User Manual for the MDTB-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 Tchebycheffian B-splines (MDTB-splines) accompanying the article

> Hendrik Speleers. Algorithm 1020: Computation of Multi-Degree Tchebycheffian B-Splines. *ACM Trans. Math. Softw.* 48, 1, Article 12 (2022), 31 pages.

The toolbox has been developed in MATLAB R2018a 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.

It is a redesigned and object-oriented extension of the MDB-spline toolbox 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 syntax of the new MDTB-spline toolbox is (almost fully) compatible with the one of the original toolbox and all the original function calls are still available (under the minor restriction that the names start now with "`(MD)TB_*`" instead of "`(MD)B_*`", so as to emphasize the Tchebycheffian nature of the extension). The user manual is also organized with this in mind, following the same style for describing the MATLAB functions.

The toolbox assumes that the input parameters specified by the user lead to valid extended complete Tchebycheff spaces (ECT-spaces) and valid multi-degree Tchebycheffian spline (MDT-spline) spaces, i.e., there exists a valid Tchebycheffian Bernstein basis and a valid MDTB-spline basis, respectively.

2. MATLAB CLASS DIAGRAM

The internal class diagram of the MDTB-spline toolbox is shown in Figure 1. The central class in the toolbox is the class `MDTB_patch`, which provides functionality for computing the MDTB-spline extraction matrix, both in the periodic and non-periodic spline setting. Furthermore, it allows for evaluating, differentiating, and visualizing the obtained MDTB-spline basis functions and any MDT-spline function represented in such basis.

The class `MDTB_patch` is built upon the class `TB_patch`, which mainly identifies a local ECT-space. Each object of type `MDTB_patch` contains a (heterogeneous) array of objects of type `TB_patch`. In order to reflect the heterogeneous nature of ECT-spaces, the class `TB_patch` is *abstract*. The functionality of evaluation and differentiation of the Tchebycheffian Bernstein basis functions is delegated to (specialized) child classes, as well as the option to provide a separate implementation for end-point derivatives because of their importance in the MDTB-spline framework. Manipulation of functions represented in the Bernstein basis and their visualization is handled in the class `TB_patch` itself.

As indicated in Figure 1, there are several child classes of the class `TB_patch` available in the toolbox. They provide functionality to work with general ECT-spaces based on

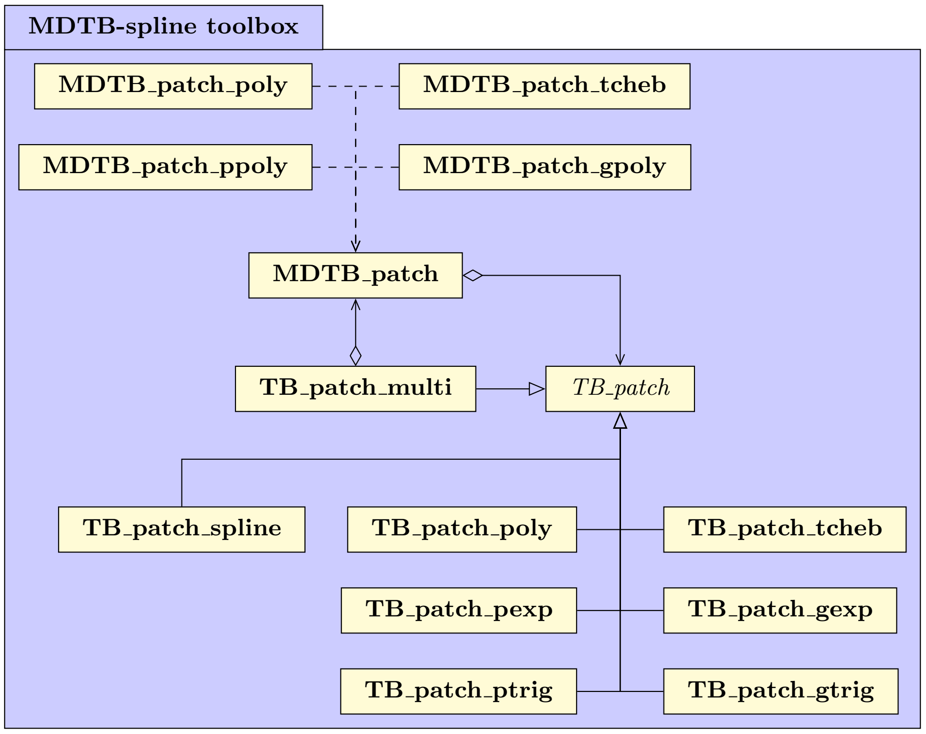


Fig. 1. Class diagram of the MDTB-spline toolbox in Matlab.

constant-coefficient linear differential operators (class `TB_patch_tcheb`), algebraic polynomial spaces (classes `TB_patch_poly` and `TB_patch_spline`), other polynomial-type spaces (classes `TB_patch_pexp` and `TB_patch_ptrig`), and generalized polynomial spaces (classes `TB_patch_gexp` and `TB_patch_gtrig`). Thanks to the object-oriented structure of the toolbox, other ECT-spaces and/or specialized implementations can be easily incorporated by adding new child classes of `TB_patch`. Such child class must provide at least an implementation for the methods `TB_evaluation_all`, `TB_differentiation_all`, and `TB_diffend_all`.

The purpose of the class `TB_patch_multi` is to encapsulate an object of the class `MDTB_patch` and a non-periodic extraction matrix such that the corresponding multi-degree spline space can be treated as if it is an instance of type `TB_patch`. In this way, already constructed multi-degree spline spaces can be embedded into larger multi-degree spline spaces without the need for recomputing them.

Finally, the classes `MDTB_patch_tcheb`, `MDTB_patch_poly`, `MDTB_patch_ppoly`, and `MDTB_patch_gpoly` are factory classes for `MDTB_patch`. They provide simplified functionality to initialize objects of type `MDTB_patch` consisting of local ECT-spaces based on constant-coefficient linear differential operators, algebraic polynomial spaces, other polynomial-type spaces, and generalized polynomial spaces, respectively.

In the following two sections, we discuss all the functions available in the MDTB-spline toolbox, divided into two groups: functions dealing with TB-spline patches and functions dealing with MDTB-spline multi-patches.

3. TB-SPLINE PATCHES

The main single-patch data-structure is called *TB-spline patch*, and it identifies either an ECT-space equipped with a Tchebycheffian Bernstein basis or a Tchebycheffian spline space equipped with a B-spline basis. It contains the TB-spline degree, space dimension, interval vector, and possibly also some local parameters for the specification of the basis computation. The term *TB-spline* means either a Tchebycheffian Bernstein function or a Tchebycheffian B-spline function, depending on the context.

The following Matlab functions are provided for constructing TB-spline patches. These functions are actually the constructor methods of the classes having the same name (see Figure 1).

- `TB_patch`: abstract construction of a TB-spline patch, and has no stand-alone usage;
- `TB_patch_tcheb`: construction of a TB-spline patch based on constant-coefficient linear differential operators;
- `TB_patch_poly`: construction of a TB-spline patch based on algebraic polynomials;
- `TB_patch_spline`: construction of a TB-spline patch based on algebraic polynomial splines;
- `TB_patch_pexp`: construction of a TB-spline patch based on polynomials of exponential type;
- `TB_patch_ptrig`: construction of a TB-spline patch based on polynomials of trigonometric type;
- `TB_patch_gexp`: construction of a TB-spline patch based on generalized polynomials of exponential type;
- `TB_patch_gtrig`: construction of a TB-spline patch based on generalized polynomials of trigonometric type;
- `TB_patch_multi`: construction of a TB-spline patch based on an MDTB-spline patch.

The following Matlab functions are provided for working with TB-spline patches.

- `TB_domain`: computation of the end points of the domain related to a patch;
- `TB_greville`: computation of the Greville points;
- `TB_evaluation_all`: evaluation of all TB-spline basis functions in given points;
- `TB_evaluation_spline`: evaluation of a spline in given points;
- `TB_evaluation_curve`: evaluation of a spline curve in given points;
- `TB_diffend_all`: full differentiation of all TB-spline basis functions at one end point up to a given order;
- `TB_differentiation_all`: differentiation of all TB-spline basis functions in given points;
- `TB_differentiation_spline`: differentiation of a spline in given points;
- `TB_differentiation_curve`: differentiation of a spline curve in given points;
- `TB_visualization_all`: visualization of all TB-spline basis functions;
- `TB_visualization_spline`: visualization of a spline;
- `TB_visualization_curve`: visualization of a spline curve;
- `TB_conversion`: conversion from source to destination TB-spline form.

3.1 TB_patch

This abstract function prepares the data-structure for a general TB-spline patch. The TB-spline patch stores the TB-spline degree `p`, space dimension `n`, and interval vector `U`.

Syntax:

```
P = TB_patch(p, xx)
```

Input parameters:

`p`	: TB-spline degree
`xx`	: vector of end points

Output parameters:

`P`	: TB-spline patch

Discussion:

This function has no stand-alone usage.

3.2 TB_patch_tcheb

This function prepares the data-structure for a TB-spline patch where the corresponding ECT-space is the null-space of a constant-coefficient linear differential operator. Such null-space is identified by the roots of its characteristic polynomial. Let the interval $[x_0, x_1]$ be the domain of the ECT-space. A given root $\omega = \alpha + \mathrm{i}\beta$ of order μ gives rise to the following internal basis functions for $i = 0, \dots, \mu - 1$:

- if $\beta = 0$, then
$$\phi_i(x) = \frac{(x-x_0)^i}{i!} \mathrm{e}^{\alpha(x-x_0)};$$
- if $\beta \neq 0$, then the complex conjugate of ω is also a root of order μ, and
$$\phi_{2i}(x) = \frac{(x-x_0)^i}{i!} \mathrm{e}^{\alpha(x-x_0)} \cos(\beta(x-x_0)),$$
$$\phi_{2i+1}(x) = \frac{(x-x_0)^i}{i!} \mathrm{e}^{\alpha(x-x_0)} \sin(\beta(x-x_0)).$$

The TB-spline patch stores the TB-spline degree `p`, space dimension `n` (`n = p+1`), and interval vector `U`. Besides these general parameters, it also contains a four-column matrix `W` where each row represents a different root of the characteristic polynomial of the null-space. Complex conjugate roots are excluded from this matrix. The first column is the type of the root, the second and third columns are the real and imaginary parts of the root, and the fourth column is its multiplicity. There are four types of roots:

- type 0: $\alpha = 0$ and $\beta = 0$;
- type 1: $\alpha \neq 0$ and $\beta = 0$;
- type 2: $\alpha = 0$ and $\beta \neq 0$;
- type 3: $\alpha \neq 0$ and $\beta \neq 0$.

Finally, the TB-spline patch stores the cumulative dimension `mu` of the spaces related to the roots, and the internal basis conversion matrix `C`.

Syntax:

```
P = TB_patch_tcheb(p, xx, ww, mm)
```

Input parameters:

`p`	: TB-spline degree
`xx`	: vector of end points
`ww`	: TB-spline roots (optional)
`mm`	: TB-spline multiplicities (optional)

Output parameters:

`P`	: TB-spline patch

Discussion:

The parameter `p` is a non-negative integer scalar, the parameter `xx` is a vector of two strictly increasing real values (indicating the end points of the domain), the parameter `ww` is a vector of complex values, and the parameter `mm` can be a scalar or vector of positive integers. If `mm` is a scalar, each root `ww(i)` has multiplicity `mm`, and if `mm` is a vector, root `ww(i)` has multiplicity `mm(i)`, for `i = 1:length(ww)`. Hence, `length(mm)` should be equal to `1` or `length(ww)`. When no multiplicity is specified, `mm = 1` is assumed. For each non-real root, its complex conjugate is automatically inserted as root with the same multiplicity. When the same complex value appears more than once in the vector `ww`, the corresponding multiplicities are added up. If necessary, zero is added as root (or its multiplicity is increased) to match a total of exactly `p+1` roots. It is assumed that zero is at least a first-order root.

Example:

Create a TB-spline patch of degree 4 on the domain $[0, 1]$ identified by the roots $(0, \mathrm{i}, -\mathrm{i}, 3)$ with multiplicities $(1, 1, 1, 2)$, respectively:

```
>> ww = complex([0, 3], [1, 0]);
>> P = TB_patch_tcheb(4, [0, 1], ww, [1, 2])
P =
     W: [3x4 double]
    mu: [0 1 3 5]
     C: [5x5 double]
     p: 4
     n: 5
     U: [0 1]
>> W = P.W
```

```
W =
     0     0     0     1
     2     0     1     1
     1     3     0     2
```

Now, create a polynomial TB-spline patch of degree 4 on the domain $[0, 1]$:

```
>> P = TB_patch_tcheb(4, [0, 1])
P =
     W: [0 0 0 5]
    mu: [0 5]
     C: [5x5 double]
     p: 4
     n: 5
     U: [0 1]
```

3.3 TB_patch_poly

This function prepares the data-structure for a TB-spline patch where the corresponding ECT-space is an algebraic polynomial space, i.e., $\langle 1, x, \ldots, x^p \rangle$. The TB-spline patch stores the polynomial degree `p`, space dimension `n` (`n = p+1`), and interval vector `U`.

Syntax:

```
P = TB_patch_poly(p, xx)
```

Input parameters:

`p`	: polynomial degree
`xx`	: vector of end points

Output parameters:

`P`	: TB-spline patch

Discussion:

The parameter `p` is a non-negative integer scalar and the parameter `xx` is a vector of two strictly increasing real values (indicating the end points of the domain).

Example:

Create a polynomial TB-spline patch of degree 4 on the domain $[0, 1]$:

```
>> P = TB_patch_poly(4, [0, 1])
P =
     p: 4
     n: 5
     U: [0 1]
```

3.4 TB_patch_spline

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

Syntax:

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

Input parameters:

`p`	: B-spline degree
`xx`	: vector of break points
`kk`	: smoothness vector (optional)

Output parameters:

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

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

3.5 TB_patch_pexp

This function prepares the data-structure for a TB-spline patch where the corresponding ECT-space is a polynomial space of exponential type, i.e., $\langle 1, e^{\omega x}, e^{-\omega x}, \ldots, e^{q\omega x}, e^{-q\omega x}\rangle$. The TB-spline patch stores the even TB-spline degree `p`, space dimension `n` (`n = p+1`), and interval vector `U`. Besides these general parameters, it also contains the TB-spline parameter `w` and an internal vector `ss` of scaling coefficients.

Syntax:

```
P = TB_patch_pexp(p, xx, w)
```

Input parameters:

`p`	: TB-spline degree
`xx`	: vector of end points
`w`	: TB-spline parameter (optional)

Output parameters:

`P`	: TB-spline patch

Discussion:

The parameter `p` is a non-negative integer scalar; it is assumed that `p` is even. The parameter `xx` is a vector of two strictly increasing real values (indicating the end points of the domain). The parameter `w` is a real scalar. When no value is specified, `w = 1` is assumed.

Example:

Create an exponential TB-spline patch of degree 4 with shape parameter $\omega = 5$ on the domain $[0, 1]$:

```
>> P = TB_patch_pexp(4, [0, 1], 5)
P =
    w: 5
   ss: [5x1 double]
    p: 4
    n: 5
    U: [0 1]
```

3.6 TB_patch_ptrig

This function prepares the data-structure for a TB-spline patch where the corresponding ECT-space is a polynomial space of trigonometric type, i.e., $\langle 1, \cos(\omega x), \sin(\omega x), \ldots, \cos(q\omega x), \sin(q\omega x)\rangle$. The TB-spline patch stores the even TB-spline degree `p`, space dimension `n` (`n = p+1`), and interval vector `U`. Besides these general parameters, it also contains the TB-spline parameter `w` and an internal vector `ss` of scaling coefficients.

Syntax:

```
P = TB_patch_ptrig(p, xx, w)
```

Input parameters:

`p`	: TB-spline degree
`xx`	: vector of end points
`w`	: TB-spline parameter (optional)

Output parameters:

`P`	: TB-spline patch

Discussion:

The parameter `p` is a non-negative integer scalar; it is assumed that `p` is even. The parameter `xx` is a vector of two strictly increasing real values (indicating the end points of the domain). The parameter `w` is a real scalar. When no value is specified, `w = 1` is assumed.

Example:

Create a trigonometric TB-spline patch of degree 4 with shape parameter $\omega = 2$ on the domain $[0, 1]$:

```
>> P = TB_patch_ptrig(4, [0, 1], 2)
P =
     w: 2
    ss: [5x1 double]
     p: 4
     n: 5
     U: [0 1]
```

3.7 TB_patch_gexp

This function prepares the data-structure for a TB-spline patch where the corresponding ECT-space is a generalized polynomial space with exponential functions ($\mathrm{e}^{\omega x}$ and $\mathrm{e}^{-\omega x}$). Let the interval $[x_0, x_1]$ be the domain of the ECT-space, and denote the TB-spline degree with p. The internal basis functions are taken to be (if $t = 1$)

$$\phi_i(x) = \omega^i \frac{(x-x_0)^i}{i!}, \quad i = 0, \dots, p-2,$$
$$\phi_{p-1}(x) = \mathrm{e}^{\omega(x-x_0)}, \quad \phi_p(x) = \mathrm{e}^{-\omega(x-x_0)},$$

or (if $t = 0$)

$$\phi_i(x) = \frac{(x-x_0)^i}{i!}, \quad i = 0, \dots, p-2,$$
$$\phi_i(x) = \sum_{j=0}^{m} \omega^{2j} \frac{(x-x_0)^{i+2j}}{(i+2j)!}, \quad i = p-1, p,$$

for some representation parameter $m > 0$. The latter basis should be employed when the TB-spline shape parameter ω is close to zero. The TB-spline patch stores the TB-spline degree `p`, space dimension `n` (`n = p+1`), and interval vector `U`. Besides these general parameters, it also contains the TB-spline parameter `w`, representation type `t`, representation parameter `m` (if `t = 0`), and the internal basis conversion matrix `C`.

Syntax:

```
P = TB_patch_gexp(p, xx, w, t, m)
```

Input parameters:

`p`	: TB-spline degree
`xx`	: vector of end points
`w`	: TB-spline parameter (optional)
`t`	: representation type (optional)
`m`	: representation parameter (optional)

Output parameters:

`P`	: TB-spline patch

Discussion:

The parameter `p` is a positive integer scalar; it is assumed that `p >= 2`. The parameter `xx` is a vector of two strictly increasing real values (indicating the end points of the domain). The parameter `w` is a real scalar. When no value is specified, `w = 1` is assumed. The parameter `t` is a boolean scalar, or takes the values `0` or `1`. When no value is specified, `t = abs(w)*(xx(2)-xx(1)) >= 3` is assumed. The parameter `m` is a positive integer scalar. When no value is specified, `m = 10` is assumed.

Example:

Create an exponential TB-spline patch of degree 4 with shape parameter $\omega = 5$ on the domain $[0, 1]$:

```
>> P = TB_patch_gexp(4, [0, 1], 5)
P =
    w: 5
    t: 1
    m: 10
    C: [5x5 double]
    p: 4
    n: 5
    U: [0 1]
```

3.8 TB_patch_gtrig

This function prepares the data-structure for a TB-spline patch where the corresponding ECT-space is a generalized polynomial space with trigonometric functions ($\cos(\omega x)$ and $\sin(\omega x)$). Let the interval $[x_0, x_1]$ be the domain of the ECT-space, and denote the TB-spline degree with p. The internal basis functions are taken to be (if $t = 1$)

$$\phi_i(x) = \omega^i \frac{(x-x_0)^i}{i!}, \quad i = 0, \ldots, p-2,$$
$$\phi_{p-1}(x) = \cos(\omega(x-x_0)), \quad \phi_p(x) = \sin(\omega(x-x_0)),$$

or (if $t = 0$)

$$\phi_i(x) = \frac{(x - x_0)^i}{i!}, \quad i = 0, \dots, p-2,$$
$$\phi_i(x) = \sum_{j=0}^{m} (-1)^j \omega^{2j} \frac{(x - x_0)^{i+2j}}{(i+2j)!}, \quad i = p-1, p,$$

for some representation parameter $m > 0$. The latter basis should be employed when the TB-spline shape parameter ω is close to zero. The TB-spline patch stores the TB-spline degree `p`, space dimension `n` (`n = p+1`), and interval vector `U`. Besides these general parameters, it also contains the TB-spline parameter `w`, representation type `t`, representation parameter `m` (if `t = 0`), and the internal basis conversion matrix `C`.

Syntax:

```
P = TB_patch_gtrig(p, xx, w, t, m)
```

Input parameters:

`p`	: TB-spline degree
`xx`	: vector of end points
`w`	: TB-spline parameter (optional)
`t`	: representation type (optional)
`m`	: representation parameter (optional)

Output parameters:

`P`	: TB-spline patch

Discussion:

The parameter `p` is a positive integer scalar; it is assumed that `p >= 2`. The parameter `xx` is a vector of two strictly increasing real values (indicating the end points of the domain). The parameter `w` is a real scalar. When no value is specified, `w = 1` is assumed. The parameter `t` is a boolean scalar, or takes the values `0` or `1`. When no value is specified, `t = abs(w)*(xx(2)-xx(1)) >= 3` is assumed. The parameter `m` is a positive integer scalar. When no value is specified, `m = 10` is assumed.

Example:

Create a trigonometric TB-spline patch of degree 4 with shape parameter $\omega = 5$ on the domain $[0, 1]$:

```
>> P = TB_patch_gtrig(4, [0, 1], 5)
P =
    w: 5
    t: 1
    m: 10
    C: [5x5 double]
    p: 4
    n: 5
    U: [0 1]
```

3.9 TB_patch_multi

This function prepares the data-structure for a TB-spline patch, starting from an MDTB-spline patch and a non-periodic extraction matrix. The TB-spline patch stores a vector of MDTB-spline degrees `p`, space dimension `n`, and interval vector `U`. Besides this general parameters, it also contains an MDTB-spline patch `MP` (see `MDTB_patch`), a non-periodic extraction matrix `H`, and possibly the extraction matrix `Hpol` for the equivalent multi-degree spline space consisting of polynomial segments. The purpose of the latter matrix is only for computational efficiency of some routines.

Syntax:

```
P = TB_patch_multi(MP, H, Hpol)
```

Input parameters:

`MP`	: MDTB-spline multi-patch
`H`	: non-periodic extraction matrix
`Hpol`	: non-periodic polynomial extraction matrix (optional)

Output parameters:

`P`	: TB-spline patch

Discussion:

The extraction matrix `H` should be deduced from the MDTB-spline multi-patch `MP` and incorporates the smoothness. The polynomial extraction matrix `Hpol` should be deduced from an MDTB-spline multi-patch of the same structure of `MP` but consisting of only polynomial segments; this matrix is optional. Hence, `size(Hpol)` should be equal to `size(H)` or is empty.

Example:

Create a TB-spline patch from a given MDTB-spline patch defined on the domain $[0, 3]$:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> P = TB_patch_multi(MP, H)
P =
     MP: [1x1 MDTB_patch]
      H: [6x9 double]
   Hpol: []
      p: [3 4]
      n: 6
      U: [0 3]
```

3.10 TB_domain

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

Syntax:

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

Input parameters:

P	: TB-spline patch

Output parameters:

a	: left end point
b	: right end point

Example:

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

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> [a, b] = TB_domain(P)
a =
    0
b =
    1
```

3.11 TB_greville

This function computes the Greville points of a given TB-spline patch, i.e., the coefficients of the TB-spline form of the identity function (if possible).

Syntax:

```
gg = TB_greville(P)
```

Input parameters:

P	: TB-spline patch

Output parameters:

gg	: vector of Greville points

Discussion:

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

Example:

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

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> gg = TB_greville(P)
gg =
          0    0.3955    0.5000    0.6045    1.0000
```

3.12 TB_evaluation_all

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

Syntax:

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

Input parameters:

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

Output parameters:

`M`	: evaluation matrix

Discussion:

The parameter `cl` is a boolean scalar. If `cl` is `false`, then the TB-spline values are computed 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. When no value is specified, `cl = true` is assumed. Each row in the resulting matrix `M` corresponds to a TB-spline basis function and each column to an evaluation point, so `size(M)` equals `[P.n, length(xx)]`.

Example:

Create a TB-spline patch and evaluate all the corresponding TB-spline basis functions at three uniform points in the domain:

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> M = TB_evaluation_all(P, [1/4, 1/2, 3/4])
M =
    0.4422    0.1123    0.0082
    0.3713    0.2712    0.0555
    0.1228    0.2328    0.1228
    0.0555    0.2712    0.3713
    0.0082    0.1123    0.4422
```

3.13 TB_evaluation_spline

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

Syntax:

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

Input parameters:

`P`	: TB-spline patch
`cc`	: vector of coefficients
`xx`	: vector of evaluation points

Output parameters:

`ss`	: vector of spline values

Discussion:

Each element of the vector `cc` corresponds to a TB-spline basis function in the TB-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 TB-spline patch and a vector of coefficients, and then evaluate the corresponding spline in TB-spline form at three uniform points in the domain:

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> cc = [1, 3, 5, 5, 2];
>> ss = TB_evaluation_spline(P, cc, [1/4, 1/2, 3/4])
ss =
    2.4640    3.6711    3.5296
```

3.14 TB_evaluation_curve

This function evaluates a spline curve in TB-spline form at a given set of points, and stores the corresponding curve points in a matrix.

Syntax:

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

Input parameters:

`P`	: TB-spline patch
`cc`	: matrix of control points (1D, 2D or 3D)
`xx`	: vector of evaluation points

Output parameters:

`ss`	: matrix of spline curve points

Discussion:

Each column of the matrix `cc` corresponds to a TB-spline basis function in the TB-spline patch. Hence, `size(cc,2)` should be equal to `P.n`. This matrix consists of 1, 2 or 3 rows, identifying a function, a 2D curve or a 3D curve, respectively. Each column of the resulting matrix `ss` corresponds to an evaluation point and this matrix has the same number of rows as `cc`, so `size(ss)` equals `[size(cc,1), length(xx)]`.

Example:

Create a TB-spline patch and a matrix of 2D control points, and then evaluate the corresponding 2D spline curve in TB-spline form at three uniform points in the domain:

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> cc = [1, 3, 5, 3, 1; 2, 1, 3, 4, 2];
>> ss = TB_evaluation_curve(P, cc, [1/4, 1/2, 3/4])
ss =
    2.3448    3.0163    2.3448
    1.8625    2.5041    2.8099
```

3.15 TB_diffend_all

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

Syntax:

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

Input parameters:

`P`	: TB-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 scalar and the parameter `el` is a boolean scalar. 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 TB-spline basis function and each column to a derivative, so `size(K)` equals `[P.n, r+1]`.

Example:

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

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> K = TB_diffend_all(P, 4, false)
```

```
K =
         0         0         0         0   53.0395
         0         0         0  -27.5186  -69.5800
         0         0    5.7516   16.5404 -110.7090
         0   -2.5285   -7.2714   21.1504  112.2041
    1.0000    2.5285    1.5198  -10.1722   15.0453
```

3.16 TB_differentiation_all

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

Syntax:

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

Input parameters:

`P`	: TB-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 scalar and the parameter `cl` is a boolean scalar. If `cl` is `false`, then the TB-spline values are computed 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. When no value is specified, `cl = true` is assumed. Each row in the resulting matrix `M` corresponds to a TB-spline basis function and each column to an evaluation point, so `size(M)` equals `[P.n, length(xx)]`.

Example:

Create a TB-spline patch and evaluate the first derivative of all the corresponding TB-spline basis functions at three uniform points in the domain:

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> M = TB_differentiation_all(P, 1, [1/4, 1/2, 3/4])
M =
   -1.8337   -0.8068   -0.1277
    0.4016   -0.9241   -0.5861
    0.7183         0   -0.7183
    0.5861    0.9241   -0.4016
    0.1277    0.8068    1.8337
```

3.17 TB_differentiation_spline

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

Syntax:

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

Input parameters:

`P`	: TB-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 scalar. Each element of the vector `cc` corresponds to a TB-spline basis function in the TB-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 TB-spline patch and a vector of coefficients, and then evaluate the first derivative of the corresponding spline in TB-spline form at three uniform points in the domain:

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> cc = [1, 3, 5, 5, 2];
>> ss = TB_differentiation_spline(P, 1, cc, [1/4, 1/2, 3/4])
ss =
    6.1485    2.6551   -3.8180
```

3.18 TB_differentiation_curve

This function evaluates the r-th order derivative of a spline curve in TB-spline form at a given set of points, and stores the corresponding curve points in a matrix.

Syntax:

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

Input parameters:

`P`	: TB-spline patch
`r`	: order of derivative
`cc`	: matrix of control points (1D, 2D or 3D)
`xx`	: vector of evaluation points

Output parameters:

`ss`	: matrix of `r`-th order derivative spline curve points

Discussion:

The parameter `r` is a non-negative integer scalar. Each column of the matrix `cc` corresponds to a TB-spline basis function in the TB-spline patch. Hence, `size(cc,2)` should be equal to `P.n`. This matrix consists of 1, 2 or 3 rows, identifying a function, a 2D curve or a 3D curve, respectively. Each column of the resulting matrix `ss` corresponds to an evaluation point and this matrix has the same number of rows as `cc`, so `size(ss)` equals `[size(cc,1), length(xx)]`.

Example:

Create a TB-spline patch and a matrix of 2D control points, and then evaluate the first derivative of the corresponding 2D spline curve in TB-spline form at three uniform points in the domain:

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> cc = [1, 3, 5, 3, 1; 2, 1, 3, 4, 2];
>> ss = TB_differentiation_curve(P, 1, cc, [1/4, 1/2, 3/4])
ss =
    4.8486   -0.0000   -4.8486
    1.4889    2.7724   -0.9354
```

3.19 TB_visualization_all

This function visualizes all TB-spline basis functions of a TB-spline patch.

Syntax:

```
TB_visualization_all(P, n, specs)
```

Input parameters:

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

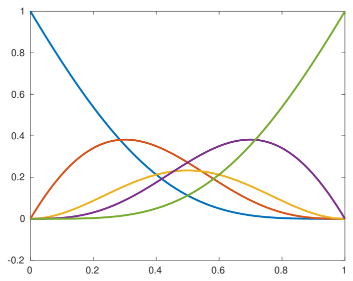
Discussion:

The parameter `n` is a positive integer scalar. 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 TB-spline patch and plot all the corresponding TB-spline basis functions:

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> TB_visualization_all(P, 100, 'LineWidth', 2);
```



3.20 TB_visualization_spline

This function visualizes a spline in TB-spline form.

Syntax:

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

Input parameters:

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

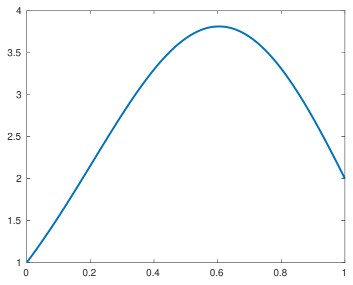
Discussion:

Each element of the vector `cc` corresponds to a TB-spline basis function in the TB-spline patch. Hence, `length(cc)` should be equal to `P.n`. The parameter `n` is a positive integer scalar. 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 TB-spline patch and a vector of coefficients, and then plot the corresponding spline in TB-spline form:

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> cc = [1, 3, 5, 5, 2];
>> TB_visualization_spline(P, cc, 100, 'LineWidth', 2);
```



3.21 TB_visualization_curve

This function visualizes a spline curve in TB-spline form.

Syntax:

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

Input parameters:

`P`	: TB-spline patch
`cc`	: matrix of control points (1D, 2D or 3D)
`n`	: number of evaluation points (optional)
`specs`	: pass any number of plot specifications (optional)

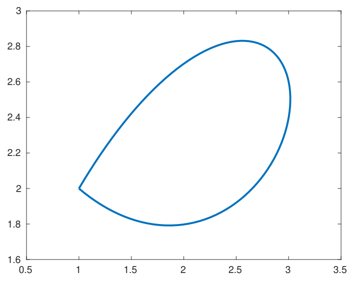
Discussion:

Each column of the matrix `cc` corresponds to a TB-spline basis function in the TB-spline patch. Hence, `size(cc,2)` should be equal to `P.n`. This matrix consists of 1, 2 or 3 rows, identifying a function, a 2D curve or a 3D curve, respectively. The parameter `n` is a positive integer scalar. 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/plot3`. We refer the reader to the documentation of `plot/plot3` for all the plotting options.

Example:

Create a TB-spline patch and a matrix of 2D control points, and then plot the corresponding spline curve in TB-spline form:

```
>> P = TB_patch_gtrig(4, [0, 1], 5);
>> cc = [1, 3, 5, 3, 1; 2, 1, 3, 4, 2];
>> TB_visualization_curve(P, cc, 100, 'LineWidth', 2);
```



3.22 TB_conversion

This function converts a spline in TB-spline form into another TB-spline form. The conversion is exact when the source and destination TB-spline patches imply nested spaces. It is assumed that Greville points can be computed.

Syntax:

```
ccd = TB_conversion(Pd, Ps, ccs, sh)
```

Input parameters:

`Pd`	: destination TB-spline patch
`Ps`	: source TB-spline patch
`ccs`	: source coefficient vector
`sh`	: shift of the source patch (optional)

Output parameters:

`ccd`	: destination coefficient vector

Discussion:

Different types of TB-spline patches for `Ps` and `Pd` may be mixed. Each element of the vector `ccs` corresponds to a TB-spline basis function in the TB-spline patch `Ps`. Hence, `length(ccs)` should be equal to `Ps.n`. Similarly, each element of the resulting vector `ccd` corresponds to a TB-spline basis function in the TB-spline patch `Pd`. The parameter `sh` is a real scalar. When no value is specified, `sh = 0` is assumed.

Example:

Create a TB-spline patch of degree 4 and a vector of coefficients; then, raise the degree to 9 and compute the coefficients of the new TB-spline form:

```
>> Ps = TB_patch_gtrig(4, [0, 1], 5);
>> ccs = [1, 3, 5, 5, 2];
>> Pd = TB_patch_gtrig(9, [0, 1], 5);
>> ccd = TB_conversion(Pd, Ps, ccs)
ccd =
  Columns 1 through 7
    1.0000    1.5889    2.2889    3.0908    3.8289    4.2506    4.1879
  Columns 8 through 10
    3.6730    2.8833    2.0000
```

Now, convert it into a more general TB-spline patch that is shifted by 10:

```
>> ww = complex([0, 3], [5, 0]);
>> Pe = TB_patch_tcheb(9, [10, 11], ww, [1, 2]);
>> cce = TB_conversion(Pe, Ps, ccs, 10)
  Columns 1 through 7
    1.0000    1.6283    2.3704    3.2081    3.9310    4.2723    4.1184
  Columns 8 through 10
    3.5658    2.8059    2.0000
```

It is no surprise that the vectors `ccd` and `cce` are similar because the corresponding basis functions look similar as well, up to the shift.

4. MDTB-SPLINE MULTI-PATCHES

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

The following MATLAB functions are provided for constructing MDTB-spline multi-patches.

- `MDTB_patch`: construction of an MDTB-spline multi-patch from TB-spline segments;
- `MDTB_patch_tcheb`: construction of an MDTB-spline multi-patch from Tchebycheff segments based on constant-coefficient linear differential operators;
- `MDTB_patch_poly`: construction of an MDTB-spline multi-patch from algebraic polynomial or algebraic polynomial spline segments;
- `MDTB_patch_ppoly`: construction of an MDTB-spline multi-patch from polynomial segments (algebraic/exponential/trigonometric);
- `MDTB_patch_gpoly`: construction of an MDTB-spline multi-patch from generalized polynomial segments (algebraic/exponential/trigonometric).

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

- `MDTB_extraction`: computation of the multi-degree spline extraction matrix;
- `MDTB_extraction_periodic`: computation of the multi-degree spline extraction matrix with periodicity;
- `MDTB_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 MDTB-splines. Thanks to the multi-degree spline extraction operator, their implementation can be easily redirected to their TB-spline analogues described in Section 3.

- `MDTB_domain`: computation of the end points of the domain related to a multi-patch;
- `MDTB_greville`: computation of the multi-degree Greville points;
- `MDTB_evaluation_all`: evaluation of all MDTB-spline basis functions in given points;
- `MDTB_evaluation_spline`: evaluation of a multi-degree spline in given points;
- `MDTB_evaluation_curve`: evaluation of a multi-degree spline curve in given points;
- `MDTB_differentiation_all`: differentiation of all MDTB-spline basis functions in given points;
- `MDTB_differentiation_spline`: differentiation of a multi-degree spline in given points;
- `MDTB_differentiation_curve`: differentiation of a multi-degree spline curve in given points;
- `MDTB_visualization_all`: visualization of all MDTB-spline basis functions;
- `MDTB_visualization_spline`: visualization of a multi-degree spline;
- `MDTB_visualization_curve`: visualization of a multi-degree spline curve;
- `MDTB_conversion`: conversion from source to destination MDTB-spline form.

4.1 MDTB_patch

This function prepares the data-structure for an MDTB-spline multi-patch, starting from a sequence of TB-spline segments. The MDTB-spline multi-patch stores a vector of TB-spline patches `P` and the corresponding cumulative dimension `mu`.

Syntax:

```
MP = MDTB_patch(PP)
```

Input parameters:

`PP`	: vector of B-spline patches

Output parameters:

`MP`	: MDTB-spline multi-patch

Example:

Create an MDTB-spline multi-patch consisting of two TB-spline patches with different degrees (3 and 4):

```
>> P1 = TB_patch_gexp(3, [0, 1], 5);
>> P2 = TB_patch_gtrig(4, [1, 3], 2);
>> MP = MDTB_patch([P1, P2])
MP =
     P: [1x2 TB_patch]
    mu: [0 4 9]
>> MP.P(1)
ans =
     w: 5
     t: 1
     m: 10
     C: [4x4 double]
     p: 3
     n: 4
     U: [0 1]
>> MP.P(2)
ans =
     w: 2
     t: 1
     m: 10
     C: [5x5 double]
     p: 4
     n: 5
     U: [1 3]
```

4.2 MDTB_patch_tcheb

This function prepares the data-structure for an MDTB-spline multi-patch, starting from a sequence of Tchebycheff segments based on constant-coefficient linear differential operators (see `TB_patch_tcheb`). The MDTB-spline multi-patch stores a vector of TB-spline patches `P` and the corresponding cumulative dimension `mu`.

Syntax:

```
MP = MDTB_patch_tcheb(pp, xx, ww, mm)
```

Input parameters:

`pp`	: vector of TB-spline degrees
`xx`	: vector of break points
`ww`	: cell array of TB-spline roots (optional)
`mm`	: cell array of TB-spline multiplicities (optional)

Output parameters:

`MP`	: MDTB-spline multi-patch

Discussion:

The parameter `xx` is a vector consisting of a strictly increasing sequence of real values (indicating the different segments). The parameter `pp` can be a scalar or a vector whose elements are non-negative integers. If `pp` is a scalar, then degree `pp` is used on each interval. On the other hand, if `pp` is a vector, then `pp(i)` is the degree on the interval `[xx(i), xx(i+1)]` for `i = 1:length(xx)-1`. Hence, `length(pp)` should be equal to `1` or `length(xx)-1`. The parameter `ww` is a cell array of vectors of roots (complex values) and the parameter `mm` is a cell array of the corresponding multiplicities (integer values). Both cell arrays should have a length equal to `1` or `length(xx)-1`. If the length is `1`, then the same root parameters are considered on each interval; otherwise, `ww{i}` and `mm{i}` are considered on the interval `[xx(i), xx(i+1)]` for `i = 1:length(xx)-1`. When no roots are specified, `ww = {0}` is assumed. When no multiplicities are specified, `mm = {1}` is assumed. We refer the reader to the documentation of `TB_patch_tcheb` for a description of how to construct a valid set of root parameters.

Example:

Create an MDTB-spline multi-patch consisting of two TB-spline patches with different degrees (3 and 4) by specifying the roots of the characteristic polynomials:

```
>> ww = {complex(1, 1), complex([0, 3], [1, 0])};
>> mm = {1, [1, 2]};
>> MP = MDTB_patch_tcheb([3, 4], [0, 1, 3], ww, mm)
MP =
     P: [1x2 TB_patch_tcheb]
    mu: [0 4 9]
```

```
>> MP.P(1)
ans =
     W: [2x4 double]
    mu: [0 2 4]
     C: [4x4 double]
     p: 3
     n: 4
     U: [0 1]
>> MP.P(2)
ans =
     W: [3x4 double]
    mu: [0 1 3 5]
     C: [5x5 double]
     p: 4
     n: 5
     U: [1 3]
```

4.3 MDTB_patch_poly

This function prepares the data-structure for an MDTB-spline multi-patch, starting from a sequence of algebraic polynomial segments and smoothness relations. The MDTB-spline multi-patch stores a vector of TB-spline patches `P` and the corresponding cumulative dimension `mu`. Consecutive polynomial segments of the same degree are merged into a single TB-spline patch, unless specified otherwise (see `TB_patch_poly` and `TB_patch_spline`).

Syntax:

```
[MP, rr] = MDTB_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`	: MDTB-spline multi-patch
`rr`	: MDTB-spline smoothness vector (optional)

Discussion:

The parameter `xx` is a vector consisting of a strictly increasing sequence of real values (indicating the different segments). The parameter `pp` can be a scalar or a vector whose elements are non-negative integers. If `pp` is a scalar, then degree `pp` is used on each interval. On the other hand, if `pp` is a vector, then `pp(i)` is the degree on the interval `[xx(i), xx(i+1)]` for `i = 1:length(xx)-1`. Hence, `length(pp)` should be equal to `1` or `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` is a boolean scalar. If `mg` is `true`, then consecutive polynomial segments of the same degree are merged into a single polynomial TB-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 TB-spline patches, so `length(rr)` equals `length(MP.P)-1`.

Example:

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

```
>> [MP, rr] = MDTB_patch_poly([3, 3, 4, 4], [0, 1, 3, 4, 6], 2)
MP =
    P: [1x2 TB_patch_spline]
   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]
```

4.4 MDTB_patch_ppoly

This function prepares the data-structure for an MDTB-spline multi-patch, starting from a sequence of polynomial-type segments. There are three types of segments:

- type 0: algebraic (see `TB_patch_poly`);
- type 1: exponential (see `TB_patch_pexp`);
- type 2: trigonometric (see `TB_patch_ptrig`).

The MDTB-spline multi-patch stores a vector of TB-spline patches `P` and the corresponding cumulative dimension `mu`.

Syntax:

```
MP = MDTB_patch_ppoly(pp, xx, tp, ww)
```

Input parameters:

`pp`	: vector of TB-spline degrees
`xx`	: vector of break points
`tp`	: vector of TB-spline types (optional)
`ww`	: vector of TB-spline parameters (optional)

Output parameters:

`MP`	: MDTB-spline multi-patch

Discussion:

The parameter `xx` is a vector consisting of a strictly increasing sequence of real values (indicating the different segments). The parameter `pp` can be a scalar or a vector whose elements are non-negative integers. If `pp` is a scalar, then degree `pp` is used on each interval. On the other hand, if `pp` is a vector, then `pp(i)` is the degree on the interval `[xx(i), xx(i+1)]` for `i = 1:length(xx)-1`. Hence, `length(pp)` should be equal to `1` or `length(xx)-1`. The parameter `tp` can be a scalar or a vector whose elements are `0`, `1` or `2`. If `tp` is a scalar, then TB-spline type `tp` is used on each interval. On the other hand, if `tp` is a vector, then `tp(i)` is the TB-spline type on the interval `[xx(i), xx(i+1)]` for `i = 1:length(xx)-1`. Hence, `length(tp)` should be equal to `1` or `length(xx)-1`. When no value is specified, `tp = 0` is assumed. The parameter `ww` can be a scalar or a vector of TB-spline type-specific parameters. If `ww` is a scalar, then the same value is considered on each interval; otherwise, `ww(i)` is considered on the interval `[xx(i), xx(i+1)]` for `i = 1:length(xx)-1`. Hence, `length(ww)` should be equal to `1` or `length(xx)-1`. In the case of exponential and trigonometric segments, we refer the reader to the documentation of `TB_patch_pexp` and `TB_patch_ptrig` for a description of these type-specific parameters and their default values. In the case of algebraic polynomial segments, these parameters have no meaning and can be set arbitrarily.

Example:

Create an MDTB-spline multi-patch consisting of two polynomial-type segments with different degrees (2 and 4):

```
>> MP = MDTB_patch_ppoly([2, 4], [0, 1, 3], [1, 2], [5, 1])
MP =
    P: [1x2 TB_patch_poly]
   mu: [0 3 8]
>> MP.P(1)
ans =
    w: 5
   ss: [3x1 double]
    p: 2
    n: 3
    U: [0 1]
```

```
>> MP.P(2)
ans =
    w: 1
   ss: [5x1 double]
    p: 4
    n: 5
    U: [1 3]
```

4.5 MDTB_patch_gpoly

This function prepares the data-structure for an MDTB-spline multi-patch, starting from a sequence of generalized polynomial segments. There are three types of segments:

- type 0: algebraic (see `TB_patch_poly`);
- type 1: exponential (see `TB_patch_gexp`);
- type 2: trigonometric (see `TB_patch_gtrig`).

The MDTB-spline multi-patch stores a vector of TB-spline patches `P` and the corresponding cumulative dimension `mu`.

Syntax:

```
MP = MDTB_patch_gpoly(pp, xx, tp, ww, tt, mm)
```

Input parameters:

`pp`	: vector of TB-spline degrees
`xx`	: vector of break points
`tp`	: vector of TB-spline types (optional)
`ww`	: vector of TB-spline parameters (optional)
`tt`	: vector of representation types (optional)
`mm`	: vector of representation parameters (optional)

Output parameters:

`MP`	: MDTB-spline multi-patch

Discussion:

The parameter `xx` is a vector consisting of a strictly increasing sequence of real values (indicating the different segments). The parameter `pp` can be a scalar or a vector whose elements are non-negative integers. If `pp` is a scalar, then degree `pp` is used on each interval. On the other hand, if `pp` is a vector, then `pp(i)` is the degree on the interval `[xx(i), xx(i+1)]` for `i = 1:length(xx)-1`. Hence, `length(pp)` should be equal to `1` or `length(xx)-1`. The parameter `tp` can be a scalar or a vector whose elements are `0`, `1` or `2`. If `tp` is a scalar, then TB-spline type `tp` is used on each interval. On the other hand, if `tp` is a vector, then `tp(i)` is the TB-spline type on the interval `[xx(i), xx(i+1)]` for `i = 1:length(xx)-1`. Hence, `length(tp)` should be equal to `1` or `length(xx)-1`. When no value is specified, `tp = 0` is assumed. The parameters `ww`, `tt` and `mm` can be scalars or vectors of TB-spline type-specific parameters. If they are scalars, then the same values are considered on each interval; otherwise, `ww(i)`, `tt(i)` and `mm(i)` are considered on the interval `[xx(i), xx(i+1)]` for `i = 1:length(xx)-1`. Hence, their length should be equal to `1` or `length(xx)-1`. In the case of exponential and trigonometric segments, we refer the reader to the documentation of `TB_patch_gexp` and `TB_patch_gtrig` for a description of these type-specific parameters and their default values. In the case of algebraic polynomial segments, these parameters have no meaning and can be set arbitrarily.

Example:

Create an MDTB-spline multi-patch consisting of two generalized polynomial segments with different degrees (3 and 4):

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2])
MP =
    P: [1x2 TB_patch]
   mu: [0 4 9]
>> MP.P(1)
ans =
    w: 5
    t: 1
    m: 10
    C: [4x4 double]
    p: 3
    n: 4
    U: [0 1]
>> MP.P(2)
ans =
    w: 2
    t: 1
    m: 10
    C: [5x5 double]
    p: 4
    n: 5
    U: [1 3]
```

4.6 MDTB_extraction

This function computes the multi-degree spline extraction matrix representing a set of MDTB-spline basis functions in terms of the TB-spline basis functions related to a given sequence of TB-spline patches.

Syntax:

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

Input parameters:

`MP`	: MDTB-spline multi-patch
`rr`	: MDTB-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, then `rr` should be less than `min([MP.P.p])`, and smoothness `rr` is imposed between all consecutive TB-spline patches. On the other hand, if `rr` is a vector, then `rr(i)` should be less than `min(MP.P(i).p,MP.P(i+1).p)`, and smoothness `rr(i)` is imposed between TB-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 MDTB-spline basis function and each column to a TB-spline basis function in one of the TB-spline patches, so `size(H,2)` equals `MP.mu(end)`.

Example:

Create an MDTB-spline multi-patch and compute the multi-degree spline extraction matrix for a given smoothness pattern:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> Hfull = full(H)
Hfull =
  Columns 1 through 7
    1.0000         0         0         0         0         0         0
         0    1.0000    0.5476    0.4267    0.4267         0         0
         0         0    0.4524    0.5210    0.5210    0.7632         0
         0         0         0    0.0523    0.0523    0.2368    1.0000
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
  Columns 8 through 9
         0         0
         0         0
         0         0
         0         0
    1.0000         0
         0    1.0000
```

4.7 MDTB_extraction_periodic

This function computes the periodic multi-degree spline extraction matrix representing a set of periodic MDTB-spline basis functions in terms of the TB-spline basis functions related to a given sequence of TB-spline patches.

Syntax:

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

Input parameters:

`MP`	: MDTB-spline multi-patch
`rr`	: MDTB-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, then `rr` should be less than `min([MP.P.p])`, and smoothness `rr` is imposed between all consecutive TB-spline patches. On the other hand, if `rr` is a vector, then `rr(i)` should be less than `min(MP.P(i).p,MP.P(i+1).p)`, and smoothness `rr(i)` is imposed between TB-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 MDTB-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 MDTB-spline basis function and each column to a TB-spline basis function in one of the TB-spline patches, so `size(H,2)` equals `MP.mu(end)`.

Example:

Create an MDTB-spline multi-patch and compute the periodic multi-degree spline extraction matrix for a given smoothness pattern:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> Hper = MDTB_extraction_periodic(MP, 2, 1);
>> Hfull = full(Hper)
Hfull =
  Columns 1 through 7
    0.2208         0         0         0         0         0         0
    0.7792    1.0000    0.5476    0.4267    0.4267         0         0
         0         0    0.4524    0.5210    0.5210    0.7632         0
         0         0         0    0.0523    0.0523    0.2368    1.0000
  Columns 8 through 9
    1.0000    0.2208
         0    0.7792
         0         0
         0         0
```

4.8 MDTB_extraction_local

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

Syntax:

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

Input parameters:

`MP`	: MDTB-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 MDTB-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 MDTB-spline basis function and each column to a TB-spline basis function in the selected TB-spline patch, so `size(Hl)` equals `[size(H,1), MP.P(ip).n]`.

Example:

Create an MDTB-spline multi-patch with smoothness and show the Bézier extraction matrix corresponding to the first patch:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> Hl = MDTB_extraction_local(MP, H, 1);
>> B = full(Hl(any(Hl, 2), :))
B =
    1.0000         0         0         0
         0    1.0000    0.5476    0.4267
         0         0    0.4524    0.5210
         0         0         0    0.0523
```

4.9 MDTB_domain

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

Syntax:

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

Input parameters:

MP	: MDTB-spline multi-patch

Output parameters:

a	: left end point
b	: right end point

Example:

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

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> [a, b] = MDTB_domain(MP)
a =
    0
b =
    3
```

4.10 MDTB_greville

This function computes the Greville points of a given MDTB-spline multi-patch with smoothness, i.e., the coefficients of the MDTB-spline form of the identity function (if possible).

Syntax:

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

Input parameters:

MP	: MDTB-spline multi-patch
H	: extraction matrix

Output parameters:

gg	: vector of Greville points

Discussion:

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

Example:

Create an MDTB-spline multi-patch with smoothness and show its Greville points:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> gg = MDTB_greville(MP, H)
gg =
    0.0000    0.1891    1.5638    2.0000    2.3329    3.0000
```

4.11 MDTB_evaluation_all

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

Syntax:

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

Input parameters:

`MP`	: MDTB-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 MDTB-spline multi-patch `MP` and incorporates the smoothness. Each row in the resulting matrix `M` corresponds to an MDTB-spline basis function and each column to an evaluation point, so `size(M)` equals `[size(H,1), length(xx)]`.

Example:

Create an MDTB-spline multi-patch with smoothness and evaluate all the corresponding MDTB-spline basis functions at five uniform points in the domain:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> M = MDTB_evaluation_all(MP, H, [1/2, 1, 3/2, 2, 5/2])
M =
    0.0513         0         0         0         0
    0.7163    0.4267    0.1688    0.0393    0.0027
    0.2297    0.5210    0.4986    0.2434    0.0417
    0.0027    0.0523    0.2760    0.3693    0.1768
         0    0.0000    0.0503    0.2561    0.3833
         0         0    0.0064    0.0920    0.3955
```

Now, do the same with periodic MDTB-spline basis functions:

```
>> Hper = MDTB_extraction_periodic(MP, 2, 1);
>> Mper = MDTB_evaluation_all(MP, Hper, [1/2, 1, 3/2, 2, 5/2])
Mper =
    0.0113    0.0000    0.0517    0.2764    0.4706
    0.7563    0.4267    0.1737    0.1109    0.3109
    0.2297    0.5210    0.4986    0.2434    0.0417
    0.0027    0.0523    0.2760    0.3693    0.1768
```

4.12 MDTB_evaluation_spline

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

Syntax:

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

Input parameters:

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

Output parameters:

ss	: vector of spline values

Discussion:

The extraction matrix `H` should be deduced from the MDTB-spline multi-patch `MP` and incorporates the smoothness. Each element of the vector `cc` corresponds to an MDTB-spline basis function. 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 MDTB-spline multi-patch with smoothness and a vector of coefficients, and then evaluate the corresponding spline in MDTB-spline form at three uniform points in the domain:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> cc = [1, 3, 5, 4, 0, 2];
>> ss = MDTB_evaluation_spline(MP, H, cc, [1/2, 3/2, 5/2])
ss =
    3.3595    4.1159    1.7149
```

Now, do the same with periodic MDTB-spline basis functions:

```
>> Hper = MDTB_extraction_periodic(MP, 2, 1);
>> ccper = [1, 3, 5, 4];
>> ssper = MDTB_evaluation_spline(MP, Hper, ccper, [1/2, 3/2, 5/2])
ssper =
    3.4394    4.1697    2.3190
```

4.13 MDTB_evaluation_curve

This function evaluates a spline curve in (periodic) MDTB-spline form at a given set of points, and stores the corresponding curve points in a matrix.

Syntax:

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

Input parameters:

`MP`	: MDTB-spline multi-patch
`H`	: extraction matrix
`cc`	: matrix of control points (1D, 2D or 3D)
`xx`	: vector of evaluation points

Output parameters:

`ss`	: matrix of spline curve points

Discussion:

The extraction matrix `H` should be deduced from the MDTB-spline multi-patch `MP` and incorporates the smoothness. Each column of the matrix `cc` corresponds to an MDTB-spline basis function. Hence, `size(cc,2)` should be equal to `size(H,1)`. This matrix consists of 1, 2 or 3 rows, identifying a function, a 2D curve or a 3D curve, respectively. Each column of the resulting matrix `ss` corresponds to an evaluation point and this matrix has the same number of rows as `cc`, so `size(ss)` equals `[size(cc,1), length(xx)]`.

Example:

Create an MDTB-spline multi-patch with smoothness and a matrix of 2D control points, and then evaluate the corresponding 2D spline curve in MDTB-spline form at three uniform points in the domain:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> Hper = MDTB_extraction_periodic(MP, 2, 1);
>> ccper = [1, 3, 5, 4; 2, 0, 2, 4];
>> ssper = MDTB_evaluation_curve(MP, Hper, ccper, [1/2, 3/2, 5/2])
ssper =
    3.4394    4.1697    2.3190
    0.4928    2.2046    1.7319
```

4.14 MDTB_differentiation_all

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

Syntax:

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

Input parameters:

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

Example:

Create an MDTB-spline multi-patch with smoothness and evaluate the first derivative of all the corresponding MDTB-spline basis functions at five uniform points in the domain:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> M = MDTB_differentiation_all(MP, H, 1, [1/2, 1, 3/2, 2, 5/2])
M =
   -0.3708         0         0         0         0
   -0.2995   -0.6397   -0.3844   -0.1467   -0.0213
    0.6509    0.3631   -0.3779   -0.5373   -0.2300
    0.0194    0.2766    0.4451   -0.1291   -0.5298
         0         0    0.2673    0.4694   -0.1198
         0         0    0.0500    0.3437    0.9010
```

4.15 MDTB_differentiation_spline

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

Syntax:

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

Input parameters:

`MP`	: MDTB-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 MDTB-spline multi-patch `MP` and incorporates the smoothness. The parameter `r` is a non-negative integer scalar. Each element of the vector `cc` corresponds to an MDTB-spline basis function. 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 MDTB-spline multi-patch with smoothness and a vector of coefficients, and then evaluate the first derivative of the corresponding spline in MDTB-spline form at three uniform points in the domain:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> cc = [1, 3, 5, 4, 0, 2];
>> ss = MDTB_differentiation_spline(MP, H, 1, cc, [1/2, 3/2, 5/2])
ss =
    2.0628    -1.1626   -1.5313
```

4.16 MDTB_differentiation_curve

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

Syntax:

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

Input parameters:

`MP`	: MDTB-spline multi-patch
`H`	: extraction matrix
`r`	: order of derivative
`cc`	: matrix of control points (1D, 2D or 3D)
`xx`	: vector of evaluation points

Output parameters:

`ss`	: matrix of `r`-th order derivative spline curve points

Discussion:

The extraction matrix `H` should be deduced from the MDTB-spline multi-patch `MP` and incorporates the smoothness. The parameter `r` is a non-negative integer scalar. Each column of the matrix `cc` corresponds to an MDTB-spline basis function. Hence, `size(cc,2)` should be equal to `size(H,1)`. This matrix consists of 1, 2 or 3 rows, identifying a function, a 2D curve or a 3D curve, respectively. Each column of the resulting matrix `ss` corresponds to an evaluation point and this matrix has the same number of rows as `cc`, so `size(ss)` equals `[size(cc,1), length(xx)]`.

Example:

Create an MDTB-spline multi-patch with smoothness and a matrix of 2D control points, and then evaluate the first derivative of the corresponding 2D spline curve in MDTB-spline form at three uniform points in the domain:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> Hper = MDTB_extraction_periodic(MP, 2, 1);
>> ccper = [1, 3, 5, 4; 2, 0, 2, 4];
>> ssper = MDTB_differentiation_curve(MP, Hper, 1, ccper, [1, 3, 5]/2)
ssper =
    1.4849   -0.8674   -1.1480
    1.2155    1.5813   -2.4209
```

4.17 MDTB_visualization_all

This function visualizes all (periodic) MDTB-spline basis functions of an MDTB-spline multi-patch with smoothness.

Syntax:

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

Input parameters:

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

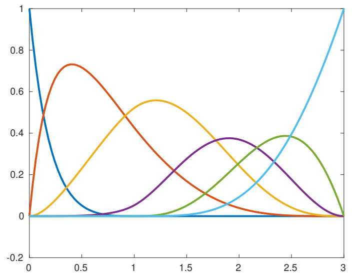
Discussion:

The extraction matrix `H` should be deduced from the MDTB-spline multi-patch `MP` and incorporates the smoothness. The parameter `n` is a positive integer scalar. 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 MDTB-spline multi-patch with smoothness and plot all the corresponding MDTB-spline basis functions:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> MDTB_visualization_all(MP, H, 100, 'LineWidth', 2);
```



4.18 MDTB_visualization_spline

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

Syntax:

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

Input parameters:

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

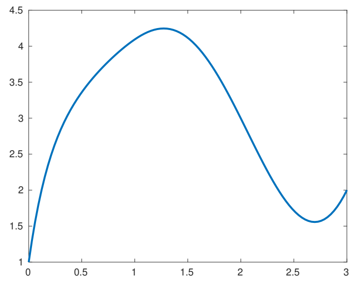
Discussion:

The extraction matrix `H` should be deduced from the MDTB-spline multi-patch `MP` and incorporates the smoothness. Each element of the vector `cc` corresponds to an MDTB-spline basis function. Hence, `length(cc)` should be equal to `size(H,1)`. The parameter `n` is a positive integer scalar. 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 MDTB-spline multi-patch with smoothness and a vector of coefficients, and then plot the corresponding spline in MDTB-spline form:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> H = MDTB_extraction(MP, 2);
>> cc = [1, 3, 5, 4, 0, 2];
>> MDTB_visualization_spline(MP, H, cc, 100, 'LineWidth', 2);
```



4.19 MDTB_visualization_curve

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

Syntax:

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

Input parameters:

`MP`	: MDTB-spline multi-patch
`H`	: extraction matrix
`cc`	: matrix of control points (1D, 2D or 3D)
`n`	: number of evaluation points (optional)
`specs`	: pass any number of plot specifications (optional)

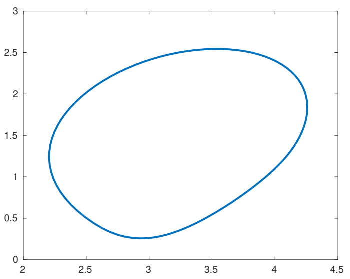
Discussion:

The extraction matrix `H` should be deduced from the MDTB-spline multi-patch `MP` and incorporates the smoothness. Each column of the matrix `cc` corresponds to an MDTB-spline basis function. Hence, `size(cc,2)` should be equal to `size(H,1)`. This matrix consists of 1, 2 or 3 rows, identifying a function, a 2D curve or a 3D curve, respectively. The parameter `n` is a positive integer scalar. 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/plot3`. We refer the reader to the documentation of `plot/plot3` for all the plotting options.

Example:

Create an MDTB-spline multi-patch with smoothness and a matrix of 2D control points, and then plot the corresponding 2D spline curve in MDTB-spline form:

```
>> MP = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> Hper = MDTB_extraction_periodic(MP, 2, 1);
>> ccper = [1, 3, 5, 4; 2, 0, 2, 4];
>> MDTB_visualization_curve(MP, Hper, ccper, 100, 'LineWidth', 2);
```



4.20 MDTB_conversion

This function converts a spline in (periodic) MDTB-spline form into another (periodic) MDTB-spline form. The conversion is exact when the source and destination MDTB-spline multi-patches with smoothness imply nested spaces. It is assumed that Greville points can be computed.

Syntax:

```
ccd = MDTB_conversion(MPd, Hd, MPs, Hs, ccs, sh, gl)
```

Input parameters:

`MPd`	: destination MDTB-spline multi-patch
`Hd`	: destination extraction matrix
`MPs`	: source MDTB-spline multi-patch
`Hs`	: source extraction matrix
`ccs`	: source coefficient vector
`sh`	: shift of the source patch (optional)
`gl`	: global conversion if true (optional)

Output parameters:

`ccd`	: destination coefficient vector

Discussion:

Different types of TB-spline patches within `MPs` and `MPd` may be mixed. 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 MDTB-spline basis function related to `Hs`. Hence, `length(ccs)` should be equal to `size(Hs,1)`. Similarly, each element of the resulting vector `ccd` corresponds to an MDTB-spline basis function related to `Hd`. The parameter `sh` is a real scalar. When no value is specified, `sh = 0` is assumed. The parameter `gl` is a boolean scalar. If `gl` is `true`, then a global conversion is computed; otherwise, a local patch-wise conversion is computed. The local conversion is more efficient, but requires that the MDTB-spline multi-patches `MPs` and `MPd` share the same number of TB-spline patches. Hence, `length(MPs.P)` should be equal to `length(MPd.P)`. Moreover, `MPd` should contain the same break points as `MPs` for the best local conversion results. When no value is specified, `gl = true` is assumed.

Example:

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

```
>> MPs = MDTB_patch_gpoly([3, 4], [0, 1, 3], [1, 2], [5, 2]);
>> Hs = MDTB_extraction(MPs, 2);
>> ccs = [1, 3, 5, 4, 0, 2];
>> MPd = MDTB_patch_gpoly([5, 7], [0, 1, 3], [1, 2], [5, 2]);
>> Hd = MDTB_extraction(MPd, 2);
>> ccd = MDTB_conversion(MPd, Hd, MPs, Hs, ccs, 0, false)
ccd =
  Columns 1 through 7
    1.0000    2.6857    3.2379    3.6200    4.3739    4.4464    3.7652
  Columns 8 through 11
    2.4723    1.3250    1.0897    2.0000
```

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

```
>> MPe = MPs;
>> He = MDTB_extraction(MPe, 1);
>> cce = MDTB_conversion(MPe, He, MPs, Hs, ccs, 0, false)
cce =
    1.0000    3.0000    3.9047    4.7632    4.0000   -0.0000    2.0000
```

Finally, find an approximation of the original spline using polynomial segments of the same multi-degree $(3, 4)$:

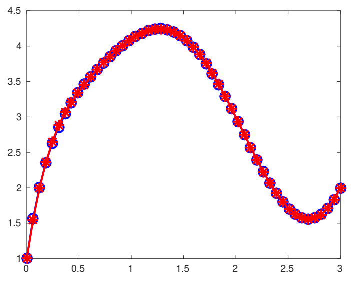
```
>> MPf = MDTB_patch_poly([3, 4], [0, 1, 3]);
>> Hf = MDTB_extraction(MPf, 2);
>> ccf = MDTB_conversion(MPf, Hf, MPs, Hs, ccs, 0, false)
ccf =
    1.0000    3.5162    4.7362    3.8197    0.1696    2.0000
```

Modifying the local Tchebycheff structure does not preserve the exact shape of the original spline, but it forms a reasonable approximation. The approximation can be improved by using locally polynomial spline segments (of uniform degree):

```
>> [MPg, rg] = MDTB_patch_poly([3, 3, 4, 4], [0, 1/2, 1, 2, 3], 2);
>> Hg = MDTB_extraction(MPg, rg);
>> ccg = MDTB_conversion(MPg, Hg, MPs, Hs, ccs, 0, false)
ccg =
  Columns 1 through 7
    1.0000    2.6365    3.4343    4.2898    4.4048    3.0828    1.4169
  Columns 8 through 9
    1.2354    2.0000
```

Make a visual comparison between the original spline (blue) and the polynomial spline approximation (red):

```
>> MDTB_visualization_spline(MPs, Hs, ccs, 50, 'LineWidth', 2, ...
>>              'Marker', 'o', 'MarkerSize', 10, 'Color', 'blue');
>> hold on;
>> MDTB_visualization_spline(MPg, Hg, ccg, 50, 'LineWidth', 2, ...
>>              'Marker', '*', 'MarkerSize', 10, 'Color', 'red');
>> hold off;
```



4.5
4
3.5
3
2.5
2
1.5
1
0
0.5
1
1.5
2
2.5
3