Alternatives to the **akima** package

Roger Bivand[1] and Albrecht Gebhardt[2]

1. Norwegian School of Economics, 5045 Bergen, Norway
2. Alpen-Adria-Universität, Klagenfurt, Austria

The **akima** package has been widely used as a convenient way of interpolating 2D points to create a surface, but it has a non-commercial ACM license; we show how to use alternatives instead.

akima: the problem

- Many users and packages wish to display an image of a surface based on interpolation from possibly irregularly scattered points.
- Probably because an implementation of Akima's interpolation function (Akima, 1978b) was provided in S-Plus, developers and users familiar with it took up Albrecht Gebhardt's early port to R (on CRAN since 1998).
- Users of S-Plus were shielded from the ACM non-commercial use license because such issues were handled by their software provider.
- Users of the R **akima** (Akima and Gebhardt, 2017) package do however face the license conditions themselves, but have largely regarded the package as a useful adjunct to visualization.

The scale of the problem

- A survey in mid-February 2017 indicated that **akima** functions affected by the ACM license conditions were used by 38 CRAN, 2 Bioconductor and 21 github-only packages (github searched using Google with `akima description site:github.com`, hint by Edzer Pebesma); very few required user acceptance of the ACM license.
- Of these, 43 used `akima::interp(..., linear = TRUE, ...)` with varying settings of the `duplicated=` argument, and 12 used `akima::interp(..., linear = FALSE, ...)` permitting extrapolation.
- The remaining affected **akima** functions were used by 13 packages, where some packages use more than one **akima** function.
- While we are not addressing the equivalent licensing issues of the **tripack** package for trianguation of irregularly spaced data, **akima** and **tripack** are closely related.

2D interpolation and approximation

- Interpolation for irregular scattered data uses ACM algorithm 526 from 1978 (Akima, 1978b,a), argument `linear = TRUE`, and ACM algorithm 761 from 1996 (Akima, 1996), `linear = FALSE`; current **akima** uses a revised version of the ACM 761 code by Renka and Brown (1998) — Renka's triangulation is more efficient ($O(n\log(n))$, Renka, 1998) while Akima's triangulation was $O(n^2)$.
- A discussion of many of the issues involved in smoothing and interpolation is provided by Ripley (1981, pp. 38–44). A continuous function is generated to reproduce the z-values at the given locations (x_i, y_i) and to ensure a certain degree of smoothness of the interpolating function.
- The piecewise linear interpolation with `linear = TRUE` is not a spline interpolator; it is a linear function per triangle, and the first derivatives are not continuous at the borders of the triangle (see Figure 1). This approach, barycentric interpolation, has been known for almost 200 years (Möbius, 1827).

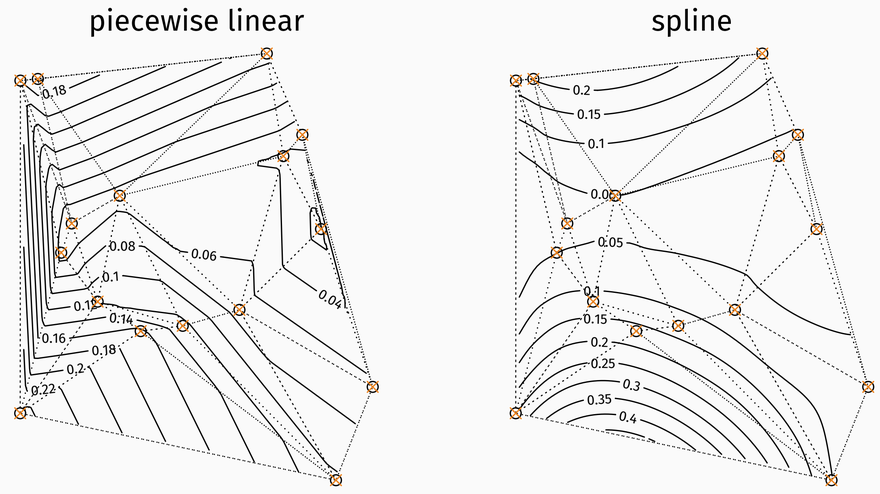

Figure 1: Piecewise linear (left, `linear = TRUE`) and irregular Akima spline interpolation (right, `linear = FALSE`); note the triangular surfaces and linear isolines for the piecewise linear case.

Alternatives and results

- The new **interp** CRAN package is a re-implementation of Akima's algorithms not using the questioned Fortran ACM code, nor using Renka's TRIPACK code. It uses new sweep-hull C++ triangulation code ($O(n\log(n))$, Sinclair, 2016) with **RcppEigen** glue (Bates and Eddelbuettel, 2013), and incidentally will also be a drop-in replacement for much of **tripack** (Renka and Gebhardt, 2016).
- For the `linear = TRUE` piecewise linear interpolator, **interp** can replace **akima**; for `linear = FALSE` and extrapolation, you may use the **MBA** approximation, or watch this space for an **interp** implementation (under development).

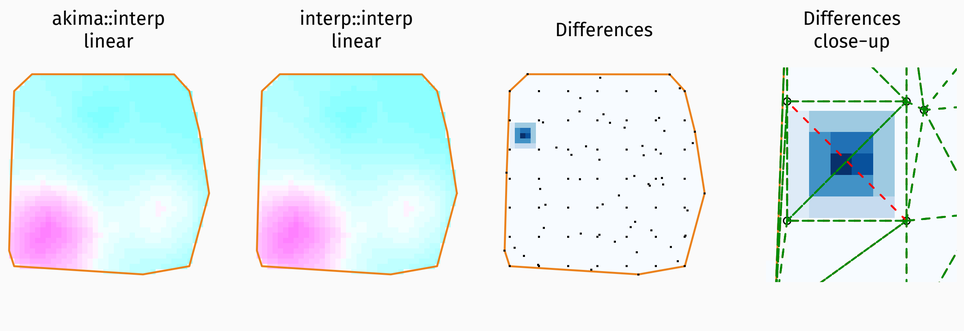

Figure 2: Interpolation output on a 40x40 grid for **akima** and **interp** `linear = TRUE` piecewise linear interpolators; the fourth panel shows that the difference in interpolated surface values is caused by the different choice of diagonal (tripack: green, sweep-hull: red).

- The **MBA** package (Finley et al., 2017) uses an implementation by Øyvind Hjelle (2001) of multilevel B-splines as described by Lee et al. (1997). The `mba.surf` function is used to fit a multilevel approximate B-spline model from the scattered input points, and then to predict to grid points.
- We can use `akima::interpp()` and `MBA::mba.points()` to demonstrate the difference between interpolation and approximation.

Table 1: Interpolation and approximation: distributions of differences between input `z` values and Akima spline, Akima piecewise linear, Akima spline with extrapolation and multivariate B-spline point output.

	Min.	1st Qu.	Median	3rd Qu.	Max.	NA's
spline	-3.735E-12	-8.195E-13	1.592E-14	6.496E-13	3.856E-12	6
linear	-5.196E-12	-8.262E-13	-3.959E-14	6.980E-13	3.764E-11	6
extrap	-3.735E-12	-6.436E-13	5.168E-14	6.451E-13	3.856E-12	0
mba	-9.888E-04	-1.847E-08	-6.318E-09	2.267E-09	9.318E-04	0

- `akima::interp()` is an interpolator, with possible use for extrapolation when `linear = FALSE`, where the fitted surface passes within machine precision of the `z` values at the input points, apart from six NAs on convex hull points.
- In contrast, the multilevel B-spline implementation only approximates the `z` values at the input points. However, multilevel B-spline approximation scales well as the number of points increases, because it does not need to triangulate the input points, and accepts gridded input.