HiPPIS A High-Order Positivity-Preserving Mapping Software for Structured Meshes User Manual

Timbwaoga A, J, Ouermi
University of Utah School of Computing
U of U Scientific Computing and Imaging Institute
touermi@cs.utah.edu

Robert M. Kirby

University of Utah School of Computing
U of U Scientific Computing and Imaging Institute
kirby@cs.utah.edu

Martin Berzins
University of Utah School of Computing
U of U Scientific Computing and Imaging Institute
mb@cs.utah.edu

October 2023

Contents

1 Getting Started **2**

1.1 Dependencies . . . 2

1.2 Installing . . . 4

1.3 Executing program . . . 4

1.3.1 Matlab . . . 4

1.3.2 Fortran . . . 4

1.4 Help . . . 4

1.5 Manuscript Examples . . . 5

1.5.1 Matlab Examples . . . 5

1.5.2 Fortran Examples . . . 5

2 Implementation and Performance **6**

3 Module and Functions **8**

3.1 adaptiveInterpolation1D(...) . . . 8

3.2 adaptiveInterpolation2D(...) . . . 9

3.3 adaptiveInterpolation3D(...) . . . 10

3.4 newtonPolyVal(...) . . . 12

3.5 divdiff(...) . . . 12

1 Getting Started

HiPPIS is a polynomial-based data-bounded and positivity-preserving interpolation software for function approximation and mapping data values between structure (uniform and nonuniform) meshes.

1.1 Dependencies

- Matlab is required for the version of the software implemented in Matlab. No special toolboxes are required. The folder structure of the Matlab version is organized as follows:

```
Matlab
|  Src
|  |  adaptiveInterpolation1D.m
|  |  adaptiveInterpolation2D.m
|  |  adaptiveInterpolation3D.m
```

```
|  |  divdiff.m
|  |  newtonPolyVal.m
|  Drivers
|  |  Tutorial
|  |  |  main.m
|  |  |  testing.m
|  |  Mapping
|  |  |  main.m
|  Extras
|  |  ...
```

The core files used for the implementation of the data-bounded and positivity-preserving interpolation methods are in the "Src" folder. The folder "Drivers" contains tutorials showing how to use HiPPIS and examples to produce the results in the paper manuscript. The remaining folder "Extras" has supplemental files and scripts for displaying and plotting figures.

- The Fortran version requires the installation of Intel (ifort), or gnu (gfortran) compilers with OpenMP4. The vectorized version of the code required the use of Intel compilers. The Make files *Fortran/Mappin/Makefile* and *Fortran/BOMEX/Makefile* can be modified for different compilers including Intel, gnu, HPE cray. The folder structure of the Matlab version is organized as follows:

```
Fortran
|  Src
|  |  mod_adaptiveInterpolation.F90
|  |  MExFiles
|  |  |  apdaptiveInterpolation1D.F90
|  |  |  apdaptiveInterpolation2D.F90
|  |  |  apdaptiveInterpolation3D.F90
|  Drivers
|  |  Tutorial
|  |  |  main.F90
|  |  |  testing.F90
|  |  |  main.m
|  |  Mapping
|  |  |  main.F90
```

```
|  |  BOMEX
|  |  |  main.F90
|  Extras
|  |  ...
```

The core files used for the implementation of the data-bounded and positivity-preserving interpolation methods are in the "Src" folder. The folder "MexFiles" contains wrappers for Matlab and The folder "Drivers" contains tutorials showing how to use HiPPIS and examples to produce the results in the paper manuscript. The remaining folder "Extras" has third-party routines for the BOMEX and Mapping examples. The different supplemental files and scripts for displaying and plotting figures are also in the folder "Extras".

1.2 Installing

- Downloading HiPPIS (Matlab and/or Fortran) is sufficient for installation.

1.3 Executing program

1.3.1 Matlab

Open Matlab and run the script "tutorial.m" in the folder "Matlab".

```
cd Matlab/Drivers/Tutorial/
main
```

1.3.2 Fortran

```
cd Fortran/Drivers/Tutorial/
make
./main
```

1.4 Help

The file "main.m" (in "Matlab/Drivers/Tutorial") or "main.m" or "main.F90" (in "Fortran/Drivers/Tutorial") is a good place to start. Both "main.m" and "main.F90" provide simple examples showing how to use HiPPIS with 1D, 2D, and 3D structured tensor product meshes.

Note:

The choices of parameters *st*, *eps0*(ϵ_0), and *eps1*(ϵ_1) influence the quality of the approximation in 1D, 2D, and 3D examples with significant oscillations.

The choice for parameters *st*, *eps0*(ϵ_0), and *eps1*(ϵ_1) is dependent on the data. In the case of 2D. For cases when smoothness is the primary goal, $st = 1$ is a suitable choice. In cases where $st = 1$ led to biased stencils $st = 2$ can be used to prioritize symmetric stencils. In cases where the size of the intervals varies greatly $st = 3$ is a better choice because it prioritizes locality. The parameters *eps0*(ϵ_0) and *eps1*(ϵ_1) should be chosen to be small enough such that hidden extrema are recovered without introducing new large oscillations.

1.5 Manuscript Examples

More examples can be found in "/Matlab/Mappin/main.m" or "/Fortran/Mapping/main.F90". These examples are used to produce the results presented in a manuscript submitted for publication entitled "Algorithm xxxx: HiPPIS: A High-Order Positivity-Preserving Mapping Software for Mtructured Meshes". Each example produces results that are saved in files. The saved data are then used to calculate the errors and produce the figures and tables in the manuscript.

Note:

Producing the results in the Manuscript takes a few hours because 1) the PDE problem in folder "BOMEX" is run multiple times and uses a small time step, and 2) the 2D approximation examples are run multiple times and use 1000x1000 output mesh.

1.5.1 Matlab Examples

```
cd Matlab/Drivers/Mapping
main
```

The driver for the examples is "/Matlab/Drivers/Mapping/main.m".

1.5.2 Fortran Examples

```
cd Fortran/Extras/
sh run_manuscript_examples.sh
```

Open the Matlab software and run

```
plot_manuscript_examples
```

The drivers for the examples are "Fortran/Mapping/main.F90" and "Fortran/BOMEX/main.F90".

Testing

Supplemental tests are provided in the files "Mapping/Tutorial/testing.m" and "Fortran/Tutorial/testing.F90".

2 Implementation and Performance

This section provides more details on the implementation and different techniques used to improve the performance of the DBI and PPI methods. HiPPIS is implemented using Fortran90 and Matlab. The Fortran implementation includes a Makefile and examples that are used to build an executable. The Makefiles require an Intel compiler with openMP4. The Makefiles can be modified for other compilers such as gfortran (gnu), HPE Cray, etc. The Fortran version contains all the examples presented in this manuscript. The Matlab version requires only the installation of the Matlab software to be able to use HiPPIS. The Matlab version does not include the BOMEX example and the MQSI method because they are both based on third-party Fortran implementations with no available Matlab versions. Vectorizing adaptive algorithms is challenging, especially when the conditions used for the adaptive decisions are not known a priori. Both the DBI and PPI approaches are adaptive methods that are not suitable for vectorization and therefore fail to take advantage of vector units in computational resources. Let "i-loops" be the loops that iterate over the intervals from the input mesh, and "j-loops" be the loops that iterate from one to the target polynomial degree d. Although ENO methods are adaptive, they can be vectorized along the "i-loops" and "j-loops" [4]. The vectorization along the "j-loops" is possible because the values (divided differences) used for the adaptive decisions are known before entering the loops. This type of vectorization is not suitable for the DBI and PPI methods. When adding a point to go from $\mathcal{V}_{j-1}$ to $\mathcal{V}_j$, the point selection depends on the selection order of previously added points, the conditions for data-boundedness and positivity, and the user-supplied parameters. These dependencies, which are not known a priori, and many nested conditionals inside the DBI and PPI algorithms prevent vectorization along the "j-loops". For a given interval, the interpolation approximation is local and independent of interpolation procedures in other intervals. Therefore, the interpolation approximation for each interval (along "i-loops") can be calculated in parallel. Because of the dependencies and conditionals, vectorization of the DBI and PPI algorithms along the "i-loops" is more complex than the approach used for the ENO method in [4].

Although there are no dependencies between the intervals, the dependencies along the "j-loops", and complex control flows remain. The implementation enables vectorization by structuring the code such that the "j-loops" and "i-loops" are the outer and inner loops, respectively, by removing the complex control flows used, by introducing local variables to reduce dependencies inside each loop, and by placing the OpenMP directive OMP SIMD before the "i-loops". Similar ideas for code restructuring are introduced in [2, 3] in the context of vectorizing complex numerical weather prediction codes.

Table 1 shows the runtimes for the PCHIP, MQSI [1], the original unvectorized, and the vectorized DBI and PPI codes on a Knights Landing (KNL) core with a 2018 Intel compiler. The execution times are obtained from the Fortran implementation of the different methods. The KNL architecture has a clock

frequency of 1.3 GHz and AVX-512 vector processing units. Each vector unit has a maximum speed-up of 16x for double-precision operations. Further code transformation is required for the gfortran (gnu) compiler as it is unable to vectorize the modified loops. The vectorization report indicates that the combination of control flow and the remaining conditionals prevents vectorizations. In the case of the Intel compiler, placing the directive $OMP SIMD is sufficient to vectorize the modified loops. In Table 1 and remaining Sections, $\mathcal{P}_d$ represents the space of polynomial of degree d with the parameter d being an input in the algorithms described above. The PCHIP, MQSI, DBI, and PPI methods use N and $N \times N$ uniformly spaced points to approximate the example functions $f_1(x) = 0.1/(0.1 + 25x^2)$ and $f_4(x, y) = 0.1/(0.1+25(x^2+y^2))$, respectively. The approximated functions are evaluated at $N+1$ and $(N + 1) \times (N + 1)$ uniformly spaced points for the 1D and 2D cases. We exclusively report the performance results for $f_1(x)$ and $f_4(x, y)$ because the 1D examples have similar execution times, and likewise, the 2D examples exhibit similar execution times. The PCHIP method has smaller execution times compared to the MQSI, DBI, and PPI methods. The MQSI algorithm is computationally more expensive than the other approaches because it enforces C^2-continuity and uses a search algorithm based on bisection to adjust the first and second derivative values. The PCHIP method requires fewer data and has fewer complex control flows compared to the MQSI, DBI and PPI methods. The runtimes for the vectorized version of the DBI and PPI methods are closer to the runtimes for the PCHIP method. The results from Table 1 show that reorganizing the code to improve vectorization and locality leads to smaller runtimes compared to unvectorized code. These performance improvements correspond to a minimum and maximum speed-up of 1.89 and 4.13 over the unvectorized version.

N	PCHIP	MQSI	PPI and DBI					
			$\mathcal{P}_4$		$\mathcal{P}_8$		$\mathcal{P}_{16}$	
			unvec	vec	unvec	vec	unvec	vec
1D $f_1(x) = 0.1/(0.1 + 25x^2)$ $x \in [-1, 1]$								
17	8.11E-3	9.73E-1	2.34E-2	1.24E-2	3.30E-2	1.22E-2	5.82E-2	1.85E-2
33	1.41E-2	1.85	4.09E-2	1.65E-2	6.31E-2	2.09E-2	1.18E-1	3.20E-2
65	2.52E-2	3.80	7.30E-2	2.72E-2	1.21E-1	3.89E-2	2.31E-1	6.13E-2
127	4.14E-2	7.53	1.30E-1	4.59E-2	2.46E-1	7.98E-2	4.74E-1	1.26E-1
257	8.73E-2	1.49E+1	2.28E-1	8.80E-2	4.83E-1	1.52E-1	9.75E-1	2.36E-1
2D $f_4(x, y) = 0.1/(0.1 + 25(x^2 + y^2))$ $x, y \in [-1, 1]$								
17^2	1.89E-1	3.45E+1	5.42E-1	2.53E-1	1.11	4.36E-1	2.02	6.48E-1
33^2	7.18E-1	1.29E+2	2.00	8.20E-1	4.19	1.45	7.97	2.18
65^2	2.79	4.98E+2	7.52	2.92	1.63E+1	5.22	3.19E+1	7.99
127^2	1.08E+1	1.90E+3	2.96E+1	1.19E+1	6.34E+1	2.12E+1	1.24E+2	3.40E+1
257^2	4.45E+1	8.49E+3	1.16E+2	4.45E+1	2.54E+2	8.05E+1	5.16E+2	1.26E+2

Table 1: Execution times in milliseconds (ms) for the different interpolation methods with varying resolution and target polynomial degree. The columns "unvec" and "vec" represent the vectorized and unvectorized execution times, respectively. $\mathcal{P}_d$ indicate the space of polynomial of degree d. The execution times for the DBI and PPI methods are the same.

3 Module and Functions

This section describes HiPPIS interface, the input and output parameters used in the different functions, and subroutine calls.

3.1 adaptiveInterpolation1D(...)

The description of ***adaptiveInterpolation1D(...)*** is the same for ***adaptiveInterpolation1D_vec(...)*** and both the Matlab and Fortran versions.

adaptiveInterpolation1D(x, y, n, xout, yout, m, degree, interpolation_type, st, eps0, eps1, deg)

The subroutine adaptiveInterpolation1D is a polynomial interpolation method that builds a piece-wise function based on the input (x,y). This piece-wise function is then evaluated at the output points xout to give (xout, yout). The interpolation method preserves positivity or data-boundedness. The data-bounded or positivity-preserving interpolant is constructed for each interval based on the theory in https://arxiv.org/abs/2204.06168 and the algorithm in the manuscript [REF].

INPUT:

n: the number of points in the 1D vector x.

m: the number of points in the 1D vector xout.

x: 1D mesh points of length n. For $i = 1, ..., n-1$ $x_i > x_{i+1}$

y: 1D vector that have the data values associated with the points x_i for $i = 1, ..., n$

xout: 1D vector of length m that represents the locations where we wish to interpolate to.

Interpolation_type: used to determine the type of interpolation to be used to build interpolant.

- interpolation_type=1: a data-bounded interpolant is built for each interpolant.
- interpolation_type=2: a positivity-preserving interpolant is built for each interpolant.

degree: target polynomial degree and maximum polynomial degree used for each interval.

st (optional): used guide point selection process in cases when adding the next point to the right or left both meet the requirements for positivity or data-boundedness.

- st=1 (default): the point with the smallest divided difference is added (ENO stencil).

– st=2 the point to the left of the current stencil is selected if the number of points to the left of x_i is smaller than the number of points to the right of x_i ($i - si < ei - i$). Similarly, the point to the right is selected if the number of points to the right of x_i is smaller than the number of points to the left ($i - si > ei - i$). When both the number of points to the right and left are the same, the algorithm chooses the point with the smallest lambda.

– st=3 the point closest to the starting interval is chosen.

eps0 (optional): positive parameter used to constrain the bounds of the positive interpolant in intervals with no extremum detected.

eps1 (optional): positive parameter used to constrain the bounds of the positive interpolant in intervals with extremum detected.

OUTPUT:

yout: results of evaluating interpolants at the locations xout.

deg (optional): 1D vector that holds the degree of the interpolant used for each interval

3.2 adaptiveInterpolation2D(...)

The description of ***adaptiveInterpolation2D(...)*** is the same for ***adaptiveInterpolation2D_vec(...)*** and both the Matlab and Fortran versions.

adaptiveInterpolation2D(x, y, nx, ny, v, xout, yout, mx, my, vout, degree, interpolation_type, st, eps0, eps1)

The subroutine **adaptiveInterpolation2D(...)** is the vectorized version of the subroutine ***adaptiveInterpolation2D(...)*** that adaptively builds a 2D tensor product interpolation based ***adaptiveInterpolation1D(...)***.

INPUT:

nx: the number of points in the 1D vector x.

ny: the number of points in the 1D vector y.

mx: the number of points in the 1D vector xout.

my: the number of points in the 1D vector xout.

x: 1D mesh points of length nx used to build tensor product mesh. For $i = 1, ..., n-1 x_i < x_{i+1}$

y: 1D mesh points of length ny used to build tensor product mesh. For $i = 1, ..., n-1 y_i < y_{i+1}$

v: 2D array that has the data values associated with the tensor product mesh obtained from x and y

xout: 1D vector of length mx used to construct the output tensor product mesh.

yout: 1D vector of length my used to construct the output tensor product mesh.

interpolation_type: used to determine the type of interpolation to be used to build interpolant.

- interpolation_type=1: a data-bounded interpolant is built for each interpolant.
- interpolation_type=2: a positivity-preserving interpolant is built for each interpolant.

degree: target polynomial degree and maximum polynomial degree used for each interval.

st (optional): used guide point selection process in cases when adding the next point to the right or left both meet the requirements for positivity or data-boundedness.

- st=1 (default): the point with the smallest divided difference is added (ENO stencil).
- st=2 the point to the left of the current stencil is selected if the number of points to the left of x_i is smaller than the number of points to the right of x_i ($i - si < ei - i$). Similarly, the point to the right is selected if the number of points to the right of x_i is smaller than the number of points to the left ($i - si > ei - i$). When both the number of points to the right and left are the same, the algorithm chooses the point with the smallest lambda.
- st=3 the point closest to the starting interval is chosen.

eps0 (optional): positive parameter used to constrain the bound of the positive interpolant in intervals with no extremum detected.

eps1 (optional): positive parameter use constrain the bound of the positive interpolant in intervals with extremum detected.

OUTPUT:

- vout: results of evaluating interpolants on tensor product mesh obtained from xout and yout.

3.3 adaptiveInterpolation3D(...)

The description of ***adaptiveInterpolation3D(...)*** is the same for ***adaptiveInterpolation3D_vec(...)*** and both the Matlab and Fortran versions.

adaptiveInterpolation3D(x, y, z, nx, ny, nz, v, xout, yout, zout, mx, my, mz, vout, degree, interpolation_type, st, eps0, eps1)

The subroutine adaptiveInterpolation3Dvec(...) is the vectorized version of the subroutine adaptiveInterpolatioin3D(...) that adaptively builds a 3D tensor product interpolation based on the 1D subroutine adaptiveInterpolation1D(...).

INPUT:

- nx: the number of points in the 1D vector x.
- ny: the number of points in the 1D vector y.
- nz: the number of points in the 1D vector z.
- mx: the number of points in the 1D vector xout.
- my: the number of points in the 1D vector yout.
- mz: the number of points in the 1D vector zout.
- x: 1D mesh points of length nx used to build tensor product mesh. For $i = 1, ..., n-1 x_i < x_{i+1}$
- y: 1D mesh points of length ny used to build tensor product mesh. For $i = 1, ..., n-1 y_i < y_{i+1}$
- z: 1D mesh points of length ny used to build tensor product mesh. For $i = 1, ..., n-1 z_i < z_{i+1}$
- v: 3D array that has the data values associated with the tensor product mesh obtained from x, y, and z
- xout: 1D vector of length mx used to construct the output tensor product mesh.
- yout: 1D vector of length my used to construct the output tensor product mesh.
- zout: 1D vector of length my used to construct the output tensor product mesh.
- Interpolation_type: used to determine the type of interpolation to be used to build interpolant.
  - interpolation_type=1: a data-bounded interpolant is built for each interpolant.
  - interpolation_type=2: a positivity-preserving interpolant is built for each interpolant.
- degree: target polynomial degree and maximum polynomial degree used for each interval.
- st (optional): used guide point selection process in cases when adding the next point to the right or left both meet the requirements for positivity or data-boundedness.
  - st=1 (default): the point with the smallest divided difference is added (ENO stencil).

- st=2 the point to the left of the current stencil is selected if the number of points to left of x_i is smaller than the number of points to the right of x_i $(i - si < ei - i)$. Similarly, the point to the right is selected if the number of points to the right of x_i is smaller than the number of points to the left $(i - si > ei - i)$. When both the number of points to the right and left are the same, the algorithm chooses the point with the smallest lambda.
- st=3 the point closest to the starting interval is chosen.

eps0 (optional): positive parameter used to constrain the bound of the positive interpolant in intervals with no extremum detected.

eps1 (optional): positive parameter used to constrain the bound of the positive interpolant in intervals with extremum detected.

OUTPUT:

- vout: results of evaluating interpolants on tensor product mesh obtained from xout and yout.

3.4 newtonPolyVal(...)

The description of ***newtonPolyVal(...)*** is the same for both the Matlab and Fortran versions.

newtonPolyVal(x, u, d, xout, yout) The newtonPolyVal subroutine builds a Newton interpolant using the mesh points in x and the divided differences in u. The constructed interpolant is then evaluated at xout.

INPUT:

- x: mesh points to be used to build the interpolant.
- u: divided differences needed to build the interpolant.
- d: interpolant degree.
- xout: output mesh points where we wish to evaluate the interpolant.

OUTPUT:

- yout: results from evaluating the interpolant at xout.

3.5 divdiff(...)

The description of ***newtonPolyVal(...)*** is the same for ***newtonPolyVal_vec(...)*** and both the Matlab and Fortran versions.

divdiff(x, y, n, d, table) The subroutine divdiff(...) is a subroutine divdiff that computes the table of divided differences.

INPUT:

- x: 1D vector of x coordinates
- y: 1D vector that holds the data values associated with the locations x.
- d: the maximum number of consecutive mesh points used to compute divided differences.
- n: the number of points in x.

OUTPUT:

- table: array of dimensions of $n \times (d + 1)$

References

[1] Thomas Lux, Layne T. Watson, Tyler Chang, and William Thacker. Algorithm 1031: Mqsi–monotone quintic spline interpolation. *ACM Trans. Math. Softw.*, 49(1), mar 2023.

[2] Timbawaoga A. J. Ouermi, Aaron Knoll, Robert M. Kirby, and Martin Berzins. Openmp 4 fortran modernization of wsm6 for knl. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, PEARC17, pages 12:1–12:8, New York, NY, USA, 2017. ACM.

[3] Timbwaoga A. J. Ouermi, Robert Kirby, and Martin Berzins. Performance Optimization Strategies for WRF Physics Schemes Used in Weather Modeling. *International Journal of Networking and Computing*, 8(2):301–327, 2018.

[4] Chi-Wang Shu, Thomas A. Zang, Gordon Erlebacher, David Whitaker, and Stanley Osher. High-order eno schemes applied to two- and three-dimensional compressible flow. *Applied Numerical Mathematics*, 9(1):45–71, 1992.