The Journal of Open Source Software

LaplaceInterpolation.jl: A Julia package for fast interpolation on a grid

Vishwas Rao[1], **Charlotte L. Haley**[1], **and Mihai Anitescu**[1,2]

1 Argonne National Laboratory **2** University of Chicago

DOI: 10.21105/joss.03766

Software

- Review
- Repository
- Archive

Editor: Viviane Pons

Reviewers:

- @wkearn
- @eviatarbach

Submitted: 09 September 2021
Published: 02 February 2022

Summary

We implement a linear-time algorithm for interpolation on a regular multidimensional grid in the Julia language. The algorithm is an approximate Laplace interpolation (Press, 1992) when no parameters are given; and when parameters $m \in \mathbb{Z}$ and $\epsilon > 0$ are set, the interpolant approximates a Matérn kernel, of which radial basis functions and polyharmonic splines are a special case. We implement, in addition, Neumann, Dirichlet (trivial), and average boundary conditions with potentially different aspect ratios in the different dimensions. The interpolant functions in arbitrary dimensions.

Mathematical Background

Radial basis functions and splines can be unified conceptually through the notion of Green's functions and eigenfunction expansions (Fasshauer, 2012). The general multivariate Matérn kernels are of the form

$$K(\mathbf{x};\mathbf{z}) = K_{m-d/2}(\epsilon||\mathbf{x}-\mathbf{z}||)(\epsilon||\mathbf{x}-\mathbf{z}||)^{m-d/2}$$

for $m > d/2$, where K is the modified Bessel function of the second kind with parameter ν and can be obtained as Green's kernels of

$$L = (\epsilon^2 I - \Delta)^m,$$

where Δ denotes the Laplacian operator in d dimensions. Polyharmonic splines, including thin plate splines, are a special case of the above.

The discrete gridded interpolation seeks to find an interpolation $u(\mathbf{x})$ that satisfies the differential operator in d dimensions on the nodes $\mathbf{x}_i$ where there is no data and equals y_i everywhere else. Discretely, one solves the matrix problem

$$\mathbf{C}(\mathbf{u}-\mathbf{y}) - (1-\mathbf{C})L\mathbf{u} = 0,$$

where $\mathbf{y}$ contains the y_i's and placeholders where there is no data, L denotes the discrete matrix operator, and C is a diagonal matrix that indicates whether node $\mathbf{x}_i$ is observed.

In d dimensions the matrix $A^{(d)}$ of size $M \times M$, where M is the product of the number of grid locations in each dimension, i.e. $M = \prod_{i=1}^{d} N_i$, expands the first-order finite difference curvature, and its (i,j)th entry is -1 when node j is in the set of neighbors of the node $\mathbf{x}_i$ and has the number of such neighbors on the diagonal. Note that if node i is a boundary

node, the ith row of $A^{(d)}$ has -1s in the neighboring node spots and the number of such nodes on the diagonal. In general, the rows of $A^{(d)}$ sum to zero.

Denote by $L = A^{(d)}$ the discrete analog of the Laplacian operator. To use the Matérn operator, one substitutes

$$L = B^{(d)}(m, \epsilon) = ((A^{(d)})^m - \epsilon^2 I).$$

Algorithmic Complexity and Performance

The sparsity of the matrix A means that the algorithmic complexity for both Laplace and Matérn interpolation is $O(M)$. Furthermore, A contains at most 5 nonzero entries per row when $d = 2$ and 7 nonzero entries per row when $d = 3$ and so on, which implies that the constant hidden by the big-O notation is small. The Matérn matrix $B^{(d)}(m, \epsilon)$ is also sparse, having $2(m + d) - 1$ nonzero entries per row.

Performance comparisons

As a synthetic example, we generate a one dimensional realization of data from a Gaussian process (GP) with a Matérn covariance. The function to predict is

$$f(x) = \sin(2\pi x 0.1) + 0.05\zeta_x,$$

where $\zeta_x \sim N(0, 1)$ independently for every integer x. Prediction was done using Matérn covariance with length scale 1.0 and roughness 1.5 which corresponds to $m = 2$ and $\epsilon = \sqrt{3}$, and these data were given for $N_{\text{sample}} = 100$ random points out of $N_{\text{total}} = 1100$. Fig. 1 shows a toy version of this process with $N_{\text{sample}} = 10$ points and $N_{\text{total}} = 20$ in x along with the Laplacian and Matérn interpolations using the true parameters. Note that, by construction, a continuous Matérn operator with the same parameters ought to perfectly reconstruct the data, and the errors we analyze here are due to the finite differencing. Boundary conditions add additional error.

The timing and accuracy of the interpolation is shown in Fig. 1. Benchmarking the timings, one finds the results in panel (b). Note the log scale on the $y-$axis in this figure, mean processing times were $785, 975$, and 9736 μs for the Laplace, Matérn, and GP interpolation, respectively, representing an order of magnitude difference in timing. Furthermore, relative errors, shown in panel (c) had means of $4.2, 3.1$, and 3.9 for the Laplace interpolation, Matérn Interpolation, and the GP prediction at the missing points, respectively. That is, the discrete (Matérn) version of the interpolation gave results with similar accuracy as the GP prediction.

Figure 1: Left panel shows a realization of GP data generated using a 1D Matérn kernel. The data used to create the GP is greyed out, while known points and interpolated points using the three interpolation methods are shown. (Colors are green for truth, brown for Laplace, turqouise for Matern(2, 1.05), and magenta for GP prediction.) The center panel shows the timings of the algorithm, and the right panel shows the accuracy. Parameters are given in text.

A Jupyter notebook is included in the repository to reproduce these performance figures.

Statement of Need

While numerous implementations of interpolation routines exist that fill missing data points on arbitrary grids, these are largely restricted to one and two dimensions and are slow to

run. The implementation we propose is dimension-agnostic, based on a linear-time algorithm, and implements an approximate Matérn kernel interpolation (of which thin plate splines, polyharmonic splines, and radial basis functions are a special case).

Why Is It So Fast?

The implementation is fast because the problem largely boils down to the solution of $Ax = b$ (Mainberger et al., 2011), where the square matrix A's size is the product of the number of points in each of the dimensions and is dense. For the special case where the data points are on a regular grid and the Matérn kernel interpolant is used, a remarkable simplification occurs in which a discrete approximation to the Green's function for the operator results in an interpolant having sparse matrix representation.

Other Software for Interpolation

As of the time of this writing, related software includes the following:

Julia

- Interpolations.jl, which does B-splines and Lanczos interpolation and has support for irregular grids (Lycken et al., 2014)
- Dierckx.jl, a Julia-wrapped Fortran package for 1-D and 2-D splines (Barbary & contributors, 2014)
- GridInterpolations.jl (Kochenderfer & Laboratory, 2016)
- Laplacians.jl, whose function `harmonic_interp` is similar to our vanilla implementation (Spielman & contributors, 2015)

Python

- astropy.convolve, which will interpolate gridded data by rescaling a convoution kernel when it encounters missing values (Astropy Collaboration et al., 2018)
- scipy.interpolate.RBF (Virtanen et al., 2020)

Python Wrapper

- gridinterppy, which serves as a Python wrapper for LaplaceInterpolation.jl (Haley, 2021)

Acknowledgments

The authors would like to thank Ray Osborn, Matt Krogstad, and Stefan Rosenkranz for their input on this work.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, Materials Sciences and Engineering Division, under contract DE-AC02-06CH11357.

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science

laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan

References

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., Shupe, D. L., Craig, M. W., Dencheva, N., Ginsburg, A., Vand erPlas, J. T., Bradley, L. D., Pérez-Suárez, D., de Val-Borro, M., Aldcroft, T. L., Cruz, K. L., Robitaille, T. P., Tollerud, E. J., … Astropy Contributors. (2018). The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. *156*(3), 123. https://doi.org/10.3847/1538-3881/aabc4f

Barbary, K., & contributors. (2014). *kbarbary/Dierckx.jl: a Julia package for 1-d and 2-d splines*. https://github.com/kbarbary/Dierckx.jl

Fasshauer, G. E. (2012). Green's functions: Taking another look at kernel approximation, radial basis functions, and splines. In *Approximation theory XIII: San Antonio 2010* (pp. 37–63). Springer. https://doi.org/10.1007/978-1-4614-0772-0_4

Haley, C. L. (2021). *lootie/gridinterppy: Python wrapper to LaplaceInterpolation.jl*. https://github.com/lootie/gridinterppy

Kochenderfer, M., & Laboratory, S. I. S. (2016). *sisl/GridInterpolations.jl: Multidimensional grid interpolation in arbitrary dimensions*. https://github.com/sisl/GridInterpolations.jl

Lycken, T., T., H., & contributors. (2014). *JuliaMath/Interpolations.jl: Fast, continuous interpolation of discrete datasets in Julia*. https://github.com/JuliaMath/Interpolations.jl

Mainberger, M., Hoffmann, S., Weickert, J., Tang, C. H., Johannsen, D., Neumann, F., & Doerr, B. (2011). Optimising spatial and tonal data for homogeneous diffusion inpainting. *International Conference on Scale Space and Variational Methods in Computer Vision*, 26–37. https://doi.org/10.1007/978-3-642-24785-9_3

Press, W. H. (1992). *Numerical recipes in Fortran 77: The art of scientific computing* (3rd ed.). Cambridge University Press.

Spielman, D., & contributors. (2015). *danspielman/Laplacians.jl: Algorithms inspired by graph Laplacians: linear equation solvers, sparsification, clustering, optimization, etc.* https://github.com/danspielman/Laplacians.jl

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2