User Manual for the MDB-Spline Toolbox in MATLAB

HENDRIK SPELEERS
University of Rome Tor Vergata

---

1. INTRODUCTION

This guide explains the usage and functionality of the MATLAB toolbox on multi-degree B-splines (MDB-splines) accompanying the article

> Hendrik Speleers. Algorithm 999: Computation of Multi-Degree B-Splines. *ACM Trans. Math. Softw.* 45, 4, Article 43 (2019), 15 pages.

The toolbox has been developed in MATLAB R2016a but should work with other MATLAB versions as well. It can be downloaded from the ACM Collected Algorithms (CALGO). For its installation, just place the toolbox in any directory on your drive, and then add it to the MATLAB path.

2. MATLAB FUNCTIONS

The toolbox is divided in two parts: functions dealing with B-splines and functions dealing with MDB-splines.

2.1 B-splines

The main B-spline data-structure is called *B-spline patch* and identifies a certain B-spline space. It contains the open knot vector, spline degree, and spline dimension. Note that it would be sufficient to store only the knot vector; however, the other parameters simplify further operations.

The following MATLAB functions are provided for working with B-spline patches.

- `B_patch`: construction of a B-spline patch with open knot vector;
- `B_domain`: computation of the end points of the domain related to a patch;
- `B_greville`: computation of the Greville points;
- `B_evaluation_all`: evaluation of all B-splines in given points;
- `B_evaluation_spline`: evaluation of a spline in given points;
- `B_diffend_all`: full differentiation of all B-splines at one end point up to a given order;
- `B_differentiation_all`: differentiation of all B-splines in given points;
- `B_differentiation_spline`: differentiation of a spline in given points;
- `B_visualization_all`: visualization of all B-splines;
- `B_visualization_spline`: visualization of a spline;
- `B_conversion`: conversion from source to destination B-spline form.

2.1.1 *B_patch*

This function prepares the data-structure for a B-spline patch, starting from a sequence of polynomial segments of fixed degree and smoothness relations. The B-spline patch is a structure array containing the open knot vector `U`, spline degree `p`, and spline dimension `n`.

Syntax:

```
P = B_patch(p, xx, kk)
```

Input parameters:

`p` : B-spline degree

`xx` : vector of break points

`kk` : smoothness vector (optional)

Output parameters:

`P` : B-spline patch

Discussion:

The parameter `p` is a non-negative integer scalar, the parameter `xx` is a vector consisting of a strictly increasing sequence of real values (indicating the different segments), and the parameter `kk` can be a scalar or a vector whose elements are non-negative integers less than the value of `p`. If `kk` is a scalar, smoothness `kk` is imposed at the break point `xx(i+1)`, and if `kk` is a vector, smoothness `kk(i)` is imposed at the break point `xx(i+1)`, for `i = 1:length(xx)-2`. Hence, `length(kk)` should be equal to `1` or `length(xx)-2`. When no smoothness is specified, `kk = 0` is assumed.

Example:

Create a B-spline patch of degree 4 and smoothness C^2 defined on a domain partitioned in the two intervals $[0, 3]$ and $[3, 4]$:

```
>> P = B_patch(4, [0, 3, 4], 2)
P =
    p: 4
    n: 7
    U: [0 0 0 0 0 3 3 4 4 4 4 4]
```

2.1.2 *B_domain*

This function computes the end points of the domain specified by a given B-spline patch.

Syntax:

```
[a, b] = B_domain(P)
```

Input parameters:

`P` : B-spline patch

Output parameters:

`a` : left end point

`b` : right end point

Example:

Create a B-spline patch and show the end points of its domain:

```
>> P = B_patch(4, [0, 3, 4], 2);
>> [a, b] = B_domain(P)
a =
     0
b =
     4
```

2.1.3 *B_greville*

This function computes the Greville points of a given B-spline patch, i.e., the coefficients of the B-spline form of the identity function.

Syntax:

```
gg = B_greville(P)
```

Input parameters:

`P` : B-spline patch

Output parameters:

`gg` : vector of Greville points

Discussion:

Each element of the vector `gg` corresponds to a B-spline, so `length(gg)` equals `P.n`.

Example:

Create a B-spline patch and show its Greville points:

```
>> P = B_patch(4, [0, 3, 4], 2);
>> gg = B_greville(P)
gg =
         0    0.7500    1.5000    2.5000    3.5000    3.7500    4.0000
```

2.1.4 *B_evaluation_all*

This function evaluates all B-splines of a B-spline patch at a given set of points, and stores the corresponding values in a matrix.

Syntax:

```
M = B_evaluation_all(P, xx, cl)
```

Input parameters:

`P` : B-spline patch
`xx` : vector of evaluation points
`cl` : closed domain if true (optional)

Output parameters:

`M` : evaluation matrix

Discussion:

The parameter `cl` takes a boolean value. If `cl` is `false`, then the B-spline values are computed by the B-spline recurrence relation on the half-open domain of the patch and are zero outside; otherwise, at the right end point, they are computed by taking limits from the left (making the spline space symmetric on the closed domain). When no value is specified, `cl = true` is assumed. Each row in the resulting matrix `M` corresponds to a B-spline and each column to an evaluation point, so `size(M)` equals `[P.n, length(xx)]`.

Example:

Create a B-spline patch and evaluate all the corresponding B-splines at the break points of the patch:

```
>> P = B_patch(4, [0, 3, 4], 2);
>> M = B_evaluation_all(P, [0, 3, 4])
M =
    1.0000         0         0
         0         0         0
         0    0.0625         0
         0    0.3750         0
         0    0.5625         0
         0         0         0
         0         0    1.0000
```

2.1.5 *B_evaluation_spline*

This function evaluates a spline in B-spline form at a given set of points, and stores the corresponding values in a vector.

Syntax:

```
ss = B_evaluation_spline(P, cc, xx)
```

Input parameters:

`P` : B-spline patch

`cc` : vector of coefficients

`xx` : vector of evaluation points

Output parameters:

`ss` : vector of spline evaluation values

Discussion:

Each element of the vector `cc` corresponds to a B-spline in the B-spline patch. Hence, `length(cc)` should be equal to `P.n`. Each element of the resulting vector `ss` corresponds to an evaluation point, so `length(ss)` equals `length(xx)`.

Example:

Create a B-spline patch and a vector of coefficients, and then evaluate the corresponding spline in B-spline form at the break points of the patch:

```
>> P = B_patch(4, [0, 3, 4], 2);
>> cc = [1, 2, 3, 4, 4, 3, 2];
>> ss = B_evaluation_spline(P, cc, [0, 3, 4])
ss =
    1.0000    3.9375    2.0000
```

2.1.6 *B_diffend_all*

This function evaluates all the derivatives, up to a certain order r, of all B-splines of a B-spline patch at one of the two end points of the domain, and stores the corresponding values in a matrix.

Syntax:

```
K = B_diffend_all(P, r, el)
```

Input parameters:

`P` : B-spline patch
`r` : maximum order of derivative
`el` : left end if true, right end otherwise (optional)

Output parameters:

`K` : differentiation matrix at end point up to `r`-th order

Discussion:

The parameter `r` is a non-negative integer. The parameter `el` takes a boolean value. If `el` is `true`, then the left end point of the domain is selected; otherwise, the right end point. When no value is specified, `el = true` is assumed. Each row in the resulting matrix `K` corresponds to a B-spline and each column to a derivative, so `size(K)` equals `[P.n, r+1]`.

Example:

Create a B-spline patch and evaluate all derivatives, up to fourth order, of all the corresponding B-splines at the right end point of the domain:

```
>> P = B_patch(4, [0, 3, 4], 2);
>> K = B_diffend_all(P, 4, false)
```

```
K =

         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0    1.5000
         0         0         0   -6.0000  -15.0000
         0         0   12.0000   54.0000   85.5000
         0   -4.0000  -24.0000  -72.0000  -96.0000
    1.0000    4.0000   12.0000   24.0000   24.0000
```

2.1.7 *B_differentiation_all*

This function evaluates the r-th order derivative of all B-splines of a B-spline patch at a given set of points, and stores the corresponding values in a matrix.

Syntax:

```
M = B_differentiation_all(P, r, xx, cl)
```

Input parameters:

`P` : B-spline patch

`r` : order of derivative

`xx` : vector of evaluation points

`cl` : closed domain if true (optional)

Output parameters:

`M` : differentiation matrix

Discussion:

The parameter `r` is a non-negative integer. The parameter `cl` takes a boolean value. If `cl` is `false`, then the B-spline values are computed by the B-spline recurrence relation on the half-open domain of the patch and are zero outside; otherwise, at the right end point, they are computed by taking limits from the left (making the spline space symmetric on the closed domain). When no value is specified, `cl = true` is assumed. Each row in the resulting matrix `M` corresponds to a B-spline and each column to an evaluation point, so `size(M)` equals `[P.n, length(xx)]`.

Example:

Create a B-spline patch and evaluate the first derivative of all the corresponding B-splines at the break points of the patch:

```
>> P = B_patch(4, [0, 3, 4], 2);
>> M = B_differentiation_all(P, 1, [0, 3, 4])
```

```
M =
   -1.3333         0         0
    1.3333         0         0
         0   -0.2500         0
         0   -0.5000         0
         0    0.7500         0
         0         0   -4.0000
         0         0    4.0000
```

2.1.8 *B_differentiation_spline*

This function evaluates the r-th order derivative of a spline in B-spline form at a given set of points, and stores the corresponding values in a vector.

Syntax:

```
ss = B_differentiation_spline(P, r, cc, xx)
```

Input parameters:

`P` : B-spline patch

`r` : order of derivative

`cc` : vector of coefficients

`xx` : vector of evaluation points

Output parameters:

`ss` : vector of `r`-th order derivative spline values

Discussion:

The parameter `r` is a non-negative integer. Each element of the vector `cc` corresponds to a B-spline in the B-spline patch. Hence, `length(cc)` should be equal to `P.n`. Each element of the resulting vector `ss` corresponds to an evaluation point, so `length(ss)` equals `length(xx)`.

Example:

Create a B-spline patch and a vector of coefficients, and then evaluate the first derivative of the corresponding spline in B-spline form at the break points of the patch:

```
>> P = B_patch(4, [0, 3, 4], 2);
>> cc = [1, 2, 3, 4, 4, 3, 2];
>> ss = B_differentiation_spline(P, 1, cc, [0, 3, 4])
ss =
    1.3333    0.2500   -4.0000
```

2.1.9 *B_visualization_all*

This function visualizes all B-splines of a B-spline patch.

Syntax:

```
B_visualization_all(P, n, specs)
```

Input parameters:

`P` : B-spline patch
`n` : number of evaluation points (optional)
`specs` : pass any number of plot specifications (optional)

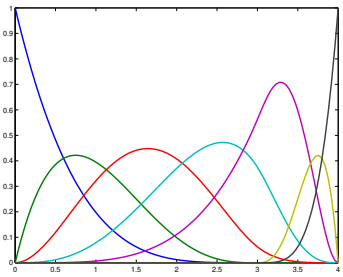
Discussion:

The parameter `n` is a positive integer. When no value is specified, `n = 100` is assumed. The parameter `specs` allows for any number of input arguments, which are passed on to the function `plot`. We refer the reader to the documentation of `plot` for all the plotting options.

Example:

Create a B-spline patch and plot all the corresponding B-splines:

```
>> P = B_patch(4, [0, 3, 4], 2);
>> B_visualization_all(P, 100, 'LineWidth', 2);
```



2.1.10 *B_visualization_spline*

This function visualizes a spline in B-spline form.

Syntax:

```
B_visualization_spline(P, cc, n, specs)
```

Input parameters:

`P` : B-spline patch
`cc` : vector of coefficients
`n` : number of evaluation points (optional)
`specs` : pass any number of plot specifications (optional)

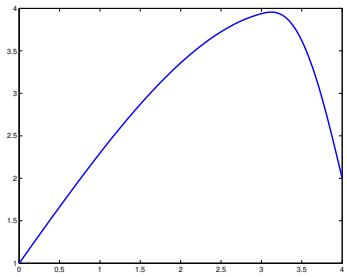
Discussion:

Each element of the vector `cc` corresponds to a B-spline in the B-spline patch. Hence, `length(cc)` should be equal to `P.n`. The parameter `n` is a positive integer. When no value is specified, `n = 100` is assumed. The parameter `specs` allows for any number of input arguments, which are passed on to the function `plot`. We refer the reader to the documentation of `plot` for all the plotting options.

Example:

Create a B-spline patch and a vector of coefficients, and then plot the corresponding spline in B-spline form:

```
>> P = B_patch(4, [0, 3, 4], 2);
>> cc = [1, 2, 3, 4, 4, 3, 2];
>> B_visualization_spline(P, cc, 100, 'LineWidth', 2);
```



2.1.11 *B_conversion*

This function converts a spline in B-spline form into another B-spline form. The conversion is exact when the source and destination B-spline patches imply nested spaces.

Syntax:

```
ccd = B_conversion(Pd, Ps, ccs)
```

Input parameters:

`Pd` : destination B-spline patch

`Ps` : source B-spline patch

`ccs` : source coefficient vector

Output parameters:

`ccd` : destination coefficient vector

Discussion:

The B-spline patches `Ps` and `Pd` preferably share the same break points for the best conversion results. Each element of the vector `ccs` corresponds to a B-spline in the B-

spline patch `Ps`. Hence, `length(ccs)` should be equal to `Ps.n`. Similarly, each element of the resulting vector `ccd` corresponds to a B-spline in the B-spline patch `Pd`.

Example:

Create a B-spline patch of degree 4 and a vector of coefficients; then, raise the degree to 7 and compute the coefficients of the new B-spline form:

```
>> Ps = B_patch(4, [0, 3, 4], 2);
>> ccs = [1, 2, 3, 4, 4, 3, 2];
>> Pd = B_patch(7, [0, 3, 4], 2);
>> ccd = B_conversion(Pd, Ps, ccs)
ccd =
  Columns 1 through 7
    1.0000    1.5714    2.1429    2.6857    3.1696    3.5625    3.9196
  Columns 8 through 13
    3.9911    3.8839    3.6000    3.1429    2.5714    2.0000
```

2.2 MDB-splines

The main MDB-spline data-structure is called *MDB-spline multi-patch*. It contains a vector of B-spline patches and the corresponding cumulative dimension.

The following MATLAB functions are provided for core operations on MDB-spline multi-patches.

- `MDB_patch`: construction of an MDB-spline multi-patch from B-spline segments;
- `MDB_patch_poly`: construction of an MDB-spline multi-patch from polynomial segments;
- `MDB_extraction`: computation of the multi-degree spline extraction matrix;
- `MDB_extraction_periodic`: computation of the multi-degree spline extraction matrix with periodicity;
- `MDB_nullspace`: computation of the left null-space of a column of matrix `L` in the extraction procedure; this is an auxiliary function for the functions `MDB_extraction` and `MDB_extraction_periodic`, and has no stand-alone usage;
- `MDB_extraction_local`: computation of the local multi-degree spline extraction matrix corresponding to a single patch;

Furthermore, the following MATLAB functions are provided for working with MDB-splines. Thanks to the multi-degree spline extraction operator, their implementation can be easily redirected to their B-spline analogues described in Section 2.1.

- `MDB_domain`: computation of the end points of the domain related to a multi-patch;
- `MDB_greville`: computation of the multi-degree Greville points;
- `MDB_evaluation_all`: evaluation of all MDB-splines in given points;
- `MDB_evaluation_spline`: evaluation of a multi-degree spline in given points;
- `MDB_differentiation_all`: differentiation of all MDB-splines in given points;
- `MDB_differentiation_spline`: differentiation of a multi-degree spline in given points;

- `MDB_visualization_all`: visualization of all MDB-splines;
- `MDB_visualization_spline`: visualization of a multi-degree spline;
- `MDB_conversion`: conversion from source to destination MDB-spline form.

2.2.1 *MDB_patch*

This function prepares the data-structure for an MDB-spline multi-patch, starting from a sequence of B-spline segments. The MDB-spline multi-patch is a structure array containing a vector of B-spline patches `P` and the corresponding cumulative dimension `mu`.

Syntax:

```
MP = MDB_patch(PP)
```

Input parameters:

`PP` : vector of B-spline patches

Output parameters:

`MP` : MDB-spline multi-patch

Example:

Create an MDB-spline multi-patch consisting of two B-spline patches with different degrees (3 and 4) but with the same smoothness C^2:

```
>> P1 = B_patch(3, [0, 1, 3], 2);
>> P2 = B_patch(4, [3, 4, 6], 2);
>> MP = MDB_patch([P1, P2])
MP =
     P: [1x2 struct]
    mu: [0 5 12]
>> MP.P(1)
ans =
    p: 3
    n: 5
    U: [0 0 0 0 1 3 3 3 3]
>> MP.P(2)
ans =
    p: 4
    n: 7
    U: [3 3 3 3 3 4 4 6 6 6 6 6]
```

2.2.2 *MDB_patch_poly*

This function prepares the data-structure for an MDB-spline multi-patch, starting from a sequence of polynomial segments and smoothness relations. The MDB-spline multi-patch is a structure array containing a vector of B-spline patches `P` and the corresponding cumulative dimension `mu`. Consecutive polynomial segments of the same degree are merged into a single B-spline patch (unless specified otherwise).

Syntax:

```
[MP, rr] = MDB_patch_poly(pp, xx, kk, mg)
```

Input parameters:

`pp` : vector of polynomial degrees
`xx` : vector of break points
`kk` : smoothness vector (optional)
`mg` : same degree merged if true (optional)

Output parameters:

`MP` : MDB-spline multi-patch
`rr` : MDB-spline smoothness vector (optional)

Discussion:

The parameter `pp` is a vector consisting of non-negative integer values, and the parameter `xx` is a vector consisting of a strictly increasing sequence of real values (indicating the different segments), such that `length(pp)` equals `length(xx)-1`. The parameter `kk` can be a scalar or a vector whose elements are non-negative integers. If `kk` is a scalar, then `kk` should be less than `min(pp)`, and smoothness `kk` is imposed at the break point `xx(i+1)` for `i = 1:length(xx)-2`. On the other hand, if `kk` is a vector, then `kk(i)` should be less than `min(pp(i),pp(i+1))`, and smoothness `kk(i)` is imposed at the break point `xx(i+1)` for `i = 1:length(xx)-2`. Hence, `length(kk)` should be equal to `1` or `length(xx)-2`. When no smoothness is specified, `kk = 0` is assumed. The parameter `mg` takes a boolean value. If `mg` is `true`, then consecutive polynomial segments of the same degree are merged into a single B-spline patch; otherwise, they are not merged. When no value is specified, `mg = true` is assumed. The resulting vector `rr` represents the corresponding smoothness between the B-spline patches, so `length(rr)` equals `length(MP.P)-1`.

Example:

Create an MDB-spline multi-patch consisting of four polynomial segments with different degrees (3 and 4) that are connected with smoothness C^2:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2)
MP =
     P: [1x2 struct]
    mu: [0 5 12]
rr =
     2
```

```
>> MP.P(1)
ans =
    p: 3
    n: 5
    U: [0 0 0 0 1 3 3 3 3]
>> MP.P(2)
ans =
    p: 4
    n: 7
    U: [3 3 3 3 3 4 4 6 6 6 6 6]
```

2.2.3 *MDB_extraction*

This function computes the multi-degree spline extraction matrix representing a set of MDB-splines in terms of the B-splines related to a given sequence of B-spline patches.

Syntax:

```
H = MDB_extraction(MP, rr)
```

Input parameters:

`MP` : MDB-spline multi-patch

`rr` : MDB-spline smoothness vector (optional)

Output parameters:

`H` : extraction matrix

Discussion:

The parameter `rr` can be a scalar or a vector whose elements are integers. If `rr` is a scalar, smoothness `rr` is imposed between all consecutive B-spline patches. On the other hand, if `rr` is a vector, smoothness `rr(i)` is imposed between B-spline patches `MP.P(i)` and `MP.P(i+1)` for `i = 1:length(MP.P)-1`. Hence, `length(rr)` should be equal to `1` or `length(MP.P)-1`. A negative value indicates no active smoothness imposition. When no smoothness is specified, `rr = 0` is assumed. The resulting matrix `H` is encoded in sparse format; each row in `H` corresponds to an MDB-spline and each column to a B-spline in one of the B-spline patches, so `size(H,2)` equals `MP.mu(end)`.

Example:

Create an MDB-spline multi-patch with related smoothness vector, and then compute the multi-degree spline extraction matrix:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> H = MDB_extraction(MP, rr);
>> Hfull = full(H)
```

```
Hfull =
  Columns 1 through 7
    1.0000         0         0         0         0         0         0
         0    1.0000         0         0         0         0         0
         0         0    1.0000    0.1864    0.0508    0.0508         0
         0         0         0    0.8136    0.4158    0.4158    0.2667
         0         0         0         0    0.5333    0.5333    0.7333
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
  Columns 8 through 12
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
    1.0000         0         0         0         0
         0    1.0000         0         0         0
         0         0    1.0000         0         0
         0         0         0    1.0000         0
         0         0         0         0    1.0000
```

2.2.4 *MDB_extraction_periodic*

This function computes the periodic multi-degree spline extraction matrix representing a set of periodic MDB-splines in terms of the B-splines related to a given sequence of B-spline patches.

Syntax:

```
H = MDB_extraction_periodic(MP, rr, rp)
```

Input parameters:

`MP` : MDB-spline multi-patch
`rr` : MDB-spline smoothness vector (optional)
`rp` : periodicity smoothness (optional)

Output parameters:

`H` : extraction matrix

Discussion:

The parameter `rr` can be a scalar or a vector whose elements are integers. If `rr` is a scalar, smoothness `rr` is imposed between all consecutive B-spline patches. On the other hand, if `rr` is a vector, smoothness `rr(i)` is imposed between B-spline patches `MP.P(i)` and `MP.P(i+1)` for `i = 1:length(MP.P)-1`. Hence, `length(rr)` should be equal to `1`

or `length(MP.P)-1`. A negative value indicates no active smoothness imposition. When no smoothness is specified, `rr = 0` is assumed. The parameter `rp` should be an integer scalar less than half the dimension (floored) of the related non-periodic MDB-spline space. When no periodicity smoothness is specified, `rp = -1` is assumed. The resulting matrix `H` is encoded in sparse format; each row in `H` corresponds to a periodic MDB-spline and each column to a B-spline in one of the B-spline patches, so `size(H,2)` equals `MP.mu(end)`.

Example:

Create an MDB-spline multi-patch with related smoothness vector and specify periodicity smoothness; then, compute the periodic multi-degree spline extraction matrix:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> Hper = MDB_extraction_periodic(MP, rr, 2);
>> Hfull = full(Hper)
Hfull =
  Columns 1 through 7
    0.1667         0         0         0         0         0         0
    0.6398    0.6774         0         0         0         0         0
    0.1935    0.3226    1.0000    0.1864    0.0508    0.0508         0
         0         0         0    0.8136    0.4158    0.4158    0.2667
         0         0         0         0    0.5333    0.5333    0.7333
         0         0         0         0         0         0         0
  Columns 8 through 12
         0         0    1.0000    0.4167    0.1667
         0         0         0    0.5833    0.6398
         0         0         0         0    0.1935
         0         0         0         0         0
    1.0000         0         0         0         0
         0    1.0000         0         0         0
```

2.2.5 *MDB_nullspace*

This auxiliary function computes the left null-space of a column of the matrix `L` used in `MDB_extraction` and `MDB_extraction_periodic`.

Syntax:

```
Hbar = MDB_nullspace(ll)
```

Input parameters:

`ll` : a column of L

Output parameters:

`Hbar` : null-space matrix of `ll`

Discussion:

This function has no stand-alone usage.

2.2.6 *MDB_extraction_local*

This function returns the local extraction matrix corresponding to a single B-spline patch.

Syntax:

```
Hl = MDB_extraction_local(MP, H, ip)
```

Input parameters:

`MP` : MDB-spline multi-patch
`H` : extraction matrix
`ip` : index of patch

Output parameters:

`Hl` : local extraction matrix

Discussion:

The extraction matrix `H` should be deduced from the MDB-spline multi-patch `MP` and incorporates the smoothness. The parameter `ip` takes an integer value between 1 and `length(MP.P)`. The resulting matrix `Hl` is encoded in sparse format; each row in `Hl` corresponds to an MDB-spline and each column to a B-spline in the selected B-spline patch, so `size(Hl)` equals `[size(H,1), MP.P(ip).n]`.

Example:

Create an MDB-spline multi-patch with related smoothness vector, and show the Bézier extraction matrix corresponding to the first patch:

```
>> [MP, rr] = MDB_patch_poly([3, 4, 4], [1, 3, 4, 6], 2, false);
>> H = MDB_extraction(MP, rr);
>> Hl = MDB_extraction_local(MP, H, 1);
>> B = full(Hl(any(Hl, 2), :))
B =
    1.0000         0         0         0
         0    1.0000    0.2558    0.0698
         0         0    0.7442    0.3969
         0         0         0    0.5333
```

2.2.7 *MDB_domain*

This function computes the end points of the domain specified by a given MDB-spline multi-patch.

Syntax:

```
[a, b] = MDB_domain(MP)
```

Input parameters:

`MP` : MDB-spline multi-patch

Output parameters:

`a` : left end point
`b` : right end point

Example:

Create an MDB-spline multi-patch and show the end points of its domain:

```
>> MP = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> [a, b] = MDB_domain(MP)
a =
     0
b =
     6
```

2.2.8 *MDB_greville*

This function computes the Greville points of a given MDB-spline multi-patch with smoothness, i.e., the coefficients of the MDB-spline form of the identity function.

Syntax:

```
gg = MDB_greville(MP, H)
```

Input parameters:

`MP` : MDB-spline multi-patch
`H` : extraction matrix

Output parameters:

`gg` : vector of Greville points

Discussion:

The extraction matrix `H` should be deduced from the MDB-spline multi-patch `MP` and incorporates the smoothness. Each element of the vector `gg` corresponds to an MDB-spline, so `length(gg)` equals `size(H,1)`.

Example:

Create an MDB-spline multi-patch with related smoothness vector, and show its Greville points:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> H = MDB_extraction(MP, rr);
>> gg = MDB_greville(MP, H)
gg =
  Columns 1 through 7
         0    0.3333    1.3333    2.5625    3.5000    4.2500    5.0000
  Columns 8 through 9
    5.5000    6.0000
```

2.2.9 *MDB_evaluation_all*

This function evaluates all (periodic) MDB-splines of an MDB-spline multi-patch with smoothness at a given set of points, and stores the corresponding values in a matrix.

Syntax:

```
M = MDB_evaluation_all(MP, H, xx)
```

Input parameters:

`MP` : MDB-spline multi-patch
`H` : extraction matrix
`xx` : vector of evaluation points

Output parameters:

`M` : evaluation matrix

Discussion:

The extraction matrix `H` should be deduced from the MDB-spline multi-patch `MP` and incorporates the smoothness. Each row in the resulting matrix `M` corresponds to an MDB-spline and each column to an evaluation point, so `size(M)` equals `[size(H,1), length(xx)]`.

Example:

Create an MDB-spline multi-patch with related smoothness vector, and evaluate all the corresponding MDB-splines at the break points of the multi-patch:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> H = MDB_extraction(MP, rr);
>> M = MDB_evaluation_all(MP, H, [0, 1, 3, 4, 6])
M =
    1.0000         0         0         0         0
         0    0.4444         0         0         0
         0    0.4652    0.0508         0         0
         0    0.0904    0.4158         0         0
         0         0    0.5333    0.4444         0
         0         0         0    0.4444         0
         0         0         0    0.1111         0
         0         0         0         0         0
         0         0         0         0    1.0000
```

Now, do the same with periodic MDB-splines:

```
>> Hper = MDB_extraction_periodic(MP, rr, 2);
>> Mper = MDB_evaluation_all(MP, Hper, [0, 1, 3, 4, 6])
```

```
Mper =
    0.1667         0         0    0.1111    0.1667
    0.6398    0.3011         0         0    0.6398
    0.1935    0.6085    0.0508         0    0.1935
         0    0.0904    0.4158         0         0
         0         0    0.5333    0.4444         0
         0         0         0    0.4444         0
```

2.2.10 *MDB_evaluation_spline*

This function evaluates a spline in (periodic) MDB-spline form at a given set of points, and stores the corresponding values in a vector.

Syntax:

```
ss = MDB_evaluation_spline(MP, H, cc, xx)
```

Input parameters:

`MP` : MDB-spline multi-patch
`H` : extraction matrix
`cc` : vector of coefficients
`xx` : vector of evaluation points

Output parameters:

`ss` : vector of spline evaluation values

Discussion:

The extraction matrix `H` should be deduced from the MDB-spline multi-patch `MP` and incorporates the smoothness. Each element of the vector `cc` corresponds to an MDB-spline. Hence, `length(cc)` should be equal to `size(H,1)`. Each element of the resulting vector `ss` corresponds to an evaluation point, so `length(ss)` equals `length(xx)`.

Example:

Create an MDB-spline multi-patch with related smoothness vector and a vector of coefficients, and then evaluate the corresponding spline in MDB-spline form at the break points of the multi-patch:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> H = MDB_extraction(MP, rr);
>> cc = [1, 2, 3, 4, 2, 2, 4, 3, 2];
>> ss = MDB_evaluation_spline(MP, H, cc, [0, 1, 3, 4, 6])
ss =
    1.0000    2.6460    2.8825    2.2222    2.0000
```

Now, do the same with periodic MDB-splines:

```
>> Hper = MDB_extraction_periodic(MP, rr, 2);
```

```
>> ccper = [1, 2, 3, 4, 3, 2];
>> ssper = MDB_evaluation_spline(MP, Hper, ccper, [0, 1, 3, 4, 6])
ssper =
    2.0269    2.7893    3.4158    2.3333    2.0269
```

2.2.11 *MDB_differentiation_all*

This function evaluates the r-th order derivative of all (periodic) MDB-splines of an MDB-spline multi-patch with smoothness at a given set of points, and stores the corresponding values in a matrix.

Syntax:

```
M = MDB_differentiation_all(MP, H, r, xx)
```

Input parameters:

`MP` : MDB-spline multi-patch
`H` : extraction matrix
`r` : order of derivative
`xx` : vector of evaluation points

Output parameters:

`M` : differentiation matrix

Discussion:

The extraction matrix `H` should be deduced from the MDB-spline multi-patch `MP` and incorporates the smoothness. The parameter `r` is a non-negative integer. Each row in the resulting matrix `M` corresponds to an MDB-spline and each column to an evaluation point, so `size(M)` equals `[size(H,1), length(xx)]`.

Example:

Create an MDB-spline multi-patch with related smoothness vector, and evaluate the first derivative of all the corresponding MDB-splines at the break points of the multi-patch:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> H = MDB_extraction(MP, rr);
>> M = MDB_differentiation_all(MP, H, 1, [0, 1, 3, 4, 6])
M =
   -3.0000         0         0         0         0
    3.0000   -0.6667         0         0         0
         0    0.3955   -0.2034         0         0
         0    0.2712   -0.5966         0         0
         0         0    0.8000   -0.8889         0
         0         0         0    0.4444         0
         0         0         0    0.4444         0
         0         0         0         0   -2.0000
         0         0         0         0    2.0000
```

2.2.12 *MDB_differentiation_spline*

This function evaluates the r-th order derivative of a spline in (periodic) MDB-spline form at a given set of points, and stores the corresponding values in a vector.

Syntax:

```
ss = MDB_differentiation_spline(MP, H, r, cc, xx)
```

Input parameters:

`MP` : MDB-spline multi-patch
`H` : extraction matrix
`r` : order of derivative
`cc` : vector of coefficients
`xx` : vector of evaluation points

Output parameters:

`ss` : vector of `r`-th order derivative spline values

Discussion:

The extraction matrix `H` should be deduced from the MDB-spline multi-patch `MP` and incorporates the smoothness. The parameter `r` is a non-negative integer. Each element of the vector `cc` corresponds to an MDB-spline. Hence, `length(cc)` should be equal to `size(H,1)`. Each element of the resulting vector `ss` corresponds to an evaluation point, so `length(ss)` equals `length(xx)`.

Example:

Create an MDB-spline multi-patch with related smoothness vector and a vector of coefficients, and then evaluate the first derivative of the corresponding spline in MDB-spline form at the break points of the multi-patch:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> H = MDB_extraction(MP, rr);
>> cc = [1, 2, 3, 4, 2, 2, 4, 3, 2];
>> ss = MDB_differentiation_spline(MP, H, 1, cc, [0, 1, 3, 4, 6])
ss =
    3.0000    0.9379   -1.3966    0.8889   -2.0000
```

2.2.13 *MDB_visualization_all*

This function visualizes all (periodic) MDB-splines of an MDB-spline multi-patch with smoothness.

Syntax:

```
MDB_visualization_all(MP, H, n, specs)
```

Input parameters:

`MP` : MDB-spline multi-patch
`H` : extraction matrix
`n` : number of evaluation points (optional)
`specs` : pass any number of plot specifications (optional)

Discussion:

The extraction matrix `H` should be deduced from the MDB-spline multi-patch `MP` and incorporates the smoothness. The parameter `n` is a positive integer. When no value is specified, `n = 100` is assumed. The parameter `specs` allows for any number of input arguments, which are passed on to the function `plot`. We refer the reader to the documentation of `plot` for all the plotting options.

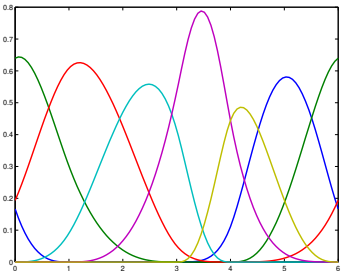
Example:

Create an MDB-spline multi-patch with related smoothness vector, and plot all the corresponding MDB-splines:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> H = MDB_extraction(MP, rr);
>> MDB_visualization_all(MP, H, 100, 'LineWidth', 2);
```

Now, do the same with periodic MDB-splines:

```
>> Hper = MDB_extraction_periodic(MP, rr, 2);
>> MDB_visualization_all(MP, Hper, 100, 'LineWidth', 2);
```



2.2.14 *MDB_visualization_spline*

This function visualizes a spline in (periodic) MDB-spline form.

Syntax:

```
MDB_visualization_spline(MP, H, cc, n, specs)
```

Input parameters:

`MP`	: MDB-spline multi-patch
`H`	: extraction matrix
`cc`	: vector of coefficients
`n`	: number of evaluation points (optional)
`specs`	: pass any number of plot specifications (optional)

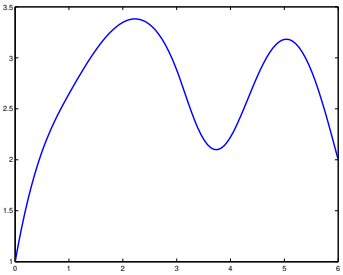
Discussion:

The extraction matrix `H` should be deduced from the MDB-spline multi-patch `MP` and incorporates the smoothness. Each element of the vector `cc` corresponds to an MDB-spline. Hence, `length(cc)` should be equal to `size(H,1)`. The parameter `n` is a positive integer. When no value is specified, `n = 100` is assumed. The parameter `specs` allows for any number of input arguments, which are passed on to the function `plot`. We refer the reader to the documentation of `plot` for all the plotting options.

Example:

Create an MDB-spline multi-patch with related smoothness vector and a vector of coefficients, and then plot the corresponding spline in MDB-spline form:

```
>> [MP, rr] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> H = MDB_extraction(MP, rr);
>> cc = [1, 2, 3, 4, 2, 2, 4, 3, 2];
>> MDB_visualization_spline(MP, H, cc, 100, 'LineWidth', 2);
```



2.2.15 *MDB_conversion*

This function converts a spline in (periodic) MDB-spline form into another (periodic) MDB-spline form. The conversion is exact when the source and destination MDB-spline multi-patches with smoothness imply nested spaces.

Syntax:

```
ccd = MDB_conversion(MPd, Hd, MPs, Hs, ccs)
```

Input parameters:

`MPd` : destination MDB-spline multi-patch

`Hd` : destination extraction matrix

`MPs` : source MDB-spline multi-patch

`Hs` : source extraction matrix

`ccs` : source coefficient vector

Output parameters:

`ccd` : destination coefficient vector

Discussion:

The MDB-spline multi-patches `MPs` and `MPd` must share the same number of B-spline patches, and preferably `MPd` contains also the same break points as `MPs` for the best conversion results. The extraction matrix `Hs` should be deduced from `MPs` and incorporates the smoothness. Similarly, the extraction matrix `Hd` should be deduced from `MPd` and incorporates the smoothness. Each element of the vector `ccs` corresponds to an MDB-spline related to `Hs`. Hence, `length(ccs)` should be equal to `size(Hs,1)`. Similarly, each element of the resulting vector `ccd` corresponds to an MDB-spline related to `Hd`.

Example:

Create an MDB-spline multi-patch of multi-degree $(3, 4)$ with related smoothness vector and a vector of coefficients; then, raise the multi-degree to $(5, 7)$ and compute the coefficients of the new MDB-spline form:

```
>> [MPs, rrs] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2);
>> Hs = MDB_extraction(MPs, rrs);
>> ccs = [1, 2, 3, 4, 2, 2, 4, 3, 2];
>> [MPd, rrd] = MDB_patch_poly([5, 5, 7, 7], [0, 1, 3, 4, 6], 2);
>> Hd = MDB_extraction(MPd, rrd);
>> ccd = MDB_conversion(MPd, Hd, MPs, Hs, ccs)
ccd =
  Columns 1 through 7
    1.0000    1.6000    2.0000    2.2646    2.8460    3.3714    3.5577
  Columns 8 through 14
    3.4016    2.4308    2.2081    2.0673    2.0317    2.2222    2.9841
  Columns 15 through 19
    3.3778    3.4476    3.1429    2.5714    2.0000
```

Now, keep the same multi-degree of the original spline but lower its smoothness, and compute the coefficients of the new MDB-spline form:

```
>> [MPe, rre] = MDB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 1);
>> He = MDB_extraction(MPe, rre);
>> cce = MDB_conversion(MPe, He, MPs, Hs, ccs)
cce =
  Columns 1 through 7
    1.0000    2.0000    2.3333    3.2712    3.8136    2.5333    2.0000
  Columns 8 through 12
    2.0000    2.6667    4.0000    3.0000    2.0000
```

Finally, find an approximation of the original spline using a lower multi-degree $(2, 3)$:

```
>> [MPf, rrf] = MDB_patch_poly([2, 2, 2, 3, 3], [0, 1, 2, 3, 4, 6], 1);
>> Hf = MDB_extraction(MPf, rrf);
>> ccf = MDB_conversion(MPf, Hf, MPs, Hs, ccs)
ccf =
  Columns 1 through 7
    1.0000    2.3034    3.1291    3.5633    2.3867    1.8105    3.1871
  Columns 8 through 9
    3.7606    2.0000
```

Lowering the degree does not preserve the exact shape of the original spline, but it forms a reasonable approximation. Make a visual comparison between the original spline (blue) and the lower-degree spline (red):

```
>> MDB_visualization_spline(MPs, Hs, ccs, 50, 'LineWidth', 2, ...
>>             'Marker', 'o', 'MarkerSize', 10, 'Color', 'blue');
>> hold on;
>> MDB_visualization_spline(MPf, Hf, ccf, 50, 'LineWidth', 2, ...
>>             'Marker', '*', 'MarkerSize', 10, 'Color', 'red');
>> hold off;
```