# Higher Order Spline Interpolations for Tracking Particles in Discretized Fields

Cristian-Constantin Lalescu

**Abstract**

The need to interpolate a discretized field appears in the context of the numerical simulation of particles moving in a turbulent field, when this field is obtained from an independent numerical simulation of fluid equations. This paper describes the construction of a particular type of interpolation, that satisfies a set of necessary constraints.

The necessity of using a smooth approximation of the fields in which the particle is moving has been touched upon in the literature, [1, 2], but the full problem has not been treated. High order integration schemes, or solvers, should be used in the study of particle transport for several reasons, as discussed in previous work [3]. The properties of high order solvers all depend, among other requirements, on the continuity of the first few derivatives of the forcing entering the particle evolution equations, thus the smoothness of the fields that enter this forcing. The need for an interpolation method that results in smooth approximation becomes then clear, and here such a method is constructed explicitely. Additionally, an implementation is presented in some detail, and several examples are given together with a discussion of efficiency.

# Chapter 1

# Construction of the method

## 1.1 The problem of interpolation

We are interested in the problem of charged particle motion in the presence of a turbulent electromagnetic field. Simulating the evolution of such particles implies describing the turbulent fields; generally these are found from simulations of fluid equations, and they are known only on some discrete spatial grid. The numerical methods used to integrate the trajectory of a given particle will require the value of the force acting on the particle at arbitrary locations in space; thus a method of approximating this value from the values on the grid is required. This is the interpolation method (interpolation because the position is between nodes of the grid). The fields to be interpolated will be threedimensional, but the method to be described is built for the 1D case, and it is then easily generalized to any dimension.

An interpolation method resulting in an approximation to be used for tracking a particle has to satisfy a smoothness constraint. Consider a first order ordinary differential equation $dx/dt = f(x)$ and the Taylor expansion of its solution:

$$x(t) = x(0) + tf(x)|_{x(0)} + \frac{t^2}{2}f'(x)|_{x(0)} + \cdots + \frac{t^m}{n!}f^{(m-1)}(x)|_{x(0)} + \ldots \quad (1.1)$$

the expansion for any $m$th order numerical integration scheme (solver) should coincide with this Taylor expansion up to the $m$th order. However, if the order $m$ derivative of the force field is ill-defined, any solver will have at most a global order $m$.

Consider the functions of $D$ variables

$$f : \mathbb{R}^D \to \mathbb{R} \quad (1.2)$$

and assume they can only be computed on the grid

$$G = \prod_{j=1}^{D} h_j \mathbb{Z}, \tag{1.3}$$

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:

$$C_{(z_1,z_2,...,z_D)} \equiv \prod_{j=1}^{D} [h_j z_j, h_j(z_j + 1)] \tag{1.4}$$

Basically, the values $f(x)$ are only available if $x \in G$. 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, 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 (this is true for 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 $f$ and its derivatives are well defined on the grid $G$.

## 1.2 One dimensional case

### 1.2.1 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

$$s^{(n)}(x) = \sum_{k=0}^{n} a_k^{(n)} x^k \tag{1.5}$$

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:

$$\left[\frac{d^l s^{(n)}}{dx^l}(x) = f^{(l)}(x)\right]_{x\in\{0,1\}}, \; l = \overline{0, m} \tag{1.6}$$

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

By solving the linear system of equations (1.6) the coefficients $a_k^{(n)}$ depending on $f(0), f(1), f'(0) \dots$ can be easily found:

$$a_k^{(n)} = \sum_{l=0}^{m} \sum_{i=0}^{1} b_{kli}^{(n)} f^{(l)}(i) \,. \tag{1.7}$$

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

$$\begin{aligned} s^{(n)}(x) &= \sum_{k=0}^{n} \left( \sum_{l=0}^{m} \sum_{i=0}^{1} b_{kli}^{(n)} f^{(l)}(i) \right) x^k \\ &= \sum_{l=0}^{m} \sum_{i=0}^{1} f^{(l)}(i) \left( \sum_{k=0}^{n} b_{kli}^{(n)} x^k \right) \\ s^{(n)}(x) &= \sum_{l=0}^{m} \sum_{i=0,1} f^{(l)}(i) \alpha_i^{(n,l)}(x) \end{aligned} \tag{1.8}$$

where the $\alpha$ 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:

$$\tfrac{d^l}{dx^l} \alpha_i^{(n,l_0)}(x) = \delta_{i,j} \delta_{l_0,l} \Big|_{j,i \in \{0,1\}, x=j}, \; l = \overline{0,m} \tag{1.9}$$

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:

$$\alpha_0^{(n,l)}(x) = \frac{x^l}{l!} (1-x)^{m+1} \sum_{k=0}^{m-l} \frac{(m+k)!}{m!k!} x^k \tag{1.10}$$

$$\alpha_1^{(n,l)}(x) = \frac{(x-1)^l}{l!} x^{m+1} \sum_{k=0}^{m-l} \frac{(m+k)!}{m!k!} (1-x)^k \tag{1.11}$$

(with $\alpha_1^{(n,l)}(x) = (-1)^l \alpha_0^{(n,l)}(1-x)$); see appendix A 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 $|s^{(n+1)}(x) - s^{(n)}(x)|$ over the interval.

### 1.2.2 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

$$s^{(n)}(x) = \sum_{l=0}^{m} \sum_{i=0,1} t_i^{(l)} \alpha_i^{(n,l)}(x). \tag{1.12}$$

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

$$\left[ \sum_{k=0}^{2g} t_{(i,g)}^{(k)} x^k = f(i+x) \right]_{x=\overline{-g,g}}. \tag{1.13}$$

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:

$$f^{<q,l>}(i) \equiv t_{(i,g)}^{(l)} = \sum_{k=-g}^{g} c_k^{(q,l)} f(i+k) \tag{1.14}$$

where the coefficients $c_k^{(q,l)}$ 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 $-g, \ldots, 0, 1, \ldots, g+1$:

$$s^{(n,q)}(x) = \sum_{i=0-g}^{1+g} f(i) \beta_i^{(n,q)}(x) \tag{1.15}$$

where the SPLINE POLYNOMIALS OF THE SECOND KIND $\beta_i^{(n,q)}(x)$ can be found for a fixed $n$ and $q$ by symbolic computation.

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

$$\beta_{-1}^{(5,4)}(x) = \frac{1}{2}(x-1)^3\, x\ (2\,x+1) \tag{1.16}$$

$$\beta_{0}^{(5,4)}(x) = -\frac{1}{2}\,(x-1)\ \left(6\,x^4 - 9\,x^3 + 2\,x + 2\right) \tag{1.17}$$

$$\beta_{1}^{(5,4)}(x) = \frac{1}{2}x\ \left(6\,x^4 - 15\,x^3 + 9\,x^2 + x + 1\right) \tag{1.18}$$

$$\beta_{2}^{(5,4)}(x) = -\frac{1}{2}\,(x-1)\ x^3\ (2\,x-3) \tag{1.19}$$

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.

## 1.3 Tensor product splines and physical fields

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 $[0,1]^D$):

$$s^{(n)}(x_1, x_2, \ldots, x_D) = \sum_{l_1,\ldots,l_D=0}^{m} \sum_{i_1,\ldots,i_D=0,1} f^{(l_1,\ldots,l_D)}(i_1,\ldots,i_D) \prod_{k=1}^{D} \alpha_{i_k}^{(n,l_k)}(x_k) \tag{1.20}$$

$$s^{(n,q)}(x_1, x_2, \ldots, x_D) = \sum_{i_1,\ldots,i_D=0-g}^{1+g} \left( f(i_1,\ldots,i_D) \prod_{j=1}^{D} \beta_{i_j}^{(n,q)}(x_j) \right) \tag{1.21}$$

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:

$$\left(\prod_{j=1}^{N}\left(\frac{\partial}{\partial x_j}\right)^{l_j}\right) s^{(n)}, \left(\prod_{j=1}^{N}\left(\frac{\partial}{\partial x_j}\right)^{l_j}\right) s^{(n,q)} \in \mathcal{C}^{m-\max\{l_j\}} \tag{1.22}$$

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

The formal operator that transforms a field $f$ into its spline approximation $\hat{f}$ will be named $S^{(n,q)}$:

$$\hat{f} = S^{(n,q)}\, f. \tag{1.23}$$

In regards to the interpolation of a physical vector field $\mathbf{F}$, we have two choices. First, the direct interpolation

$$\hat{\mathbf{F}}^{(n,q)}(\mathbf{r}) = S^{(n,q)}(\mathbf{r})\, \mathbf{F} \tag{1.24}$$

Second, we note that any vector field can be written as deriving from a scalar and vector potential

$$\mathbf{F} = -\nabla\phi_{\mathbf{F}} + \nabla \times \mathbf{A}_{\mathbf{F}} \tag{1.25}$$

and so we can approximate the field as

$$\tilde{\mathbf{F}}^{(n,q)}(\mathbf{r}) = -\nabla(S^{(n,q)}(\mathbf{r})\, \phi_{\mathbf{F}}) + \nabla \times (S^{(n,q)}(\mathbf{r})\, \mathbf{A}_{\mathbf{F}}) \tag{1.26}$$

The second approach, suggested in [1], is most likely the best, as it will yield a divergence-free magnetic field everywhere in space, and a curl-free electric field when the fields are stationary ($\mathbf{E} = -\nabla\phi_{\mathbf{E}}$, because $\partial_t \mathbf{A} = 0$). With the method described here, it is also trivial to implement, as derivatives of the spline polynomials can be used in the tensor product.

## 1.4 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.

### Full expressions of the 1D grid splines

A computer algebra system should be used. For $q = 4$, define the following functions:

$$t_{(0,1)}(h) = t^0_{(0,1)} + t^1_{(0,1)}h + t^2_{(0,1)}\frac{h^2}{2!} \tag{1.27}$$

$$t_{(1,1)}(h) = t^0_{(1,1)} + t^1_{(1,1)}h + t^1_{(1,1)}\frac{h^2}{2!} \tag{1.28}$$

$$s^{(3,4)}(x) = s^{(3,4)}_0 + s^{(3,4)}_1 x + s^{(3,4)}_2 x^2 + s^{(3,4)}_3 x^3 \tag{1.29}$$

$$s^{(5,4)}(x) = s^{(5,4)}_0 + s^{(5,4)}_1 x + s^{(5,4)}_2 x^2 + s^{(5,4)}_3 x^3 + s^{(5,4)}_4 x^4 + s^{(5,4)}_5 x^5 \tag{1.30}$$

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

$$\begin{cases} t_{(0,1)}(0) = f(0), & t_{(0,1)}(-1) = f(-1), & t_{(0,1)}(1) = f(1), \\ t_{(1,1)}(0) = f(1), & t_{(1,1)}(-1) = f(0), & t_{(1,1)}(1) = f(2), \end{cases} \tag{1.31}$$

$$\begin{cases} s^{(3,4)}(0) = t^0_{(0,1)}, & s^{(3,4)}(1) = t^0_{(1,1)}, \\ \left[\dfrac{d}{dx} \quad s^{(3,4)}(x)\right]_{x=0} = t^1_{(0,1)}, & \left[\dfrac{d}{dx} \quad s^{(3,4)}(x)\right]_{x=1} = t^1_{(1,1)}, \end{cases} \tag{1.32}$$

$$\begin{cases} s^{(5,4)}(0) = t^0_{(0,1)}, & s^{(5,4)}(1) = t^0_{(1,1)}, \\ \left[\dfrac{d}{dx} \quad s^{(5,4)}(x)\right]_{x=0} = t^1_{(0,1)}, & \left[\dfrac{d}{dx} \quad s^{(5,4)}(x)\right]_{x=1} = t^1_{(1,1)}, \\ \left[\dfrac{d^2}{dx^2} \quad s^{(5,4)}(x)\right]_{x=0} = t^2_{(0,1)}, & \left[\dfrac{d^2}{dx^2} \quad s^{(5,4)}(x)\right]_{x=1} = t^2_{(1,1)}, \end{cases} \tag{1.33}$$

They can be solved either one at a time (and afterwards the solution of (1.31) should be replaced into the expressions of the splines), or they can all be solved at once. For larger numbers of grid points $q$, the process is identical.

### 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:

$$\alpha^{(n,l)}_i(x) \equiv \frac{d}{d(f^{(l)}(i))} s^{(n)}(x) \tag{1.34}$$

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):

$$\beta_i^{(n,q)}(x) \equiv \frac{d}{d(f(i))} s^{(n,q)}(x) \tag{1.35}$$

These two equations are the simplest way to find the polynomials. Noting 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.

**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 $\xi$
   - 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 $\gamma_{ij}$ (by calling the spline polynomials subroutine), then compute the sum

$$\hat{f} = \sum_{i_1,\ldots,i_D=0-g}^{1+g} \left( \tilde{f}(i_1, \ldots, i_D) \prod_{j=1}^{D} \gamma_{ij} \right); \tag{1.36}$$

  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 for 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.

# Chapter 2

# Examples and conclusions

We consider a set of particles with random initial conditions (but fixed initial kinetic energy). We then evolve this same set of particles in a fixed discretized electromagnetic field, using several approximations $\hat{\mathbf{F}}^{(n,q)}$ and $\tilde{\mathbf{F}}^{(n,q)}$ and two solvers, the fourth order composition scheme CM4 by Blanes and Moan [4] and the sixth order CM6 by Kahan and Li [5] — the use of CMs for charged particles is discussed in [3].

The first example is an order test for the two solvers, with different interpolation methods, see figure 2.1. Note that using an interpolation through the potentials implies using a field that has one less continuous derivative. This is one good reason to use higher order spline interpolations, as differentiating cubic splines would result in solvers having at most of second order.

One argument to using an interpolation through potentials instead of directly is given by figure 2.2. Energy conservation, when studying particle transport, is highly significant; it is also important because it is a first measure of the stability of the numerical method. The divergence-free of the magnetic field is also important, as particle transport depends on the structure of invariant sets in phase space.

Furthermore, the hierarchy of interpolations constructed in this paper allows for a direct control of the errors introduced strictly by the approximation of the field; several approximations can be used simultaneously, and the results obtained can be compared afterwards. It is likely that for most cases very smooth approximations are not crucial to obtaining results; however, the fact remains that better smoothness allows for the use of higher order solvers, and these prove to be more efficient, even when there is some performance loss due to the complexity of the interpolation formula.

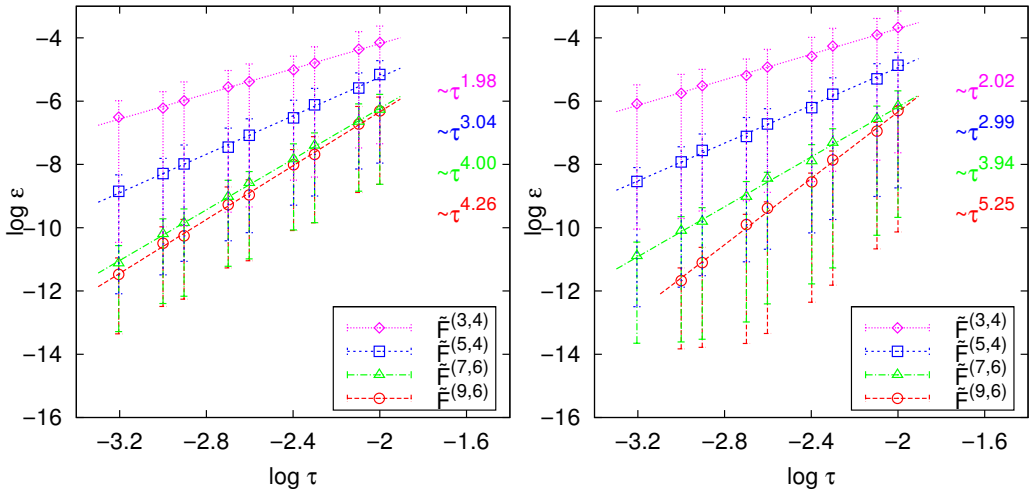


Figure 2.1: The order of the integration schemes is measured by evaluating the error $\varepsilon$ in the trajectories as a function of the time step $\tau$. These graphs show that this order is directly affected by the smoothness of the interpolation scheme $\tilde{\mathbf{F}}$. The exponent $n_{\text{eff}}$ in the power $\varepsilon(\tau) \propto \tau^{-n_{\text{eff}}}$ shows the effective order of the scheme. Left and right figures are respectively obtained with the CM4 and CM6 schemes.

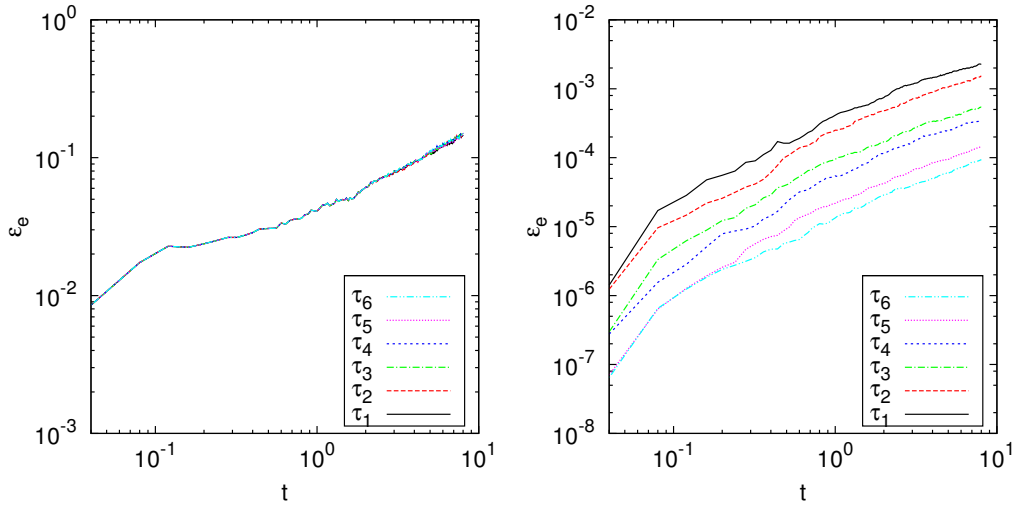


Figure 2.2: The energy conservation gives another quality indicator of the numerical solution. In frozen fields, the total kinetic plus potential energy is theoretically conserved. By computing the differences between the energy of a particle at time $t$ and its initial energy, for a trajectory obtained with a timestep $\tau$, we obtain a family of average energy errors, $\varepsilon_e(\tau, t)$. Left and right figures respectively correspond to $\hat{\mathbf{F}}^{(3,4)}$ and $\tilde{\mathbf{F}}^{(3,4)}$. Both figures are obtained with the CM4 scheme. We find that generally the energy error introduced by interpolating directly on the fields (using $\hat{\mathbf{F}}$) dominates the error related to the size of the timestep.

# Appendix A

# 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 $\alpha$ polynomials up to $n = 19$, it can be seen that all these spline polynomials of the first kind have a simple form:

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

$$\alpha_1^{(n,l)}(x) = \frac{(x-1)^l}{l!}x^{m+1}\sum_{k=0}^{m-l}\frac{(m+k)!}{m!k!}(1-x)^k \tag{A.2}$$

(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 $\alpha$ 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

$$\alpha_0^{(n,l)}(x) = \frac{1}{m!l!}\sum_{k=0}^{m-l}\sum_{j=0}^{m+1}\frac{(m+k)!}{k!}\frac{(m+1)!(-1)^j}{j!(m+1-j)!}x^{l+k+j} \tag{A.3}$$

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:

$$l + k + j = l_0 \tag{A.4}$$
$$k + j = l_0 - l \tag{A.5}$$
$$\text{but } k + j \geq 0 \tag{A.6}$$
$$\text{so } l_0 - l \geq 0 \tag{A.7}$$

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

$$\frac{1}{l!}\frac{(m+0)!}{m!0!}\frac{(m+1)!(-1)^0}{0!(m+1-0)!} = \frac{1}{l!} \tag{A.8}$$

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

$$g(m, l, \Delta l) = \frac{1}{l!}\sum_{k=0}^{m-l}\sum_{j=0}^{m+1}\frac{(m+k)!}{m!k!}\frac{(m+1)!(-1)^j}{j!(m+1-j)!}\delta_{k+j,\Delta l} \tag{A.9}$$

(with the Kronecker $\delta$). This formula simplifies to

$$g(m, l, \Delta l) = \frac{1}{l!}\sum_{k=0}^{\Delta l}\frac{(m+k)!}{m!k!}\frac{(m+1)!(-1)^{\Delta l-k}}{(\Delta l-k)!(m+1-\Delta l+k)!} \tag{A.10}$$

And in fact this sum is, from [6]:

$$g(m, l, \Delta l) = \frac{(-1)^{\Delta l}\sin(\pi\Delta l)}{l!\pi\Delta l} \tag{A.11}$$

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

# Bibliography


[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] C.C. Lalescu. Implementation of splitting-composition schemes for the numerical study of charged particles. *internal report*, 2008.

[4] S. Blanes and P.C. Moan. Practical symplectic patitioned runge-kutta and runge-kutta-nyström methods. *Journal of Computational and Applied Mathematics*, 142:313–330, 2002.

[5] William Kahan and Ren-Cang Li. Composition constants for raising the orders of unconventional schemes for ordinary differential equations. *Math. Comput.*, 66(219):1089–1099, 1997.

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