In our last meeting we discussed how to proceed after I got FLEXI successfully running. Our main goal is to compare initally identical MHD turbulence setups in FLASH and in FLEXI. One option is to transfer a fully developed turbulence stored in a FLASH snapshot file to a format FLEXI understands. The other option is using an initally smooth velocity field which leads to turbulence during simulation. Interpolation is in order. So before that the transfer/interpolation scheme from FLASH all the way through FLEXI has to be tested on sensible test cases. The focus lies on the correctnes of handling smooth setups and the resolution of shocks. Following document testifies just that, shows many plots, reveals encountered problems and puts the result into perspective.

1 Definitions and Preliminary Remarks

1.1 Grids, Cells and Elements

Since there is no precise nomenclature yet for grids, cells and elements, I will propose one for this document. The following definitions reflect the common way of implementation in actual code as well and shall be framework agnostic. This is open for debate of course.

Grid A grid consists of either cells or elements. The grid can be structured or unstructured. It contains the necessary information where to find cells/elements and what their (spatial) relationship to neighbors are.

Cell The atomic container type of a grid. They contain the actual data which can be a scalar, arrays of scalars, vectors, tensors, etc. What cells distinguish from points is that they have an expanse. Hence, one must specificy if the data is defined in the cell-center, at their corners or at their faces.

Element Elements are spatially extended objects like cells. However, they group a list of points called *nodes* on which the data is pinned on. When an element interacts with the outside world it must extract the necessary values from these nodes via polynomial interpolation.

Specifically for this document, it is important to keep in mind that cells refer to the *finite volumes (FV)* of the FLASH grid and elements to HOPR mesh scheme. Remark: No nodal information is stored in the HOPR mesh file. *FLEXI* on the other hand just stores the data as arrays of scalars refering to an element in the mesh. The exact assignment of values to the nodes is based on convention only.

1.2 Grid Spaces and Transformation

In this work, four grid spaces are of importance: *Face-centered grid (FCG)*, *body-centered grid (BCG)*, *Gauss nodal grid (GNG)* and *Gauss-Lobatto nodal grid (LNG)*. A visual representation can be found in figure 1. This figure shows a one-dimensional grid of eight cells (dashed lines) or alternatively a grid with two elements (thick lines) each consisting of four nodes which implies a polynomial order of three. Major part of the work is the transformation back and forth between these grid spaces via interpolation. Another significant aspect is the relationship of elements to cells. First one overlays both grids. Ideally, they are of the same shape and cover the same physical domain. When transforming between both grid types, cells get grouped together in numbers equal to the nodal number of the superincumbent element. Interpolation happens in each group/element independently.

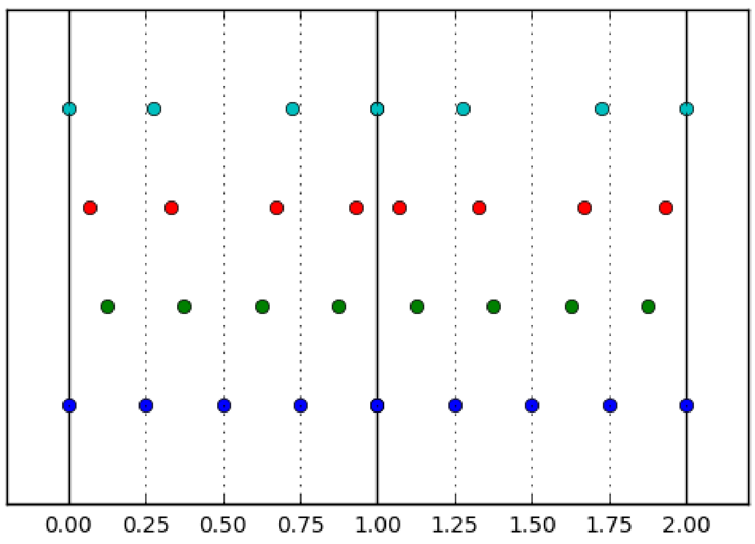

Figure 1: Nodes in a two element grid each consisting of four cells. From bottom to top: face-centered, body-centered, Gauss nodes (n = 3), Gauss-Lobatto nodes (n = 3).

1.3 1D Lagrange Interpolation

For illustrative purposes we begin with the simplest case: one-dimensional LAGRANGE interpolation.

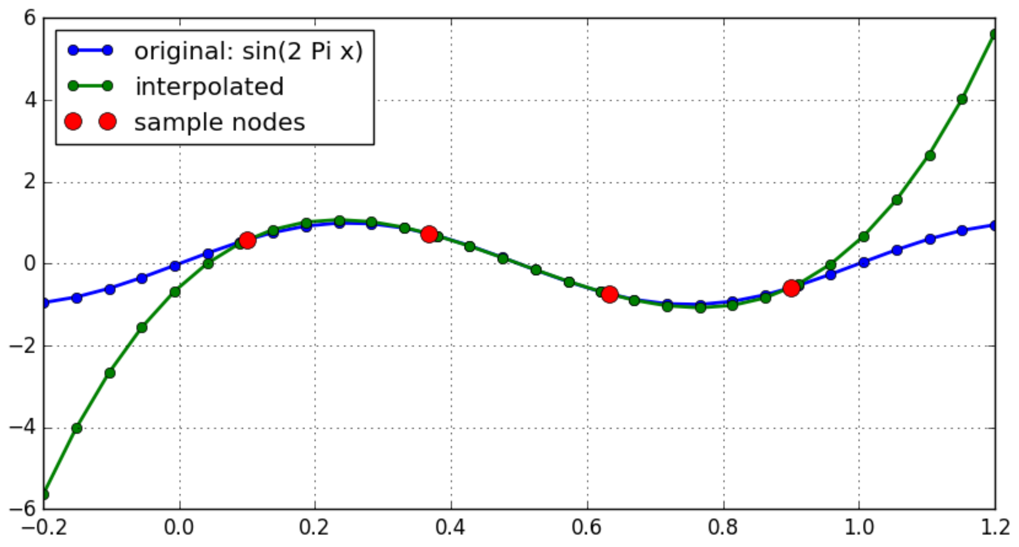

Figure 2: One-dimensinal Lagrange interpolation with four sample nodes.

The Lagrange polynome of third order needs four anchor nodes with their associated values in order to interpolate any other point in between. This works very well and does not cause any headaches. Unfortunately, considering previous section, extrapolation is necessary, too. Lagrange polynomes tend to explode going further away from the outer anchor nodes. In pathological interpolation cases this effect yields erroneous results. The end of the document shows specific examples.

1.4 Multidimensional Lagrange Interpolation

Since we operate in three-dimensional space we need to generalize the one-dimensional Lagrange interpolation to higher dimensions. One approach is called: *Multivariate Lagrange Interpolation.* Their exists extensive amout of literature about this. Interestingly, this method needs lesser anchor nodes than the following one, we use: *Tensor Ansatz Lagrange Interpolation.* The ansatz is straigtforward. For the three dimensional case it reads:

$$p(x,y,z) = \sum_{i,j,k=0}^{n_x,n_y,n_z} f_{ijk} \cdot l_i(x) \cdot l_j(y) \cdot l_k(z) \tag{1}$$

Life is easy, so we set $n = n_x = n_y = n_z$. If we go crazy we could introduce tensor notation.

$$p(x,y,z) = \mathbb{F} \odot (\underline{l(x)} \otimes \underline{l(y)} \otimes \underline{l(z)}) \tag{2}$$

where $\odot$ is the *relentless* contraction operator. If the target nodes x_q, y_r, z_s are known beforehand, $(\underline{l(x)} \otimes \underline{l(y)} \otimes \underline{l(z)})$ can be precomputed and stays the same for all interpolations within an element and it's cell group.

1.5 Transformation from FLASH to FLEXI

All following examples/figures in this document are produced by the very same procedure where we set the interpolation order $n = 3$. First the intial values are generated (or read) in the FLASH grid format. The grid is split into groups of $4^3 = 64$ neighboring cells. These groups get mapped to the associated element of the HOPR grid. Looping over all group-element pairs a three-dimensional Lagrange Polynome gets constructed according to the body-centered values of the cells. Then the new values at the Gauss/Gauss-Lobatto nodes get interpolated and stored according to the ordering convention of FLEXI.

In order to do useful analysis and visualization, the FLEXI data gets again back-interpolated to BCG. This is similar to what the visualization routines in FLEXI do.

1.6 Interpolation Error Estimate

As a meassure of interpolation error of an original function f and its interpolated counterfeit $\widetilde{f}$ I opted for the relative root-means-sqare variance. Taking the absolute error would of course be another valid option.

$$\text{rms} = \sqrt{\frac{1}{N}\sum_{i}^{N} f_i^2} \tag{3}$$

$$\text{rmse} = \sqrt{\frac{1}{N}\sum_{i}^{N} (f_i - \widetilde{f}_i)^2} \tag{4}$$

$$\text{relative rmse} = \frac{\text{rmse}}{\text{rms}} \tag{5}$$

One has to ensure that f_i and $\widetilde{f}_i$ live in the same grid space.

2 Interpolation Test Cases

This section displays a variety of test cases in order to confirm the correctness of the implementation. The two-dimensional interpolation is implemented in the same manner as the three-dimensinal. As expected functions of sufficient low polynomial order get interpolated exactly.

2.1 2D Interpolation in one element

Here we see the interpolation of two-dimensional functions within an element consisting of four nodes as usual. If not specified otherwise, the anchor points are four-times-four BCG nodes.

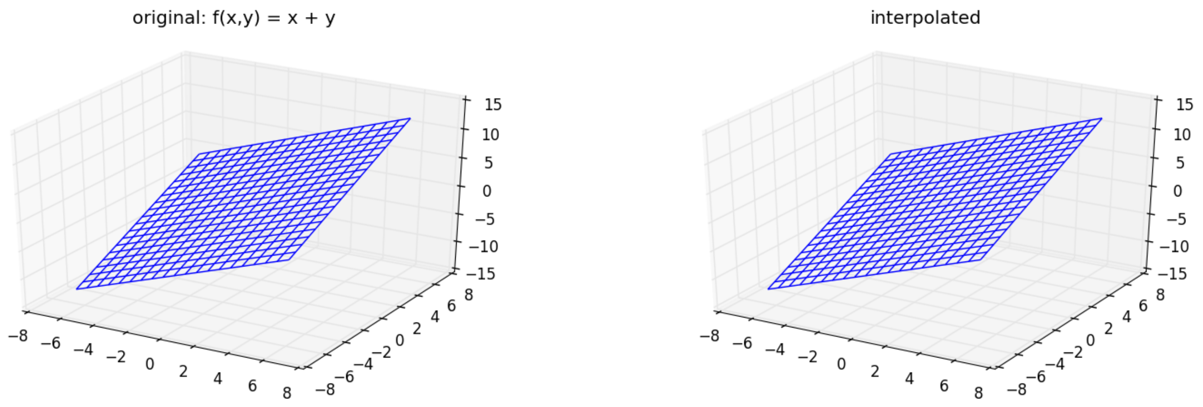

Figure 3: Third-order 2d interpolation of the ascending plane.
rms = 5.393455 | rms error = 0.000000 | relative rms error = 0.000000%

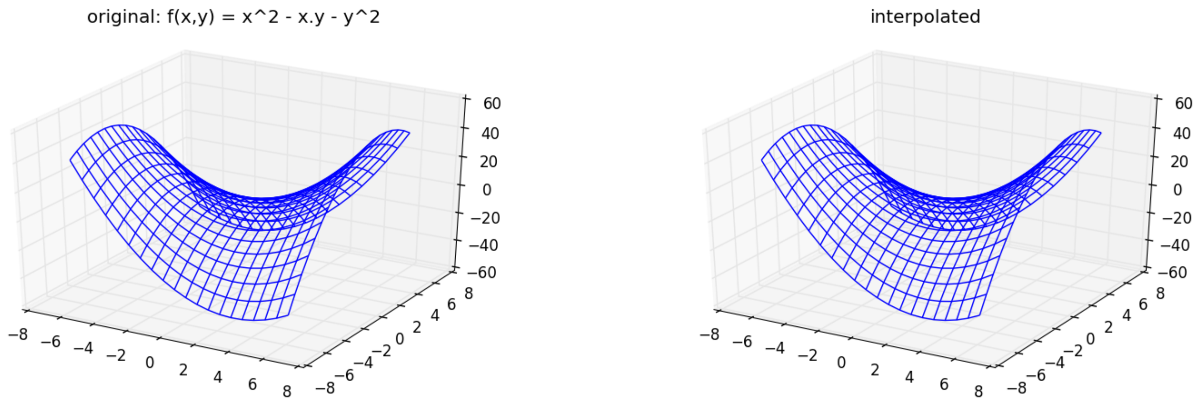

Figure 4: Third-order 2d interpolation of a second-order polynome.
rms = 23.398272 | rms error = 0.000000 | relative rms error = 0.000000%

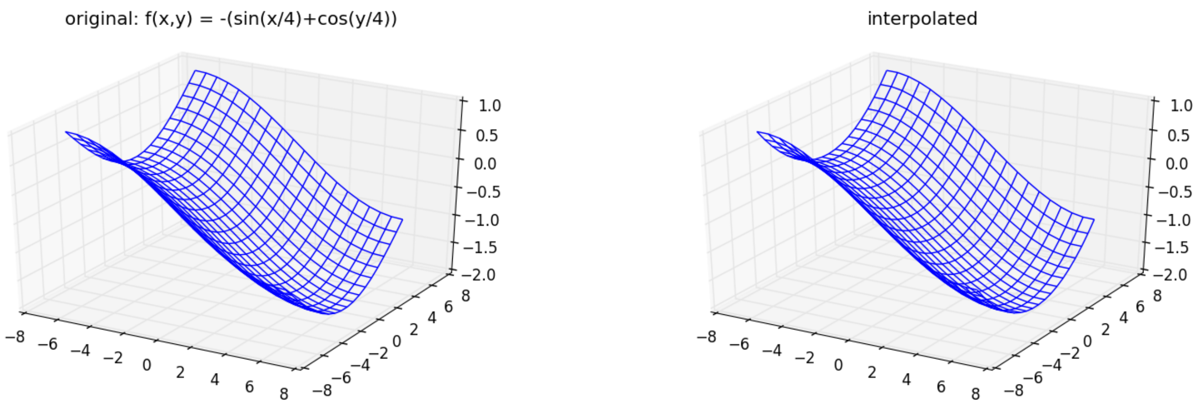

Figure 5: Third-order 2d interpolation of a moderate sine-cosine superposition.
rms = 1.000000 | rms error = 0.028277 | relative rms error = 2.827652%

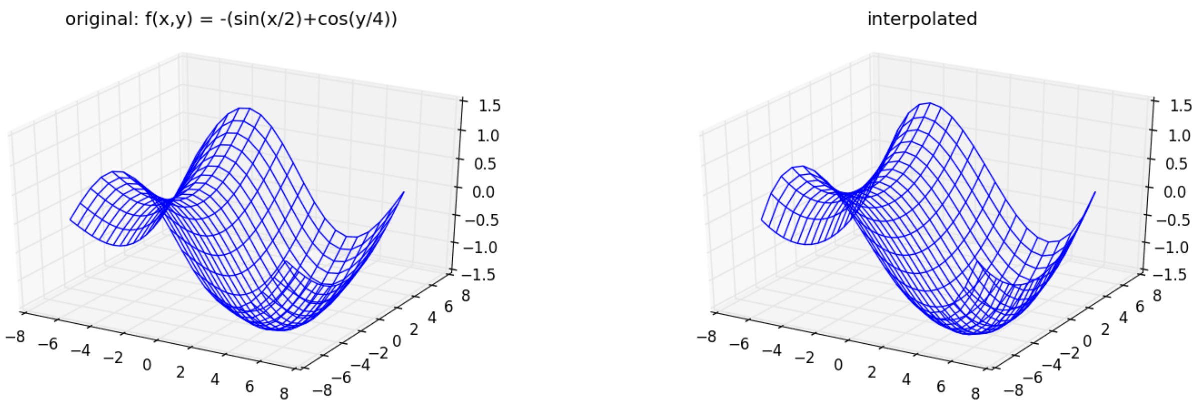

Figure 6: Third-order 2d interpolation of a more difficult sine-cosine superposition.
rms = 0.974679 | rms error = 0.110654 | relative rms error = 11.352840%

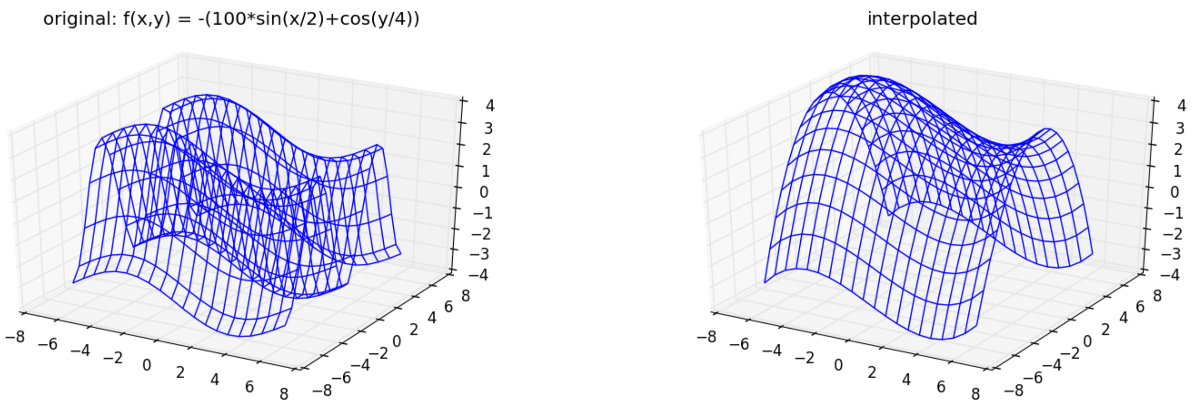

Figure 7: Third-order 2d interpolation of an incommensurable sine-cosine superposition.
rms = 2.280351 | rms error = 3.157907 | relative rms error = 138.483376%

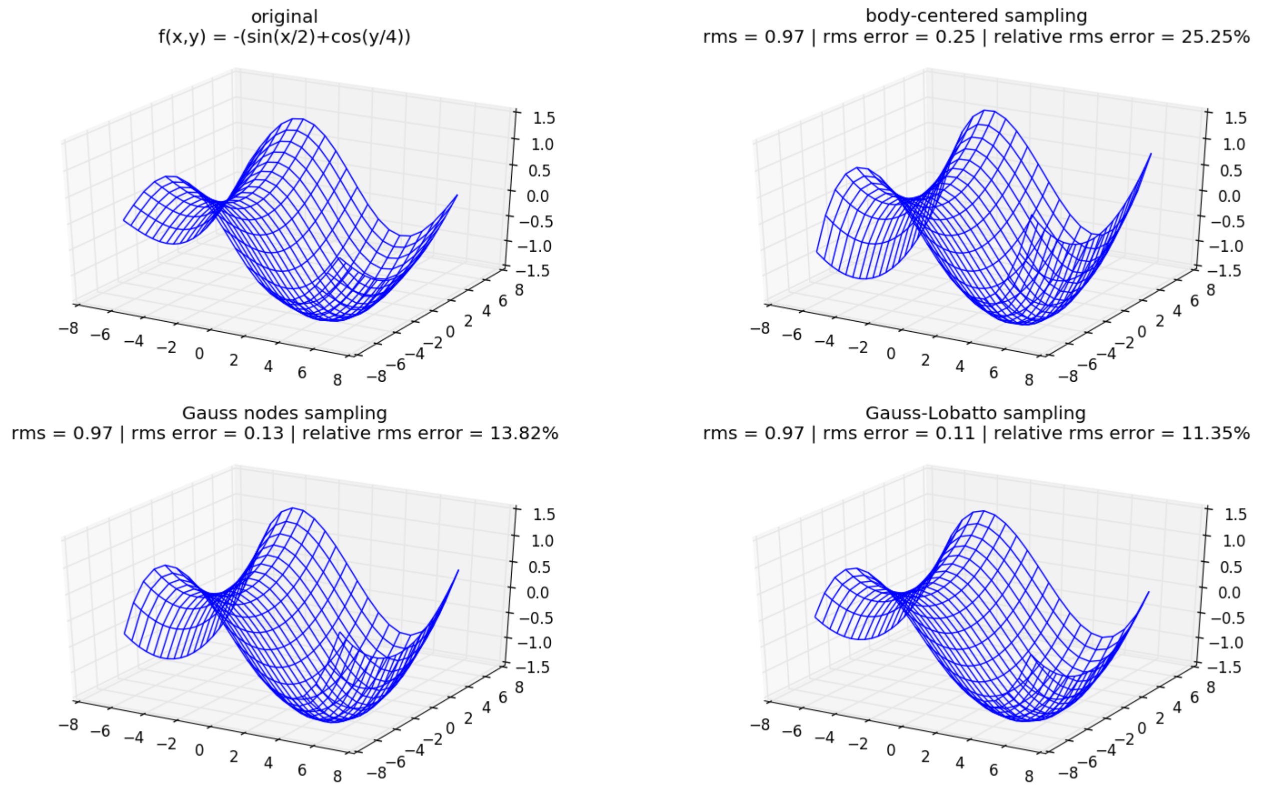

Figure 8: Comparison of third-order 2d interpolation with different sampling grids.

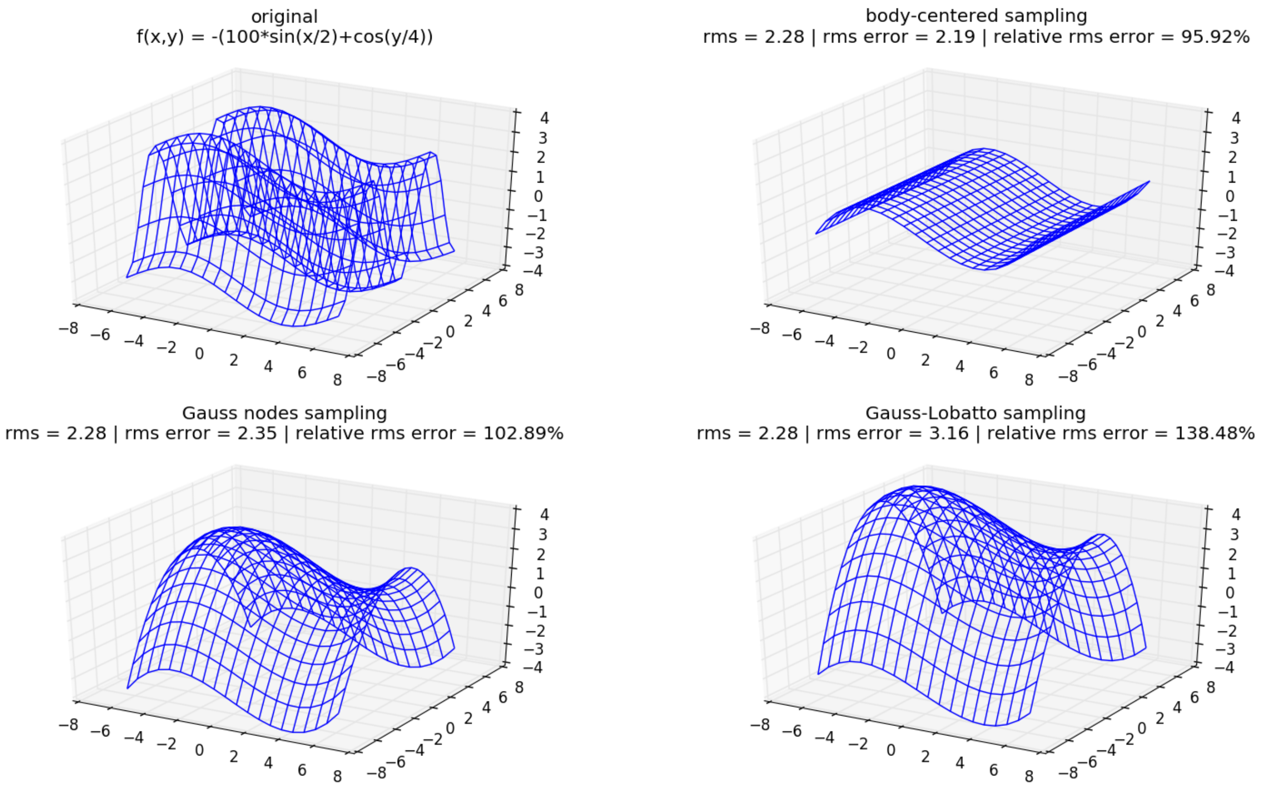

Figure 9: Comparison of third-order 2d interpolation with different sampling grids.

2.2 3D Interpolation over the grid

Following compares the original data on the FLASH grid (BCG) to the back-interpolated data of FLEXI (BCG!).

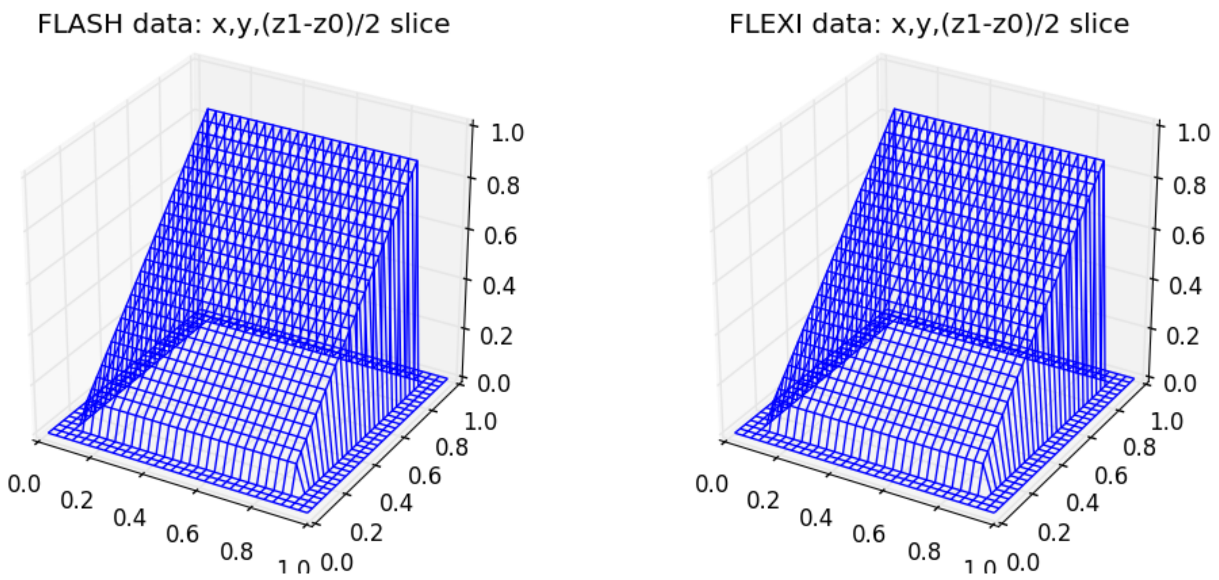

Figure 10: Ascending plane in x-direction: Third-order 3d interpolation of FLASH data to FLEXI data.
rms = 0.404423 | rms error = 0.000000 | relative rms error = 0.000000%

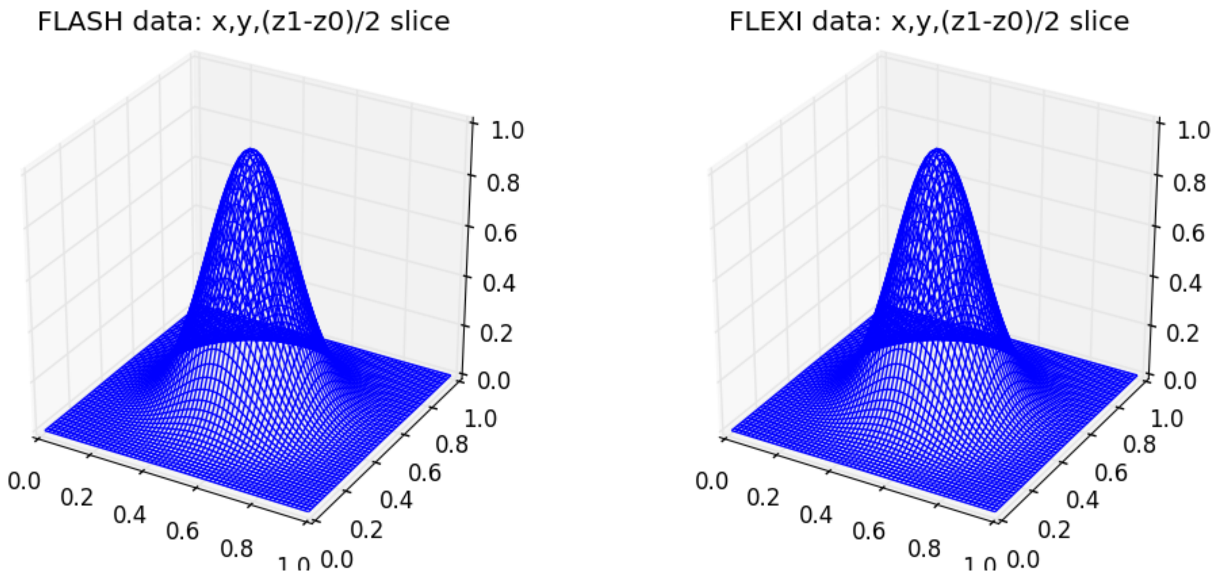

Figure 11: 3D Gaussian: Third-order 3d interpolation of FLASH data to FLEXI data.
rms = 0.125497 | rms error = 0.000000 | relative rms error = 0.000000%

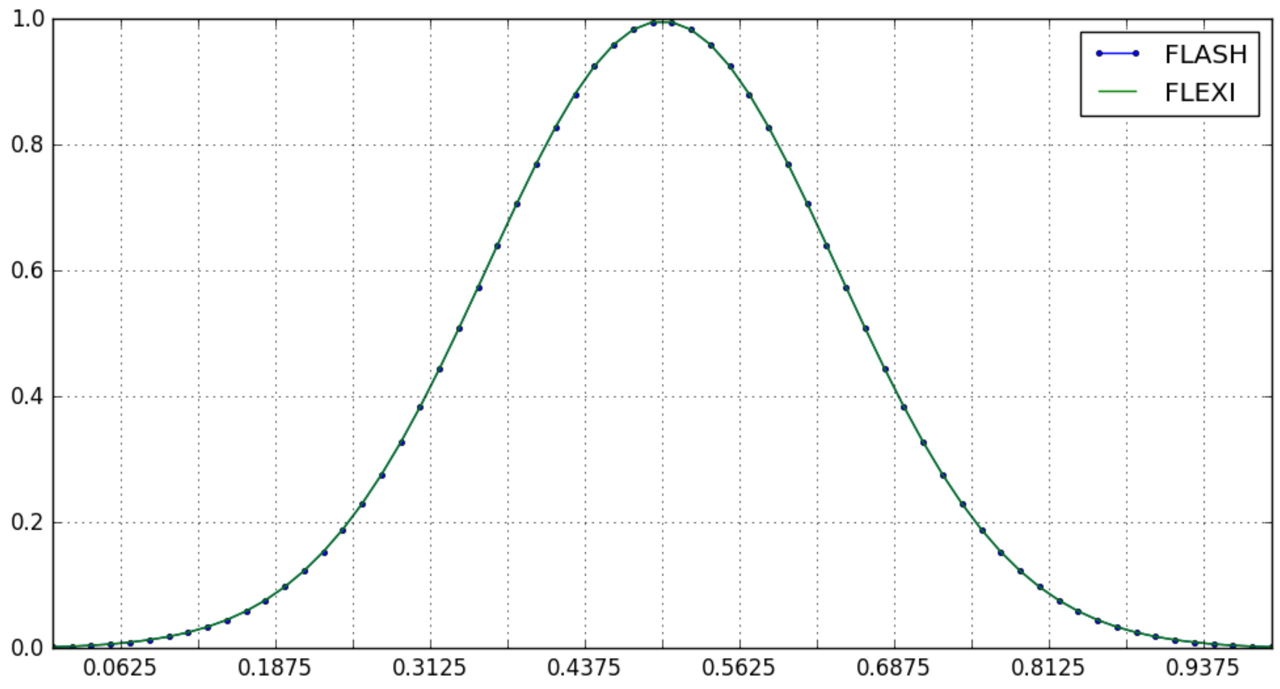

Figure 12: 3D Gaussian - ray through the center: Third-order 3d interpolation of FLASH data to FLEXI data.

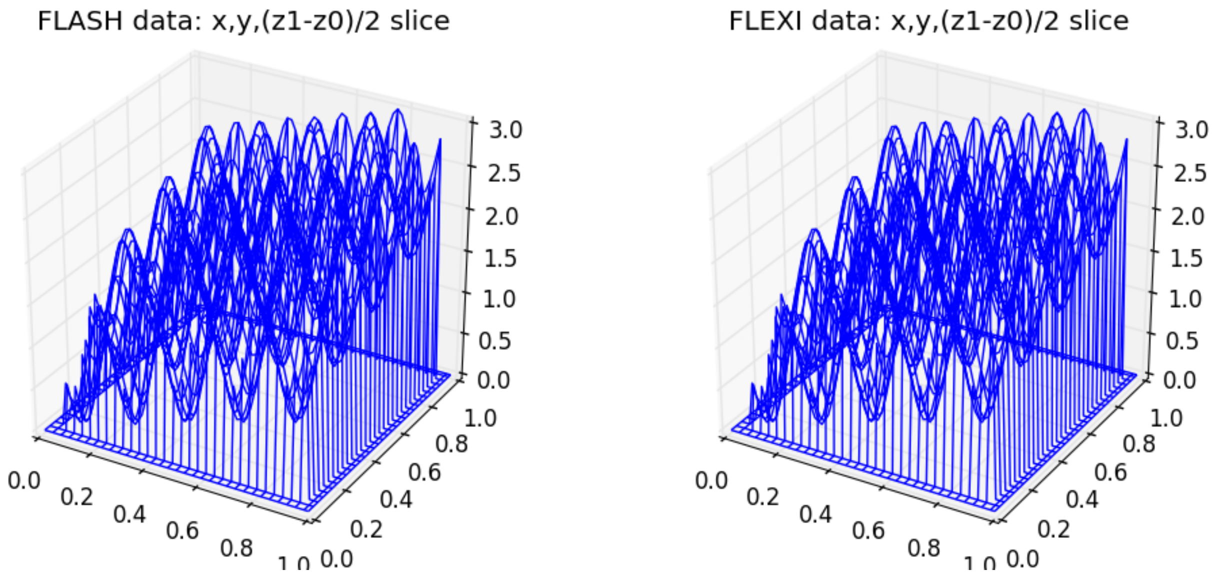

Figure 13: Ascending plane in all directions superposed by sines and cosines. Third-order 3d interpolation of FLASH data to FLEXI data.
rms = 1.377285 | rms error = 0.000000 | relative rms error = 0.000000%

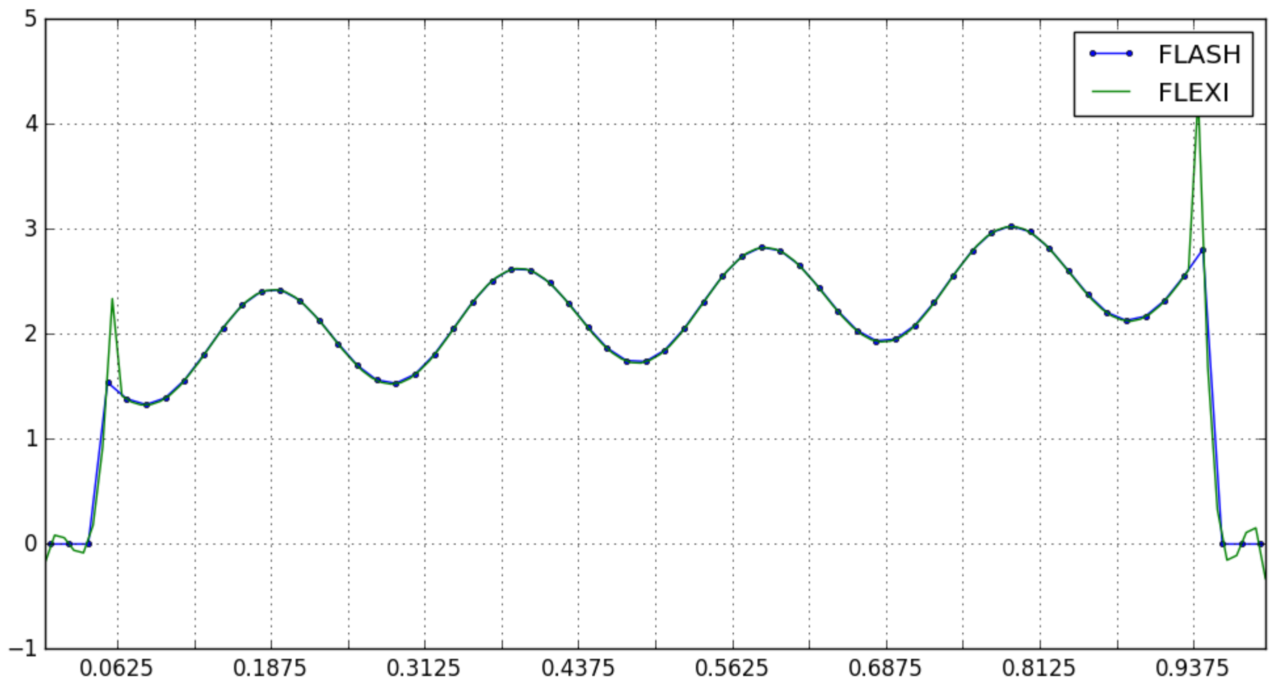

Figure 14: Ascending plane in all directions superposed by sines and cosines - ray through the center: Third-order 3d interpolation of FLASH data to FLEXI data.

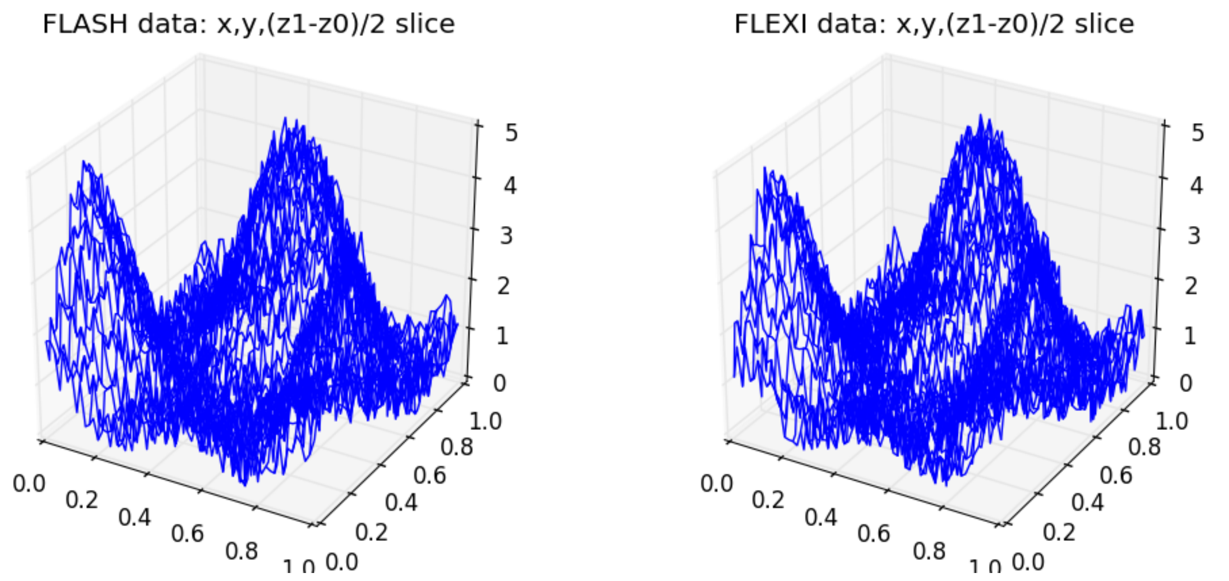

Figure 15: Stress test: 3D superposed sines and cosines with random noise: Third-order 3d interpolation of FLASH data to FLEXI data.
rms = 2.731093 | rms error = 0.409321 | relative rms error = 14.987457%

The bottom line is that for nice and smooth setups the transformation scheme is exact up to machine precision. Nasty discontinuities cause the Lagrange polynomial to break out but within bearable magnitude. The global solution still remains stable and resembles the overall structure. More on this in the next chapter. Remark: Considering figure 14, how can this setup still yield zero error estimate? The green curve depicts more points than just the (back-interpolated) BCG nodes. The error estimate only considers the values at the same position as the FLASH data which coincide.

3 3D Interpolation of Shocks / Real world data

Following figures depict the very same as the previous section but with real world FLASH data of a fully developed turbulence.

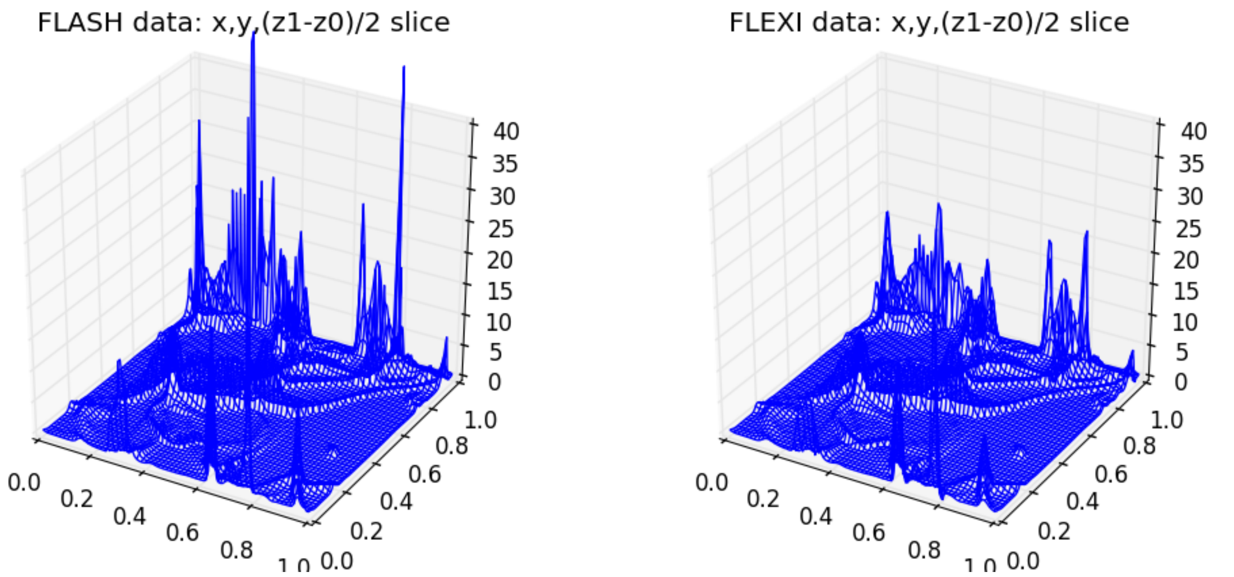

Figure 16: Fully developed turbulence: Third-order 3d interpolation of FLASH data to FLEXI data. rms = 3.162914 | rms error = 0.879976 | relative rms error = 27.821685%

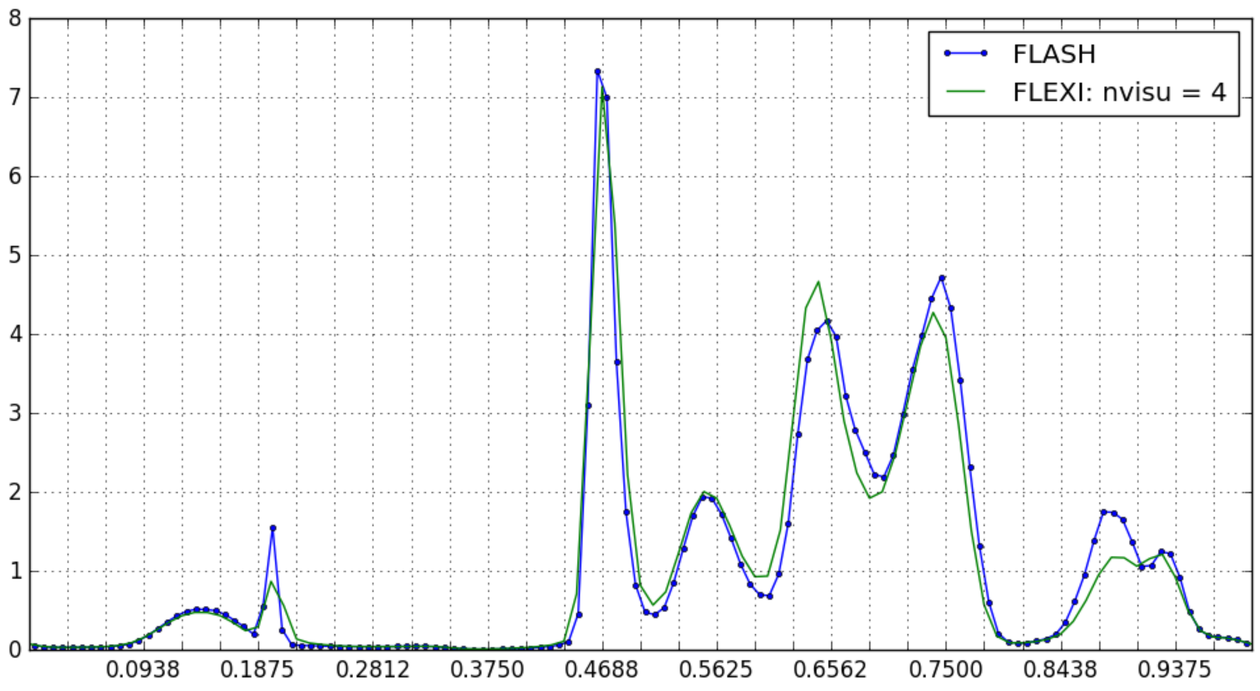

Figure 17: Fully developed turbulence - ray through the center: Third-order 3d interpolation of FLASH data to FLEXI data. The vertical dashed lines depict the boundaries of an element.

As one can see the result is not satisfiying. Shocks do not get resolved very well with this scheme. Suprisingly, the huge jumps within an element do not cause any harm. This turbulence snapshot has a resolution of 128^3 nodes in BCG space. Hence, higher resolution like 256^3, which is necessary for serious simulation anyway, in the first place would mitigate interpolation errors considerably.

The bigger problem is the necessary extrapolation near the element boundaries during conversion from BCG space to nodal space. As noted in the beginning Lagrange polynomes tend to explode outside their anchor nodes. In conjunction with the influence of the other dimensions this might explain the shift of the interpolant at shocks. To exemplify this phenomena following artificial shock cases were generated.

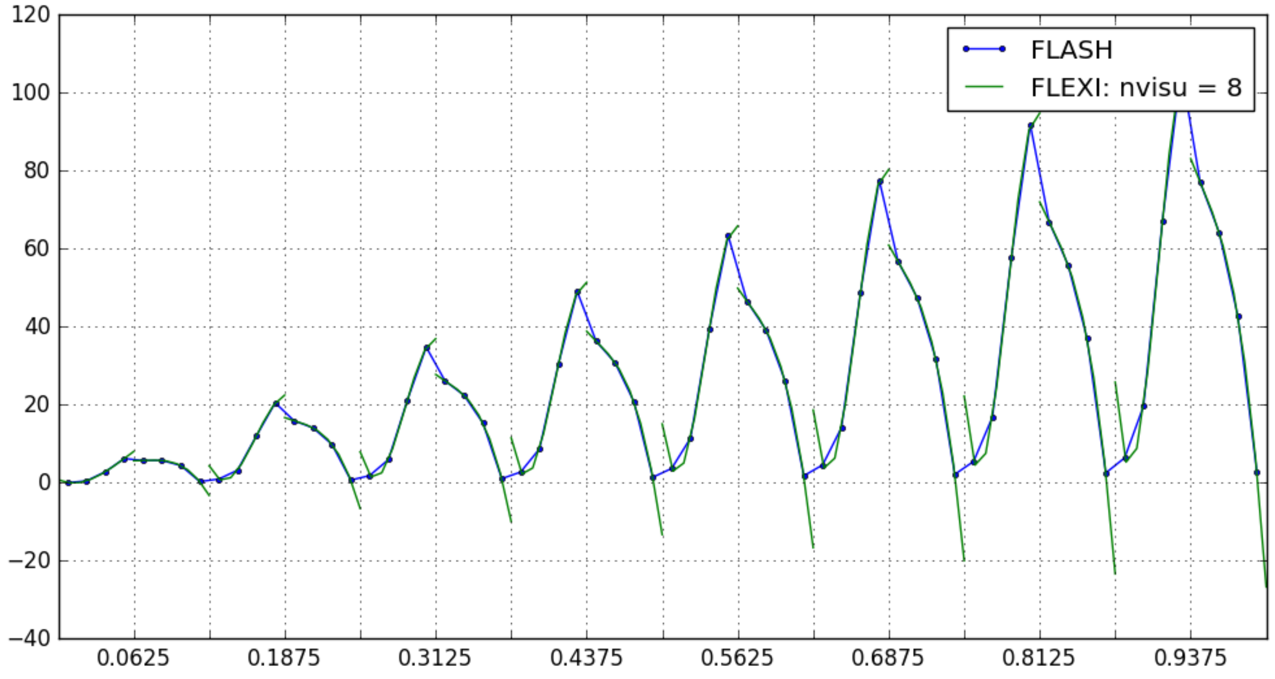

Figure 18: Superposed sines and cosines - ray through the center: Third-order 3d interpolation of FLASH data to FLEXI data. The vertical dashed lines depict the boundaries of an element.

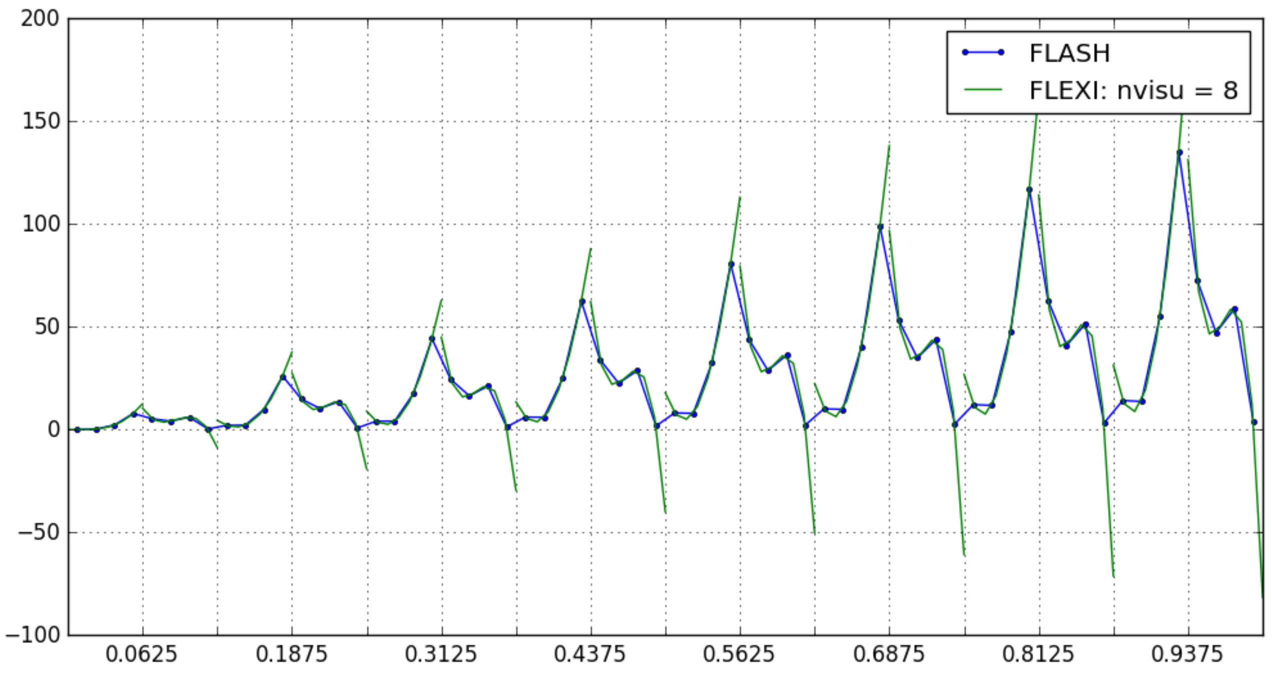

Figure 19: Superposed sines and cosines - ray through the center: Third-order 3d interpolation of FLASH data to FLEXI data. The vertical dashed lines depict the boundaries of an element.

Bearing figure 1 (comparison of grid spaces) in mind extrapolation from BCG to nodal space clearly yields erroneous values. Remedy would be to enforce the Lagrange polynome to consider the via a different method obtained values at element boundaries. If one could achieve that I would bet for considerably better shock resolution. Suggestions are welcome.

4 Conclusion

This document shows the correctness and stability of the current transformation scheme from FLASH (BCG) to FLEXI (GNG/LNG) and back to BCG for various setups. Shocks do not get resolved very well due to extrapolation errors inherent to the Lagrange interpolation.

Smooth or not too harsh setups are transformed up to machine precision.