BSPLINES Reference Guide

Trach-Minh Tran, Stephan Brunner, Kurt Appert v0.3, February 2012

Generalized splines of any order on irregular grids for interpolation and solving PDEs with FEM.

Contents

1 Properties of Splines

In this section, several properties of splines will be shown in a more or less rigorous way. The aim is mainly to provide a minimum mathematical background for using the module `BSPLINES` for the *interpolation* problem as well as the *Finite Element Method* to solve PDEs. More rigorous mathematical proofs can be found in the book by de Boor [1].

1.1 Recurrence Relation

We start by defining a finite interval $[a, b]$ subdivided into N_x intervals:

$$a = t_0 \leq t_1 \leq \ldots \leq t_{N_x} = b. \tag{1}$$

The sequence $t_i, i = 0, \ldots, N_x$ can be irregularly spaced. The j^{th} spline of degree p defined on this sequence of grid points (also called **knots**), is denoted by Λ^p_j and can be constructed using the following recurrence relation. Starting with the *constant* spline

$$\Lambda^0_i(x) = \begin{cases} 1 & \text{if } t_i \leq x < t_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

the splines of degree $p > 0$ for $t_i \leq x < t_{i+1}$ can be constructed from

$$\Lambda^p_i = w^p_i \Lambda^{p-1}_i + (1 - w^p_{i+1})\Lambda^{p-1}_{i+1}, \tag{3}$$

$$w^p_i = \frac{x - t_i}{t_{i+p} - t_i}. \tag{4}$$

Thus the values of all *non-zero* splines up to degree p in the interval $[t_i, t_{i+1}]$ fit into the triangular array as shown in Fig. 1. Starting from the first column with $\Lambda^0_i = 1$, one can compute each of the $p + 1$ entries in a subsequent column with Eq. (3). Applying this procedure to generate splines on every intervals $[t_i, t_{i+1}], i = 0, \ldots, N_x - 1$ would produce the sequence of $N_x + p$ splines of degree p: $\Lambda^p_{-p}, \ldots, \Lambda^p_{N_x-1}$.

1.2 Support and positivity

The linear spline

$$\Lambda^1_i = w^1_i \Lambda^0_i + (1 - w^1_{i+1})\Lambda^0_{i+1} = \frac{x - t_i}{t_{i+1} - t_i}\Lambda^0_i + \frac{t_{i+2} - x}{t_{i+2} - t_{i+1}}\Lambda^0_{i+1}$$

consists of 2 *linear pieces* on $[t_i, t_{i+2}]$, forming a C^0 function which breaks at t_{i+1} and vanishes outside of this interval. Likewise, the quadratic spline

$$\begin{aligned} \Lambda^2_i &= w^2_i \Lambda^1_i + (1 - w^2_{i+1})\Lambda^1_{i+1} \\ &= w^2_i w^1_i \Lambda^0_i + [w^2_i(1 - w^1_{i+1}) + w^1_{i+1}(1 - w^2_{i+1})]\Lambda^0_{i+1} + (1 - w^2_{i+1})(1 - w^1_{i+2})\Lambda^0_{i+2} \end{aligned}$$

consists of 3 *parabolic pieces* on $[t_i, t_{i+3}]$ that join to form a C^1 function which breaks at t_{i+1} and t_{i+2} and vanishes outside of this interval. In general the spline of degree p can be expressed as:

$$\Lambda^p_i = \sum_{r=0}^{p} b^p_{i+r} \Lambda^0_{i+r} \tag{5}$$

$$
\begin{array}{ccccccc}
 & & & & & & 0 \\
 & & & & & 0 & \\
 & & & & \cdot & & \Lambda^p_{i-p} \\
 & & & 0 & & \Lambda^{p-1}_{i-p+1} & \\
 & & 0 & & \cdot & & \Lambda^p_{i-p+1} \\
 & 0 & & \Lambda^2_{i-2} & & \Lambda^{p-1}_{i-p+2} & \\
 & & \Lambda^1_{i-1} & & \cdot & & \Lambda^p_{i-p+2} \\
 & \Lambda^0_i & & \Lambda^2_{i-1} & & \cdot & \\
 & & \Lambda^1_i & & \cdot & & \cdot \\
 & 0 & & \Lambda^2_i & & \Lambda^{p-1}_{i-1} & \\
 & & 0 & & \cdot & & \Lambda^p_{i-1} \\
 & & & 0 & & \Lambda^{p-1}_i & \\
 & & & & \cdot & & \Lambda^p_i \\
 & & & & & 0 & \\
 & & & & & & 0
\end{array}
$$

Figure 1: The array of all the splines of degree up to p that are non-zero in $[t_i, t_{i+1}]$.

where b^p_{i+r} is a sum of products of p linear functions, resulting in $p+1$ polynomials of degree p, joining to form a C^{p-1} function which breaks at $t_i, \ldots, t_{i+p+1}$ and vanishes outside of the *support* $[t_i, t_{i+p+1}]$. From its construction, Λ^p_i is clearly *strictly positive* on the interior of $[t_i, t_{i+p+1}]$.

$$\Lambda^p_i(x) > 0, \qquad t_i < x < t_{i+p+1}. \tag{6}$$

1.3 Sum of Splines

For $t_i \le x < t_{i+1}$

$$\begin{aligned}
\sum_j \Lambda^0_j &= \Lambda^0_i = 1, \\
\sum_j \Lambda^1_j &= \Lambda^1_{i-1} + \Lambda^1_i = (1 - w^1_i)\Lambda^0_i + w^1_i \Lambda^0_i = 1.
\end{aligned}$$

Thus assuming that for $p > 1$:

$$\sum_{j=i-p+1}^{i} \Lambda^{p-1}_j = 1,$$

or that the sum of the next to last column in Fig. 1 is 1, we have, using the recurrence relation (3)

$$\begin{aligned}
\sum_{j=i-p}^{i} \Lambda^p_j &= \sum_{j=i-p}^{i} \left(w^p_j \Lambda^{p-1}_j + (1 - w^p_{j+1})\Lambda^{p-1}_{j+1} \right) \\
&= \sum_{j=i-p+1}^{i} w^p_j \Lambda^{p-1}_j + \sum_{j=i-p+1}^{i} (1 - w^p_j)\Lambda^{p-1}_j \\
&= \sum_{j=i-p+1}^{i} \Lambda^{p-1}_j = 1.
\end{aligned}$$

1.4 Derivative of Splines

The derivative of the splines of degree p can be expressed in terms of the splines of degree $p-1$ by the following relation:

$$\frac{d}{dx}\Lambda_i^p = p\left(\frac{\Lambda_i^{p-1}}{t_{i+p}-t_i} - \frac{\Lambda_{i+1}^{p-1}}{t_{i+p+1}-t_{i+1}}\right). \tag{7}$$

A straightforward consequence of this relation is that the splines of order p are C^{p-1} continuous. The demonstration of Eq.(7) is done by induction. One starts with the case $p=1$:

$$\begin{aligned}\frac{d}{dx}\Lambda_i^1 &= \frac{d}{dx}\left[w_i^1\Lambda_i^0 + (1-w_{i+1}^1)\Lambda_{i+1}^0\right]\\ &= \frac{d\,w_i^1}{dx}\Lambda_i^0 + \frac{d\,(1-w_{i+1}^1)}{dx}\Lambda_{i+1}^0 + w_i^1\frac{d\,\Lambda_i^0}{dx} + (1-w_{i+1}^1)\frac{d\,\Lambda_{i+1}^0}{dx}\\ &= \frac{1}{t_{i+1}-t_i}\Lambda_i^0 - \frac{1}{t_{i+2}-t_{i+1}}\Lambda_{i+1}^0,\end{aligned}$$

having used Eq.(3) and $d\,\Lambda_i^0/dx = 0$. One then assumes Eq.(7) true for $p-1$ and demonstrates that it remains true for p. This is done as follows:

$$\begin{aligned}\frac{d}{dx}\Lambda_i^p &= \frac{d}{dx}\left[w_i^p\Lambda_i^{p-1} + (1-w_{i+1}^p)\Lambda_{i+1}^{p-1}\right] &(8)\\ &= \frac{d\,w_i^p}{dx}\Lambda_i^{p-1} + \frac{d\,(1-w_{i+1}^p)}{dx}\Lambda_{i+1}^{p-1} + w_i^p\frac{d\,\Lambda_i^{p-1}}{dx} + (1-w_{i+1}^p)\frac{d\,\Lambda_{i+1}^{p-1}}{dx}\\ &= \frac{\Lambda_i^{p-1}}{t_{i+p}-t_i} - \frac{\Lambda_{i+1}^{p-1}}{t_{i+p+1}-t_{i+1}}\\ &\quad + w_i^p(p-1)\left(\frac{\Lambda_i^{p-2}}{t_{i+p-1}-t_i} - \frac{\Lambda_{i+1}^{p-2}}{t_{i+p}-t_{i+1}}\right) + (1-w_{i+1}^p)(p-1)\left(\frac{\Lambda_{i+1}^{p-2}}{t_{i+p}-t_{i+1}} - \frac{\Lambda_{i+2}^{p-2}}{t_{i+p+1}-t_{i+2}}\right) &(9)\end{aligned}$$

having used Eq.(3) to obtain (8), and the induction hypothesis to obtain Eq.(9). Now, rearranging the last two terms of Eq.(9), one easily obtains:

$$\begin{aligned}\frac{d}{dx}\Lambda_i^p &= \frac{\Lambda_i^{p-1}}{t_{i+p}-t_i} - \frac{\Lambda_{i+1}^{p-1}}{t_{i+p+1}-t_{i+1}}\\ &\quad + (p-1)\left[\frac{1}{t_{i+p}-t_i}\left(\frac{x-t_i}{t_{i+p-1}-t_i}\Lambda_i^{p-2} + \frac{t_{i+p}-x}{t_{i+p}-t_{i+1}}\Lambda_{i+1}^{p-2}\right)\right.\\ &\qquad \left. - \frac{1}{t_{i+p+1}-t_{i+1}}\left(\frac{x-t_{i+1}}{t_{i+p}-t_{i+1}}\Lambda_{i+1}^{p-2} + \frac{t_{i+p+1}-x}{t_{i+p+1}-t_{i+2}}\Lambda_{i+2}^{p-2}\right)\right]\\ &= \frac{\Lambda_i^{p-1}}{t_{i+p}-t_i} - \frac{\Lambda_{i+1}^{p-1}}{t_{i+p+1}-t_{i+1}} + (p-1)\left(\frac{\Lambda_i^{p-1}}{t_{i+p}-t_i} - \frac{\Lambda_{i+1}^{p-1}}{t_{i+p+1}-t_{i+1}}\right) &(10)\\ &= p\left(\frac{\Lambda_i^{p-1}}{t_{i+p}-t_i} - \frac{\Lambda_{i+1}^{p-1}}{t_{i+p+1}-t_{i+1}}\right)\end{aligned}$$

having again used Eq.(3) to obtain (10). This completes the demonstration of relation (7).

1.5 Integrals of Splines

With the proper normalization all splines of all degrees have unitary surface:

$$\frac{p+1}{t_{i+p+1}-t_i}\int \Lambda_i^p(x)dx = 1. \tag{11}$$

This relation holds trivially for $p=0$ and $p=1$. A recursive proof of the general statement (11) starts assuming

$$\frac{p}{t_{i+p}-t_i}\int \Lambda_i^{p-1}(x)dx = 1 \tag{12}$$

to be true. Then using Eq.(7) multiplied by x and integrating one obtains:

$$\int x\frac{d}{dx}\Lambda_i^p dx = -\int \Lambda_i^p dx = p\int\left(\frac{x\Lambda_i^{p-1}}{t_{i+p}-t_i} - \frac{x\Lambda_{i+1}^{p-1}}{t_{i+p+1}-t_{i+1}}\right)dx.$$

Completing the fractions in the big parentheses in view of using Eq.(3) one has

$$\begin{aligned}\int \Lambda_i^p dx &= -p\int\frac{x-t_i}{t_{i+p}-t_i}\Lambda_i^{p-1}dx - p\int\frac{t_i}{t_{i+p}-t_i}\Lambda_i^{p-1}dx \\ &\quad -p\int\frac{t_{i+p+1}-x}{t_{i+p+1}-t_{i+1}}\Lambda_{i+1}^{p-1}dx + p\int\frac{t_{i+p+1}}{t_{i+p+1}-t_{i+1}}\Lambda_{i+1}^{p-1}dx,\end{aligned}$$

where the first and the third terms on the right correspond to $-p\int\Lambda_i^p dx$, Eq.(3), and can be combined with the left side to yield

$$(1+p)\int\Lambda_i^p dx = t_{i+p+1}-t_i, \tag{13}$$

where relation (12) has been used for the rest on the right hand side. This concludes the proof of Eq.(11).

1.6 Boundary Conditions

Applying the recurrence relation to generate *all* the splines on the finite domain $[t_0, t_{N_x}]$ yields the $N_x + p$ splines of degree p:

$$\Lambda_{-p}^p, \Lambda_{-p+1}^p, \ldots, \Lambda_{N_x-1}^p. \tag{14}$$

Note that *additional* knots beyond both ends of $[t_0, t_{N_x}]$ have to be defined to generate all these splines.

1.6.1 Periodic splines

The extra knots are simply defined through periodicity.:

$$t_{-\nu} = t_{N_x-\nu} - (b-a), \tag{15}$$

$$t_{N_x+\nu} = t_\nu + (b-a), \qquad \nu = 0,\ldots,p. \tag{16}$$

The $p+1$ leftmost splines in (14) are thus identical to the rightmost splines:

$$\Lambda_{-\nu}^p = \Lambda_{N_x-\nu}^p, \qquad \nu = 0,\ldots,p. \tag{17}$$

1.6.2 Non-periodic splines

The choice made in `BSPLINES` is simply:

$$t_{-p} = \cdots = t_0 = a, \qquad b = t_{N_x} = \cdots = t_{N_x+p}. \tag{18}$$

Thus in the first interval $[t_0, t_1]$, the first spline Λ_{-p}^p is constructed (refer to the first entry on each of the column of Fig. 1, with $i = 0$) as follow:

$$\begin{aligned}\Lambda_{-1}^1 &= (1-w_0^1)\Lambda_0^0 = \frac{t_1-x}{t_1-t_0}\Lambda_0^0 \\ \Lambda_{-2}^2 &= (1-w_{-1}^2)\Lambda_{-1}^1 = \frac{t_1-x}{t_1-t_{-1}}\Lambda_{-1}^1 = \left(\frac{t_1-x}{t_1-t_0}\right)^2\Lambda_0^0 \\ &\vdots \\ \Lambda_{-p}^p &= (1-w_{-p+1}^p)\Lambda_{-p+1}^p = \frac{t_1-x}{t_1-t_{-p+1}}\Lambda_{-p+1}^{p-1} = \left(\frac{t_1-x}{t_1-t_0}\right)^p\Lambda_0^0\end{aligned}$$

In the same manner, the generation of the *last* spline $\Lambda^p_{N_x-1}$ (last entry on each of the column of Fig. 1, with $i = N_x - 1$) yields:

$$\begin{aligned}
\Lambda^1_{N_x-1} &= w^1_{N_x-1}\Lambda^0_{N_x-1} = \frac{x - t_{N_x-1}}{t_{N_x} - t_{N_x-1}}\Lambda^0_{N_x-1} \\
\Lambda^2_{N_x-1} &= w^2_{N_x-1}\Lambda^1_{N_x-1} = \frac{x - t_{N_x-1}}{t_{N_x+1} - t_{N_x-1}}\Lambda^1_{N_x-1} = \left(\frac{x - t_{N_x-1}}{t_{N_x} - t_{N_x-1}}\right)^2 \Lambda^0_{N_x-1} \\
&\cdot \qquad\qquad \cdot \qquad\qquad \cdot \\
\Lambda^p_{N_x-1} &= w^p_{N_x-1}\Lambda^p_{N_x-1} = \frac{x - t_{N_x-1}}{t_{N_x+p-1} - t_{N_x-1}}\Lambda^1_{N_x-1} = \left(\frac{x - t_{N_x-1}}{t_{N_x} - t_{N_x-1}}\right)^p \Lambda^0_{N_x-1}
\end{aligned}$$

Since the sum of all splines is 1 and using the positivity of splines, all the non-periodic splines, except the first (last) spline should vanish at $x = a$ ($x = b$):

$$\Lambda^p_r(a) = \delta_{r,-p}, \qquad \Lambda^p_r(b) = \delta_{r,N_x-1} \tag{19}$$

The spline derivatives at the boundaries $x = a$ and $x = b$ can be derived using Eq.(7) as follow. At $x = a$ (interval $[t_0, t_1]$), by noting that only the spline Λ^{p-1}_{-p+1} is non-zero at $x = a$ (see next to last column of Fig.1, with $i = 0$), it is easy to see that there are only 2 non-zero derivatives given by

$$\begin{aligned}
\frac{d}{dx}\Lambda^p_{-p}(a) &= -\frac{p\,\Lambda^{p-1}_{-p+1}(a)}{t_1 - t_{-p}} = -\frac{p}{t_1 - t_0}, \\
\frac{d}{dx}\Lambda^p_{-p+1}(a) &= \frac{p\,\Lambda^{p-1}_{-p+1}(a)}{t_1 - t_{-p+1}} = \frac{p}{t_1 - t_0},
\end{aligned} \tag{20}$$

where we have used $t_0 = t_{-1} = \ldots = t_{-p} = a$. Likewise, the 2 non-zero derivatives of spline at the other boundary $x = b$ are

$$\begin{aligned}
\frac{d}{dx}\Lambda^p_{N_x-2}(b) &= -\frac{p\,\Lambda^{p-1}_{N_x-1}(b)}{t_{N_x+p-1} - t_{N_x-1}} = -\frac{p}{t_{N_x} - t_{N_x-1}}, \\
\frac{d}{dx}\Lambda^p_{N_x-1}(b) &= \frac{p\,\Lambda^{p-1}_{N_x-1}(b)}{t_{N_x+p-1} - t_{N_x-1}} = \frac{p}{t_{N_x} - t_{N_x-1}},
\end{aligned} \tag{21}$$

where we have used $t_{N_x} = t_{N_x+1} = \ldots = t_{N_x+p} = b$.

1.6.3 Spline expansion

In summary, the approximation of a function f defined in the interval $[a, b]$ using a basis (**Is this obvious?**) of splines of degree p associated with the sequence of knots $t_i, i = -p, \ldots, N_x + p$ can be written as

$$f(x) = \sum_{j=-p}^{N_x-1} c_j \Lambda^p_j(x), \qquad \begin{array}{l}\text{support of } \Lambda^p_j\text{:}\quad [t_j, t_{j+p+1}], \\ t_i \le x < t_{i+1} \Longrightarrow \Lambda^p_{i-p}(x), \ldots, \Lambda^p_i(x) \ge 0.\end{array} \tag{22}$$

Note that the *last* spline in the interval $[t_i, t_{i+1}]$, which can be written as

$$\Lambda^p_i(x) = w^p_i(x)\Lambda^{p-1}_i(x) = \ldots = w^p_i(x)w^{p-1}_i(x)\ldots w^1_i(x)\Lambda^0_i(x)$$

vanishes at the knot $x = t_i$. Thus at any position x, the sum involves $p + 1$ terms except at the knots t_i where there are only p terms.

It is sometimes more convenient to renumber the spline index j so that it starts from 0. With this new numbering, the spline expansion becomes

$$f(x) = \sum_{j=0}^{N_x+p-1} c_j \Lambda^p_j(x), \qquad \begin{array}{l}\text{support of } \Lambda^p_j\text{:}\quad [t_{j-p}, t_{j+1}], \\ t_i \le x < t_{i+1} \Longrightarrow \Lambda^p_i(x), \ldots, \Lambda^p_{i+p}(x) \ge 0.\end{array} \tag{23}$$

In the *periodic* case, there are N_x *independent* spline coefficients since

$$c_{N_x+\nu} = c_\nu, \qquad \nu = 0, \ldots, p-1. \tag{24}$$

In the *non-periodic* case, the first and the last spline coefficients c_0, c_{N_x+p-1} are respectively the values of f at the end points a and b.

The basis functions for both non-periodic and periodic cubic splines ($p = 3$) are shown in Fig .2 where this new numbering is used.

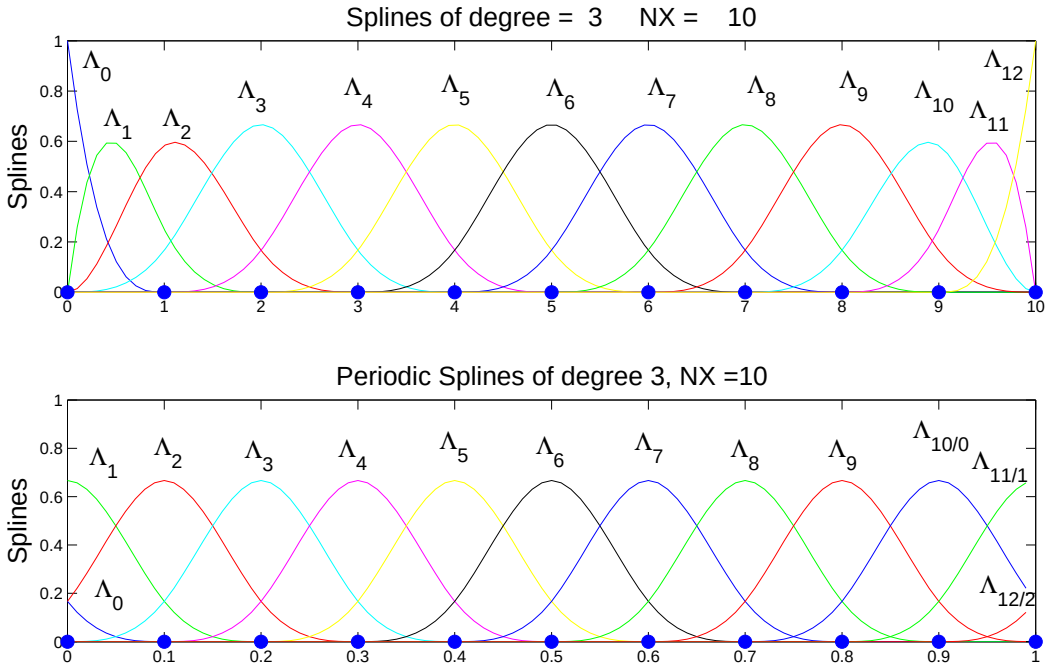

Figure 2: The basis of non-periodic and periodic cubic splines. The periodic splines Λ_{10}, Λ_{11}, Λ_{12} denote the same splines as Λ_0, Λ_1, Λ_2 respectively.

1.7 Spline Initialization with `SET_SPLINE`

The initialization of a spline is performed by calling the routine `SET_SPLINE`, passing the desired degree p and the sequence of grid points (or knots) $t_j, j = 0, \ldots, N_x$. If Gauss points on each of the intervals $[t_j, t_{j+1}]$ are needed, a non-zero value of `NGAUSS` should be specified. The other input argument is the *optional* `LOGICAL` argument `PERIOD` to define the periodicity of the splines. By default it is `.FALSE.`. The routine returns the 1d spline `SP` which is of type `TYPE(spline1d)`:

```
SUBROUTINE set_spline(p, ngauss, grid, sp, period)
  INTEGER, INTENT(in) :: p, ngauss
  DOUBLE PRECISION, INTENT(in)  :: grid(:)
  LOGICAL, OPTIONAL, INTENT(in) :: period
  TYPE(spline1d), INTENT(out)   :: sp
  LOGICAL, OPTIONAL, INTENT(in) :: period
```

Besides the main characteristics of the spline (degree p of splines, number of grid intervals, dimension of splines $N_x + p$, etc.) the following quantities will be determined and stored in `SP`:

- values and all the p derivatives of the $p+1$ non-vanishing splines on each knots t_j. These quantities will be used to speed up computation of the spline expansion (23).
- integrals of splines $I_i = \int \Lambda_i(x)\,dx$.

For a 2d spline

$$\Lambda_{ij}^{p+q}(x,y) = \Lambda_i^p(x)\Lambda_j^q(y), \tag{25}$$

on a 2d structured mesh defined by the grid points `grid1(0:N1), grid2(0:N2)`, the same call as in the 1d case can be used, except that the scalars `p, ngauss, period` become 2 element arrays and the output `SP` is now of type `TYPE(spline2d)`:

```
  INTEGER :: p(2), ngauss(2)
  LOGICAL, OPTIONAL :: period(2)
  DOUBLE PRECISION, dimension(:) :: grid1, grid2
  TYPE(spline2d) :: sp2d
...
  CALL set_spline(p, ngauss, grid1, grid2, sp2d, period)
```

The derived type `spline2d` is a *wrapper* of 2 `spline1d` objects which can be accessed through `sp2d%sp1` and `sp2d%sp2`.

Once `SET_SPLINE` is called, the routine `GET_DIM` can be called to inquire the spline's essential characteristics such as dimension, number of intervals and degree, for both 1d and 2d splines:

```
SUBROUTINE get_dim(sp, dim, nx, nidbas)
  TYPE(spline1d), INTENT(in) :: sp
  INTEGER, INTENT(out) :: dim
  INTEGER, OPTIONAL, INTENT(out) :: nx, nidbas
```

Integral of function $\int_a^b f(x)dx$ is computed from its spline `sp` and splines coefficients in:

```
DOUBLE PRECISION FUNCTION fintg(sp, c)
  TYPE(spline1d), INTENT(in) :: sp
  DOUBLE PRECISION, INTENT(in) :: c(:)
```

For a 2d functions, the same function should be called with a 2d spline `sp` and 2d array c.

Finally `DESTROY_SP(sp)` should be called when a spline `sp` is not needed anymore to clean up memory space.

1.8 Generating Splines with `DEF_BASFUN`

```
SUBROUTINE def_basfun(xp, sp, fun, left)
  DOUBLE PRECISION, INTENT(in) :: xp
  TYPE(spline1d), INTENT(in) :: sp
  DOUBLE PRECISION, INTENT(out) :: fun(:,:)
  INTEGER, OPTIONAL, INTENT(out) :: left
```

This routine computes, for a given point xp $\in [t_0, t_{N_x}]$, the value and optionally the m derivatives of the $p+1$ splines `sp` which were previously defined and returns them in `fun(1:p+1,1:m+1)` with $m \le p$. The maximum number of computed derivatives m is determined by the size of the second dimension of the array `fun`. The subroutine will return the optional integer `left` defined such that:

$$t_{\text{left}} \le xp < t_{\text{left}+1}, \qquad 0 \le \text{left} \le N_{x.-1}.$$

1.9 Example 1: Values and derivatives of all splines

In this example, we first initialize a cubic spline with the knot sequence $t_0, \ldots, t_{N_x}$ with `SET_SPLINE` and then call `DEF_BASFUN` to compute its values, first and second derivatives on the mesh points `xp(1:npts)`.

```
  USE BSPLINES
  INTEGER, PARAMETER :: nx=10, npts=100
  DOUBLE PRECISION :: t(0:nx), xp(npts)
  DOUBLE PRECISION, ALLOCATABLE :: fxp0(:,:), fxp1(:,:), fxp2(:,:)
  DOUBLE PRECISION :: fun(4,3)   ! 4 cubic splines at a given xp
                                 ! plus first and second derivatives.
  INTEGER :: i, dim, left
  TYPE(spline1d) :: sp
!
!   Define t(0:nx), xp(npts)
!
  CALL set_spline(3, 0, t, sp, period=.FALSE.)
  CALL get_dim(sp, dim)
  ALLOCATE(fxp0(npts,0:dim-1), fxp1(npts,0:dim-1), fxp2(npts,0:dim-1)
  fxp0 = 0.0
  fxp1 = 0.0
  fxp2 = 0.0
  DO i=1,npts
     CALL def_basfun(xp(i), sp,  fun, left=left)
     fxp0(i, left:left+3) = fun(1:4, 1)   ! Value
     fxp1(i, left:left+3) = fun(1:4, 2)   ! 1st derivative
     fxp2(i, left:left+3) = fun(1:4, 3)   ! 2nd derivative
  END DO
  DEALLOCATE(fxp0, fxp1, fxp2)
  CALL destroy_sp(sp)
```

This code fragment will store `dim=nx+3=13` splines and theirs first 2 derivatives in `fxp0`, `fxp1` and `fxp2`. Change the `period` to `.TRUE.` to obtain *periodic* splines.

2 Spline Interpolation

Given the interval $[a, b]$ discretized into $\{x_k,\, k = 0, \ldots, N_g\}$ with $x_0 = a$ and $x_{N_g} = b$, the problem of interpolating $f(x), x \in [a, b]$ with splines of degree p is to solve the following equations for the spline coefficients c_i:

$$\sum_{i=0}^{N_x+p-1} c_i \Lambda_i^p(x_k) = f(x_k), \quad k = 0, \ldots, N_g. \tag{26}$$

The sequence of knots $t_0, \ldots, t_{N_x}$ defines completely the splines Λ_i^p and its choice will be described in the following section.

2.1 Choice of knots

If Eqs. (26) are the only conditions for our interpolation problem, the number of equations should match the number of unknowns c_i. The number of knot intervals N_x hence has to verify

$$N_x = N_g - p + 1. \tag{27}$$

For the *periodic* case, taking into account the p periodic spline conditions (24) on c_i and $f(a) = f(b)$, this condition reduces to:

$$N_x = N_g. \tag{28}$$

For *odd* values of the spline degree p, the knots t_i could be placed on the *interpolation sites* x_k while when p is even, t_i should not be on x_k to avoid a *badly conditioned* linear system when solving Eq. (26). This leads to the following choice for t_i in `BSPLINES`:

2.1.1 Periodic splines

The number of knots $N_x + 1$ is *equal* to the number of interpolation points $N_g + 1$ with

$$t_i = \begin{cases} x_i & p \text{ odd} \\ (x_{i-1} + x_i)/2 & p \text{ even} \end{cases}, \qquad i = 0, \ldots, N_x \tag{29}$$

2.1.2 Non-Periodic splines

In order to satisfy the equality (27), first, the 2 end points are retained as knots:

$$t_0 = x_0, \qquad t_{N_x} = x_{N_g}. \tag{30}$$

For even p, the first $p/2$ interpolation intervals are *skipped*:

$$t_i = (x_{i+p/2-1} + x_{i+p/2})/2, \qquad i = 1, \ldots, N_x - 1, \tag{31}$$

while for odd p, $(p-1)/2$ interpolation points are *skipped*:

$$t_i = x_{i+(p-1)/2}, \qquad i = 1, \ldots, N_x - 1. \tag{32}$$

Instead of skipping grid points, an alternative would be to supplement the system of equations (26) with conditions on derivatives of $f(x)$ at one or both ends of $[a, b]$. This type of boundary conditions is not implemented in the present version of the `BSPLINES` module.

2.2 The collocation matrix

The *collocation matrix* $\Lambda_i^p(x_k)$ of the interpolation problem (26) is a square matrix. Each row has at most $p + 1$ non-zero terms. Let us consider separately the non-periodic and the periodic cases.

2.2.1 The non-periodic case

Even spline degree From (31), there are $p/2+1$ interpolation points $x_0, \ldots, x_{p/2}$ in the first knot interval $[t_0, t_1)$. Since there are at most $p + 1$ non-zero splines for any points in each interval (except for x_0 where $\Lambda_i(x_0) = \delta_{i,0}$, the collocation matrix starts as:

$$\begin{pmatrix} \Lambda_0(x_0) & 0 & \cdots & \cdots & \cdots & \cdots \\ \Lambda_0(x_1) & \Lambda_1(x_1) & \cdots & \Lambda_p(x_1) & 0 & \cdots \\ \vdots & \vdots & \cdots & \vdots & 0 & \cdots \\ \Lambda_0(x_{p/2}) & \Lambda_1(x_{p/2}) & \cdots & \Lambda_p(x_{p/2}) & 0 & \cdots \\ 0 & \Lambda_1(x_{p/2+1}) & \cdots & \Lambda_p(x_{p/2+1}) & \Lambda_{p+1}(x_{p/2+1}) & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \tag{33}$$

The number of *upper-diagonals* (non including the diagonal) is obviously determined by the second row of the matrix above, which yields $p-1$. Since the knot placement is identical for both ends of the interpolation mesh, the matrix $\Lambda_i(x_k)$ is *banded* with half-bandwidths

$$kl = ku = p - 1 \tag{34}$$

Odd spline degree Applying the same procedure, it is straightforward to show for p odd and from (32), that $x_0, \ldots . x_{(p-1)/2}$ are located in the first knot interval $[t_0, t_1)$ and that the matrix has again the same half-bandwidths as in the even p case.

The resulting interpolation problem can then be solved with the usual *banded matrix factorization* followed by a *back-solve* phase.

2.2.2 The periodic case

Let consider the matrix for $p = 3$ and $N_x = 10$ (see lower figure of Fig. (2):

$$\begin{pmatrix} \Lambda_0(x_0) & \Lambda_1(x_0) & \Lambda_2(x_0) & 0 & \cdots & \\ 0 & \Lambda_1(x_1) & \Lambda_2(x_1) & \Lambda_3(x_1) & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & 0 & \Lambda_7(x_7) & \Lambda_8(x_7) & \Lambda_9(x_7) \\ \Lambda_0(x_8) & 0 & 0 & \cdots & \Lambda_8(x_8) & \Lambda_9(x_8) \\ \Lambda_0(x_9) & \Lambda_1(x_9) & 0 & 0 & \cdots & \Lambda_9(x_9) \end{pmatrix} \tag{35}$$

The matrix is "almost triangular" (except for the last 2 rows) and is not *diagonally dominant*! A more satisfactory (and symmetric in shape) matrix is however obtained by simply renumbering the splines such that the sequence starts with $-\lfloor p/2 \rfloor$ instead of 0. This renumbered splines are shown in Fig. 3 for the cubic and quadratic periodic splines. With this renumbering, the matrix (35) has a more symmetric shape and is diagonally dominant:

$$\begin{pmatrix} \Lambda_0(x_0) & \Lambda_1(x_0) & 0 & \cdots & \Lambda_9(x_0) \\ \Lambda_0(x_1) & \Lambda_1(x_1) & \Lambda_2(x_1) & 0 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots \\ 0 & \cdots & \Lambda_7(x_8) & \Lambda_8(x_8) & \Lambda_9(x_8) \\ \Lambda_0(x_9) & 0 & 0 & \Lambda_8(x_9) & \Lambda_9(x_9) \end{pmatrix} \tag{36}$$

In general, for arbitrary p (even and odd values), the collocation matrix $A = \Lambda_j(x_i)$ can be written as

$$A = B + UV^T \tag{37}$$

where B is a banded matrix with half-bandwidths $kl = ku = b = \lfloor p/2 \rfloor$ and rank N_x. U and V are $N_x \times 2b$ sparse matrices:

$$U = \begin{pmatrix} I & 0 \\ 0 & 0 \\ 0 & I \end{pmatrix}, \qquad V = \begin{pmatrix} 0 & D^T \\ 0 & 0 \\ C^T & 0 \end{pmatrix}, \qquad V^T = \begin{pmatrix} 0 & 0 & C \\ D & 0 & 0 \end{pmatrix}, \tag{38}$$

where C, D are the $b \times b$ *off-band* sub-matrices and I, the identity matrix. In the cubic spline example considered above, the *off-band* matrices are simply 1×1 matrices with $C = \Lambda_9(x_0)$ and $D = \Lambda_0(x_9)$.

The inverse of A can be deduced from the *Sherman-Morrison-Woodbury formula* [2]:

$$\begin{aligned} A^{-1} &= B^{-1} - B^{-1}U(1 + V^T B^{-1}U)^{-1}V^T B^{-1} \\ &= B^{-1} - ZW^T B^{-1}, \end{aligned}$$

where

$$\begin{aligned} Z &= B^{-1}U, \\ H &= 1 + V^T B^{-1}U \\ W^T &= H^{-1}V^T. \end{aligned}$$

The solution of the interpolation problem $Ax = f$ can then be reduced to a *factorization* and a *back-solve* phase:

1. Factorization
   (a) Factor: $B \longleftarrow L_B U_B$
   (b) Solve: $(L_B U_B)Z = U, \quad U \longleftarrow Z$
   (c) Compute: $H = 1 + V^T Z$
   (d) Factor: $H = L_H U_H$
   (e) Solve: $(L_H U_H)W^T = V^T, \quad V^T \longleftarrow W^T$
2. Back-solve
   (a) Solve: $(L_B U_B)y = f$
   (b) Compute: $t = W^T y$
   (c) Compute: $x = y - Zt$

At the end of the factorization, only the (updated) matrices B, U and V^T, required in the back-solve phase, need to be saved. Note that we avoid to store the product ZW^T because it is a *big* $N_x \times N_x$ matrix.

After the *back-solve* step, the solution x is *shifted back* (by $\lfloor p/2 \rfloor$) and the appropriate periodicity condition is applied to obtain the spline coefficients c_j, $j = 0, \ldots, N_x + p - 1$, as defined in (23).

2.3 `PP` representation

The computation of $f(x)$ using directly the spline expansion Eq. (23) can be costly, because of the evaluation of the splines $\Lambda_j^p(x)$, especially when interpolating on large number of points. Expanding $f(x)$, using truncated Taylor series in each interval $[t_\mu, t_{\mu+1}]$, we obtain the following *Piecewise Polynomial Function* representation (or `ppform`) of $f(x)$:

$$f(x) = \sum_{k=0}^{p} \Pi_{k\mu}(x - t_\mu)^k, \quad t_\mu \le x < t_{\mu+1}, \tag{39}$$

where

$$\Pi_{k\mu} = \frac{1}{k!}\frac{d^k}{dx^k}f(t_\mu) = \frac{1}{k!}\sum_j c_j \frac{d^k}{dx^k}\Lambda_j(t_\mu) = \sum_j c_j \alpha_{j\mu k}. \tag{40}$$

Note that

$$\alpha_{j\mu k} = \frac{1}{k!}\frac{d^k}{dx^k}\Lambda_j(t_\mu) \tag{41}$$

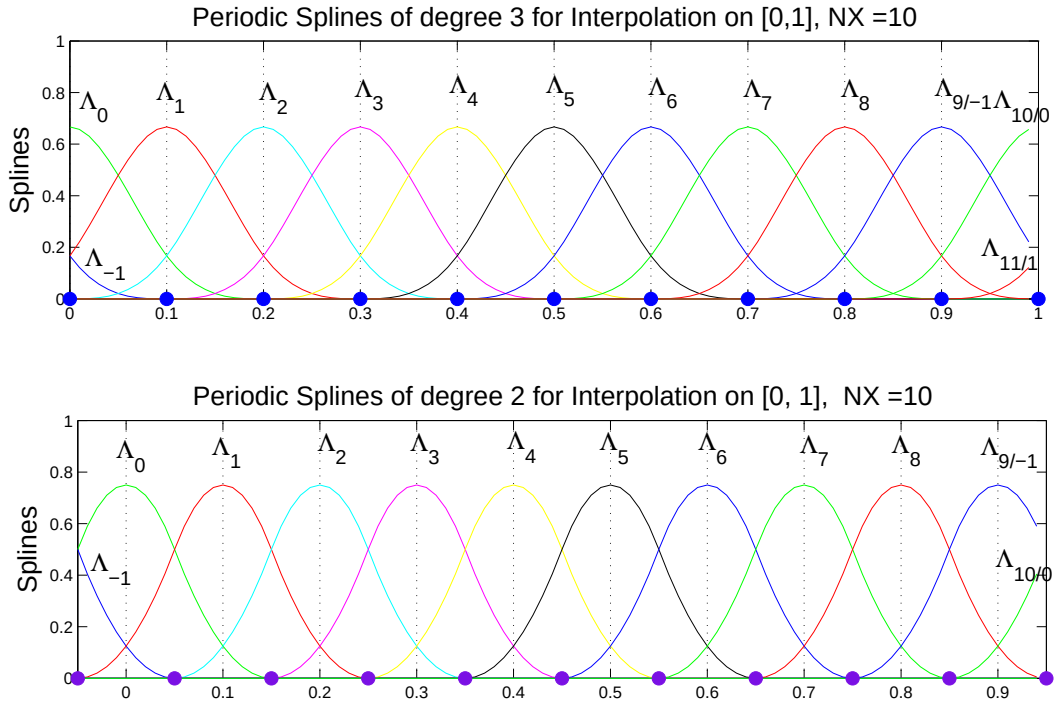

Figure 3: The periodic cubic and quadratic splines used for interpolation. The spline knots are indicated by *blue full circles* and the interpolation points, by *dashed vertical lines*

depend only on the spline specifications. They are *pre-calculated* in the spline setup routine `SET_SPLINE` and stored in the 3d arrays `sp%val0`. The `PP` coefficients $\Pi_{k\mu}$ can be computed once the spline coefficients c_j are available, using (40). Then the interpolated function values together with the p derivatives can be calculated *efficiently* using the power series.

$$
\begin{aligned}
f(x) &= \sum_{k=0}^{p} \Pi_{k\mu}(x - t_\mu)^k \\
f'(x) &= \sum_{k=1}^{p} k\,\Pi_{k\mu}(x - t_\mu)^{k-1} \\
f''(x) &= \sum_{k=2}^{p} k(k-1)\,\Pi_{k\mu}(x - t_\mu)^{k-2} \\
&\vdots
\end{aligned}
$$

These 2 steps are performed in `GRIDVAL`. Note that the first step (computation of $\Pi_{k\mu}$ from c_j) can be skipped for subsequent calls to `GRIDVAL` with the same function f, for example to compute f or its derivatives at any others points x.

2.4 Example 2: Cubic spline interpolation

Given a function f with its grid values `f(1:nx)`, the following example determines the spline coefficients `c(1:dim)`. Using these coefficients, the interpolated f and derivative f' are then computed on the mesh

points `xp(1:npts)`. Note that the second call to `GRIDVAL` does not include the spline coefficients `c` to signal that the previously calculated `PP` coefficients will be reused.

```
  USE BSPLINES
  INTEGER, PARAMETER :: nx=10, npts=100
  DOUBLE PRECISION :: x(0:nx), f(0:nx), xp(npts), fp0(npts), fp1(npts)
  DOUBLE PRECISION, ALLOCATABLE :: c(:)
  INTEGER :: dim
  TYPE(spline1d) :: sp
!
!   Define x and f
!
  CALL set_splcoef(3, x, sp, period=.FALSE.)
  CALL get_dim(sp, dim)
  ALLOCATE(c(dim))
  CALL get_splcoef(sp, f, c)   ! compute spline coefs c(1:dim)
!
!   Compute interpolated f and f' on xp(npts)
!
  CALL gridval(sp, xp, fp0, 0, c)
  CALL gridval(sp, xp, fp1, 1)
!
  DEALLOCATE(c)
  CALL destroy_sp(sp)
```

The description of each of the routines called in the example above is briefly given below:

`SET_SPLCOEF` Determines the spline knots, sets up the splines with `SET_SPLINE`, assembles the collocation matrix and performs its *factorization*.

`GET_SPLCOEF` Computes the spline *coefficients* c from the input grid values of function f (*back-solve phase*), using the factorized matrix.

`GRIDVAL` Compute the `PP` coefficients using (40) *if c is provided*, locates the interval containing the given point x and computes interpolated function values or derivatives using the `PP` representation (40).

2.5 2d interpolation

Consider the spline interpolation on the plane (x, y), using a tensor product of splines defined as follow

$$\Lambda_{ij}^{p,q}(x,y) = \Lambda_i^p(x)\Lambda_j^q(y), \qquad \begin{array}{l} i = 1,\ldots,d_1 = N_1 + p, \\ j = 1,\ldots,d_2 = N_2 + q, \end{array} \tag{42}$$

where (p, q) are the spline degrees , (N_1, N_2), the number of knot intervals in each direction:

$$t_0 \le x < t_{N_1}, \quad s_0 \le y < s_{N_2}.$$

2.5.1 Spline coefficients (GET_SPLCOEF)

The 2d version of (26) can be written as:

$$\begin{aligned} \sum_{ij} c_{ij}\Lambda_i^p(x_\mu)\Lambda_j^q(y_\nu) &= f(x_\mu, y_\nu) \qquad && \mu = 0, \ldots, N_{g1} \\ &= f_{\mu\nu} && \nu = 0, \ldots, N_{g2} \end{aligned} \tag{43}$$

where (x_μ, y_ν) are the *interpolation sites* on the (x, y) plane. These equations can be rearranged into

$$\sum_i \bar{c}_{i\nu}\Lambda_i(x_\mu) = f_{\mu\nu}, \tag{44}$$

$$\sum_j c_{ij}\Lambda_j(y_\nu) = \bar{c}_{i\nu}. \tag{45}$$

Such a 2 step procedure is implemented, using the 1d version of `GET_SPLCOEF` in the 2d version of `GET_SPLCOEF` by the following code fragment:

```
TYPE(spline2d) :: sp
DOUBLE PRECISION :: ctr(SIZE(c,2), SIZE(c,1))
CALL get_splcoefn(sp%sp1, f, c)
CALL get_splcoefn(sp%sp2, TRANSPOSE(c), ctr)
c = TRANSPOSE(ctr)
```

2.5.2 PP representation (GRIDVAL)

Let us start with the *spline representation*, for $f(x, y)$ with $t_\mu \le x < t_{\mu+1}$ and $s_\nu \le y < s_{\nu+1}$:

$$f(x, y) = \sum_{j=1}^{d_2} \left(\sum_{i=1}^{d_1} c_{ij}\Lambda_i^p(x) \right) \Lambda_j^q(y). \tag{46}$$

Applying successively (39) to the x and y functional dependency yields the following `PP` representation for $f(x, y)$:

$$f(x, y) = \sum_{k'=0}^{q} \left(\sum_{k=0}^{p} \Pi_{k\mu k'\nu}(x - t_\mu)^k \right) (y - s_\nu)^{k'} \tag{47}$$

The 2d `PP` coefficient is the *tensor product* of 2 1d `PP` coefficients:

$$\Pi_{k\mu k'\nu} = \sum_{l'=1}^{d_2} \left(\sum_{l=1}^{d_1} c_{ll'}\, \alpha_{l\mu k} \right) \alpha'_{l'\nu k'}. \tag{48}$$

where α and α' are respectively the derivatives (41) of all splines in x and y direction. All the derivatives of f can be deduced straightforwardly from the `PP` representation (47).

In the present version, the 2d `GRIDVAL` can be called, either for (1) points on a 2d structured mesh: `XP(1:NPX), YP(1:NPY)` and returns the array `FP(1:NPX,1:NPY)`, or (2) with a 1d sequence of points `XP(1:NP), YP(1:NP)` and returns the 1d array `FP(1:NP)`.

3 Finite Elements using Splines

3.1 The weak form

3.2 The matrix assembly

3.3 The boundary conditions

3.3.1 Dirichlet condition

Dirichlet BC can be simply applied by imposing the conditions on the spline coefficients and the boundary point. For the BC $u(x=0) = u_1 = c$ for example, the discrete equations can be expressed as:

$$\begin{pmatrix} 1 & 0 & \cdots \\ A_{21} & A_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} c \\ f_2 \\ \vdots \\ f_N \end{pmatrix} \tag{49}$$

A more appropriate transformed system which preserves any symmetry of the original system is:

$$\begin{pmatrix} 1 & 0 & \cdots \\ 0 & A_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} c \\ f_2 - cA_{21} \\ \vdots \\ f_N - cA_{N1} \end{pmatrix} \tag{50}$$

3.3.2 Unicity on the axis

Denoting the N solutions at the axis by $(u_1, \ldots, u_N)$, and their transforms by $(\hat{u}_1, \ldots, \hat{u}_N)$ defined by

$$\begin{array}{c} u_1 - u_N = \hat{u}_1 \\ u_2 - u_N = \hat{u}_2 \\ \vdots \\ u_{N-1} - u_N = \hat{u}_{N-1} \\ u_N = \hat{u}_N \end{array} \quad \Longrightarrow \quad \begin{array}{c} u_1 = \hat{u}_1 + \hat{u}_N \\ u_2 = \hat{u}_2 + \hat{u}_N \\ \vdots \\ u_{N-1} = \hat{u}_{1-1} + \hat{u}_N \\ u_N = \hat{u}_N, \end{array} \tag{51}$$

the unicity condition can be specified by simply imposing

$$\hat{u}_1 = \hat{u}_2 = \ldots = \hat{u}_{N-1} = 0. \tag{52}$$

From (51), the *transformation matrix* $\mathbf{U}$ is defined as

$$\mathbf{u} = \mathbf{U} \cdot \hat{\mathbf{u}}, \qquad \mathbf{U} = \begin{pmatrix} 1 & 0 & \ldots & 0 & 1 \\ 0 & 1 & \ldots & 0 & 1 \\ & & \ddots & & \vdots \\ 0 & 0 & \ldots & 1 & 1 \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix}, \quad \mathbf{U^T} = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ & & \ddots & & \vdots \\ 0 & 0 & \ldots & 1 & 0 \\ 1 & 1 & \ldots & 1 & 1 \end{pmatrix}. \tag{53}$$

Matrix product $\mathbf{A} \cdot \mathbf{U}$

$$\mathbf{A} \cdot \mathbf{U} = \begin{pmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,N-1} & \sum_j A_{1,j} \\ A_{2,1} & A_{2,2} & \dots & A_{2,N-1} & \sum_j A_{2,j} \\ & & \ddots & & \vdots \\ A_{N-1,1} & A_{N-1,2} & \dots & A_{N-1,N-1} & \sum_j A_{N-1,j} \\ A_{N,1} & A_{N,2} & \dots & A_{N,N-1} & \sum_j A_{N,j} \end{pmatrix}. \tag{54}$$

Thus *right multiply by* $\mathbf{U}$ is equivalent to put the *the sum of each row on the last column.*

Matrix product $\mathbf{U^T} \cdot \mathbf{A}$

$$\mathbf{U^T} \cdot \mathbf{A} = \begin{pmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,N-1} & A_{1,N} \\ A_{2,1} & A_{2,2} & \dots & A_{2,N-1} & A_{2,N} \\ & & \ddots & & \vdots \\ A_{N-1,1} & A_{N-1,2} & \dots & A_{N-1,N-1} & A_{N-1,N} \\ \sum_i A_{i,1} & \sum_i A_{i,2} & \dots & \sum_i A_{i,N-1} & \sum_i A_{i,N} \end{pmatrix}. \tag{55}$$

Thus *left multiply by* $\hat{\mathbf{U}}$ is equivalent to put the *the sum of each column on the last row.*

Product $\hat{\mathbf{U}} \cdot \mathbf{b}$

$$\hat{\mathbf{b}} = \mathbf{U^T} \cdot \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \\ \sum_i b_i \end{pmatrix}, \tag{56}$$

Transformation of the original matrix equation The full original linear system, obtained from the discretization of the 2D r, θ polar coordinates can be written as:

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{c} \end{pmatrix}, \tag{57}$$

where the solution array is split into the solutions $\mathbf{u}$ at $r = 0$ and the solutions $\mathbf{v}$ on the remaining domain. The transformed system can thus be written as

$$\begin{pmatrix} \mathbf{U^T} & 0 \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{U} & 0 \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{u}} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{U^T} & 0 \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{b} \\ \mathbf{c} \end{pmatrix},$$

$$\implies \begin{pmatrix} \mathbf{U^T A U} & \mathbf{U^T B} \\ \mathbf{CU} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{u}} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{U^T b} \\ \mathbf{c} \end{pmatrix}, \tag{58}$$

Notice that the transformation preserves any symmetry existing in the original system (57). The transformed matrix is finally given in the following where only the modified elements are shown and the sum is only over

the first N points in θ direction. The $\times$ symbol denotes unmodified elements.

$$\begin{pmatrix} \times & \times & \times & \times & \sum_j A_{1,j} & \times & \times \\ \times & \times & \times & \times & \sum_j A_{2,j} & \times & \times \\ \times & \times & \times & \times & \vdots & \times & \times \\ \times & \times & \times & \times & \sum_j A_{N-1,j} & \times & \times \\ \sum_i A_{i,1} & \sum_i A_{i,2} & \dots & \sum_i A_{i,N-1} & \sum_{i,j} A_{i,j} & \sum_i A_{i,N+1} & \dots \\ \times & \times & \times & \times & \sum_j A_{N+1,j} & \times & \times \\ \times & \times & \times & \times & \vdots & \times & \times \end{pmatrix} \tag{59}$$

Only the N^{th} column and the N^{th} row are affected by the transformation. Applying now the unicity condition (52) the final transformed system reads:

$$\begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & \vdots & 0 & 0 \\ 0 & 0 & \dots & 1 & 0 & 0 & 0 \\ 0 & 0 & \dots & 0 & \sum_{i,j} A_{i,j} & \sum_i A_{i,N+1} & \dots \\ 0 & 0 & \dots & 0 & \sum_j A_{N+1,j} & \times & \times \\ 0 & 0 & \dots & 0 & \vdots & \times & \times \end{pmatrix} \begin{pmatrix} \hat{u}_1 \\ \hat{u}_2 \\ \vdots \\ \hat{u}_{N-1} \\ \hat{u}_N \\ u_{N+1} \\ \vdots \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \sum_i b_i \\ b_{N+1} \\ \vdots \end{pmatrix}. \tag{60}$$

References

[1] C. de Boor, *A Practical Guide to Splines*, Applied Mathematical Sciences, vol. 27 (Springer, NY, 2001).

[2] G.H. Golub, C.F. Van Loan, *Matrix Computation, 3rd Edition*, p.5 (The John Hopkins University Press, 1996).