The smint package

Computing details

2015-06-10

Yves Deville

Yves Deville Ingénieur-conseil. Siret 424 966 091 00011
37 rue Lamartine 73000 Chambéry
Maill. deville.yves@alpestat.com Tél. 04 79 96 95 50

Contents

Chapter 1

Grid interpolation

1.1 Linear interpolation and interpolation factors

Consider the interpolation problem with n distinct nodes $\mathbf{x}_i \in \mathbb{R}^d$. Recall that for a *linear* interpolation method we can find a *cardinal* basis ψ_j for $j = 1, 2, \dots, n$ such that $\psi_j(\mathbf{x}_i) = \delta_{i,j}$. In other words, the Gramian matrix $[\psi_j(\mathbf{x}_i)]_{i,j}$ boils down to the identity matrix. The interpolation operator P (Cheney W. and Light W. 2009, chap. 2) can then be written as

$$P.f(x) = \sum_i f(\mathbf{x}_i)\, \psi_i(x).$$

If n^{new} locations $x_i^{\text{new}} \in \mathbb{R}^d$ are given, a matrix of *interpolation factors* Hébert (2013) can be defined as

$$\mathbf{H} = \begin{bmatrix} \psi_1(\mathbf{x}_1^{\text{new}}) & \psi_2(\mathbf{x}_1^{\text{new}}) & \dots & \psi_n(\mathbf{x}_1^{\text{new}}) \\ \psi_1(\mathbf{x}_2^{\text{new}}) & \psi_2(\mathbf{x}_2^{\text{new}}) & \dots & \psi_n(\mathbf{x}_2^{\text{new}}) \\ \vdots & & & \\ \psi_1(\mathbf{x}_{n^{\text{new}}}^{\text{new}}) & \psi_2(\mathbf{x}_{n^{\text{new}}}^{\text{new}}) & \dots & \psi_n(\mathbf{x}_{n^{\text{new}}}^{\text{new}}) \end{bmatrix},$$

with n^{new} rows and n columns. The case where $n^{\text{new}} = 1$ will often be used in tensor product interpolation. This matrix $\mathbf{H}$ can be used to obtain the vector $\widetilde{\mathbf{f}}^{\text{new}}$ of the n^{new} interpolated values (with length n^{new}) by

$$\widetilde{\mathbf{f}}^{\text{new}} = \mathbf{H}\mathbf{f}$$

where $\mathbf{f}$ is the vector of length n containing the values $f(\mathbf{x}_i)$.

The j-th column of the matrix $\mathbf{H}$ can be obtained by using as response vector $\mathbf{f}$ the j-th column of the identity matrix $\mathbf{I}_n$, and then by computing the interpolated value at each new value $\mathbf{x}_i^{\text{new}}$.

When $d = 1$, we can as well define a matrix $\mathbf{H}^{(r)}$ of derivated interpolation factors at order r by simply replacing each basis function ψ_j by its derivative $\psi_j^{(r)}$. Of course, this is possible only when the derivative exists for each basis function. With obvious notations, then $\{\widetilde{\mathbf{f}}^{\text{new}}\}^{(r)} = \mathbf{H}^{(r)}\mathbf{f}$. Note that the true derivative is generally not interpolated at the nodes: the derivatives values obtained there can be considered as estimation of the derivatives.

1.2 Cubic spline interpolation

Determination of the spline

We consider n distinct nodes $x_1 < x_2 < \cdots < x_n$ and the corresponding values $f_i := f(x_i)$. Recall that the natural cubic spline is the only spline of order 4 (degree 3) which interpolates the values f_i with the two border conditions

$$f^{(2)}(x_1) = 0, \qquad f^{(2)}(x_n) = 0.$$

The general cubic spline f with knots x_i has $n + 2$ degree of freedom and 2 conditions must be given to uniquely determine f. Other conditions are sometimes preferred to the natural spline (de Boor 2001, chap. IV). Up to changes in the notations, the determination of the natural interpolatin spline described here can be found in a number of books including de Boor's, Lange (2010, chap. 10) or Press W.H. and Teukolsky S.A. and Vetterling W.T. and Flannery B.P. (1993, chap. 3) among many others.

Let h_i the forward difference $h_i := x_{i+1} - x_i$ for $i = 1, 2, \ldots, n-1$. The natural cubic spline is obtained by solving a tridiagonal system $\mathbf{c}$ with elements second order derivative $c_i := f^{(2)}(x_i)$ at inner nodes x_i for $i = 2, 3, \ldots, n-1$. The system writes as

$$\mathbf{A}\mathbf{c} = \mathbf{B}\mathbf{f} \tag{1.1}$$

where $\mathbf{A}$ is a $(n-2) \times (n-2)$ tridiagonal matrix

$$\mathbf{A} = \frac{1}{6}\begin{bmatrix} 2\left[h_1 + h_2\right] & h_2 & & & \\ h_2 & 2\left[h_2 + h_3\right] & h_3 & & \\ & & & & \\ & & & & h_{n-2} \\ & & & h_{n-2} & 2\left[h_{n-2} + h_{n-1}\right] \end{bmatrix}$$

and $\mathbf{B}$ is a $(n-2) \times n$ matrix

$$\mathbf{B} = \begin{bmatrix} h_1^{-1} & -\left[h_1^{-1} + h_2^{-1}\right] & h_2^{-1} & & & \\ & h_2^{-1} & -\left[h_2^{-1} + h_3^{-1}\right] & h_3^{-1} & & \\ & & & & & \\ & & & h_{n-2}^{-1} & -\left[h_{n-2}^{-1} + h_{n-1}^{-1}\right] & h_{n-1}^{-1} \end{bmatrix}.$$

Since $\mathbf{A}$ is tridiagonal symmetric and strictly diagonal dominant, a system $\mathbf{A}\mathbf{c} = \mathbf{b}$ can be solved by Gauss elimination without pivoting, thus in $\mathcal{O}(n)$ operations.

Spline values

Here we consider a single new value x^{new} with $x_1 \leqslant x^{\text{new}} \leqslant x_n$ and compute the interpolated value, see Press W.H. and Teukolsky S.A. and Vetterling W.T. and Flannery B.P. (1993, chap. 3). The formulas are easily stated by integrating the second order derivative, taking into account the interpolation conditions.

The value of the natural spline is given by

$$f(x^{\text{new}}) = \Delta_{\mathtt{R}} f_i + \Delta_{\mathtt{L}} f_{i+1} + \left\{ f_i^{(2)}\left[\Delta_{\mathtt{R}}^3 - \Delta_{\mathtt{R}}\right] + f_{i+1}^{(2)}\left[\Delta_{\mathtt{L}}^3 - \Delta_{\mathtt{L}}\right] \right\} \frac{h_i^2}{6}$$

where the index i is such that $x_i \leqslant x \leqslant x_{i+1}$ and

$$\Delta_{\mathtt{L}} := \frac{x^{\text{new}} - x_i}{x_{i+1} - x_i}, \qquad \Delta_{\mathtt{R}} := \frac{x_{i+1} - x^{\text{new}}}{x_{i+1} - x_i}.$$

Similarly the first order derivative is given by

$$f'(x^{\text{new}}) = \frac{f_{i+1} - f_i}{h_i} + \left\{ f_i^{(2)} \left[-3\Delta_{\mathtt{R}}^2 + 1\right] + f_{i+1}^{(2)} \left[3\Delta_{\mathtt{L}}^2 - 1\right] \right\} \frac{h_i}{6},$$

while the second order derivative simply results from a linear Lagrange interpolation

$$f^{(2)}(x^{\text{new}}) = f_i^{(2)} \Delta_{\mathtt{R}} + f_{i+1}^{(2)} \Delta_{\mathtt{L}}.$$

Note that extrapolation formulas could be given as well for $x^{\text{new}} < x_1$ or $x^{\text{new}} > x_n$.

Interpolation factors

The interpolation factors matrix for $\mathbf{x}^{\text{new}}$ is obtained by solving the n linear systems (1.1) of size $n-2$ with $\mathbf{f}$ taken as the n columns of the identity matrix $\mathbf{I}_n$. The solution $\mathbf{c}$ can be completed by zeros to give the vector $\mathbf{f}^{(2)}$ of the second order derivatives

$$\mathbf{f}^{(2)} = [0,\, c_2,\, c_3\, \dots,\, c_{n-2},\, 0]^\top$$

which is required for the evaluation at x^{new}. The n vectors $\mathbf{f}^{(2)}$ are the columns of the $n \times n$ matrix

$$\begin{bmatrix} \mathbf{0}^\top \\ \mathbf{A}^{-1}\mathbf{B} \\ \mathbf{0}^\top \end{bmatrix}$$

where $\mathbf{0}$ stand for a zero vector of length n.

Minimal number of points

Some codes dedicated to the natural interpolation spline impose $n \geqslant 4$, and so does the package **splines**. Yet $n = 3$ is possible.

Chapter 2

Grid interpolation: R and C functions

2.1 General grid interpolation

2.1.1 General 1D interpolation function

The grid interpolation is an interpolation method for grid data which relies on an arbitrary one-dimensional interpolation function. This function will be provided as a R function having the following signature

```
interpFun1d <- function(x, y, xout)
```

where `x` and `y` are numeric vectors with the same length, and `xout` is a numeric vector. This function returns a vector of the same length as `xout`. This R function will be called from within a C function for speed considerations.

We will see that the vectorisation w.r.t. the argument `xout` will not be used from within the C function, so we can nearly assume that `xout` is a scalar i.e. a vector with length 1. The vectorisation can be used only during a preliminary step as explained later.

2.1.2 Vectors and arrays in R

We assume in that section that array indices are in C-style i.e. that they begin at zero. The number of nodes for dimensions 1 to d are thus n_0, n_2, …, n_{d-1}.

Define

$$N_j := \prod_{i=0}^{j-1} n_i \qquad 0 \leqslant j \leqslant d$$

where the empty product for $j = 0$ is by convention $N_0 = 1$. The total number of nodes is thus N_d.

In R, a numeric array $\widetilde{\mathbf{F}}$ is simply a numeric "atomic" vector with a *dimension* attribute. The dimension allows the use of a multi-index $[i_0,\, i_1,\, \dots,\, i_{d-1}]$ with

$$0 \leqslant i_0 < n_0, \quad 0 \leqslant i_1 < n_2, \quad \dots \quad 0 \leqslant i_{d-1} < n_{d-1}.$$

Yet the standard vector indexation can be used as well, so there is a correspondance

$$\widetilde{F}[i_0, i_1, \dots, i_{d-1}] \quad \leftrightarrow \quad \widetilde{F}[i].$$

The index i in the vector representation and the multi-index $[i_0, i_1, \dots, i_{d-1}]$ in the array representation are such that

$$i = N_0 i_0 + N_1 i_1 + \cdots + N_{d-1} i_{d-1}$$

Given a vector index i with $0 \leqslant i < N_d$, the corresponding value of i_j can be found using an integer division and a modulo operation

$$i_j = i / N_j \pmod{n_j}$$

for $j = 0, 1, \dots, d-1$.

Remark 1. Although these operators are not used here, recall that the integer division operator in R is `%/%`, while the modulo operator simply writes `%/%`.

2.1.3 Algorithm

Algorithm 1 Grid interpolation. The array $\widetilde{\mathbf{F}}$ initially has dimension $n_0 \times n_1 \times \cdots \times n_{d-1}$. At each of the $d-1$ steps of the j-loop, the array "morally" loses its last dimension, i.e. it is "flattened". For $j = 0$ the array is simply a scalar (with length 1).

```
1:  # Initialisation
2:  $\widetilde{\mathbf{F}} \leftarrow \mathbf{F}$
3:  for ($j = d-1, d-2, \dots, 0$) do
4:      $\mathbf{x}^\star \leftarrow$ xLevels[[$j$]]
5:      $x^{\text{new}} \leftarrow \mathbf{x}^{\text{new}}[j]$
6:      for ($i = 0, 1, \dots N_j - 1$) do
7:          # Fill the vector $\mathbf{f}$ of length $n_j$
8:          for ($\ell = 0, 1, \dots n_j - 1$) do
9:              $f[\ell] \leftarrow \widetilde{F}[i + N_j\, \ell]$
10:         end for
11:         # Compute $g$, a scalar
12:         $g \leftarrow$ interpFun1d(x = $\mathbf{x}^\star$, y = $\mathbf{f}$, xout = $x^{\text{new}}$)
13:         # $\widetilde{\mathbf{F}}$ will now have dimension $n_0 \times n_1 \times \cdots \times n_{j-1}$
14:         $\widetilde{F}[i] \leftarrow g$
15:     end for
16: end for
17: return $f^{\text{new}} \leftarrow \widetilde{\mathbf{F}}$
```

The algorithm 1 describes the general grid interpolation. The loop on i begining on line 6 may be thought of as a loop for indices $i_0, i_2, \dots, i_{j-1}$, while the loop on ℓ begining on line 8 is a loop over i_j.

Note that the number of calls to `interpFun1d` is

$$(n_{d-2} \times n_{d-3} \times \cdots \times n_0) + (n_{d-3} \times n_{d-4} \times \cdots \times n_0) + \cdots + (n_1 \times n_0) + n_0$$

which will generally be *very large* for $d > 5$. Fortunately, each interpolation involves a vector of length $\leqslant \max_{0 \leqslant i < d} n_i$, which will usually be small in practice. Depending on the cost of the interpolation, the ordering of the dimensions can have a non-negligible impact on the total computation time. The number of calls to `interFun` will be smaller if the numbers n_i are in decreasing order, since we then get rid sooner of the largest n_i.

2.1.4 R implementation

Several adaptations can be done. Firstly, with the interface `.Call` we have to make a copy $\widetilde{\mathbf{F}}$ of the original array $\mathbf{F}$ since this is passed as an input argument. Secondly, in practice we may have several vectors $\mathbf{x}^{\text{new}}$ provided as the n^{new} rows of a matrix $\mathbf{X}^{\text{new}}$ having d columns. Obviously it will be enough to within a loop for k (say) running from 0 to $n^{\text{new}} - 1$.

Although the interpolation function could allow `xout` to be a numeric vector and take advantage of this, it not possible to use such a vectorised call throughout the loop on j because the vector $\mathbf{f}$ passed to the `y` argument depends on the output index k, excepted for the first interpolation (for $j = d-1$), since we use then a “fresh” array $\widetilde{\mathbf{F}}$ equal to $\mathbf{F}$. A possibility is thus to vectorise on `xout` for the first interpolation only by simply using `apply` in R before calling the C function

```
Fout <- apply(Fout, MARGIN = 1L:(d - 1L), FUN = interpFun1d,
              x = xLevels[[d]], xout = Xnew[ , d])
```

As a result, the array object `Fout` will loose one “interpolation” dimension, and gain one “output” first dimension

before		$n_0 \times n_1 \times \cdots \times n_{d-2}$	$\times n_{d-1}$
after	$n^{\text{new}} \times$	$n_0 \times n_1 \times \cdots \times n_{d-2}$	

Note that for $d = 1$, no call to a C function is required, but no gain can be expected from using a C function in that case since the function `interpFun1d` can be expected to be efficiently vetorised w.r.t. `xout`.

2.2 Linear grid interpolation

2.2.1 Context

Let us now assume that the interpolation method is linear, meaning that the interpolated value is linear w.r.t. the vector $\mathbf{f}$ of n known values.

$$\widetilde{f}^{\text{new}} = \mathbf{h}(\mathbf{x}^{\text{new}})\, \mathbf{f}$$

where $\mathbf{h}(\mathbf{x}^{\text{new}})$ is a row vector the elements of which are the interpolation factors, i.e. the values at $\mathbf{x}^{\text{new}}$ of the Cardinal Basis functions.

We will assume that the interpolation factors are provided as a R function having the following signature

```
cardinalBasis1d <- function(x, xout)
```

where `x` is a numeric vector and `xout` is a numeric vector. This function returns a matrix with `length(xout)` rows and `length(x)` columns, i.e. with n^{new} rows and n columns in the math notations. This R function will be called from within a C function for speed considerations.

If $n^{\text{new}} > 1$ interpolated values have to be computed instead of one, we can proceed as follows.

- We can wrap the whole algorithm 2 within a loop for $k = 0, 1, \ldots, n^{\text{new}} - 1$. The step k will compute $f^{\text{new}}[k]$. The drawback of this approach is that we will have to compute n^{new} vectors in

  `cardinalBasis1d(x =` $\mathbf{x}^\star$`, xout =` $x^{\text{new}}[k]$`)`

  for $k = 0$ to $k = n^{\text{new}} - 1$. These computations having much in common could gain to be done simultaneously, at least when n^{new} is large.

- Therefore, a variant would consist in computing a list of d matrices $\mathbf{H}(\mathbf{x}_j^{\text{new}})$ for the dimension indices $j = 0, 1, \ldots\ d - 1$. Each of these matrices has n^{new} rows and a varying number of columns: $n_0, n_1, \ldots, n_{d-1}$ equal to the number of nodes.

Finally, a possible improvement concerns the case the interpolation method is *local*, meaning the the interpolated value at x^{new} depends only of the function values at neighbouring abscissas. The loop on ℓ begining at line 10 in the algorithm 2 could be made shorted because we can know that $h^{\text{new}}[\ell]$ is zero except when $x^\star[\ell]$ is a neighbour of x^{new}. However, the method seems very fast without these improvements.

2.2.2 Algorithm

The algorithm 2 describles the general grid interpolation. Note that the number of calls to `cardinalBasis1d` is d, making this method considerably faster than the general grid interpolation.

2.2.3 R implementation

The R implementation is very similar to that of the general grid interpolation. No preliminary `apply` step seems useful, and the code can be kept very close to the pseudo-code.

Algorithm 2 Linear Grid interpolation using **Cardinal Basis** function for $n^{\text{new}} = 1$. This is a slight modification of algorithm 1.

```
1: # Initialisation
2: $\widetilde{\mathbf{F}} \leftarrow \mathbf{F}$
3: for $(j = d-1,\, d-2,\, \dots,\, 0)$ do
4:     $\mathbf{x}^\star \leftarrow$ xLevels[[$j$]]
5:     $x^{\text{new}} \leftarrow \mathbf{x}^{\text{new}}[j]$
6:     # Compute $\mathbf{h}^{new}$, a row vector of length $n_j$
7:     $\mathbf{h}^{\text{new}} \leftarrow$ cardinalBasis1d(x = $\mathbf{x}^\star$, xout = $x^{\text{new}}$)
8:     for $(i = 0,\, 1,\, \dots\; N_j - 1)$ do
9:         $g \leftarrow 0$
10:        for $(\ell = 0,\, 1,\, \dots\; n_j - 1)$ do
11:            $g \leftarrow g + h^{\text{new}}[\ell] \times \widetilde{F}[i + N_j\, \ell]$
12:        end for
13:        # $\widetilde{\mathbf{F}}$ will now have dimension $n_0 \times n_1 \times \cdots \times n_{j-1}$
14:        $\widetilde{F}[i] \leftarrow g$
15:    end for
16: end for
17: return $f^{\text{new}} \leftarrow \widetilde{\mathbf{F}}$
```

Bibliography

Cheney W and Light W (2009). *A course in approximation theory.* Providence, RI: American Mathematical Society (AMS). ISBN 978-0-8218-4798-5/hbk.

de Boor C (2001). *A Practical Guide to Splines, Revised Rdition.* Springer-Verlag.

Hébert A (2013). "Revisiting the Ceschino Interpolation Method." `http://www.intechopen.com/download/pdf/21941`.

Lange K (2010). *Numerical Analysis for Statisticians.* 2nd edition. Springer-Verlag.

Press WH and Teukolsky SA and Vetterling WT and Flannery BP (1993). *Numerical Recipes in FORTRAN; The Art of Scientific Computing.* 2nd edition. Cambridge University Press, New York, NY, USA. ISBN 0521437164.