpolation and the standard B-spline approach. As well as the B-spline method, the algorithm is based on choosing a suitable base of the function space of cubic splines. As it doesn't require any linear equation solving method, the algorithm given in Sects. 2 and 3 is easy to compute in any programming language. The implementation effort is manageable even without much previous knowledge on function approximation. Having implemented one-dimensional interpolation, a generalization to n dimensions is straightforward. In addition, with re-scaling, we can easily manage the problem of low accuracy in steep parts of utility functions. However, the algorithm provided is more restrictive than a linear interpolation or a standard B-spline approach. It is based on the precondition that the nodes providing the interpolation data are equally spaced or follow a strictly monotone, continuous one-to-one mapping from $[0, n]$ to the interpolation interval. Hence, it is not possible to interpolate any data from arbitrarily chosen points. If one faces the problem to interpolate such data, linear interpolation or B-splines would be a reasonable instrument. Nevertheless, if the given precondition is satisfied, the algorithm presented in this paper gives a fast solution to cubic spline interpolation which can easily be applied in many approximation problems in computational economics.

Acknowledgements We would like to thank an anonymous referee for very helpful comments on an earlier version of the this paper.

References

Behforooz, G. H., & Papamichael, N. (1979). End conditions for cubic spline interpolation. *IMA Journal of Applied Mathematics, 23*, 355–366.

de Boor, C. (1978). *A practical guide to Splines*. New York: Springer.

Epstein, L. G., & Zin, S. E. (1991). Substitution, risk aversion, and the temporal behavior of consumption and asset returns: An empirical analysis. *Journal of Political Economy, 99*, 263–286.

Fehr, H., & Habermann, Ch. (2005). *Risk sharing and efficiency implications of progressive pension arrangements*. CESifo Working Paper No. 1568, München.

Heer, B., & Maussner, A. (2005). *Dynamic general equilibrium modelling*. Berlin: Springer.

İmrohoroğlu, A., İmrohoroğlu, S., & Joines, D. H. (1995). A life cycle analysis of social security. *Economic Theory, 6 (1)*, 83–114.

Judd, K. L. (1998). *Numerical methods in economics*. Massachusetts: The MIT Press.

Miranda, M. J., & Fackler, P. L. (2002). *Applied computational economics and finance*. Massachusetts: The MIT Press.

Nishiyama, S., & Smetters, K. (2005). Consumption taxes and economic efficiency with idiosyncratic wage shocks. *Journal of Political Economy, 113*, 1088–1115.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2002). *Numerical recipes in C*. Cambridge: Cambridge University Press.