Two hierarchies of spline interpolations. Practical algorithms for multivariate higher order splines.

Cristian Constantin Lalescu*†

May 21, 2009

Abstract

A systematic construction of higher order splines using two hierarchies of polynomials is presented. Explicit instructions on how to implement one of these hierarchies are given. The results are limited to interpolations on regular, rectangular grids, but an approach to other types of grids is also discussed.

Contents

*Statistical and Plasma Physics, Université Libre de Bruxelles, Campus Plaine, CP 231, B-1050 Brussels, Belgium
†*Electronic address:* clalescu@ulb.ac.be

1 Introduction

The purpose of this work is to construct smooth interpolants for functions that are only known on the nodes of a regular rectangular grid. Cubic splines have recently been used in the context of particle or virtual particle tracking in complex fields, [1, 2, 3]. A more elaborate discussion on integrating particle trajectories in interpolated fields, and the advantages of using higher order splines, is to be published elsewere[1]. In this work just a review of the construction of the spline interpolations is presented, without rigurous proofs.

Consider the functions of D variables

$$f : \mathbb{R}^D \to \mathbb{R} \tag{1}$$

and assume they can only be computed on the grid

$$G = \prod_{j=1}^{D} h_j \mathbb{Z}, \tag{2}$$

where $0 < h_j < 1$ are the grid constants and $h\mathbb{Z} = \{hz \big| z \in \mathbb{Z}\}$; a grid cell is defined as:

$$C_{(z_1,z_2,\ldots,z_D)} \equiv \prod_{j=1}^{D} [h_j z_j, h_j(z_j + 1)] \tag{3}$$

Basically, the values $f(x)$ are only available if $x \in G$ — this can happen for a variety of reasons, the simplest being that they are measured from an experiment. In the following we will say that we are computing approximations of f. In a more rigurous setting we would say that we are building an array of spline functions that converge to a certain limit under certain conditions; when the function f is sufficiently nice (a term that still needs a clear definition), it will be equal to that limit. For instance, one of the properties of this limit is that any of its Taylor expansions converge everywhere (for example any function with a finite discrete Fourier representation).

A polynomial spline is a function defined on $\mathbb{R}^D$, that is a polynomial on each cell and it has continuous derivatives *everywhere* up to a certain order. Note that the set of grid constants $\{h_j\}$ is a given, and the limit spoken of before is the limit of very large orders of the polynomials entering the spline. This limit will always exist as long as f is well defined on the grid G; it is true that for certain functions the limits for different grids will not be equal, but these functions are irrelevant here.

[1] C.C. Lalescu, B. Teaca, and D. Carati, "Implementation of high order spline interpolations for tracking test particles in discretized fields", submitted for publication.

2 One dimensional case

2.1 Hermite splines

Assume that the correct values of a function $f(x)$ are known on $h\mathbb{Z}$, and we are interested in finding an approximation for the interval $[x_0, x_0 + h]$ (the cell $C_{(x_0)}$). Without loss of generality, we express the variable x in h units, and the formulas will be deduced for the interval $[0, 1]$ (the cell $C_{(0)}$).

On $[0, 1]$ we construct the n-th order polynomial

$$s^{(n)}(x) = \sum_{k=0}^{n} a_k^{(n)} x^k \tag{4}$$

For Hermite spline interpolation, it is imposed that, on the enclosing grid nodes $s^{(n)}$ coincides with the original function and the derivatives of $s^{(n)}$ up the order $m \equiv (n-1)/2$ coincide with the derivatives of the original function:

$$\left[\frac{d^l s^{(n)}}{dx^l}(x) = f^{(l)}(x)\right]_{x\in\{0,1\}}, \; l = \overline{0, m} \tag{5}$$

where we called the l-th order derivative $f^{(l)} \equiv \frac{d^l}{dx^l} f$.

By solving the linear system of equations (5) the coefficients $a_k^{(n)}$ depending on $f(0), f(1), f'(0) \ldots$ can be easily found:

$$a_k^{(n)} = \sum_{l=0}^{m} \sum_{i=0}^{1} b_{kli}^{(n)} f^{(l)}(i)\,. \tag{6}$$

Here we will discuss a method that avoids the computation of these coefficients. This leads to rewriting the expression of the spline as:

$$\begin{aligned} s^{(n)}(x) &= \sum_{k=0}^{n} \left(\sum_{l=0}^{m} \sum_{i=0}^{1} b_{kli}^{(n)} f^{(l)}(i) \right) x^k \\ &= \sum_{l=0}^{m} \sum_{i=0}^{1} f^{(l)}(i) \left(\sum_{k=0}^{n} b_{kli}^{(n)} x^k \right) \\ s^{(n)}(x) &= \sum_{l=0}^{m} \sum_{i=0,1} f^{(l)}(i) \alpha_i^{(n,l)}(x) \end{aligned} \tag{7}$$

where the α polynomials can be found for specific values of n by symbolic computation; they can be defined equivalently as the solutions of the following system of equations:

$$\frac{d^l}{dx^l} \alpha_i^{(n,l_0)}(x) = \delta_{i,j}\delta_{l_0,l}\Big|_{j,i\in\{0,1\},x=j}, \; l = \overline{0, m} \tag{8}$$

The set $\{s^{(n)}\}$ is a hierarchy of spline approximations (Hermite splines), and it is based on the *spline polynomials of the first kind* $\alpha_i^{(n,l)}$. These polynomials

have the following explicit expressions:

$$\alpha_0^{(n,l)}(x) = \frac{x^l}{l!}(1-x)^{m+1}\sum_{k=0}^{m-l}\frac{(m+k)!}{m!k!}x^k \tag{9}$$

$$\alpha_1^{(n,l)}(x) = \frac{(x-1)^l}{l!}x^{m+1}\sum_{k=0}^{m-l}\frac{(m+k)!}{m!k!}(1-x)^k \tag{10}$$

(with $\alpha_1^{(n,l)}(x) = (-1)^l\alpha_0^{(n,l)}(1-x)$); see appendix B for a proof.

The distance $||s^{(n+1)} - s^{(n)}||$ will obvioulsy go to 0 for large n, because the splines coincide with the Taylor approximation around the two nodes up to the order m; note that this distance can be defined as the maximum of the distance $|s^{(n+1)}(x) - s^{(n)}(x)|$. And there is a class of functions f that are limits of such approximations (all piecewise polynomials for instance).

2.2 Grid splines

If centered differences are used to compute the derivatives, the formula can be further adapted. The result is called here a *grid spline* (it should be a type of b-spline), as it is found solely from the values of the function on the grid. Centered differences are used because generally for $D > 1$ dimensions — when discussing practical implementations of higher order splines — the memory cost of keeping all the derivatives necessary is prohibitive (they are as many as the coefficients $a_k^{(n)}$). A finite difference (for this rescaled variable) is just a linear combination of the numbers $f(z)$ with $z \in \mathbb{Z}$. Also, it is crucial that the use of *centered* differences imposes the continuity of the derivatives when the polynomials are put together.

To be more specific, rewrite the Hermite spline as

$$s^{(n)}(x) = \sum_{l=0}^{m}\sum_{i=0,1} t_i^{(l)}\alpha_i^{(n,l)}(x). \tag{11}$$

The smoothness of the interpolant is only determined by the fact that the coefficients of the Taylor expansions used, $t_i^{(l)}$, are determined by the grid node i (they are the same whether the node is approached from the left or from the right, or more simply they are invariant to the cell). In practice the centered differences are the coefficients of the Lagrange interpolation polynomial $\sum_{k=0}^{2g} t_{(i,g)}^{(k)}x^k$, with the coefficients determined from the equations

$$\left[\sum_{k=0}^{2g} t_{(i,g)}^{(k)}x^k = f(i+x)\right]_{x=\overline{-g,g}}. \tag{12}$$

Because the information contained in this Taylor expansion must be invariant to the cell, it must be obtained from a set of grid nodes that is symmetrical to

the current grid node i (thus the equations are solved at $x = \overline{-g, g}$, so a unique solution is obtained for a polynomial of order $2g$).

It is then obvious that a centered difference $f^{<q,l>} \approx f^{(l)}$ can be written as:

$$f^{<q,l>}(i) \equiv t^{(l)}_{(i,g)} = \sum_{k=-g}^{g} c^{(q,l)}_k f(i+k) \tag{13}$$

where the coefficients $c^{(q,l)}_k$ are numbers that only depend on g. This means that the final formula of the spline will use the values of the function on the $q = 2g + 2$ nodes $-g, \ldots, 0, 1, \ldots, g + 1$:

$$s^{(n,q)}(x) = \sum_{i=0-g}^{1+g} f(i)\beta^{(n,q)}_i(x) \tag{14}$$

where the *spline polynomials of the second kind* $\beta^{(n,q)}_i(x)$ can be found for a fixed n and q by symbolic computation.

We give the $\beta^{(5,4)}_i$-s as an example:

$$\beta^{(5,4)}_{-1}(x) = \frac{1}{2}(x-1)^3\, x\, (2\,x+1) \tag{15}$$

$$\beta^{(5,4)}_{0}(x) = -\frac{1}{2}\,(x-1)\, \left(6\,x^4 - 9\,x^3 + 2\,x + 2\right) \tag{16}$$

$$\beta^{(5,4)}_{1}(x) = \frac{1}{2}x\, \left(6\,x^4 - 15\,x^3 + 9\,x^2 + x + 1\right) \tag{17}$$

$$\beta^{(5,4)}_{2}(x) = -\frac{1}{2}\,(x-1)\, x^3\, (2\,x-3) \tag{18}$$

In terms of accuracy, using the distances $||f^{<q,l>} - f^{(l)}||$ one should be able to find the distance $||s^{(n,q)} - s^{(n)}||$.

As a sidenote, it is quite clear that the same approach can be used for irregular grids (the Taylor expansions can still be approximated using adjacent nodes). However, some of the simplifications that can be made for regular grids will no longer be possible.

3 D dimensions

Once the spline polynomials of the first and the second kind are available, the spline interpolation formula can be directly generalized to a scalar function of D variables (expressed in h_j units, and shifted to $[0, 1]^D$):

$$s^{(n)}(x_1, x_2, \ldots, x_D) = \sum_{l_1,\ldots,l_D=0}^{m} \sum_{i_1,\ldots,i_D=0,1} f^{(l_1,\ldots,l_D)}(i_1, \ldots, i_D) \prod_{k=1}^{D} \alpha^{(n,l_k)}_{i_k}(x_k) \tag{19}$$

$$s^{(n,q)}(x_1, x_2, \ldots, x_D) = \sum_{i_1,\ldots,i_D=0-g}^{1+g} \left(f(i_1, \ldots, i_D) \prod_{j=1}^{D} \beta^{(n,q)}_{i_j}(x_j)\right) \tag{20}$$

A rather interesting observation is that for the Hermite splines exactly $2^D(m+1)^D$ input values $f^{(l_1,\ldots,l_D)}(i_1,\ldots,i_D)$ are required for a given $n=2m+1$, while for grid splines q^D input values are required for a given q (and $m \leq 2g$, thus $n \leq 2q-3$). This means that grid splines generally achieve a given degree of smoothness from less information than a Hermite spline — and the price is probably a much larger error. Also, the number of input values is the number of terms in the sum to be computed, thus grid splines will be faster to compute than the corresponding Hermite splines (up to a maximum q that will depend on the order). As a final note, the mixed derivatives of these fields are also smooth:

$$\left(\prod_{j=1}^{N}\left(\frac{\partial}{\partial x_j}\right)^{l_j}\right)s^{(n)},\left(\prod_{j=1}^{N}\left(\frac{\partial}{\partial x_j}\right)^{l_j}\right)s^{(n,q)} \in \mathcal{C}^{m-\max\{l_j\}} \tag{21}$$

(whenever $m \geq \max\{l_j\}$).

4 Implementation of the grid splines

In practice the Hermite splines will probably not be very useful, as they require too much memory. Other than that, their implementation should be similar to that of the grid splines. Note that we mention "parallelized codes" in the following; this refers specifically to cases of computer programs that run on several processors at once, with the memory divided between them, and these programs work with physical fields, each processor keeping a slice of these fields in its memory.

4.1 Full expressions of the 1D grid splines

You will need to use a computer algebra system. For $q=4$, define the following functions:

$$t_{(0,1)}(h) = t^0_{(0,1)} + t^1_{(0,1)}h + t^2_{(0,1)}\frac{h^2}{2!} \tag{22}$$

$$t_{(1,1)}(h) = t^0_{(1,1)} + t^1_{(1,1)}h + t^1_{(1,1)}\frac{h^2}{2!} \tag{23}$$

$$s^{(3,4)}(x) = s^{(3,4)}_0 + s^{(3,4)}_1 x + s^{(3,4)}_2 x^2 + s^{(3,4)}_3 x^3 \tag{24}$$

$$s^{(5,4)}(x) = s^{(5,4)}_0 + s^{(5,4)}_1 x + s^{(5,4)}_2 x^2 + s^{(5,4)}_3 x^3 + s^{(5,4)}_4 x^4 + s^{(5,4)}_5 x^5 \tag{25}$$

Afterwards, solve the following systems of equations (symbolically):

$$\begin{cases} t_{(0,1)}(0)=f(0), & t_{(0,1)}(-1)=f(-1), & t_{(0,1)}(1)=f(1), \\ t_{(1,1)}(0)=f(1), & t_{(1,1)}(-1)=f(0), & t_{(1,1)}(1)=f(2), \end{cases} \tag{26}$$

$$\begin{cases} s^{(3,4)}(0) = t^0_{(0,1)}, & s^{(3,4)}(1) = t^0_{(1,1)}, \\ \left[\dfrac{d}{dx}\quad s^{(3,4)}(x)\right]_{x=0} = t^1_{(0,1)}, & \left[\dfrac{d}{dx}\quad s^{(3,4)}(x)\right]_{x=1} = t^1_{(1,1)}, \end{cases} \tag{27}$$

$$\begin{cases} s^{(5,4)}(0) = t^0_{(0,1)}, & s^{(5,4)}(1) = t^0_{(1,1)}, \\ \left[\dfrac{d}{dx} \quad s^{(5,4)}(x)\right]_{x=0} = t^1_{(0,1)}, & \left[\dfrac{d}{dx} \quad s^{(5,4)}(x)\right]_{x=1} = t^1_{(1,1)}, \\ \left[\dfrac{d^2}{dx^2} \quad s^{(5,4)}(x)\right]_{x=0} = t^2_{(0,1)}, & \left[\dfrac{d^2}{dx^2} \quad s^{(5,4)}(x)\right]_{x=1} = t^2_{(1,1)}, \end{cases} \tag{28}$$

You can either solve them one at a time (and afterwards replace the solution of (26) into the expressions of the splines), or you can solve them all at once. For larger numbers of grid points q, the process is identical.

4.2 Spline polynomials

Note that the same systems of equations need to be solved for the Hermite splines, just that the centered differences $t^q_{(i,l)}$ should be replaced with the $f^{(l)}(i)$, and the spline polynomials of the first kind can be found as:

$$\alpha_i^{(n,l)}(x) \equiv \frac{d}{d(f^{(l)}(i))} s^{(n)}(x) \tag{29}$$

For the spline polynomials of the second kind, the values of the function come into play (and the following expression makes sense after the centered differences have been introduced into the expressions of the splines):

$$\beta_i^{(n,q)}(x) \equiv \frac{d}{d(f(i))} s^{(n,q)}(x) \tag{30}$$

These two equations are the simplest way to find the polynomials. Considering that the full expressions of the splines are linear in the $t^l_{(i,g)}$-s which are linear in the $f(i)$-s, they are equivalent to rearanging the terms in the sum and extracting the "coefficients" of the $f(i)$-s, as in the definitions.

4.3 Implementation

After finding the spline polynomials of the second kind, they should be put into their Horner forms (for fast computation). The following steps depend on the programmer's taste and abilities mostly, but we recommend the implementation of the subroutines that follow. They are easy to adapt for the case of a parallelized code, and this implementation proved to be quite efficient and easy to work with (easy to expand for more cases, explain to other users, check for errors when necessary). Note that the same structure can be used for interpolating derivatives of the field if necessary, just that subroutines for computing the derivatives of the spline polynomials have to be added.

Recommended structure of the interpolation code (subroutines):

1. spline polynomials (n, q)

- input: fraction ξ
- output: $\gamma_i = \beta_i^{(n,q)}(\xi)$ (array of dimension q)

2. grid coordinates

   - input: "normal" point coordinates $(x_1, \ldots, x_D)$
   - algorithm: compute each $\hat{x}_j \equiv \lfloor x_j/h_j \rfloor$ and each $\tilde{x}_1 \equiv x_j/h_j - \hat{x}_j$.
   - output:
     - the set of integers $(\hat{x}_1, \ldots, \hat{x}_D)$
     - the set of fractions $(\tilde{x}_1, \ldots, \tilde{x}_D)$

3. spline formula

   - input:
     - the set of fractions $(\tilde{x}_1, \ldots, \tilde{x}_D)$
     - the type of spline (n, q)
     - a pointer (or similar notion) $\tilde{f}$ to an array containing the information about the local field (the values of the field on the nodes of the cell $C_{(\hat{x}_1,\ldots,\hat{x}_D)}$ and the necessary neighbouring cells), shifted such that $\tilde{f}(0, 0, \ldots, 0) = f(x_1, \ldots, x_D)$.
   - algorithm: compute the polynomials $\beta_i^{(n,q)}(\tilde{x}_j)$ in the array γ_{ij} (by calling the spline polynomials subroutine), then compute the sum

$$\hat{f} = \sum_{i_1,\ldots,i_D=0-g}^{1+g} \Big(\tilde{f}(i_1, \ldots, i_D) \prod_{j=1}^{D} \gamma_{ij} \Big); \tag{31}$$

     note that testing shows it is more efficient to introduce as little `do while` loops as possible — for our 3D implementation, for $q = 4$ the sum is written in full in the source code, as introducing `do while` type loops slows down the code considerably. For higher values of q we just have one `do while` loop for one of the variables.
   - output: the approximation $\hat{f}$.

4. wrapper

   - input:
     - "normal" point coordinates $(x_1, \ldots, x_D)$
     - a pointer f to the array containing the information about the entire field
     - the type of spline (n, q)
   - algorithm: put the local field in the array $\tilde{f}$ from f and then compute the approximation $\hat{f}$ (using the above subroutines)
   - output: the approximation $\hat{f}$

The wrapper is very useful in the case of a parallelized code. All the operations related to bringing together information spread on possibly several processors can be placed inside the wrapper, allowing for easy debugging and maintenance of the code.

As an example of a parallelized version, in our implementation a 3D field is divided along the z coordinate between processors. The field is periodic in all directions, and obtaining $\tilde{f}$ implies a little care in regards to the z coordinate, but it basically just requires a normal use of the `MODULO` operator. We impose that each processor has at least q nodes on the z direction in its memory, so that the formula contains at most information from two processors ("low" and "up"). We then compute (31) on each processor, only for the nodes that are in "its domain", and we then sum the two resulting values $\hat{f} = \hat{f}_{low} + \hat{f}_{up}$; the amount of information passed between processors is thus kept to a minimum.

A Appendix: Uniqueness

Construction for general case:

$$t(\mathbf{i}, g)(\mathbf{u}) = \sum_{k_1,\ldots,k_D=0}^{2g} t^{\mathbf{k}}_{(\mathbf{i},g)} \prod_{j=1}^{D} u_j^{k_j} \tag{32}$$

$$s^{(n,q)}(\mathbf{x}) = \sum_{k_1,\ldots,k_D=0}^{n} s^{(n,q)}_{\mathbf{k}} \prod_{j=1}^{D} x_j^{k_j} \tag{33}$$

with $i_1, \ldots, i_D \in \{0, 1\}$.

The Taylor expansions (the centered differences) are given by the system of equations

$$t_{(\mathbf{i},g)}(\mathbf{v}) = f(\mathbf{i} + \mathbf{v}) \tag{34}$$

where $v_1, \ldots, v_D = \overline{-g, g}$. This system of equations has a unique solution $t^{\mathbf{k}}_{(\mathbf{i},g)} \approx f^{(\mathbf{k})}$ (there are as many $t^{\mathbf{k}}_{(\mathbf{i},g)}$ as there are $f(\mathbf{i} + \mathbf{v})$, and this is in fact a simple Lagrange interpolation).

The splines are given by the system of equations

$$\left[\left(\prod_{j=1}^{D} \left(\frac{d}{dx_j} \right)^{l_j} \right) s^{(n,q)}(\mathbf{x}) \right]_{\mathbf{x}=\mathbf{i}} = t^{\mathbf{l}}_{(\mathbf{i},g)} \tag{35}$$

which also has a unique solution (and $q \equiv 2g + 2$), so formula (20) must give this unique solution.

A generalization of this method would be to go to other types of grids. For instance one could imagine the case where in three dimensions there is a grid made up of equilateral triangles in the (x, y) plane, and squares in the other two planes. What changes is the fact that the sums are a bit more complicated. The crucial property that has to be preserved to have a smooth approximation is that the Taylor expansion must be the same no matter from which cell we approach a given node.

B Appendix: Spline polynomials

For a rigurous analysis of the errors of these approximations, the distances $||f^{<q,l>} - f^{(l)}||$, $||s^{(n,q)} - s^{(n)}||$ and $||s^{(n)} - f||$ should be computed. After using the method presented above to find the α polynomials up to $n = 19$, it can be seen that all these spline polynomials of the first kind have a simple form:

$$\alpha_0^{(n,l)}(x) = \frac{x^l}{l!}(1-x)^{m+1} \sum_{k=0}^{m-l} \frac{(m+k)!}{m!k!} x^k \tag{36}$$

$$\alpha_1^{(n,l)}(x) = \frac{(x-1)^l}{l!} x^{m+1} \sum_{k=0}^{m-l} \frac{(m+k)!}{m!k!} (1-x)^k \tag{37}$$

(with $\alpha_1^{(n,l)}(x) = (-1)^l \alpha_0^{(n,l)}(1-x)$).

It would be useful to have an explicit formula for the α polynomials in the general case, as it would allow for a more straightforward treatment of the error $||s^{(n+1)} - s^{(n)}||$, which is close to $||s^{(n)} - f||$.

Obviously, if this form is proven to apply for $\alpha_0^{(n,l)}$, the form for $\alpha_1^{(n,l)}$ must also be correct.

First, notice that for x very close to 1 ($x = 1 - y$, with y small), the Taylor expansion of $\alpha_0^{(n,l)}$ begins at y^{m+1}, and this is half of the proof. To continue, note rewrite the polynomial as

$$\alpha_0^{(n,l)}(x) = \frac{1}{m!l!} \sum_{k=0}^{m-l} \sum_{j=0}^{m+1} \frac{(m+k)!}{k!} \frac{(m+1)!(-1)^j}{j!(m+1-j)!} x^{l+k+j} \tag{38}$$

For this proof the coefficient of x^{l_0} in $\alpha_0^{(n,l)}$, for $0 \leq l_0 \leq m$ is needed. Obviously, for $l_0 < l$ this coefficient is 0:

$$l + k + j = l_0 \tag{39}$$

$$k + j = l_0 - l \tag{40}$$

$$\text{but } k + j \geq 0 \tag{41}$$

$$\text{so } l_0 - l \geq 0 \tag{42}$$

For $l_0 = l$, the coefficient is given by the term with $k + j = 0$:

$$\frac{1}{l!} \frac{(m+0)!}{m!0!} \frac{(m+1)!(-1)^0}{0!(m+1-0)!} = \frac{1}{l!} \tag{43}$$

which is what is needed. For $l_0 - l = \Delta l \geq 1$ we have:

$$g(m, l, \Delta l) = \frac{1}{l!} \sum_{k=0}^{m-l} \sum_{j=0}^{m+1} \frac{(m+k)!}{m!k!} \frac{(m+1)!(-1)^j}{j!(m+1-j)!} \delta_{k+j,\Delta l} \tag{44}$$

(with the Kronecker δ). This formula simplifies to

$$g(m,l,\Delta l) = \frac{1}{l!}\sum_{k=0}^{\Delta l}\frac{(m+k)!}{m!k!}\frac{(m+1)!(-1)^{\Delta l-k}}{(\Delta l-k)!(m+1-\Delta l+k)!} \tag{45}$$

And in fact this sum is, from [4]:

$$g(m,l,\Delta l) = \frac{(-1)^{\Delta l}\sin(\pi\Delta l)}{l!\pi\Delta l} \tag{46}$$

which is 0 for integer values of Δl (note that $\Delta l \leq m$ for the sum to make sense).

References

[1] F. Mackay, R. Marchand, and K. Kabin. Divergence-free magnetic field interpolation and charged particle trajectory integration. *Journal of Geophysical Research*, 111:A06208, 2006.

[2] F. Lekien and J. Marsden. Tricubic interpolation in three dimensions. *Journal of Numerical Methods and Engineering*, 63:455–471, 2005.

[3] Holger Homann, Jürgen Dreher, and Rainer Grauer. Impact of the floating-point precision and interpolation scheme on the results of dns of turbulence by pseudo-spectral codes. *Computer Physics Communications*, 177:560–565, 2007.

[4] *Mathematica Version 6.0.* Wolfram Research, Inc, 2007.