Finite Element Methods with B-Splines: Sample MATLAB Programs

Klaus Höllig and Jörg Hörner

November 2011

1 Introduction

The small collection of MATLAB programs (FEMB-Programs; version 1.0) is intended as a supplement to the SIAM book

"Finite Element Methods with B-Splines" (Frontiers in Applied Mathematics, 2003),

referred to by [**FB**] in the following. Our examples illustrate that the elegance and efficiency of B-spline algorithms, familiar from numerous applications, also prevail for the numerical solution of partial differential equations. This is due to the exceptionally simple data structure of uniform B-splines with its obvious advantages. Most importantly, regardless of the degree, the finite element subspace is spanned by translates of a single basis function, and the coefficients are associated with a regular grid; no mesh generation and connectivity data structures are necessary as for conventional finite element codes.

The goal of our sample programs is not to cover a broad range of partial differential equations or to provide B-spline software ready to serve potential applications. Instead, our MATLAB routines for basic model problems are intended as examples which show how to exploit standard B-spline features. In particular, the `m`-files provide illustrations for teaching purposes. Moreover, experimenting with our sample codes can facilitate the development of B-spline-based finite element software.

Since the introduction of WEB-splines by U. Reif, J. Wipper, and the first author [6], finite element methods with uniform B-splines have been implemented for many applications (cf., in particular [12, 1] and `www.web-spline.de/publications`). Our program collection builds on the experience gained throughout the past decade and also relies on standard finite element procedures (cf., e.g., [11]). A new key feature are the modified integration algorithms which directly use an implicit domain representation. This simplifies the main assembly routines. In particular, a parametric boundary representation is no longer required.

The programs solely use weighted B-spline approximations. While the extension procedure [**FB**, section 4.4] is essential for many theoretical aspects, it is usually not necessary in practice. Recent results of U. Reif and B. Mößner [9] support this observation.

In accordance with our primary purpose to provide an easy to understand introduction to B-spline finite element programming, we illustrate the methods described in [**FB**] in a very elementary setting:

- The simulation domain is a subset of $(0,1)^2$ or $(0,1)^3$.
- All boundary conditions are homogeneous.
- Only second order problems are considered.
- A basic conjugate gradient solver is used.

Most generalizations (cf. section 8) are straightforward and have partially been implemented. Moreover, the programs can be adapted to incorporate more recent developments, such as special cases (one-patch geometries) of isogeometric [3] and weighted isogeometric [4] techniques. Our implementation of multigrid algorithms [5] can serve as an example demonstrating that the overall program structure also adapts well to variants of B-spline finite elements.

Our programs are easily ported to any of the standard programming languages. We use MATLAB [8] for several reasons. MATLAB provides an ideal environment for program development, especially in the context of numerical methods. A programmer can focus on the essentials of algorithms and has convenient access to powerful functionality, in particular for visualization. Finally, the matrix oriented syntax naturally enforces a program structure which is well suited for vectorization.

2 If you do not like to read detailed program documentations ...

... here are some brief instructions to get started!

As a basic example, the following steps are required to solve Poisson's problem

$$-\Delta u = f$$

on a domain $D \subseteq (0,1)^2$ with homogeneous Dirichlet boundary conditions:

(i) Select a grid width `1/H` `H`$\in \mathbb{N}$ and a B-spline degree `n` and set defaults for the algorithmic parameters.

(ii) Define the force function `f` as an inline function.

(iii) Write an `m-file` with a weight function `w` describing the domain, i.e., `w` is positive on D and < 0 on all points in $(0,1)^2$ outside of $\overline{D}$. Include also partial derivatives up to second order[1].

(iv) Call the fe-solver `bvp_2d`.

Here is the complete MATLAB code for the disc

$$D: w(x,y) = 1/4 - (x-1/2)^2 - (y-1/2)^2 > 0$$

and the force function $f(x,y) = \sin(y-x)$.

MATLAB script `example_BASIC.m`:

```
H = 10; n = 3; PAR = set_par(H,n);
w = @w_BASIC
f = @(x,y) sin(y-x);
bvp_2d(@p_poisson,@q_poisson,f,w,w,H,n,PAR);
```

MATLAB weight function `w_BASIC.m`

```
function [W,Wx,Wy,Wxx,Wxy,Wyy] = w_BASIC(x,y)
W = 1/4-(x-1/2).^2-(y-1/2).^2;
Wx = -2*(x-1/2); Wy = -2*(y-1/2);

Wxx = -2*ones(size(x)); Wxy = zeros(size(x)); Wyy = Wxx;
```

Both `m-files` are included in the program collection.

The program `bvp_2d` plots the domain, the grid, and the solution (view the source code for the list of optional output variables) as shown in Figure 1.

Of course, any other force function `f` can be used. You may also replace `w_BASIC` by any other of the predefined weight functions, e.g.,

```
w_product
w_rvachev
w_bezier
...
```

Changing the parameters within these functions leads to further examples.

The input variables of the program `bvp_2d` indicate that there are many options $\Longrightarrow$ please, read on. But, you might try the demos and examples first. Just skip to sections 4 or 9 and experiment with the parameters of the sample programs. In addition, it is advisable to look at the appropriate chapters in [**FB**] in order to fully understand the theoretical background.

[1] Values of second derivatives are needed only for error estimation and can be set to `NaN` if the residual for the partial differential equation is not of interest.

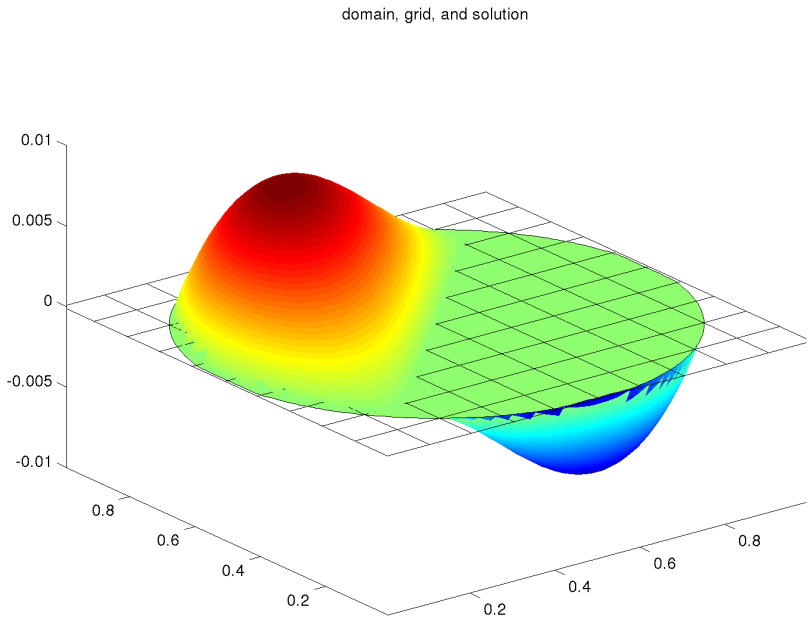


Figure 1: Solution plotted by the program `bvp_2d` as called by `example_BASIC`

3 Model Problems

The programs implement Ritz-Galerkin approximations for variational problems of the form [**FB**, section 2.4]

$$a(u,v) = \int_D fv \quad \forall v \in \mathcal{H}$$

which correspond to a partial differential equation

$$Lu = f$$

of second order. The bilinear form a is elliptic, f is a square integrable function, and $\mathcal{H}$ is a Hilbert space defined on the domain D. Moreover, $\mathcal{H}$ incorporates essential homogeneous boundary conditions:

$$u = 0 \quad \text{on} \quad \Gamma\,,$$

where Γ is a subset of the boundary ∂D of D. We consider three basic examples.

(P) Generalized Poisson problem [**FB**, section 6.3 (special case)]:

$$\begin{aligned} a(u,v) &= \int_D (\operatorname{grad} u)\, p\, (\operatorname{grad} v) \;+\; quv \\ Lu &= -\operatorname{div}(p \operatorname{grad} u) \;+\; qu\,, \end{aligned}$$

where $D \subset (0,1)^2$ or $D \subset (0,1)^3$, p is a positive and q a non-negative function on the closure of D.

(E2) Two-dimensional elasticity (plane strain and plane stress) [**FB**, section 6.6]:

$$\begin{aligned} a(u,v) &= \int_D \underline{\varepsilon}(u)\, Q\, \underline{\varepsilon}(v) \\ Lu &= -\alpha \begin{pmatrix} u^1_{xx} \\ u^2_{yy} \end{pmatrix} - (\beta+\gamma/4) \begin{pmatrix} u^2_{xy} \\ u^1_{xy} \end{pmatrix} - (\gamma/4) \begin{pmatrix} u^1_{yy} \\ u^2_{xx} \end{pmatrix}, \quad f = (f^1, f^2)\,, \end{aligned}$$

where $u = (u^1, u^2)$ is the displacement, $D \subset (0,1)^2$, $\underline{\varepsilon} = (\varepsilon_{1,1}, \varepsilon_{2,2}, \varepsilon_{1,2})$ is the two-dimensional strain tensor, and

$$Q = \begin{pmatrix} \alpha & \beta & 0 \\ \beta & \alpha & 0 \\ 0 & 0 & \gamma \end{pmatrix}$$

with α, β, γ defined in terms of material constants.

(E3) Three-dimensional elasticity [**FB**, section 6.5]:

$$\begin{aligned} a(u,v) &= \int_D q_1 \,\mathrm{trace}(u)\,\mathrm{trace}(v) + 2q_2 \varepsilon(u) : \varepsilon(v) \\ Lu &= -(q_1+q_2) \begin{pmatrix} u^1_{xx} + u^2_{xy} + u^3_{xz} \\ u^1_{xy} + u^2_{yy} + u^3_{yz} \\ u^1_{xz} + u^2_{yz} + u^3_{zz} \end{pmatrix} - q_2 \begin{pmatrix} u^1_{xx} + u^1_{yy} + u^1_{zz} \\ u^2_{xx} + u^2_{yy} + u^2_{zz} \\ u^3_{xx} + u^3_{yy} + u^3_{zz} \end{pmatrix}, \quad f = (f^1, f^2, f^3)\,, \end{aligned}$$

where $u = (u^1, u^2, u^3)$ is the displacement, $D \subset (0,1)^3$, and $q_k > 0$ are material constants.

For the elasticity problems, the constants α, β, γ, q_1, q_2 are defined in terms of the Young modulus and the Poisson ratio,

$$E, \quad \nu,$$

which characterize the elastic properties of the material.

For each of the above problems, the finite element approximation with B-splines requires the following:

- specification of the functions and constants appearing in the partial differential equations
- description of the domain and the essential boundary
- choice of the spline space.

The domain and the essential boundary are represented by weight functions.

Weight Functions. The domain D is a subset of the unit square ($d = 2$) or the unit cube ($d = 3$) described implicitly by a weight function w_D:

$$D = \{(x, y, \ldots) \in (0,1)^d : \, w_D(x, y, \ldots) > 0\} \,,$$

where the notation $(x, y, \ldots)$ stands for (x, y) or (x, y, z).

The boundary condition $u = 0$ on the essential part Γ of ∂D is incorporated by a weight function w which is of one sign on D and vanishes linearly on Γ[2]:

$$\Gamma = \{(x, y, \ldots) \in \partial D : \, w(x, y, \ldots) = 0\} \,.$$

We elaborate on these definitions in some more detail.

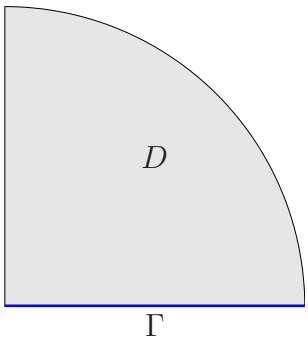
If the solution vanishes on the entire boundary (pure Dirichlet problem),

$$\Gamma = \partial D \quad \Leftrightarrow \quad w_D = w \,,$$

i.e., only a single weight function is required. On the other hand, if the normal derivative is zero on ∂D (pure Neumann problem),

$$\Gamma = \emptyset \quad \Leftrightarrow \quad w(x, y, \ldots) = 1 \,.$$

Again, we only need to construct a weight function w_D for the description of the simulation domain. For a mixed boundary value problem, the weight functions w_D and w are different. A simple example is shown in Figure 2. Since $D \subseteq (0,1)^2$, w_D describes a quarter disc. On the horizontal boundary Γ, the solution vanishes (Dirichlet or essential boundary). This part of ∂D is represented by w. On the vertical and circular boundary, the normal derivative of the solution is zero (Neumann or natural boundary).



$$\begin{aligned} D: &\quad (x,y) \in (0,1)^2 \ \wedge \ w_D(x,y) = 1 - x^2 - y^2 > 0, \\ \Gamma: &\quad (x,y) \in \partial D \ \wedge \ w(x,y) = y = 0 \end{aligned}$$

Figure 2: Weight functions for domain and essential boundary condition

[2] Slightly generalizing the definition 4.4 in [**FB**, section 4.3], it is not necessary to require that $w > 0$; it is sufficient to exclude zeros in $\overline{D} \backslash \Gamma$.

While the implicit boundary representation is exact, polynomial approximations are used in the implementation. Therefore, all functions appearing in the differential equation should be defined and continuous on all of $(0,1)^d$. Usually, in particular for explicit formulas, this does not present any problems. If it does, one can simply use 0 as function value on $(0,1)^d \backslash D$.

The finite element subspace is spanned by scaled translates of a single function, the uniform tensor product B-spline [**FB**, section 8.5]. (Run the demo `demo_spline_2d` for a visualization of this exceptionally simple basis.)

Splines. For all boundary value problems, solutions are approximated by linear combinations of weighted B-splines:

$$u(x,y,\ldots) \approx u_h(x,y,\ldots) = \sum_{k_1=1}^{H+n} \cdots \sum_{k_d=1}^{H+n} u_k \, w(x,y,\ldots) b_k(x,y,\ldots)$$

for $(x,y,\ldots) \in (0,1)^d$ with $w_D(x,y,\ldots) > 0$ and $d = 2$ or $d = 3$. Here,

- $h = 1/H$ is the grid width, i.e. $(0,1)^d$ is partitioned into H^d grid cells Q_m;
- n is the degree of the B-splines;
- the coefficients u_k are either scalars (problem (P)) or vectors (problems (E2), (E3)) with the number of indices depending on the dimension d, i.e.,

  $$u_k = U(k_1,k_2),\ U(k_1,k_2,m),\ U(k_1,k_2,k_3),\ U(k_1,k_2,k_3,m), \quad m = 1:d;$$

- b_k is the uniform tensor product B-spline with support

  $$[(k_1-n-1)h, k_1 h] \times \cdots \times [(k_d-n-1)h, k_d h]\,,$$

  i.e., $k = (k_1,\ldots,k_d)$ corresponds to the grid position.

Since, in general, D is a proper subset of $(0,1)^d$, not all B-spline coefficients are relevant. Keeping these irrelevant array entries for B-splines with no support in D is essential to maintain a simple data structure. In all programs, we work with the full set of $(H+n)^d$ B-splines which have some support in $(0,1)^d$. Of course, no problem arises, since values of irrelevant B-splines do not influence the solution on the domain D.

Unlike standard finite elements, weighted B-splines of degree $n > 1$ can be substituted into the differential equation, i.e., we can form the pointwise residual.

Residual for the partial differential equation. The residual of the Ritz-Galerkin approximation u_h is defined as

$$r(x, y, \ldots) = (Lu_h)(x, y, \ldots) - f(x, y, \ldots) \,,$$

where L is the differential operator of the boundary value problem. The relative error

$$e = \|r\|_{0,D} / \|f\|_{0,D} \,,$$

with $\| \ \|_{0,D}$ denoting the L_2-norm on D, provides a measure of accuracy for the solution without having to resort to grid refinement.

We expect that

$$e = O(H^{1-n})$$

for smooth solutions. For solutions with singularities caused, e.g., by corners or edges of the domain or discontinuities of the force function, the approximation order deteriorates. In fact, e needs not even be bounded for $H \to \infty$.

4 Finite Element Approximation

The problems (P), (E2), and (E3) are solved with the programs

```
bvp_2d, bvp_3d,
elasticity_2d,
elasticity_3d,
```

respectively. Simple illustrations are provided by the demos (cf. also section 9 for further examples)

```
demo_{program name}.
```

They do not require any input and allow interactive specification of parameters with the option of using defaults. We take the problems solved in the demos as basic examples for our three principal applications and discuss the MATLAB implementation now in more detail.

4.1 Generalized Poisson Problem in Two Dimensions

Generalized Poisson problems in two dimensions can be solved with the program

```
function [U,wuXY,X,Y,R,e] = bvp_2d(p,q,f,wD,w,H,n,PAR) .
```

As example (see the demo `demo_bvp_2d`), we consider the differential equation

$$-\operatorname{div}\Big(\underbrace{\exp(xy)}_{p}\operatorname{grad} u\Big) + \underbrace{(x+2y)}_{q}\, u = \underbrace{3x-y}_{f}$$

on the quarter disc

$$D : (x,y) \in (0,1)^2 \quad \wedge \quad w_D(x,y) = 1 - x^2 - y^2 > 0$$

with essential homogeneous boundary conditions $u = 0$ on the lower horizontal boundary

$$\Gamma : \ (x,y) \in \partial D \quad \wedge \quad w(x,y) = y = 0$$

(cf. Figure 2).

A MATLAB program begins by specifying the functions appearing in the differential equation, the weight functions for the domain and the essential boundary, and the grid width and degree of the B-splines.

```
% pde
p = @p_exp;
q = @(x,y) x+2*y;
f = @(x,y) 3*x-y;
% domain and essential boundary
wD = @(x,y) 1-x.^2-y.^2;
w  = @w_line;
% grid and degree
H = 10;
n = 3;
```

The functions `p` and `w` have to be supplied as `m-files` or local functions since they have several output arguments (function values and partial derivatives).

```
function [P,Px,Py] = p_exp(x,y)
% coefficient of the bilinear form
P = exp(x.*y);
Px = y.*P; Py = x.*P;

function [W,Wx,Wy,Wxx,Wxy,Wyy] = w_line(x,y)
% weight function for the x-axis as essential boundary
W = y;
Wx = zeros(size(x)); Wy = ones(size(y));
Wxx = Wx; Wxy = Wx; Wyy = Wx;
```

After the main input arguments have been defined, we can call the finite element solver with defaults for the algorithmic parameters.

```
% finite element approximation
PAR = set_par(H,n)
[U,wuXY,X,Y,R,e] = bvp_2d(p,q,f,wD,w,H,n,PAR);
```

The program plots the grid, the domain, and the graph of the weighted spline u_h. It returns the following output arguments.

`U(1:H+n,1:H+n)`
coefficients u_k of the finite element approximation u_h

`X(1:H*M,1:H*M), Y(1:H*M,1:H*M)`
regular grid of $(H\,M)^2$ centered evaluation points in $(0,1)^2$ with coordinates in

$$\{1, 3, \ldots, 2HM-1\}/(2HM)\,;$$

`M = PAR.values` is an algorithmic parameter with default value `ceil(64/H)` (see section 7). It controls the resolution for visualization. With the default value, approximately 64 evaluation points per coordinate direction are used.

`wuXY(1:H*M,1:H*M), R(1:H*M,1:H*M)`
numerical solution u_h and residual r on the evaluation grid; e.g., `R(k1,k2)` is the value of r at `X(k1,k2),Y(k1,k2)`.

`e`
relative error for the partial differential equation (see the definition of the residual in section 3).

The demo `demo_bvp_2d` allows to specify the input arguments `f`, `wD`, `H`, and `n` of the program `bvp_2d` interactively. For example, one can consider a sequence of grids, i.e., run the demo successively with

```
H = 2, 4, 8, ...
```

and determine the factors γ by which the error `e` is reduced in each refinement step ($\gamma \approx 2^{1-n}$ for smooth solutions[3]). It is also possible to define different domains D. For example,

```
wD = @(x,y) (x-1/2).^2+(y-1/2).^2-1/9
wD = @(x,y) 3+cos(pi*x)-4*y
...
```

are suitable choices. The only requirement is that the boundary of the domain specified by `wD` contains some part of the essential boundary $\Gamma = \{(x,0) : 0 \le x \le 1\}$.

[3] Larger factors γ can be caused by singularities due to the corners of the domain D.

4.2 Generalized Poisson Problem in Three Dimensions

Generalized Poisson problems in three dimensions can be solved with the program

```
function [U,wuXYZ,X,Y,Z,R,e] = bvp_3d(p,q,f,wD,w,H,n,PAR) .
```

An example (see the demo `demo_bvp_3d`) is the Neumann problem for the partial differential equation

$$-\Delta u + u = f, \quad f(x,y,z) = \exp(x+y+z)\,,$$

($p = 1$, $q = 1$) on a cube with a cylindrical hole:

$$D:\ (x,y,z) \in (0,1)^3 \quad \wedge \quad w_D(x,y,z) = (x-1/2)^2 + (y-1/2)^2 - 1/9 > 0\,.$$

Since the normal derivative vanishes on the entire boundary of D, the weight function `w` is identically equal to one, reflecting the fact that no Dirichlet boundary conditions are imposed ($\Gamma = \emptyset$).

The MATLAB script is completely analogous to the two-dimensional example, and we list the code without comment.

```
% pde
p = @p_poisson;
q = @(x,y,z) ones(size(x));
f = @(x,y,z) exp(x+y+z);
% domain and essential boundary
wD = @(x,y,z) (x-1/2).^2+(y-1/2).^2-1/9;
w  = @w_one;
% grid and degree
H = 10;
n = 2;
% finite element approximation
PAR = set_par(H,n);
[U,wuXYZ,X,Y,Z,R,e] = bvp_3d(p,q,f,wD,w,H,n,PAR)
```

We have used predefined functions for the "Poisson coefficient" $p(x,y,z) = 1$ and the "trivial weight" $w(x,y,z) = 1$. The finite element subspace consists of quadratic splines on a 10×10 grid. The program plots the domain boundary and visualizes the solution with a volumetric slice plot. It returns essentially the same output arguments as its two-dimensional counterpart – just the dimensions of the arrays change. Clearly, the evaluation grid has three components:

```
[X,Y,Z] = ndgrid(1/(2*H*M) : 1/(H*M) : 1)
```

with `M = PAR.values`, and the arrays of B-spline coefficients, solution values, and pointwise residuals are three-dimensional:

```
U(1:H+n,1:H+n,1:H+n
wuXYZ(1:H*M,1:H*M,1:H*M)
R(1:H*M,1:H*M,1:H*M)
```

As in the two-dimensional case, `e` is the relative error for the partial differential equation and approximates the L_2-norm of the residual divided by the L_2-norm of f.

In the demo `demo_bvp_3d`, one can choose `f`, `wD`, `H`, and `n` interactively. There are no restrictions on the weight function `wD` other than that it should lead to a "reasonable" form of the domain D. The simplest choice is

```
wD = @(x,y,z) ones(size(x))
```

defining D as the entire unit cube. In this case, an interesting experiment is to investigate the convergence for eigenfunctions of the differential operator by setting, e.g.,

```
f = @(x,y,z) cos(pi*x).*cos(2*pi*y).*cos(3*pi*z)
```

and increasing the degree in successive runs of the demo:

```
n = 2, 3, 4, ...
```

For fixed `H`, e.g., `H=6`, we expect that $e \approx \gamma^n$ with $\gamma < 1$.

4.3 Two Dimensional Elasticity: Plane Strain and Plane Stress

The elasticity problem (E2) (cf. section 3) can be solved with the program

```
function [U,wuXY,X,Y,R,e] = elasticity_2d(model,E,nu,f,wD,w,H,n,PAR) .
```

The plane strain (`model = 'strain'`) and the plane stress (`model = 'stress'`) models differ only in the coefficients of the bilinear form, which essentially depend on the Poisson ratio `nu`; the Young modulus `E` just amounts to a scaling and can be set to 1.

In the demo `demo_elasticity_2d`, we consider the deformation of a bridge-shaped, concrete structure, which is fixed at the bottom,

$$\begin{aligned} D &: (x,y) \in (0,1)^2 \quad \wedge \quad w_D(x,y) = y/(0.7) - 1 + (x-1/2)^2/(0.2)^2 > 0 \\ \Gamma &: (x,y) \in \partial D \quad \wedge \quad w(x,y) = y = 0 \end{aligned}$$

under gravity,

$$f(x,y) = (0,-1)\,.$$

For cubic B-splines on a 10×10 grid, the finite element approximation is computed with the following MATLAB script.

```
% pde
model = 'strain';
E = 1;
nu = 0.2;
f = @f_gravity;
% domain and essential boundary
height = 0.7; width = 0.4;
wD = @(x,y) y/height + (x-1/2).^2/(width/2)^2 - 1;
w = @w_line;
% grid and degree
H = 10;
n = 3;
% finite element approximation
PAR = set_par(H,n);
[U,wuXY,X,Y,R,e] = elasticity_2d(model,E,nu,f,wD,w,H,n,PAR);
```

The script uses two auxiliary functions, which are supplied in `m-files` and predefined in the program collection.

```
function [F1,F2] = f_gravity(x,y)
% normalized gravitational force
F1 = zeros(size(x)); F2 = -ones(size(y));

function [W,Wx,Wy,Wxx,Wxy,Wyy] = w_line(x,y)
...
```

(see subsection 4.1).

The program plots the domain, the grid, and the displacement $u_h \approx u$. The solution and consequently also its finite element approximation are vector-valued:

$$\begin{pmatrix} u^1(x,y) \\ u^2(x,y) \end{pmatrix} \approx \begin{pmatrix} u_h^1(x,y) \\ u_h^2(x,y) \end{pmatrix} = \sum_{k_1=1}^{H+n} \sum_{k_2=1}^{H+n} \begin{pmatrix} \mathtt{U}(k_1,k_2,1) \\ \mathtt{U}(k_1,k_2,2) \end{pmatrix} w(x,y)\, b_k(x,y)\,,$$

where `U` is the array of B-spline coefficients, computed by the program. Also returned are the values

`wuXY(1:H*M,1:H*M,1:2)`, `R(1:H*M,1:H*M,1:2)`

of the numerical solution u_h and of the residual r on the evaluation grid `X`, `Y`; `M = PAR.values`, as mentioned before. The relative error `e` for the partial differential equation is the last output argument.

In the demo `demo_elasticity_2d`, we can gradually change the domain by varying the parameters `height` and `width` in the parabola defining `wD`, e.g.,

```
height = 0.7 -> 0.75, 0.8, ...,
width = 0.4 -> 0.5, 0.6, ...
```

and observe the effect on the displacement. As usual, one can also specify `H` and `n` interactively.

Due to the corners of the domain the solution is not smooth. As a consequence, the error `e`, which involves second derivatives, increases, e.g., for the sequence

```
H: 8, 16, 32, ...
```

Nevertheless, the numerical solution converges in the L_2-norm (see subsection 9.5 for a similar example).

4.4 Three-Dimensional Elasticity

The elasticity problem (E3) (cf. section 3) can be solved with the program

```
function [U,wuXYZ,X,Y,Z,R,e] = elasticity_3d(E,nu,f,wD,w,H,n,PAR) .
```

In the demo `demo_elasticity_3d`, we compute the deformation of a rotating steel hyperboloid fixed at a cylindrical vertical axis:

$$\begin{aligned} D &: (x,y,z) \in (0,1)^3 \quad \wedge \quad 1/25 < (x-1/2)^2 + (y-1/2)^2 < (1+z^2)/9\,, \\ \Gamma &: (x,y,z) \in \partial D \quad \wedge \quad w(x,y,z) = 1/25 - (x-1/2)^2 - (y-1/2)^2 = 0\,. \end{aligned}$$

We need two auxiliary functions to describe the problem.

Centrifugal force:

```
function [F1,F2,F3] = f_centrifugal(x,y,z)
% centrifugal force:
F1 = x; F2 = y; F3 = zeros(size(z));
```

Weight function for the essential boundary:

```
function [W,Wx,Wy,Wz,Wxx,Wxy,Wxz,Wyy,Wyz,Wzz] = w_cylinder(x,y,z,r)
% cylinder with radius r and vertical axis (0,0,z)
W = r*r - x.*x-y.*y;
Wx = -2*x; Wy = -2*y; Wxx = -2+0*x;  Wyy = Wxx;
Wz = 0*z; Wxy = Wz; Wxz = Wz; Wyz = Wz; Wzz = Wz;
```

The MATLAB script for the finite element approximation has a by now familiar form.

```
% pde
E = 1;
nu = 0.28;
f = @(x,y,z) f_centrifugal(x-1/2,y-1/2,z);
% essential boundary and domain
w = @(x,y,z) w_cylinder(x-1/2,y-1/2,z,1/5);
profile = @(z) sqrt(1+z.^2)/3;
wD = @(x,y,z) -w(x,y,z).*(profile(z).^2-(x-1/2).^2-(y-1/2).^2);
% grid and degree
H = 8;
n = 1;
% finite element approximation
PAR = set_par(H,n);
[U,wuXYZ,X,Y,Z] = elasticity_3d(E,nu,f,wD,w,H,n,PAR);
```

Note that the negative product of the two elementary weight functions correctly describes the domain $D : wD > 0$. The minus sign is necessary, since `w_cylinder` is a weight function for the interior of a cylinder. To describe the cylinder with axis at $x = y = 1/2$, the function is called with shifted arguments `x-1/2,y-1/2`. Similarly `f_centrifugal` is also called with translated arguments.

The program `elasticity_3d` plots the boundary of D and visualizes the displacement as a vector field. For degree `n=1`, as chosen in the script, second derivatives do not exist. Therefore, we have omitted the output arguments `R` (residual) and `e` (error); the programs assign `NaN` to `R` and `e` in these cases. Returned are merely the B-spline coefficients `U` and the values `wuXYZ` of the displacement on the grid `X`, `Y`, `Z`:

$$\begin{aligned} u_h^m &= \sum_{k_1=1}^{H+n}\sum_{k_2=1}^{H+n}\sum_{k_3=1}^{H+n} \mathtt{U}(k_1,k_2,k_3,m)\, w\, b_k \\ \mathtt{wuXYZ}(\ell_1,\ell_2,\ell_3,m) &= u_h^m(X(\ell_1,\ell_2,\ell_3), Y(\ell_1,\ell_2,\ell_3), Z(\ell_1,\ell_2,\ell_3)), \quad \ell_\nu = 1 : H\,M \end{aligned}$$

(`M=PAR.values`) where $m = 1, 2, 3$ refers to the three components of the displacement $u_h \approx u = (u^1, u^2, u^3)$.

In the demo `demo_elasticity_3d`, it is possible to specify `H`, `n`, and the `profile` of the rotating structure, i.e., the outer radius R as a function of z. Possible choices are

```
profile = @(z) (1+z)/4              % cone
profile = @(z) (3+8*(z-1/2).^2)/10 % paraboloid
...
```

An essential restriction is

$$1/5 \le R = \mathtt{profile}(z) \le 1/2\,,$$

since otherwise the profile intersects the inner cylinder or lies outside of $(0,1)^3$.

5 Error Estimation

For points outside the domain D, the solution is not defined, and the corresponding entries of the arrays

```
wuXY, wuXYZ, R
```

are set to `NaN`. This has to be taken into account when calculating the relative error `e`, which is accomplished by the following MATLAB code segment.

```
R = R(:); F = F(:);
R(isnan(R)) = 0; F(isnan(R)) = 0;
e = norm(R)/norm(F);
```

As mentioned earlier, `e` approximates the quotient of the L_2-norms of the residual r and the force function f (`F` contains the grid values).

From the grid values

```
wuXY, wuXYZ
```

we can estimate the error of the finite element approximation in the L_2-norm in the usual way. We compare the numerical solutions on a sequence of grids with

```
H = Hmin, 2*Hmin, 4*Hmin, ...
```

To this end, it is convenient to use the same evaluation grids, i.e, the number `M = PAR.values`, which controls the number of evaluation points per grid cell, has to be adjusted:

```
M = Mmax, Mmax/2, Mmax/4, ...
```

With these choices, the same $(H_{\min} M_{\max})^d$ points are used on all grids.

As an illustration, we list a sample MATLAB code for estimating the L_2-error and the convergence rate.

```
...
grids = 4;
u_old = NaN;
for m = 1:grids;
    H = 2^m; PAR.values = 2^(grids-m);
    [U,wuXY,X,Y] = bvp_2d(p,q,f,w,w,H,n,PAR);
    wuXY(isnan(wuXY)) = 0;
```

```
    error(m) = norm(u_old(:) - wuXY(:))/norm(wuXY(:));
    u_old = wuXY;
end
rate = -diff(log(error))/log(2);
...
```

As is customary, the approximation on the finer grid takes the place of the exact solution, i.e.,

$$e_h = \frac{\|u_h - u\|_{0,D}}{\|u\|_{0,D}} \approx \frac{\|u_h - u_{h/2}\|_{0,D}}{\|u_{h/2}\|_{0,D}}\,,$$

as computed in the second last statement of the `for`-loop.

With the last MATLAB statement, we estimate the approximation order:

$$e_h \approx ch^\alpha \quad \Longrightarrow \quad \alpha \approx \log_2(e_h) - \log_2(e_{h/2})\,.$$

We expect that

$$e_h = O(H^{-n-1})\,,$$

i.e., $\alpha = n + 1$ for smooth solutions.

6 Principal Subroutines

The main programs

```
bvp_2d, bvp_3d, elasticity_2d, elasticity_3d,
```

which solve the generalized Poisson problem and the elasticity systems, have an almost identical structure. They successively call the subroutines

```
integrate_...
assemble_...
solve_...
evaluate_...
visualize_...
```

for determining Gauß parameters, assembling the Ritz-Galerkin system, computing the B-spline coefficients, evaluating the solution and the residual, and visualizing the results. There are several versions for each of these programs,

```
_... = _2d, _3d, _2de, or _3de,
```

to take the different dimension and specific features of the problems (P-2d), (P-3d), (E2), and (E3) into account.

The overall structure of the main programs resembles to some extent a simulation utilizing standard finite elements. We now describe each of the five principal steps in some more detail. This includes a discussion of various algorithmic parameters, which are combined in a structure

```
PAR
```

and mostly control the accuracy of various approximations. The fields of `PAR` can be defined separately. Usually, one will invoke the program

```
function PAR = set_par(H,n)
```

to assign default values and change only few of the algorithmic parameters.

6.1 Numerical Integration

The subroutines

```
function [int,cells] = integrate_2d(wD,H,PAR)
function [int,cells] = integrate_3d(wD,H,PAR)
```

determine Gauß parameters for integration over the relevant portions D_m of the grid cells

$$Q_m = [(m_1 - 1)h, m_1 h] \times \cdots \times [(m_d - 1)h, m_d h], \quad h = 1/H\,,$$

i.e., the sets

$$D_m = D \cap Q_m, \quad m_1, \ldots, m_d = 1 : H$$

($d = 2$ or $d = 3$).

Input for the integration routines are the weight function `wD`, describing the domain D, the number `H` of grid cells per coordinate, and the algorithmic parameters

```
PAR.fit, default: n+2,
PAR.gauss, default: n+1,
PAR.steps, default: 4,
PAR.tol_int, default: H^(-2*PAR.gauss-1).
```

These parameters specify the order of interpolation of the weight function, the minimal number of Gauß points per coordinate, the number of Gauß steps, and the local relative accuracy.

The programs return, in a structure `int`, for each grid cell Q_m Gauß coefficients and Gauß points. These parameters are used to approximate integrals of functions φ over relevant cell-subsets D_m. For example, for $d = 3$,

$$\int_{D_m} \varphi \approx \sum_{\nu} c_\nu \, \varphi(p(\nu,1), p(\nu,2), p(\nu,3)) \,,$$

where

```
c = int.c{m1,m2,m3}(:), p = int.p{m1,m2,m3}(:,1:3) .
```

In addition, the subroutines `integrate_...` return arrays

```
cells(1:H,1:H), cells(1:H,1:H,1:H)
```

with the cell types, which is convenient for further processing. The values $-1, 0, 1$ indicate an outer, boundary, and inner cell, respectively.

The programs for generating the Gauß parameters for computing finite element integrals basically use cell partitioning, as described in [**FB**, section 8.4]. However, they operate directly on the implicit domain representation, which simplifies the algorithm considerably.

In each grid cell Q_m, the weight function `wD` is interpolated by a polynomial at α^d ($\alpha =$ `PAR.fit`) equally spaced centered points. This results in an error of order

$$O(H^{-\alpha})$$

for smooth boundary portions. Hence, `PAR.fit` should be chosen larger than the B-spline degree `n` to maintain optimal accuracy. Of course, if the interpolation is exact, a smaller value might suffice (e.g., `PAR.fit=3` for quadrics).

For inner grid cells a tensor product Gauß formula with β^d ($\beta =$ `PAR.gauss`) points is used with a relative error of order

$$O(H^{-2\beta})$$

for smooth integrands. Choosing β greater than `n` is recommended to yield sufficiently accurate entries of the Ritz-Galerkin matrix.

For boundary cells, depending on the geometry, the number of Gauß points is successively raised (at most `PAR.steps` times) until the relative tolerance `PAR.tol_int` is met. If `PAR.steps=0`, the Gauß formula with β^d points is used throughout, which requires significantly less computing time. Setting `PAR.steps` equal to 1 does also not yet allow error estimation and is just equivalent to raising `PAR.gauss` by 1 for boundary cells.

As test for the local accuracy, the sums of the weights, which approximate the area ($d = 2$) or volume ($d = 3$) of $D_m = D \cap Q_m$ for consecutive Gauß formulas, are compared. Since this does not take the actual finite element integrands into account,

the estimate of the error is not very reliable and a pessimistic choice of the parameters is recommended.

This has been just a brief outline of the integration strategy. The algorithmic details cannot be explained in a short paragraph; in particular, the comments in the programs are probably not sufficient. We refer to the forthcoming publication [7].

The integration programs `integrate_2d`, `integrate_3d` require auxiliary data files

```
parameters_2d.mat, parameters_3d.mat
```

with precomputed variables which are problem-independent (Gauß parameters, interpolation matrices, etc.). These files can be generated with the programs

```
function parameters_2d(n_max)
function parameters_3d(n_max)
```

where `n_max` is an upper bound for `PAR.fit` and `PAR.gauss`. For the existing `mat`-files `n_max=10` was used. Exceeding this upper bound results in an error message.

6.2 Assembly of the Ritz-Galerkin System

Matrix and right side of the Ritz-Galerkin system are computed with the subroutines

```
function [G,F] = assemble_2d(p,q,f,w,H,n,int,PAR),
function [G,F] = assemble_3d(p,q,f,w,H,n,int,PAR),
function [G,F] = assemble_2de(model,E,nu,f,w,H,n,int,PAR),
function [G,F] = assemble_3de(E,nu,f,w,H,n,int,PAR).
```

These assembly routines convincingly demonstrate the beauty of B-spline elements: they just consist of a few statements inside fully parallelizable loops (cf. [**FB**, section 8.5]).

The input arguments pertain to the variational problem (`p,q,model,E,nu,f`), the spline space (`w,H,n`), and the Gauß parameters for numerical integration (`int`). A relative tolerance

```
PAR.tol_zero, default: 1000*eps,
```

is used to eliminate the influence of B-splines with excessively small values inside D, which are considered irrelevant.

The programs return the Ritz-Galerkin system in B-spline format, identifying matrix and vector elements with the position of the corresponding basis functions. For example, for three-dimensional elasticity,

$$\begin{aligned} G(k_1,k_2,k_3,s_1,s_2,s_3,i,j) &= a(e_i\, w\, b_k,\, e_j\, w\, b_\ell), \quad s_m = \ell_m - k_m + n + 1 \\ F(k_1,k_2,k_3,i) &= \int_D f^i\, w\, b_k\,, \end{aligned}$$

where e_1, e_2, e_3 are the unit vectors in $\mathbb{R}^3$, $e_i w b_k$ are the finite element basis functions and f^i are the components of the force function. Since the Ritz-Galerkin matrix has a regular band structure, we have identified the column indices ℓ with the diagonal shifts s_m ($s_1 = s_2 = s_3 = n + 1$ corresponds to the main diagonal). Of course, entries are nonzero only, if the supports of the B-splines b_k and b_ℓ overlap. Hence,

$$1 \le s_m \le 2n + 1\,,$$

and the dimensions of the arrays are

```
G(1:H+n,1:H+n,1:H+n,1:2*n+1,1:2*n+1,1:2+n+1,1:31,:3),
F(1:H+n,1:H+n,1:H+n,1:3),
```

still considering three-dimensional elasticity as a representative example.

6.3 Solution of the Linear System

The Ritz-Galerkin systems are solved with the subroutines

```
function U = solve_2d(G,F,PAR)
function U = solve_3d(G,F,PAR)
function U = solve_2de(G,F,PAR)
function U = solve_3de(G,F,PAR)
```

using a standard conjugate gradient algorithm with symmetric diagonal preconditioning. Of course, for the implementation of the basic matrix/vector multiplication with the scaled Ritz-Galerkin matrix, these solvers take the special B-spline format into account.

Input are the Ritz-Galerkin system, stored in the arrays `G`, `F`, and the parameters

```
PAR.tol_solve, default: H^(-n-3),
PAR.iterations, default: 100000.
```

The programs return the B-spline coefficients `U` of the Ritz-Galerkin approximation. For example, for three-dimensional elasticity,

$$u(x,y,z) \approx \sum_{m=1}^{3} \sum_{k_1=1}^{H+n} \sum_{k_2=1}^{H+n} \sum_{k_3=1}^{H+n} \mathtt{U}(k_1,k_2,k_3,m)\, w(x,y,z)\, e_m\, b_k(x,y,z)$$

where e_m is the m-th unit vector.

The algorithmic parameters are self-explanatory. The iteration stops if the initial residual is reduced by a factor `PAR.tol_solve`. The bound on the maximal number of iterations, `PAR.iterations`, is just a precaution in case of convergence failure. A sufficiently high value, e.g., 10000, is recommended.

Upon termination of the conjugate gradient procedure, we compute and print the norm of the residual to assess the accuracy of the solution.

6.4 Evaluation of the Ritz-Galerkin Approximation and of its Residual

The subroutines

```
function [wuXY,X,Y,R,e] = evaluate_2d(p,q,f,wD,w,U,H,n,PAR)
function [wuXYZ,X,Y,Z,R,e] = evaluate_3d(p,q,f,wD,w,U,H,n,PAR)
function [wuXY,X,Y,R,e] = evaluate_2de(model,E,nu,f,wD,w,U,H,n,PAR)
function [wuXYZ,X,Y,Z,R,e] = evaluate_3de(E,nu,f,wD,w,U,H,n,PAR)
```

evaluate the Ritz-Galerkin approximation on a regular grid and compute its residual for the differential equation.

The input arguments are identical to the main programs. They describe the partial differential equation (`p,q,model,E,nu,f,wD`) and the spline space (`w,U,H,n`); the parameter

```
PAR.values, default: ceil(64/H),
```

specifies the number of evaluation points per coordinate for the grid cells. For high quality images of the solution, a higher value of `PAR.values` is recommended; in particular, to avoid artifacts near the boundary.

The programs return the values `wuXY` ($d = 2$) or `wuXYZ` ($d = 3$) of the Ritz-Galerkin approximation at the grid points `X`, `Y`, ... which have coordinates in

$$\left\{ \frac{1}{2HM}, \frac{3}{2HM}, \ldots, \frac{2HM-1}{2HM} \right\}, \quad M = \texttt{PAR.values}\,.$$

For points outside the domain (`wD < 0`), the value `NaN` is assigned. In addition, the residual `R` and the relative error `e` for the partial differential equation is computed.

6.5 Visualization of the Solution

The subroutines

```
function visualize_2d(wD,wuXY,X,Y,H)
function visualize_3d(wD,wuXYZ,X,Y,Z,H)
function visualize_2de(wD,wuXY,X,Y,H)
function visualize_3de(wD,wuXYZ,X,Y,Z,H)
```

provide graphic illustrations of the numerical solution. As input they require the weight function `wD`, describing the domain, the values of the Ritz-Galerkin approximation `wuXY/wuXYZ` on the regular grid `X`, `Y`, ..., and the number `H` of grid cells per coordinate direction.

For the two-dimensional generalized Poisson problem (P), the grid, the domain, and the graph of the solution are shown. In three dimensions we visualize the solution with a volumetric slice plot. For the elasticity problems, the displacement is shown as a vector field. Moreover, in two dimensions, we color the domain D according to the size of the displacement. In three dimensions, just the boundary of D is shown.

The amount of supplementary output is controlled by the parameter

```
PAR.info, default: 2,
```

with the possible values 0 (quiet run), 1 (printed information only), and 2 (full visualization).

7 Auxiliary Programs

In the course of the finite element simulation, B-splines b and splines

$$\sum_k \mathtt{U}(k_1, k_2, \ldots) b_k, \quad b_k(x, y, \ldots) = b(xH - k_1, yH - k_2, \ldots)\,,$$

have to be evaluated many times, often together with their derivatives. This is accomplished by the following three programs.

```
function [b,db] = b_spline(x,n)
% evaluation of univariate uniform B-splines
```

```
function [uXY,X,Y] = spline_2d(U,H,n,M)
function [uXYZ,X,Y,Z] = spline_3d(U,H,n,M)
% values of a spline on a regular grid in (0,1)^d
```

While the first program uses the standard recurrence relation [2] to evaluate the uniform B-spline and its derivative, the two- and three-dimensional evaluation routines first compute the grid values of the uniform tensor product B-spline and then combine the local mask with the coefficient array. This is also a standard procedure, appropriate for all shift invariant bases.

We already mentioned the programs

```
function parameters_2d(n_max)
function parameters_3d(n_max)
```

which generate arrays used in the integration routines. Due to the regular grid, many variables can be precomputed, which accelerates the run-time of computations.

Finally, the program

```
function PAR = set_par(H,n)
```

sets default values for the algorithmic parameters.

8 Generalizations

With just a few hundred lines of code implementing B-spline solvers of arbitrary order for free-form domains, the scope of our program collection must remain rather limited. Of course, B-spline techniques apply to virtually every simulation based on partial differential equations. Moreover, even for the few model problems (P), (E1), and (E2), generalizations and enhancements are possible. Some of these are outlined below.

- Tools for constructing weight functions: Rvachev's method (cf. [10], [**FB**, section 4.3]) provides a general purpose technique for domains defined in terms of Boolean operations, as is customary, e.g., in constructive solid geometry. Moreover, numerical schemes based on evaluating distance functions have been implemented. In particular, it is possible to construct weight functions from parametric boundary descriptions (e.g., NURBS curves and surfaces). An approach yet to be explored are spline weight functions. We think that this will eventually yield the most versatile representations of free-form domains.

- Inhomogeneous boundary conditions: Boundary value problems can be reduced to homogeneous form in a standard fashion; a function which satisfies the inhomogeneous boundary conditions is subtracted from the solution. For domains constructed via R-functions, a systematic theory is available [10][4]. Alternatively, non-essential boundary conditions can be incorporated with appropriate boundary integrals into the variational formulation. If an additional parametric description of the boundary is available, such integrals can be treated in the assembly process similarly to the domain integrals. This has already been implemented. Finally, it is possible to transform boundary integrals into domain integrals, which eliminates the need of additional integration routines.

- Isogeometric elements: The routines for weighted B-splines easily extend to one-patch isogeometric and weighted isogeometric approximations [4]. This allows a particularly elegant handling of trim curves and surfaces. A natural application, for which web-spline-based programs have already been written, is the analysis of shells.

  Isogeometric approximations, which use more than one patch or nonuniform knot sequences, cannot be handled as simply as the uniform weighted B-spline elements. Programs for general isogeometric elements require connectivity arrays and resemble mesh-based finite element codes. As a consequence, much of the advantages of the regular data structure for uniform B-splines is lost.

- Multigrid solvers: The conjugate gradient solver should be replaced by a multigrid algorithm whenever possible. Due to natural refinement procedures and the regular

[4] The procedures utilize normalized weight functions and can be simplified for the standard Neumann and Robin boundary conditions.

grid, B-splines are ideally suited for this fastest solution technique (cf. **FB**, chapter 8 and [5]).

- Hierarchical refinement: Natural adaptive refinement strategies for B-spline elements are available [**FB**, section 4.5]. However, they have yet to be implemented. This seems particularly promising in conjunction with the a posteriori error bound provided by the residual. Refinements are made, where the pointwise residual `R` is larger than a given tolerance.

- Approximation of derivatives: Derivatives of the weight function `w` and the coefficient `p` in the bilinear form can be approximated on the grid cells via polynomial interpolation. Then, only routines for evaluation of these functions need to be supplied.

- Performance optimization: Some improvements of the efficiency of our algorithms are possible, but would make the codes less readable. For example, on inner grid cells, precomputed arrays with B-spline values at Gauß points avoid duplicating computations. The weight function `w` can be blended with a plateau of constant height on a subset of D, leading to a further simplification; `w` has to be constructed and evaluated only in a narrow boundary strip. Because of symmetry, only the lower triangular part of the Ritz-Galerkin systems needs to be computed. For large three-dimensional problems, it might be essential to incorporate such modifications in order to obtain the best possible performance.

Clearly, much work lies ahead. However, we are convinced that, with a joint effort of mathematicians and engineers, B-splines will prove to be as successful for finite element methods as they already are in other disciplines.

9 Demos and Examples

In addition to the demos for the main programs (see section 4),

```
function demo_bvp_2d(),
function demo_bvp_3d(),
function demo_elasticity_2d(),
function demo_elasticity_3d(),
```

demos are available for the integration routines and the functions for evaluating splines:

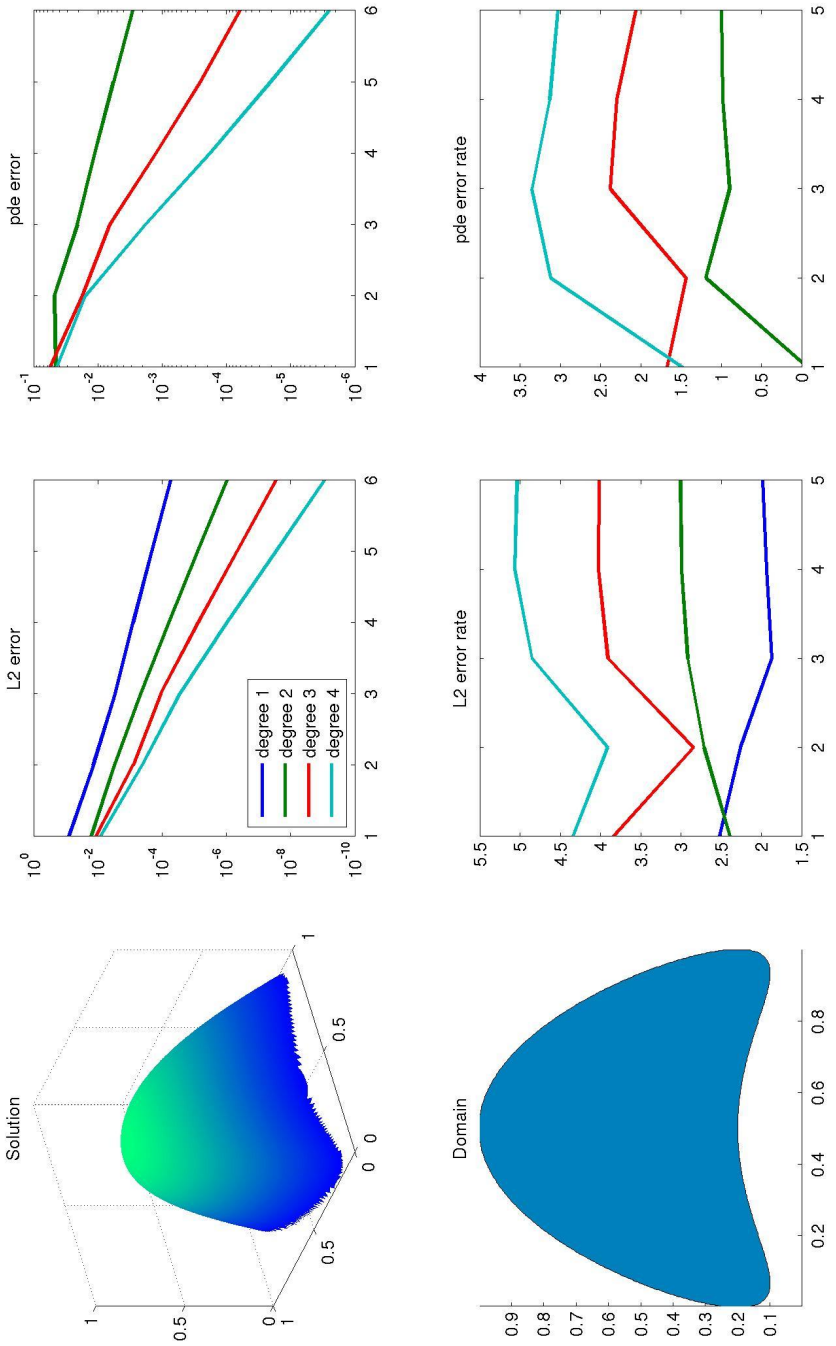
```
function demo_integrate_2d(),
function demo_integrate_3d(),

function demo_spline_2d(),
function demo_spline_3d().
```

These demos allow interactive specification of the input arguments with the option of using defaults.

The use of our main programs is also illustrated with several examples, which are listed below. The corresponding MATLAB functions do not allow any input. But, parameters in the `m-files` can be changed in order to experiment with the algorithms.

9.1 Convergence Rates for a Smooth Domain [FB, section 6.1]

```
function example_bvp_2d_convergence
% convergence analysis for the Poisson problem -Delta u = f
% on the domain D: w>0 with Dirichlet boundary conditions
% solution u = sin(w) -> f = -Delta sin(w)
...
% maximal degree n and number of grid refinements
n_max = 4; H_steps = 4;

% functions describing domain, essential boundary, and pde
% (see also LOCAL FUNCTIONS as well as m-files w_shovel.m and p_poisson.m)
w = @w_shovel; wD = w;
p = @p_poisson; q = @(x,y) zeros(size(x));
f = @(x,y) f_delta_sinw(x,y,w);

% L2 and pde errors for different degrees and grid widths
fh = figure;
for n = 1:n_max
  for k = 1:H_steps+1
    H = 2^(k+1);
    PAR=set_par(H,n); PAR.info = 1; PAR.values = 3;
    [U,wuXY,X,Y,R,error_pde(k,n)] = bvp_2d(p,q,f,w,w,H,n,PAR);
    uXY_exact = sin(w(X,Y)); uXY_exact(isnan(wuXY))=0;
    error = uXY_exact(:) - wuXY(:); error(isnan(error)) = 0;
    error_L2(k,n) = norm(error(:))/norm(uXY_exact(:));
  end
  rate_pde = -diff(log(error_pde))/log(2);
  rate_L2 = -diff(log(error_L2))/log(2);
  % visualization
  ...
end

%%%%% LOCAL FUNCTIONS
function F = f_delta_sinw(x,y,w)
% force f = -Delta u for solution u(x,y) = sin(w(x,y))
...
```

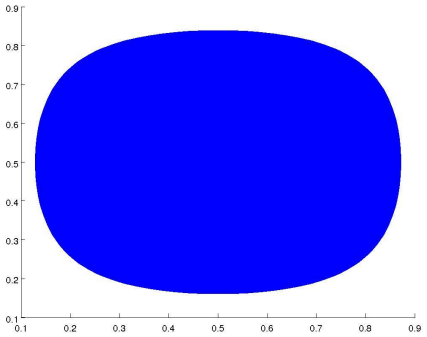

pde error
pde error rate
L2 error
L2 error rate
degree 1
degree 2
degree 3
degree 4
Solution
Domain

9.2 L_2-Error for Increasing Degree [FB, section 6.3]

```
function example_bvp_3d_convergence
% convergence (increasing degree, fixed grid width) for
%  -Delta u + u = exp(x-2y+3z)
% on a smooth domain with natural boundary conditions
...

% grid and maximal degree
H = 4; n_max = 4;

% pde, domain, and natural boundary conditions
% (see also m-files p_poisson.m, w_exp_3d.m, and w_one.m)
p  = @p_poisson;
q  = @(x,y,z) ones(size(x));
f  = @(x,y,z) exp(x-2*y+3*z);
wD = @w_exp_3d;
w  = @w_one;

% relative L2 error for different degrees
for n=1:n_max
  PAR=set_par(H,n); PAR.values = 5; PAR.info = 1;
  [U,u_H,X,Y,Z] = bvp_3d(p,q,f,wD,w,H,n,PAR);
  u_H(isnan(u_H)) = 0;
  % estimate error via difference of consecutive solutions
  if n>1
    e = u_H-u_old; error(n-1) = norm(e(:))/norm(u_old(:));
  end
  u_old = u_H;
end
... plot domain and print results
```

Printed result:

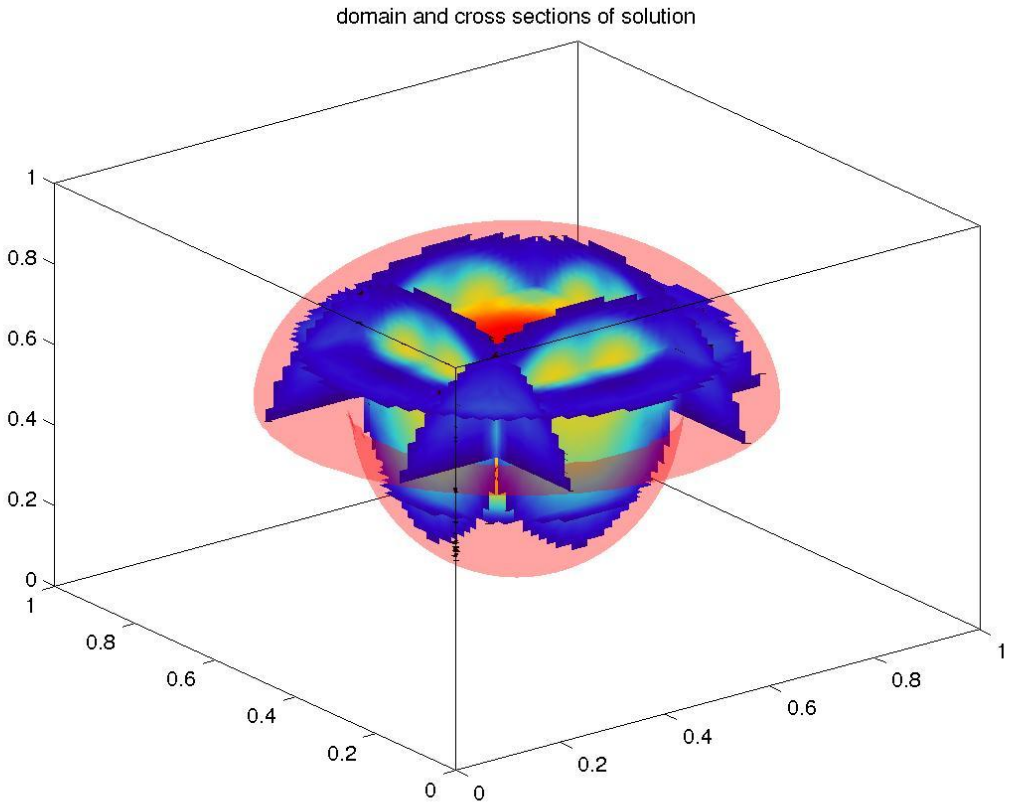
```
degree       L2 error
     1    2.96649e-03
     2    4.64664e-04
     3    5.21756e-05
```



9.3 Rvachev's Method for CSG Domains [FB, sections 4.3, 8.3]

```
function example_bvp_3d_csg
% Poisson's equation for a domain constructed with Boolean operations
...

% grid width and degree
H = 8; n = 1;

% pde (see also m-file p_poisson.m)
p  = @p_poisson;
q  = @(x,y,z) zeros(size(x));
f  = @(x,y,z) ones(size(x));

% finite element approximation
PAR = set_par(H,n); PAR.steps = 0;
bvp_3d(p,q,f,@w_csg,@w_csg,H,n,PAR);

%%%%% LOCAL FUNCTIONS
% domain
function [W,Wx,Wy,Wz,Wxx,Wxy,Wxz,Wyy,Wyz,Wzz] = w_csg(x,y,z)
% combining elementary weight functions via Rvachev's method

% ball
W1 = 4/25-(x-1/2).^2-(y-1/2).^2-(z-1/2).^2;
W1x = -2*(x-1/2); W1y = -2*(y-1/2); W1z = -2*(z-1/2);

% half space
W2 = z-1/2;
W2x = 0*x; W2y = W2x; W2z = ones(size(z));

% ellipsoid
W3 = 1-16*(x-1/2).^2-16*(y-1/2).^2-25/4*(z-1/2).^2;
W3x = -32*(x-1/2); W3y = -32*(y-1/2); W3z = -25/2*(z-1/2);

% intersection of ball with half space
s = sqrt(W1.^2+W2.^2);
W = W1+W2-s;
Wx = W1x+W2x-(W1.*W1x+W2.*W2x)./s;
Wy = W1y+W2y-(W1.*W1y+W2.*W2y)./s;
Wz = W1z+W2z-(W1.*W1z+W2.*W2z)./s;

% union with ellipsoid
s = sqrt(W.^2+W3.^2);
```

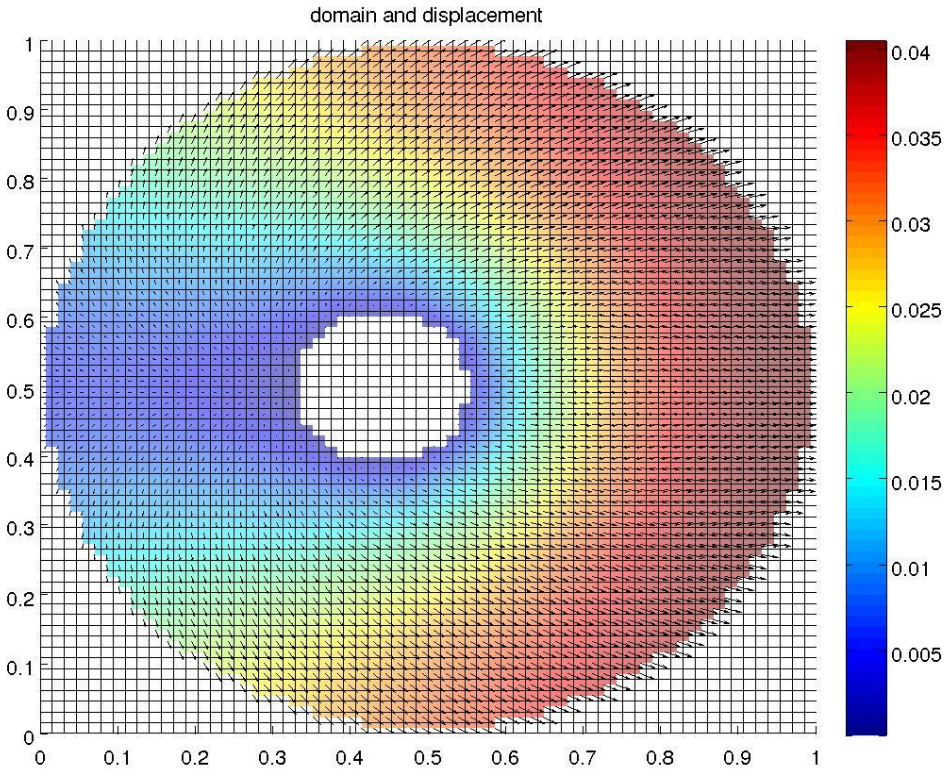
```
W = W+W3+s;
Wx = Wx+W3x+(W.*Wx+W3.*W3x)./s;
Wy = Wy+W3y+(W.*Wy+W3.*W3y)./s;
Wz = Wz+W3z+(W.*Wz+W3.*W3z)./s;

% leave second derivatives undefined
Wxx = NaN*W; Wxy = Wxx; Wxz = Wxx; Wyy = Wxx; Wyz = Wxx; Wzz = Wxx;
```



9.4 Plane Stress [FB, section 6.6]

```
function example_elasticity_2d_disc
% displacement of an excentric rotating steel disc
...

% offset and inner radius
d = 0.06; r = 0.1;

% degree and refinement steps
n = 4; H_steps = 3;
```

```
% Young modulus (normalized), Poisson ratio, and force
% (see m-file f_centrifugal.m)
E  = 1; nu = 0.28;
f  = @(x,y) f_centrifugal(x-1/2+d,y-1/2)

% domain and essential boundary
% (see m-file w_disc)
wD = @(x,y) -w_disc(x-1/2,y-1/2,1/2).*w_disc(x-1/2+d,y-1/2,r)
w  = @(x,y) w_disc(x-1/2+d,y-1/2,r)

% finite element approximations with different grid widths
figure
for k=1:H_steps+1
  H(k) = 4*2^k; PAR = set_par(H(k),n);
  [U,wuXY,X,Y,R,e(k)] = elasticity_2d('stress',E,nu,f,wD,w,H(k),n,PAR);
pause(1)
end

... compute and print error for the pde and estimated convergence rate
```



Printed result:

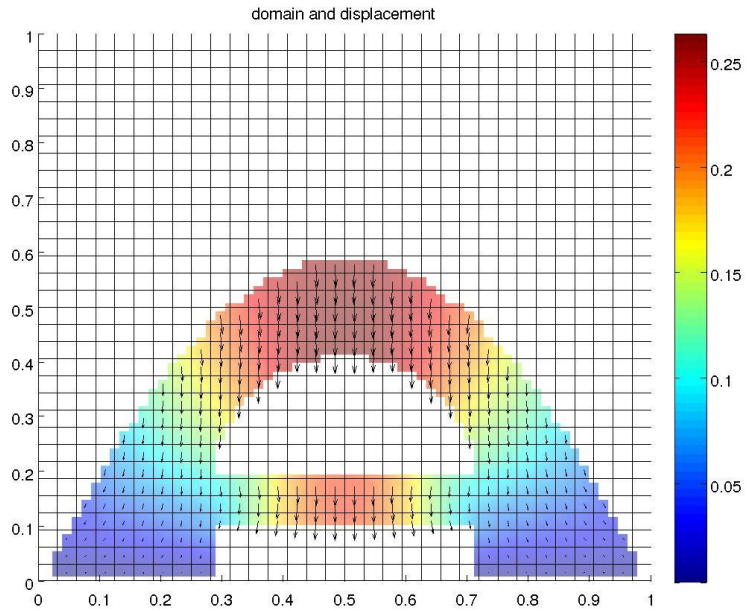
```
  H      pde error           rate
  8      1.235e+00            NaN
 16      3.236e-01      1.932e+00
 32      8.570e-03      5.239e+00
 64      2.567e-04      5.061e+00
```

9.5 Plane Strain [FB, section 6.6]

```
function example_elasticity_2d_tunnel
% displacement of a tunnel-shaped concrete structure under gravity
% (singular solution -> low accuracy)
...
% degree, refinement steps
n = 1; H_steps = 3;

% Young modulus (normalized) and Poisson ratio
E = 1; nu = 0.2;

% force, domain, and essential boundary
% (see m-files f_gravity.m, wD_tunnel.m, and w_line.m)
f = @f_gravity;
wD = @wD_tunnel; w = @w_line;

% finite element approximations with different grid widths
figure
for k=1:H_steps+1;
  H{k} = 2*2^k; PAR=set_par(H{k},n);
  [U,wuXY{k},X,Y] = elasticity_2d('strain',E,nu,f,wD,w,H{k},n,PAR);
  wuXY{k}(isnan(wuXY{k}))=0;
  pause(1)
end
... print estimated L2 error
```

Printed result:

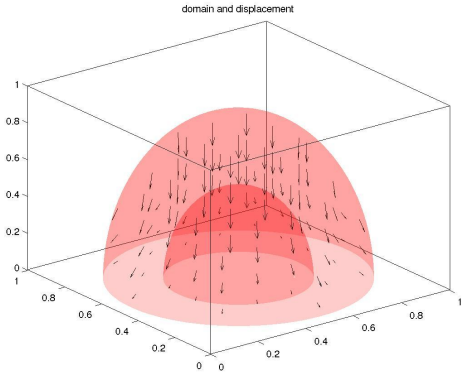
```
  H       L2 error
  4    2.61008e-01
  8    1.07042e-01
 16    7.89070e-02
```

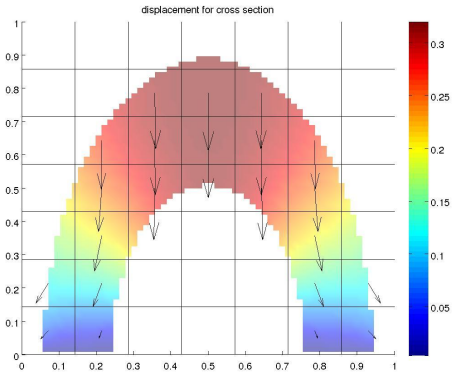


9.6 Deformation under Gravity [FB, section 6.5]

```
function example_elasticity_3d_dome
% displacement of a concrete dome under gravity
...
% inner and outer half axes of ellipsoids
ri = [5 5 10]/20; ro = [9 9 18]/20;

% degree, grid, and evaluation points
n = 3;
H = 7; % odd
M = 9; % odd

% algorithmic parameters
...

% Young modulus (scaled), Poisson ratio, and force
% (see m-file f_gravity.m)
E = 1; nu = 0.3;
f = @f_gravity;

% domain and essential boundary
% (see m-files w_ellipsoid.m and w_plane.m)
wD = @(x,y,z) -w_ellipsoid(x-1/2,y-1/2,z,ri).*w_ellipsoid(x-1/2,y-1/2,z,ro);
w  = @w_plane;
```

```
% finite element approximation
figure
[U,wuXYZ,X,Y,Z] = elasticity_3d(E,nu,f,wD,w,H,n,PAR);
... plot of  cross section
```





10 Terms of Use

The programs listed in the following section are test versions, intended only for educational use and for illustrating the methods described in the SIAM book "Finite Element Methods with B-Splines". No permission is granted for any other application, in particular for any commercial use or applications of the programs, which can result in material or physical damage. A possible lack of reliability of the numerical algorithms, which to some extent are based on heuristic strategies, has been emphasized. As a consequence, no guarantees for the correctness of the computed results can be given.

In addition to the above restrictions, the general terms of MathWorks for using MATLAB programs apply.

List of Files

The following files are provided by our program collection FEMB-Programs; version 1.0.

Main programs

bvp_2d.m	bvp_3d.m
elasticity_2d.m	elasticity_3d.m
integrate_2d.m	integrate_3d.m
spline_2d.m	spline_3d.m
assemble_2d.m	assemble_3d.m
assemble_2de.m	assemble_3de.m
solve_2d.m	solve_3d.m
solve_2de.m	solve_3de.m
evaluate_2d.m	evaluate_3d.m
evaluate_2de.m	evaluate_3de.m
visualize_2d.m	visualize_3d.m
visualize_2de.m	visualize_3de.m

Demo and example functions

demo_bvp_2d.m	f_centrifugal.m
demo_bvp_3d.m	f_gravity.m
demo_elasticity_2d.m	wD_tunnel.m
demo_elasticity_3d.m	w_ball.m
demo_integrate_2d.m	w_BASIC.m
demo_integrate_3d.m	w_bezier.m
demo_spline_2d.m	w_cylinder.m
demo_spline_3d.m	w_disc.m
example_BASIC.m	w_ellipsoid.m
example_bvp_2d_convergence.m	w_exp_3d.m
example_bvp_3d_convergence.m	w_hyperboloid.m
example_bvp_3d_csg.m	w_line.m
example_elasticity_2d_disc.m	w_one.m
example_elasticity_2d_tunnel.m	w_plane.m
example_elasticity_3d_dome.m	w_product.m
p_poisson.m	w_rvachev.m
q_poisson.m	w_shovel.m

Auxiliary functions and data files

b_spline.m	set_par.m
parameters_2d.m	parameters_3d.m
parameters_2d.mat	parameters_3d.mat
gausspar.m	vanderinv.m
Contents.m	ReadMe.txt

Acknowledgement.

We gratefully acknowledge the cooperation with Christian Apprich and Marco Boßle in many B-spline projects. Moreover, we thank Dr. Joachim Wipper, one of the co-founders of the WEB-method, for valuable suggestions and Dr. Bernadetta-Kwintiana Ane for commenting on our program collection from an engineering point of view. Irmgard Walter has carefully read various drafts of our documentation and as usual, kept administrative duties for us to a minimum. Special thanks also to our wives Elisabeth and Monika for their support and patience during many weekends and evenings of program development.

References

[1] G. Apaydin: *Finite Element Method with Web-splines for Electromagnetics,* VDM Verlag Dr. Müller, Saarbrücken, 2009.

[2] C. de Boor: *A Practical Guide to Splines,* Springer-Verlag, New York, 1978.

[3] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs: *Isogeometric Analysis,* Wiley, 2009.

[4] K. Höllig, J. Hörner, and A. Hoffacker: *Finite Element Analysis with B-splines: Weighted and Isogeometric Methods,* to appear in LNCS 6920, Springer, 2011.

[5] K. Höllig and J. Hörner: *Programming Multigrid Methods with B-splines,* Stuttgarter Mathematische Berichte, preprint 2011/016.

[6] K. Höllig, U. Reif, and J. Wipper: *Weighted Extended B-spline Approximation of Dirichlet Problems,* SIAM J. Numer. Anal. 39 (2001), 442–462.

[7] J. Hörner: *Mehrdimensionale numerische Integration für Finite-Elemente-Verfahren mit B-Splines,* in preparation.

[8] The MathWorks, Inc.: MATLAB - *The Language Of Technical Computing,* `http://www.mathworks.com/products/matlab/index.html`, Website, 2011.

[9] B. Mößner and U. Reif: *Stability of Tensor Product B-splines on Domains,* J. Approx. Theory 154 (2008), 1-19.

[10] V. L. Rvachev and T. I. Sheiko: *R-functions in Boundary Value Problems in Mechanics,* Appl. Mech. Rev. 48 (1995), 151-188.

[11] G. Strang and G. J. Fix: *An Analysis of the Finite Element Method,* Prentice–Hall, Englewood Cliffs, NJ, 1973.

[12] J. Wipper: *Finite-Elemente-Approximation mit WEB-Splines,* Shaker Verlag, Aachen, 2005.