A Re-Implementation of Akima's Spline Interpolation for Scattered Data

Albrecht Gebhardt
University Klagenfurt

Roger Bivand
Norwegian School of Economics

Abstract

This vignette presents the R package **interp** and focuses on interpolation of irregular spaced data.

This is the second of planned three vignettes for this package (not yet finished).

Keywords: interpolation, spline, R software.

1. Note

Notice: This is a preliminary and not yet complete version of this vignette. Finally three vignettes will be available for this package:

1. a first one related to partial derivatives estimation,
2. this one describing interpolation related stuff
3. and a third one dealing with triangulations and Voronoi mosaics.

2. Introduction

The main aim of this R package is to provide interpolation algorithms for both regular and irregular data grids

$$\{((x_i, y_i)^\intercal, z_i) | x_i, y_i, z_i \in \mathbb{R} \quad i = 1, \ldots, n\}$$

From the early days of S and S-Plus there was a function `interp()` which solved this task. It used Akima's spline interpolation algorithms available at `netlib`[1] twice: Once to determine a triangulation of the data which is needed for a piecewise linear interpolation. This is the default application case of this function and as shown in Bivand and Gebhardt (2017) the most common use of it, especially in other R packages depending on it. Second to get the spline interpolation based an the same triangulation. These algorithms have been available since 1998 in R via the package `akima`. Unfortunately this package inherits a non-free license from the underlying Fortran code. So the need to rewrite the algorithms under a free license,

[1] https://netlib.org/toms/526.gz

encouraged by the CRAN team, appeared convincing to the authors of this package. This is now mostly done and package `interp` provides plugin capable replacement functions for the interpolations delivered in package **akima**.

For both of these interpolations to work it has to be ensured that no duplicate points (x_i, y_i) may exist in the given point set $\{(x_i, y_i)|i = 1, \ldots, n\}$. This is reached via the argument `duplicate` of `interp::interp()`. It offers three options:

- `"error"`: Stop with an error, this is the default.
- `"strip"`: Completely remove points with duplicates, or
- `"mean"`,`"median"`,`"user"`: apply some function to them. The Implementation provides `mean()`, `median()` or a user supplied function (`"dupfun"`).

3. Bivariate Linear Interpolation

The default behaviour of the `interp::interp()` function is to produce a piecewise linear interpolation. This interpolation takes the triangles of the Delaunay triangulation as also returned by `tri.mesh()` and simply fits a plane to the three vertices $(x_i, y_i, z_i), i = 1, 2, 3$ of those triangles. As a natural consequence it is not possible to extrapolate this interpolation beyond the convex hull of the given point set.

First load the data set used by Akima in his initial work on irregular gridded data (Akima 1978b), see figure 1.

```
> data(akima)
> library(scatterplot3d)
> scatterplot3d(akima, type="h", angle=60, asp=0.2, lab=c(4,4,0))
```

The next plot in figure 2 shows the linear nature of the isolines of the interpolation generated within all triangles:

```
> li <- interp(akima$x, akima$y, akima$z, nx=150, ny=150)
> MASS::eqscplot(akima$x, akima$y)
> contour(li, nlevels=30, add=TRUE)
> plot(tri.mesh(akima), add=TRUE)
```

In case the point data set resembles a regular rectangular grid it should be noted that no unique solution to the triangulation task exists. For each rectangle of this grid there are two possibilities to form triangles compatible with the main condition of a Delaunay triangulation: The interior of the circumcircle of each triangle does not contain any other point of the data set. Generally, as long as the data set contains more then 3 points on a common circumcircle which is otherwise empty of remaining points, it will lead to non uniqueness of the triangulation. This in turn means that a piece wise linear interpolation of rectangular gridded data is not unique. Nevertheless `interp::interp()` will always produce the same

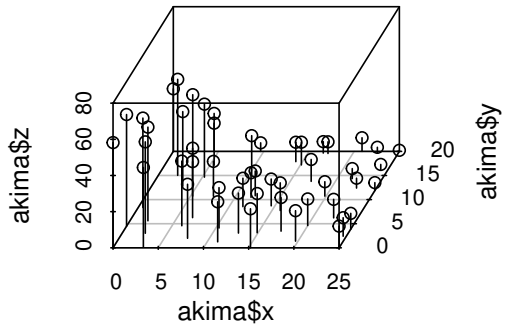

Figure 1: Akimas test data in Akima (1978b)

result as long as no jitter is applied to the data set. This can be done by explicitly via the argument `jitter` or it is applied automatically during the underlying triangulation, which applies this in some cases of collinear points to avoid error conditions.

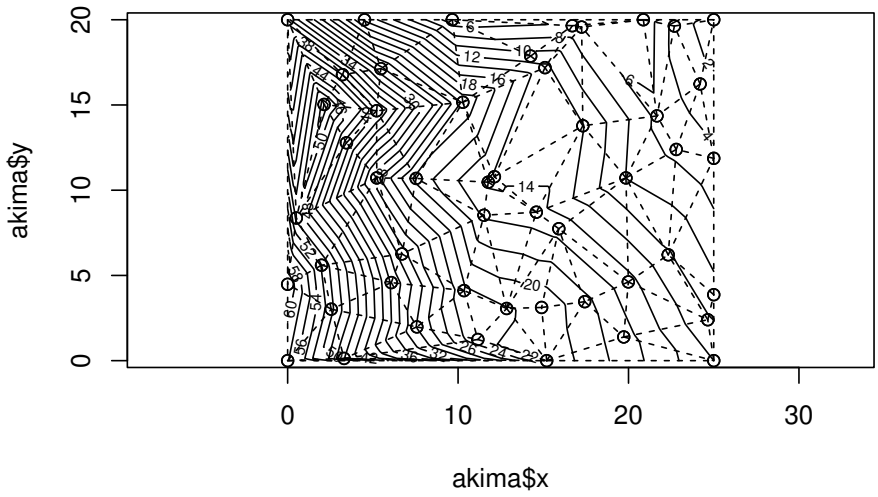

Figure 2: Piecewise linear interpolation

4. Bivariate Spline Interpolation

Akimas spline interpolator 'with the accuracy of a bicubic polynomial' (Akima 1978a) for irregular gridded data is given by the following polynomial in x and y:

$$p(x,y) = \sum_{i=0}^{5} \sum_{j=0}^{5-i} p_{i,j} x^i y^j \tag{1}$$

with 21 coefficients $p_{i,j}$, $0 \leq i \leq j \leq 5$. This polynomial is determined within each triangle (v_1, v_2, v_3) with vertexes $v_i \in \mathbb{R}^2, i = 1, 2, 3$ of the Delaunay triangulation. The solution has to fulfill the following restrictions:

1. The interpolation itself (condition (i) in (Akima 1974)) results in 3 conditions.

2. First and second order partial derivatives of $p(x, y)$ have to match estimated derivatives at the triangle vertices (Akima denotes them as condition (ii)). This makes up for 15 conditions.

3. Finally the last three equations (condition (iii)) involve the directional derivatives along the normal vectors of the triangle sides. As the spline polynomial is of degree 5 these derivatives generally will be polynomials of degree 4. Now the condition demands that they are polynomials of degree 3 in that variable that is describing the position of that normal vector along the triangle side (later denoted as s in a (s, t) coordinate system), thus setting its highest degree coefficient to zero. This can be expressed by setting the appropriate 4th derivative of this directional derivative to zero.

The same conditions are also used in an improved algorithm described in (Akima 1996b), but e.g. the estimation of the partial derivatives is different to the old algorithm and a better triangulation based on the `TRIPACK` Fortran package has been used (Renka 1996).

Next we will formulate the conditions at the triangle vertices $\underline{v_i} = (x_i, y_i)^\intercal, i = 1, 2, 3$ and for the normal vectors $\underline{n}_{ij} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \underline{t}_{ij}$ of the triangle sides $\underline{t}_{ij} = (x_j, y_j)^\intercal - (x_i, y_i)^\intercal \ (i, j) \in \{(1, 3), (3, 2), (2, 1)\}$.

$$
\begin{array}{lrcl}
(i) & p(x_i, y_i) & = & z_i, \quad i = 1, 2, 3 \\
(ii) & \frac{\partial}{\partial x} p(x_i, y_i) & = & z_{x,i}, \quad \frac{\partial}{\partial y} p(x_i, y_i) = z_{y,i}, \quad i = 1, 2, 3 \\
 & \frac{\partial^2}{\partial x \partial y} p(x_i, y_i) & = & z_{xy,i}, \quad \frac{\partial^2}{\partial x^2} p(x_i, y_i) = z_{xx,i}, \quad \frac{\partial^2}{\partial y^2} p(x_i, y_i) = z_{yy,i} \\
(iii) & \frac{\partial^4}{\partial s^4} \underline{n}_{ij} \nabla p(x, y) & = & 0 \quad (i, j) \in \{(1, 3), (3, 2), (2, 1)\}
\end{array}
\tag{2}
$$

where z_i are the values to interpolate in $\underline{v_i} = (x_i, y_i)^\intercal, i = 1, 2, 3$ and $z_{x,i} = \frac{\partial}{\partial x} p(x_i, y_i)$, $z_{y,i} = \frac{\partial}{\partial y} p(x_i, y_i)$, $z_{xx,i} = \frac{\partial^2}{\partial x^2} p(x_i, y_i)$, $z_{xy,i} = \frac{\partial^2}{\partial x \partial y} p(x_i, y_i)$ and $z_{yy,i} = \frac{\partial^2}{\partial^2 y} p(x_i, y_i)$ denote the estimates for partial derivatives at $\underline{v_i}$. Note that the scalar product $\underline{n}_{ij} \nabla p(x, y)$ represents the directional derivative mentioned above expressed in coordinates s and t.

All these conditions together ensure that the resulting spline interpolates the given data and the interpolating function is continuous and differentiable across the borders of all triangles. We now illustrate this with the same data set as above in figure 3.

```
> si <- interp(akima$x, akima$y, akima$z, method="akima", nx=150, ny=150)
> MASS::eqscplot(akima$x, akima$y)
> contour(si, nlevels=30, add=TRUE)
> plot(tri.mesh(akima), add=TRUE)
```

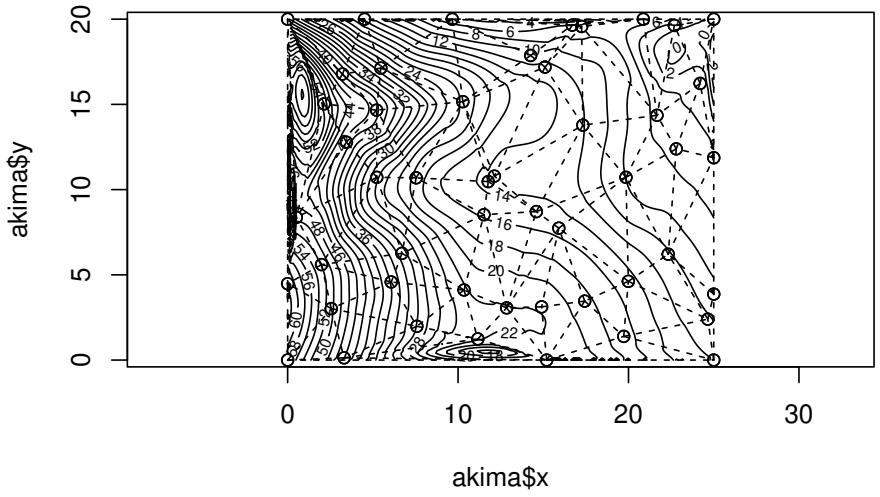

Figure 3: Bivariate Spline Interpolation

5. Implementation details

The call to `interp::interp()` follows this form:

```
interp(x, y = NULL, z, xo = seq(min(x), max(x), length = nx),
       yo = seq(min(y), max(y), length = ny),
       linear = (method == "linear"), extrap = FALSE,
       duplicate = "error", dupfun = NULL,
       nx = 40, ny = 40, input="points", output = "grid",
       method = "linear", deltri = "shull", h=0,
       kernel="gaussian", solver="QR", degree=3,
       baryweight=TRUE, autodegree=FALSE, adtol=0.1,
       smoothpde=FALSE, akimaweight=TRUE, nweight=25)
```

The arguments `duplicate` and `dupfun` have been introduced above, as well as `method` with its currently two available options `"linear"` and `"akima"`.

Generally the input will be given as three vectors `x`, `y` and `z` of equal length. Omitting `y` implicates that `x` consist of a two column matrix or dataframe containing x and y entries. Additionally the argument `input` has to be set to `"points"` (which it is by default). If `input` ↪ `="grid"` is given, `z` is treated as a matrix of z values containing $z_{i,j}$ for the x and y values given in the argument vectors `x` and `y` both of a length matching the dimensions of `z`. A similar scheme is applied to the output: If `output="grid"` is set (default) a matrix with rows and columns according to the output defining vectors `xo` and `yo` is returned. The output grid can also be specified by setting its dimension to `nx` times `ny`, it will then be chosen to cover the range of the input data. With `output="points"` `xo` and `yo` have to be of equal length and only a vector of z values of the same length is returned. Extrapolation (`extrap=TRUE`) is only possible for spline interpolation but is disabled by default. The remaining parameters control several aspects of the algorithm and are at least partially explained later.

Both methods are implemented via the `Rcpp` interface (Eddelbuettel and Balamuta 2018). As mentioned before, step 1 of these interpolation methods is the Delaunay triangulation, described in another vignette (`vignette("tri")`) which is based on the sweep hull algorithm described in (Sinclair 2016). The access to the triangulation code is done internally via C++, not via the R function `interp::tri.mesh()`.

In the second step the needed estimates for the partial derivatives up to degree 2 in all data points are determined. This is based on a local polynomial regression approach implemented in C++. These intermediate results are also available via `interp::locpoly()` described in a separate vignette (`vignette("partDeriv")`). All options of the related `interp::locpoly` ↪ `()` function are also available in `interp::interp()`, e.g. argument `kernel` specifies the kernel used. In contrast to Akima's interpolation we use a gaussian kernel by default and not a uniform one. Argument `h` contains the bandwidth, either as a scaler, or a vector of length 2. The first setting gives a percentage of the data set used for a local nearest neigbour bendwidth approach. If two bandwidths as a vector are given then two global bandwidths for x and y are chosen as the given percentage of their data range. If `h=0` then a minimum local bandwidth resulting in 10 nearest neigbours are choosen to be able to determine the 10 parameters of a `degree=3` polynomial. It is possible to choose different numerical solutions of the weighted least squares method behind the local regression via the argument `solver` (default is `"QR"`, but also `"LLT"`, `"SVD"`, `"Eigen"` and `"CPivQR"` are available) to be used in the local regression step, compare `fastLm()` in (Bates and Eddelbuettel 2013).

The third step performs the real interpolation. First the estimated derivatives are (optionally) smoothed according to the smoothing scheme detailed in (Akima 1978b). Then the system

of equations (2) is solved per triangle and the results are determined via

$$\begin{aligned} p(x,y) = y\ (y\ (y\ (y\ (p_{0,5}\,y + p_{1,4}\,x + p_{0,4}) + x\ (p_{2,3}\,x + p_{1,3}) + p_{0,3}) + x\ (x\ (p_{3,2}\,x + p_{2,2}) + p_{1,2}) \\ + p_{0,2}) + x\ (x\ (x\ (p_{4,1}\,x + p_{3,1}) + p_{2,1}) + p_{1,1}) + p_{0,1}) \\ + x\ (x\ (x\ (x\ (p_{5,0}\,x + p_{4,0}) + p_{3,0}) + p_{2,0}) + p_{1,0}) + p_{0,0} \end{aligned} \tag{3}$$

which is equivalent to (1) but numerically more stable.

Optionally some methods to improve the results can be applied. They are choosen via the following arguments:

- `akimaweight`: As mentioned above, this sort of averaging is also done in Akimas original algorithms. It takes by default 25 (parameter `nweight`) estimates of that specific partial derivative and builds a weighted sum of them with the weights beeing constructed out of normal densities with mean and standard deviations of the according estmation errors.

- `baryweight`: The system of equations (2) is solved after transforming each triangle into a standardized triangle with vertices $(0,0)^\intercal, (1,0)^\intercal, (0,1)^\intercal$. So one of the three vertices of a triangle gets transformed into $(0,0)^\intercal$. During the development of the code it became apperent that the numerical errors for points near to this vertices are minimal and increase for the two other vertices. This weighting scheme repeats the interpolation for all three possibilities to transform a vertex into $(0,0)^\intercal$ and then merges the results using the barycentric coordinates (see 7.1) of the prediction points. That way results generated from a vertex mapped to $(0,0)^\intercal$ always dominate and all three vertices can benefit from the reduced numerical errors near $(0,0)^\intercal$ after transformation. Clearly this triples the computing time. But nevertheless this option is used by default. As motivation a result with barycentric weighting turned off is given below in figure 4.

```
> si.nobw <- interp(akima$x, akima$y, akima$z, method="akima", nx=150, ny=150,
                    baryweight=FALSE)
> MASS::eqscplot(akima$x, akima$y)
> contour(si.nobw, nlevels=30, add=TRUE)
> plot(tri.mesh(akima), add=TRUE)
```

  The plot clearly shows (e.g. in the center of the upper left quadrant) the numerical problems of disconnected isolines across the triangle borders. Note, that these errors occur only on one triangle edge. It turned out this is opposite to the vertex mapped internally by the algorithm to $(0,0)^\intercal$. So we encourage to use this option even dispite the tripled computing time. Only if acurracy does not really matter one could reduce the computing time by turning it off.

- `smoothpde`: If `TRUE` smoothing of partial derivative estimates, if `akimaweight==TRUE` ↪ then Akimas weighting scheme is applied, otherwise a simple arithmetic mean is returned. Note that it is disabled by default which in turn means that also no Akima weighting is applied. If it is enabled then Akima weighting is used by default and a simple arithmetic mean if `akimaweight=FALSE` is given.

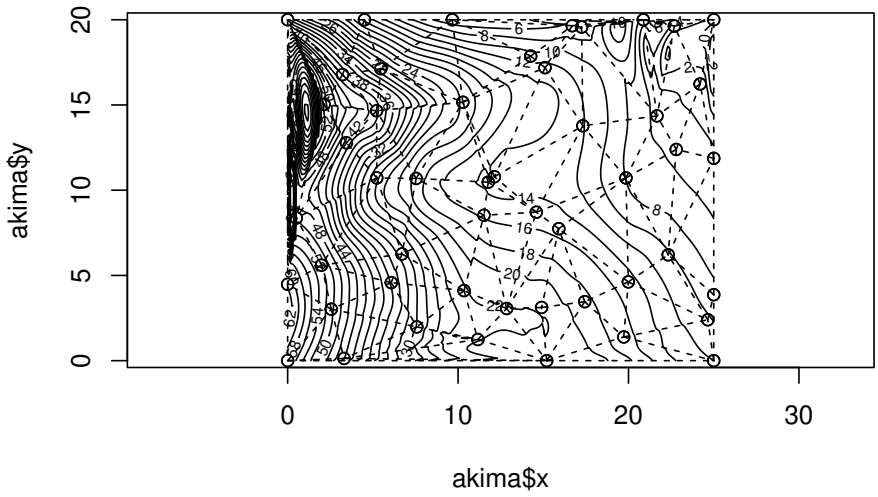

Figure 4: Bivariate Spline Interpolation (Without barycentric weighting)

- `autodegree`: If the variability of the interpolates is above `adtol` then reduce the degree of the polynomial to get a smoother result. This is also disabled by default.

If `interp::interp()` is called with regular gridded data as input, it uses the same irregular grid based algorithm. This is in contrast to the old package `akima`, this also contained Akimas code for regular gridded data, based on (Akima 1974) and (Akima 1996a). Maybe a future version of package `interp` will also contain a re-implementation of this old code.

This package also implements bilinear interpolation for rectangular grids. Given a rectangle $\{(x_1, y_1)^\intercal, (x_2, y_2)^\intercal, (x_3, y_3)^\intercal, (x_4, y_4)^\intercal\}$ and $y_1 = y_2$, $y_3 = y_4$, $x_1 = x_4$ and $x_2 = x_3$ (this makes it axis parallel) with counter clockwise indexed vertexes and according z values z_1, z_2, z_3, z_4, this algorithm can be described as follows: For a location $(x_0, y_0)^\intercal$ contained in this rectangle the interpolation is determined via:

1. Determine intermediate z values for $(x_0, y_1)^\intercal$ and $(x_0, y_3)^\intercal$ as

$$z_{01} = \frac{x_0 - x_1}{x_2 - x_1}(z_1 + z_2) \quad \text{and} \quad z_{03} = \frac{x_0 - x_1}{x_2 - x_1}(z_3 + z_4).$$

2. Now get

$$z_0 = \frac{y_0 - y_1}{y_4 - y_1}(z_{01} + z_{03}).$$

This results in a polynomial of degree 2 which is continuous but not differentiable at the borders of the rectangle.

We use Franke function 1 (Franke 1982) on a regular grid for the demonstration, see figure 5.

```
> nx <- 8; ny <- 8
> xg<-seq(0,1,length=nx)
```

```
> yg<-seq(0,1,length=ny)
> xyg<-expand.grid(xg,yg)
> fg <- outer(xg,yg,function(x,y)franke.fn(x,y,1))
> # not yet implemented this way:
> # bil <- interp(xg,yg,fg,input="grid",output="grid",method="bilinear")
> bil <- bilinear.grid(xg, yg, fg, dx=0.01, dy=0.01)
> MASS::eqscplot(xyg[,1], xyg[,2])
> contour(bil, add=TRUE)
```

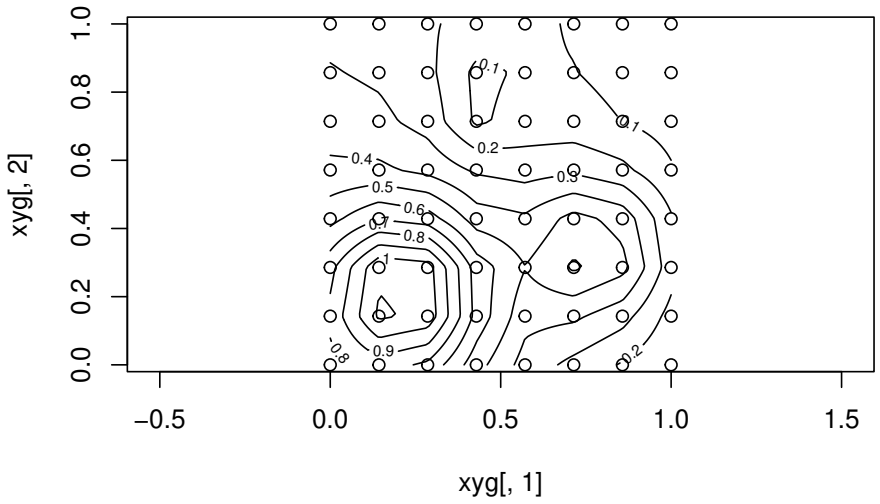

Figure 5: Bilinear interpolation of regularly gridded data

6. One-Dimensional Data

Akima also implemented algorithms for one-dimensional spline interpolation, see (Akima 1972). So it was a natural choice to include these algorithms also in the package **akima**. The functions `aspline()` and `aSpline()` are freely licensed re-implementations of this algorithm in Fortran and C++. It comes in two versions, one as described in (Akima 1972) and an improved version as described in (Akima 1991), the newer algorithm also allows for higher degrees of the polynomial, not only degree 3, compare figure 6

```
> x <- c(-3, -2, -1, 0,  1,  2, 2.5, 3)
> y <- c( 0,  0,  0, 0, -1, -1, 0,   2)
> MASS::eqscplot(x, y, ylim=c(-2, 3))
> lines(aspline(x, y, n=200, method="original"), col="red")
> lines(aspline(x, y, n=200, method="improved"), col="black", lty="dotted")
> lines(aspline(x, y, n=200, method="improved", degree=10), col="green", lty="dashed")
```

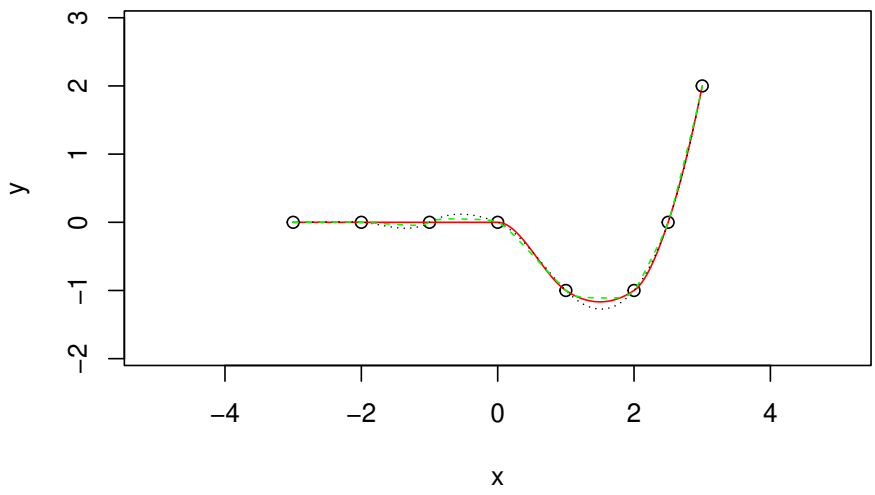

Figure 6: Spline interpolation of onedimensional data

7. Appendix

7.1. Barycentric Coordinates

Points within a triangle can be expressed in barycentric coordinates as follows:

Given a triangle with vertices $\underline{v}_i = (x_i, y_i)^\intercal, i = 1, 2, 3$ any interior point $\underline{v}_0 = (x_0, y_0)^\intercal$ of this triangle can be expressed as a convex linear combination

$$\underline{v}_0 = a \cdot \underline{v}_1 + b \cdot \underline{v}_2 + c \cdot \underline{v}_3$$

with $a, b, c \in [0, 1]$ and $a + b + c = 1$ (notation: $[a : b : c]$). The vertices itself carry the representation $[1 : 0 : 0]$, $[0 : 1 : 0]$ and $[0 : 0 : 1]$.

In section 5 we used these coordinates to build a weighted sum of three interpolation results. Component a of the barycentric coordinates of a point near vertex $\underline{v}_1$ will be close to 1 and so the interpolation result with the lowest numerical error (where vertex $\underline{v}_1$ had been transformed to $(0, 0)^\intercal$) will dominate the barycentric weighted sum mentioned above. Using this approach we cherry pick the numerically best portions of these three interpolation results.

References

Akima H (1972). "Algorithm 433: Interpolation and Smooth Curve Fitting Based on Local Procedures (E2)." *Communications of the ACM*, **15**, 914–918.

Akima H (1974). "Algorithm 474: Bivariate Interpolation and Smooth Surface Fitting Based on Local Procedures [E2]." *Communications of the ACM*, **17**, 26–31.

Akima H (1978a). "ALGORITHM 526: Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points [E1]." *ACM Transactions on Mathematical Software*, **4**, 160–164.

Akima H (1978b). "A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points." *ACM Transactions on Mathematical Software*, **4**, 148–159.

Akima H (1991). "A Method of Univariate Interpolation that Has the Accuracy of a Third-degree Polynomial." *ACM Transactions on Mathematical Software*, **17**, 341–366.

Akima H (1996a). "Algorithm 760: Rectangular-Grid-Data Surface Fitting that Has the Accuracy of a Bicubic Polynomial." *ACM Transactions on Mathematical Software*, **22**, 357–361.

Akima H (1996b). "Algorithm 761: scattered-data surface fitting that has the accuracy of a cubic polynomial." *ACM Transactions on Mathematical Software*, **22**, 362–371.

Bates D, Eddelbuettel D (2013). "Fast and Elegant Numerical Linear Algebra Using the RcppEigen Package." *Journal of Statistical Software*, **52**(5), 1–24. URL http://www.jstatsoft.org/v52/i05/.

Bivand R, Gebhardt A (2017). *Alternatives to the* ***akima*** *package*. URL https://www.user2017.brussels/uploads/bivand_gebhardt_user17_a0.pdf.

Eddelbuettel D, Balamuta JJ (2018). "Extending R with C++: A Brief Introduction to Rcpp." *The American Statistician*, **72**(1), 28–36. doi:10.1080/00031305.2017.1375990. URL https://doi.org/10.1080/00031305.2017.1375990.

Franke R (1982). "Scattered Data Interpolation: Tests of Some Methods." *MATHEMATICS OF COMPUTATION*, **38**, 181–200.

Renka RJ (1996). "Algorithm 751: TRIPACK: A Constrained Two-Dimensional Delaunay Triangulation Package." *ACM Transactions on Mathematical Software*, **22**, 1–8.

Sinclair D (2016). "S-hull: a fast radial sweep-hull routine for Delaunay triangulation." URL https://archive.org/details/arxiv-1604.01428.

List of Figures

Affiliation:

Albrecht Gebhardt Institut für Statistik
Universität Klagenfurt 9020 Klagenfurt, Austria
E-mail: albrecht.gebhardt@aau.at