GeoPDES

an Octave/Matlab Software for Research on IGA

J. Corno[1,2]

[1]Graduate School CE and [2]Institut für Theorie Elektromagnetischer Felder, TU Darmstadt, Germany

TECHNISCHE UNIVERSITÄT DARMSTADT

GRADUATE SCHOOL computational engineering

Institut für Theorie Elektromagnetischer Felder

Computational Electromagnetics Research Group at GSCE

Motivation and Introduction

Starting point: different codes, different problems, different developers.

Primary goal: uniform implementation of the codes, clear and easy to use, for didactic purposes.

In the presentation we will see two different packages.

- The NURBS toolbox, for geometry definition and manipulation.
- GeoPDEs, for isogeometric analysis, discretisation of PDEs.

Both packages are implemented in Octave (compatible with Matlab), open source and free.

Contributors

- Main developer:
  - **Rafael Vázquez** from IMATI-CNR (Pavia, IT) and EPFL (Lausanne, CH).
- List of contributors:
  - **Viacheslav Balobanov**, Bernoulli-Euler beam, error in H^2 norm.
  - **Andrea Bressan**, implementation of SubGrid methods and other contributions for Stokes problem.
  - **Elena Bulgarello**, contributions to the 3D div-conforming splines.
  - **Adriano Côrtes**, Nitsche's method for enforcing Dirichlet conditions with div-conforming splines.
  - **Jacopo Corno**, beta testing and geometry of the Tesla cavity.
  - **Luca Dedè**, advection-diffusion equation.
  - **Carlo de Falco**, contributions to the design of the code, implementation and maintenance of oct-files, implementation of many low level functions, and constant advice and support.
  - **Sara Frizziero**, contributions to the 3D div-conforming splines.
  - **Eduardo M. Garau**, algorithms and implementation of adaptive methods with hierarchical B-splines.
  - **Monica Montardini**, collocation methods.
  - **Marco Pingaro**, implementation of the bilaplacian and the Kirchhoff-Love plate.
  - **Alessandro Reali**, contributions to the original design of the code.
  - **Anna Tagliabue**, advection-diffusion equation.
  - **Lorenzo Tamellini**, collocation methods.

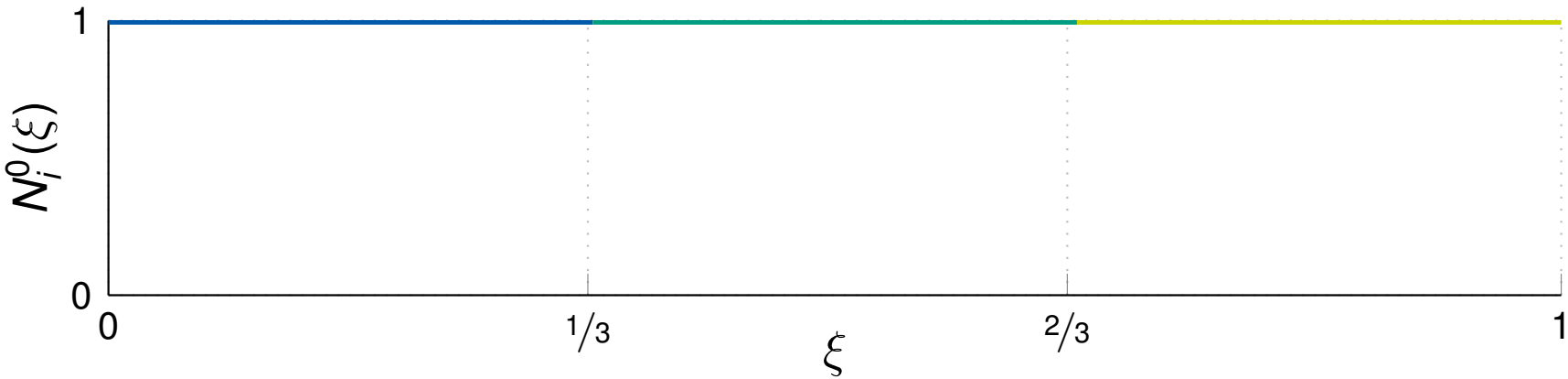
Outline

B-Spline Basis Functions

Given a *non-uniform knot vector* $\Xi = [\xi_1, ..., \xi_{n+p+1}]$, in the parametric domain $[0, 1]$, B-Spline basis functions are:

$$N_i^0(\xi) = \begin{cases} 1 \text{ if } \xi_i \leq \xi < \xi_{i+1} \\ 0 \text{ otherwise} \end{cases}$$

$$N_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1}^{p-1}(\xi).$$

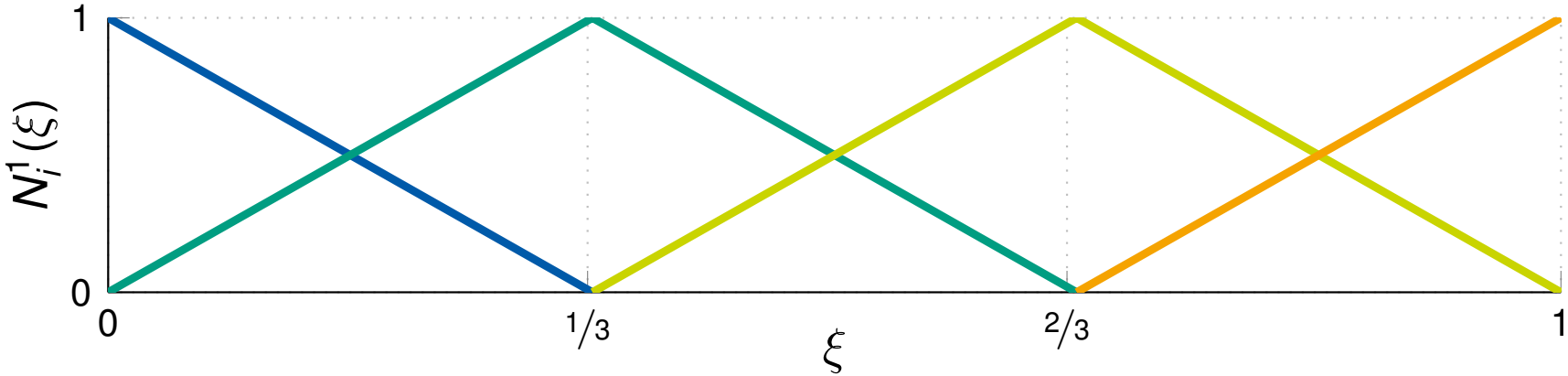


B-Spline Basis Functions

Given a *non-uniform knot vector* $\Xi = [\xi_1, ..., \xi_{n+p+1}]$, in the parametric domain $[0, 1]$, B-Spline basis functions are:

$$N_i^0(\xi) = \begin{cases} 1 \text{ if } \xi_i \leq \xi < \xi_{i+1} \\ 0 \text{ otherwise} \end{cases}$$

$$N_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1}^{p-1}(\xi).$$

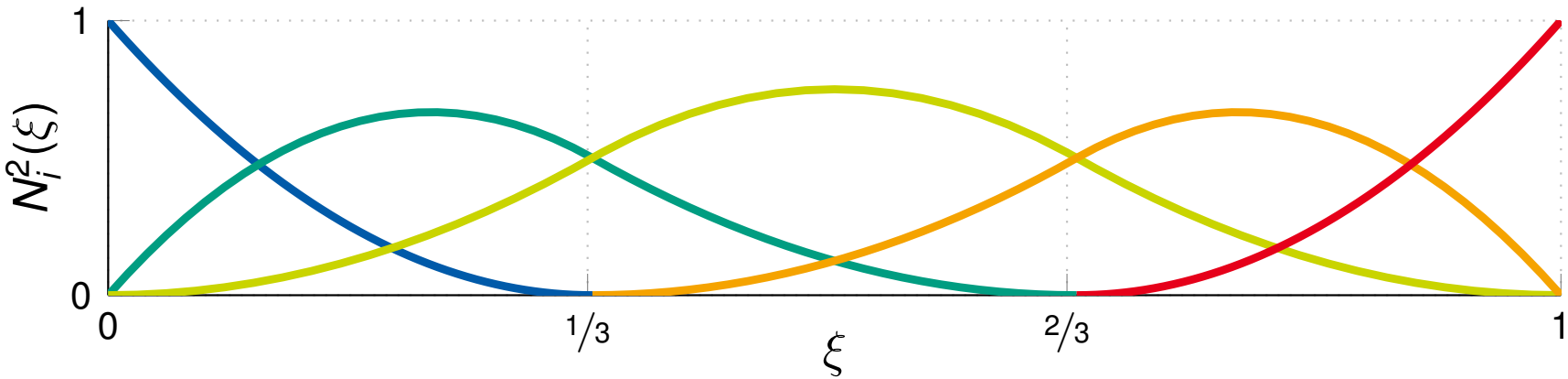


B-Spline Basis Functions

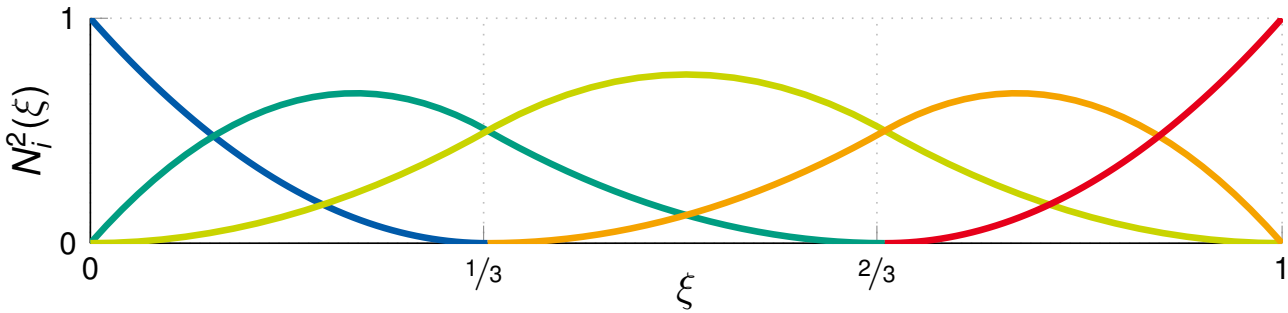
Given a *non-uniform knot vector* $\Xi = [\xi_1, ..., \xi_{n+p+1}]$, in the parametric domain $[0, 1]$, B-Spline basis functions are:

$$N_i^0(\xi) = \begin{cases} 1 \text{ if } \xi_i \leq \xi < \xi_{i+1} \\ 0 \text{ otherwise} \end{cases}$$

$$N_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1}^{p-1}(\xi).$$



B-Splines: Main Properties



B-spline basis functions have the following properties:

- N_i^p are non-negative with partition-of-unity property
- Locally linearly independent on each knot span (ξ_i, ξ_{i+1})
- Support of N_i^p is (ξ_i, ξ_{i+p+1})
- Piecewise polynomials of degree p, and regularity at most $p - 1$

The regularity at ξ_i is controlled by the knot multiplicity m_i

- Number of continuous derivatives: $p - m_i$, with $m_i \leq p + 1$
- For $m_i = p$, B-Splines become interpolatory at the knot
- Multiple knots also reduce the support of B-Splines

B-Splines: Basis Functions Evaluation

To compute B-spline basis functions, the data we need are:

- Degree `p` (order $p+1$)
- Knot vector `knt`, including knot repetitions (length = m)
- Number of basis functions `ncp` ($n_{cp} = m - p - 1$)

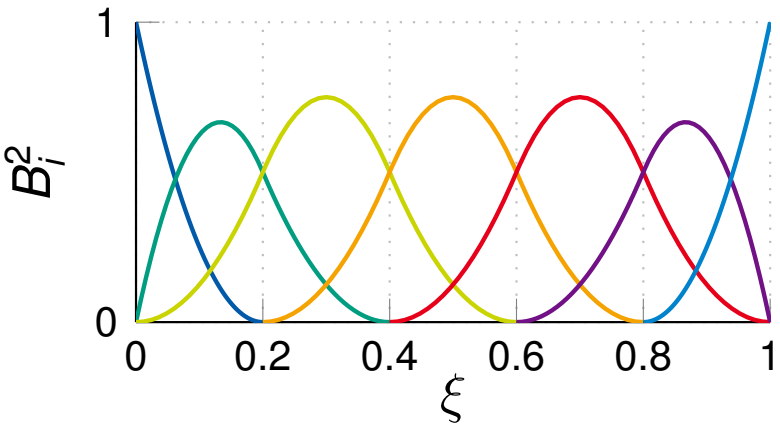
In the NURBS toolbox, they are evaluated using **`basisfun`**.
The derivatives can be computed using **`basisfunder`**.

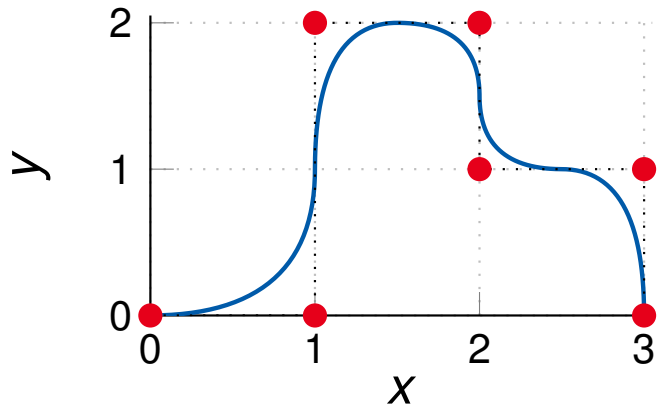
```
iv   = findspan (ncp-1, p, x, knt);
N    = basisfun (iv, x, p, knt);
Nder = basisfunder (iv, p, x, knt, nder);
```

For each point, only the $p+1$ non-vanishing functions are computed.
The global numbering is recovered using **`numbasisfun`**.

```
num = numbasisfun (iv, x, p, knt) + 1;  % Compute the numbering
```

B-Spline Curves: Definition





A B-Spline curve in $\mathbb{R}^d$ is defined as a linear combination of B-Splines:

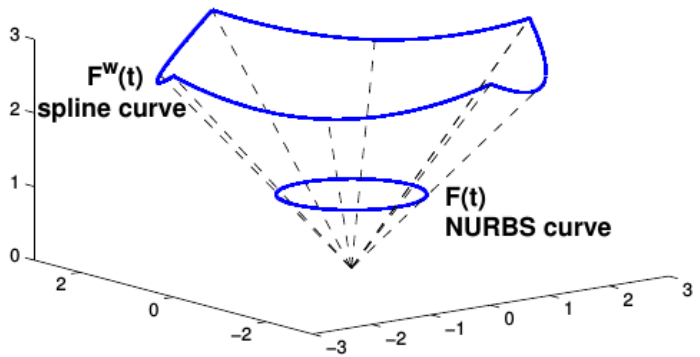
$$\mathbf{F}(\xi) = \sum_{i=0}^{n} \mathbf{C}_i N_i^p(\xi)$$

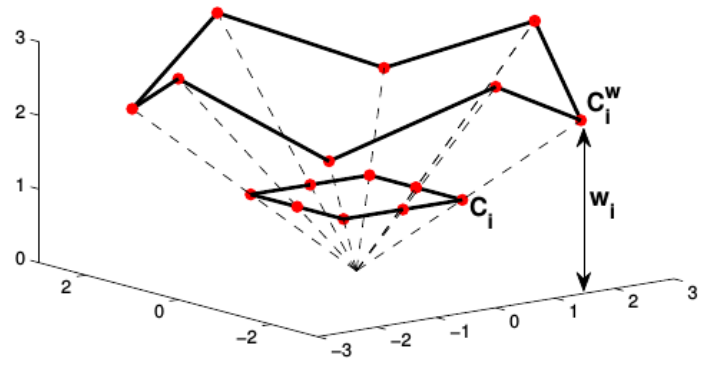
To define the parametrisation **F** we only need:

- The basis functions N_i^p, given by the knot vector
- The control points $\mathbf{C}_i \in \mathbb{R}^d$

NURBS Curves: Definition

A NURBS object in $\mathbb{R}^d$ is the projection of a B-Spline object in $\mathbb{R}^{d+1}$





In practice, a weight w_i is associated to each B-Spline function:

$$w_i = (\mathbf{C}_i^w)_{d+1}, \qquad R_i^p = \frac{N_i^p w_i}{\sum_{j=1}^{n} N_j^p w_j} \qquad \mathbf{F}(\xi) = \sum_{i=0}^{n} \mathbf{C}_i R_i^p(\xi)$$

- NURBS basis functions are piecewise rational polynomials
- They maintain most of the properties of B-splines
- Exact representation of conic sections

NURBS Curves: Definition in the NURBS Toolbox

To define B-Splines (or NURBS) curves, we use the command **`nrbmak`**.
It requires as input the knot vector and the control points.

```
crv = nrbmak (coefs, knt);
```

It creates a Matlab structure, which contains the following fields:

- **number**: the number of control points, `ncp`.
- **coefs**: control points coordinates (for NURBS also the weights).
- **order**: the degree plus one, $p + 1$.
- **knots**: the knot vector, `knt`.

The curve can be visualised with the commands **`nrbplot`**.

```
nrbplot (crv, 1000);
```

NURBS Curves: Definition in the NURBS Toolbox

To define B-Splines (or NURBS) curves, we use the command **`nrbmak`**.
It requires as input the knot vector and the control points.

```
crv = nrbmak (coefs, knt);
```

It creates a Matlab structure, which contains the following fields:

- **number**: the number of control points, `ncp`.
- **coefs**: control points coordinates (for NURBS also the weights).
- **order**: the degree plus one, $p + 1$.
- **knots**: the knot vector, `knt`.

The curve can be visualised with the commands **`nrbplot`**.

```
nrbplot (crv, 1000);
```

Important: coefs contains the **"weighted" control points**, i.e. $\mathbf{C^w}$.

NURBS Curves: Evaluation

Other functions for visualisation based on the previous structure: **`nrbkntplot`** and **`nrbctrlplot`**.

```
nrbnktplot (crv);  % Plot the curve with knots on it
nrbctrlplot (crv); % Plot the curve with the control polygon
```

NURBS Curves: Evaluation

Other functions for visualisation based on the previous structure: **`nrbkntplot`** and **`nrbctrlplot`**.

```
nrbnktplot (crv);  % Plot the curve with knots on it
nrbctrlplot (crv); % Plot the curve with the control polygon
```

The parametrisation **F** is evaluated with **`nrbeval`**:

```
x = 0:0.001:1;
F = nrbeval (nrb, x);
```

NURBS Curves: Evaluation

Other functions for visualisation based on the previous structure: **`nrbkntplot`** and **`nrbctrlplot`**.

```
nrbnktplot (crv); % Plot the curve with knots on it
nrbctrlplot (crv); % Plot the curve with the control polygon
```

The parametrisation **F** is evaluated with **`nrbeval`**:

```
x = 0:0.001:1;
F = nrbeval (nrb, x);
```

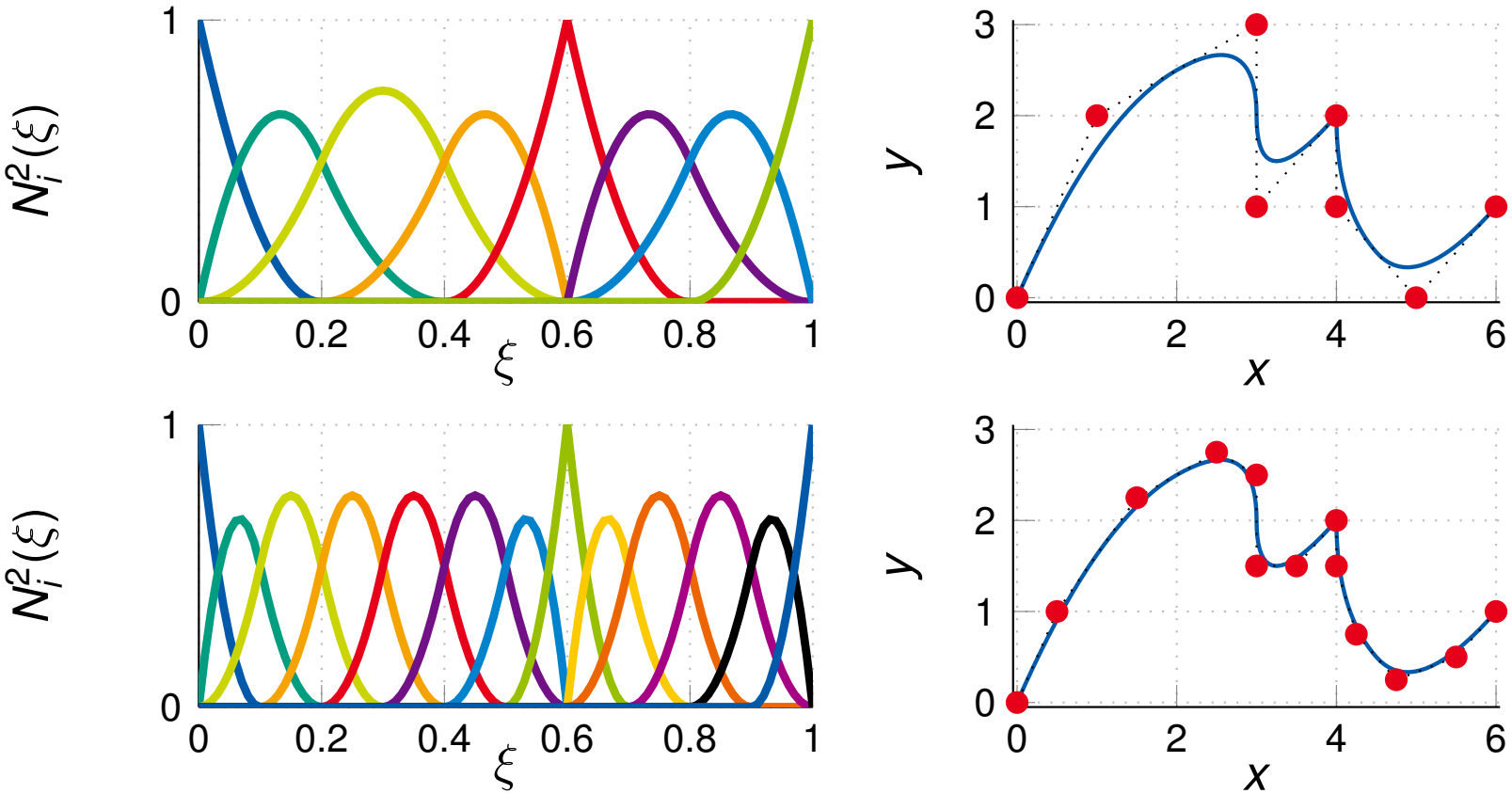
For IGA, we will also need the derivatives of the parametrisation, $\mathbf{F}'(\xi)$. They are computed with **`nrbderiv`** and **`nrbdeval`**:

```
dcrv = nrbderiv (crv);
[F, dF] = nrbdeval (crv, dcrv, x);
```

The second derivative can be computed in a very similar way.

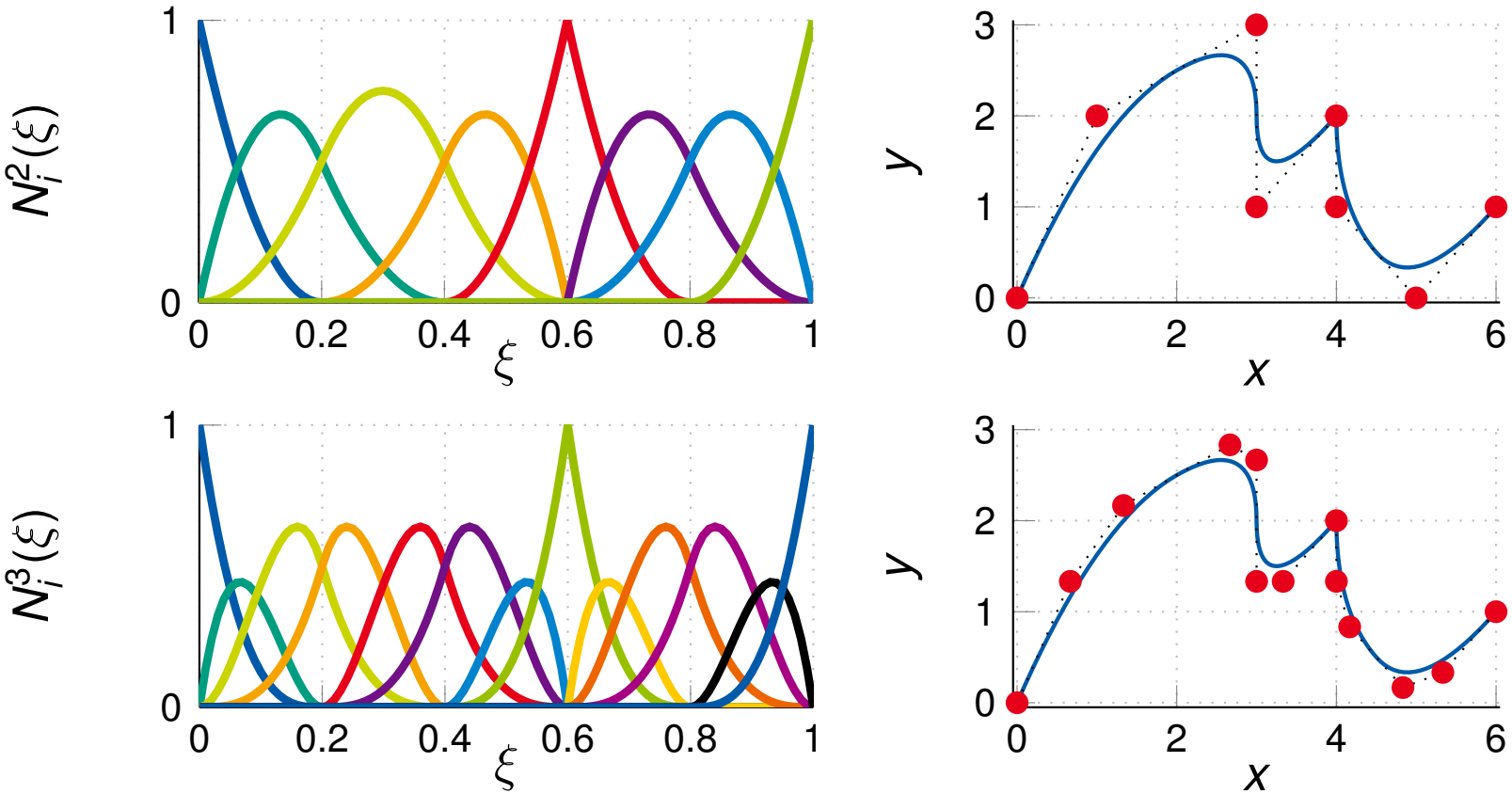
Knot Insertion (*h*-refinement)

Knot insertion: add more knots maintaining the parametrisation **F**.
In CAD it is used to add detail in some regions.
In IGA it is the analogous of *h*-refinement for FEM.



Degree Elevation (*p*-refinement)

Degree elevation: raise the degree maintaining the parametrisation **F**.
In IGA it is the analogous of *p*-refinement for FEM.



NURBS Toolbox: Knot Insertion and Degree Elevation

Knot insertion and degree elevation are implemented in the toolbox. The first input and the output are NURBS structures.

- For knot insertion, the function is **`nrbkntins`**. For instance, to add once the knot 1/10, and twice the knot 3/10:

```
new_crv = nrbkntins (crv, [1/10, 3/10, 3/10]);
```

NURBS Toolbox: Knot Insertion and Degree Elevation

Knot insertion and degree elevation are implemented in the toolbox. The first input and the output are NURBS structures.

- For knot insertion, the function is **`nrbkntins`**. For instance, to add once the knot 1/10, and twice the knot 3/10:

```
new_crv = nrbkntins (crv, [1/10, 3/10, 3/10]);
```

- For degree elevation, the function is **`nrbdegelev`**. For instance, to raise the degree by 2:

```
new_crv = nrbdegelev (crv, 2);
```

NURBS Toolbox: Knot Insertion and Degree Elevation

Knot insertion and degree elevation are implemented in the toolbox. The first input and the output are NURBS structures.

- For knot insertion, the function is **`nrbkntins`**. For instance, to add once the knot 1/10, and twice the knot 3/10:

```
new_crv = nrbkntins (crv, [1/10, 3/10, 3/10]);
```

- For degree elevation, the function is **`nrbdegelev`**. For instance, to raise the degree by 2:

```
new_crv = nrbdegelev (crv, 2);
```

- *k*-refinement can be obtained as a combination of both.

NURBS Toolbox: Knot Insertion and Degree Elevation

Knot insertion and degree elevation are implemented in the toolbox. The first input and the output are NURBS structures.

- For knot insertion, the function is **`nrbkntins`**. For instance, to add once the knot 1/10, and twice the knot 3/10:

```
new_crv = nrbkntins (crv, [1/10, 3/10, 3/10]);
```

- For degree elevation, the function is **`nrbdegelev`**. For instance, to raise the degree by 2:

```
new_crv = nrbdegelev (crv, 2);
```

- *k*-refinement can be obtained as a combination of both.
- If you want to perform a uniform refinement, the toolbox contains a function (**`kntrefine`**) to find the new knots to be inserted.

Surfaces and Volumes

The NURBS structure is also valid for surfaces and volumes.

It contains the knot vector in each direction, and a grid of control points.

Most of the functions for curves also work for surfaces and volumes: **`nrbmak`**, **`nrbctrlplot`**, **`nrbeval`**, **`nrbderiv`**, **`nrbkntins`**, **`nrbdegelev`**, ...

Surfaces and Volumes

The NURBS structure is also valid for surfaces and volumes.

It contains the knot vector in each direction, and a grid of control points.

Most of the functions for curves also work for surfaces and volumes: **`nrbmak`**, **`nrbctrlplot`**, **`nrbeval`**, **`nrbderiv`**, **`nrbkntins`**, **`nrbdegelev`**, ...

It is possible to create the NURBS by hand, using **`nrbmak(coefs,knt)`**.

- The knot vectors are given in a cell-array, of dimension 2 or 3.
- The control points are in an array of size (4, n1, n2) or (4, n1, n2, n3).
- The fourth coordinate is always the weight, equal to one for B-splines.

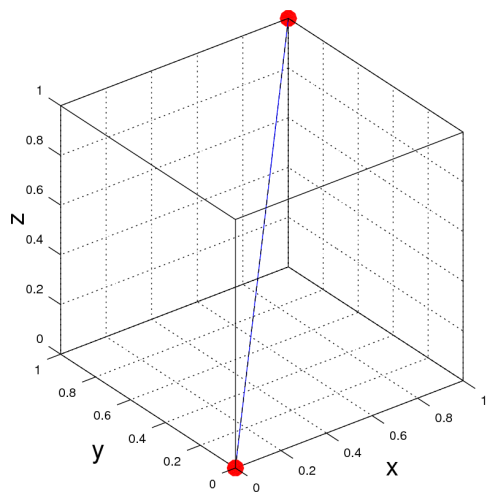
It is easier to create the geometry with functions of the toolbox.

Simple Geometric Entities

The starting point should be one of the simplest geometries.

- **`nrbline`**: creates a straight line in $\mathbb{R}^d$
- **`nrb4surf`**: creates a bilinear surface in $\mathbb{R}^d$ (including planar surfaces)
- **`nrbcirc`**: creates an arc of circunference in $\mathbb{R}^2$
- **`nrbcylind`**: creates the surface of a cylinder, in $\mathbb{R}^3$

```
1 crv = nrbline ([0 0 0], [1 1 1]);
  nrbctrlplot (crv);
```

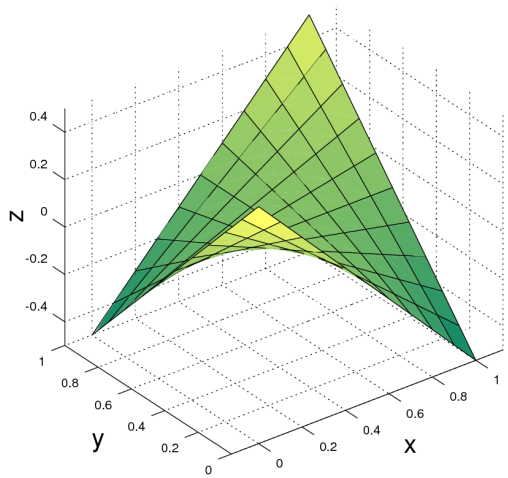


Simple Geometric Entities

The starting point should be one of the simplest geometries.

- **nrbline**: creates a straight line in $\mathbb{R}^d$
- **nrb4surf**: creates a bilinear surface in $\mathbb{R}^d$ (including planar surfaces)
- **nrbcirc**: creates an arc of circunference in $\mathbb{R}^2$
- **nrbcylind**: creates the surface of a cylinder, in $\mathbb{R}^3$

```
srf = nrb4surf ([0 0 0.5], [1 0 -0.5],
                [0 1 -0.5], [1 1 0.5]);
nrbplot (srf, [10, 10]);
```

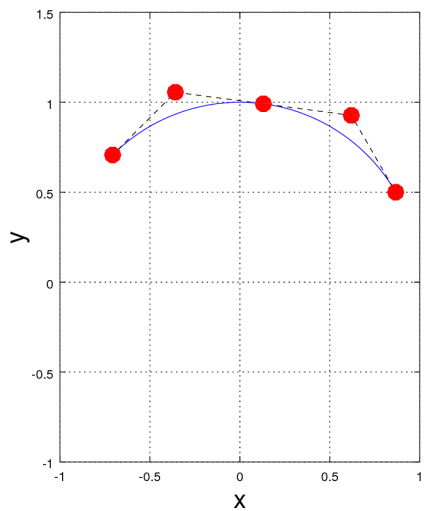


Simple Geometric Entities

The starting point should be one of the simplest geometries.

- **`nrbline`**: creates a straight line in $\mathbb{R}^d$
- **`nrb4surf`**: creates a bilinear surface in $\mathbb{R}^d$ (including planar surfaces)
- **`nrbcirc`**: creates an arc of circunference in $\mathbb{R}^2$
- **`nrbcylind`**: creates the surface of a cylinder, in $\mathbb{R}^3$

```
1 crv = nrbcirc (1, [0 0], pi/6, 3/4*pi);
  nrbctrlplot (crv);
```

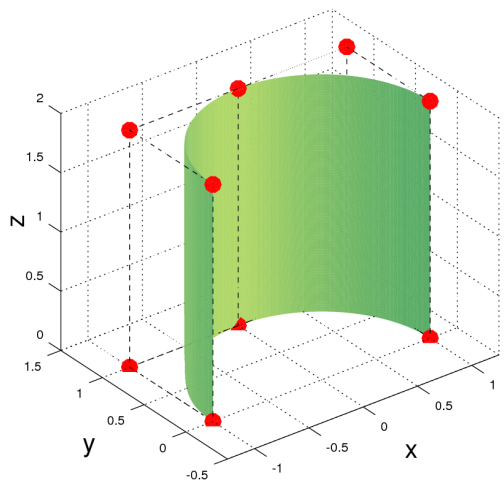


Simple Geometric Entities

The starting point should be one of the simplest geometries.

- **`nrbline`**: creates a straight line in $\mathbb{R}^d$
- **`nrb4surf`**: creates a bilinear surface in $\mathbb{R}^d$ (including planar surfaces)
- **`nrbcirc`**: creates an arc of circunference in $\mathbb{R}^2$
- **`nrbcylind`**: creates the surface of a cylinder, in $\mathbb{R}^3$

```
cyl = nrbcylind (2, 1, [0 0], 0, pi);
nrbctrlplot (cyl)
```

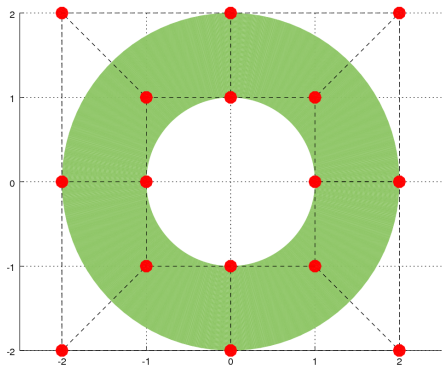


Revolution and Extrusion

The toolbox allows to create geometries by revolution and extrusion.

`nrbrevolve`: create a surface by revolution of a curve, or create a volume by revolution of a surface.

```
crv = nrbline ([1 0], [2 0]);
srf = nrbrevolve (crv, [0 0 0], [0 0 1], 2*pi);
```

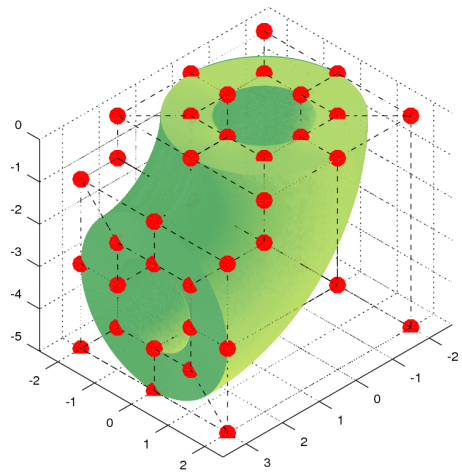


Revolution and Extrusion

The toolbox allows to create geometries by revolution and extrusion.
`nrbrevolve`: create a surface by revolution of a curve, or create a volume by revolution of a surface.

```
crv = nrbline ([1 0], [2 0]);
srf = nrbrevolve (crv, [0 0 0], [0 0 1], 2*pi);
```

```
vol = nrbrevolve (srf, [3 0 0], [0 -1 0], pi/2);
```



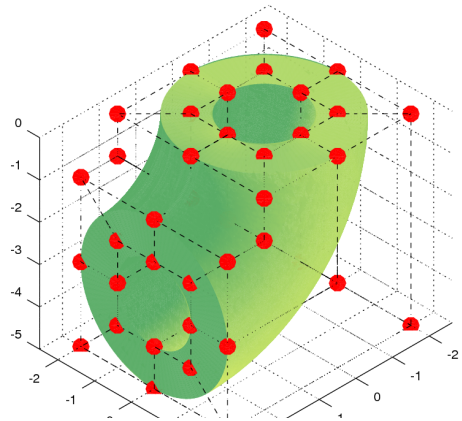
Revolution and Extrusion

The toolbox allows to create geometries by revolution and extrusion.

`nrbrevolve`: create a surface by revolution of a curve, or create a volume by revolution of a surface.

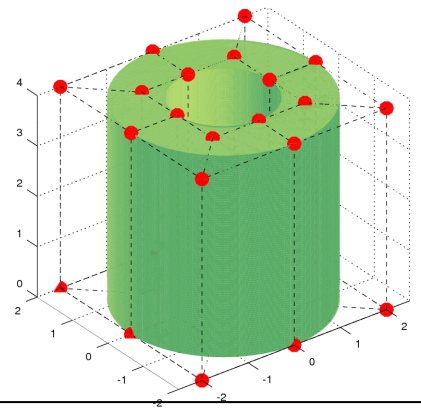
```
crv = nrbline ([1 0], [2 0]);
srf = nrbrevolve (crv, [0 0 0], [0 0 1], 2*pi);
```

```
vol = nrbrevolve (srf, [3 0 0], [0 -1 0], pi/2);
```



`nrbextrude`: create surface or volume by extrusion

```
vol = nrbextrude (srf, [0 0 4]);
```

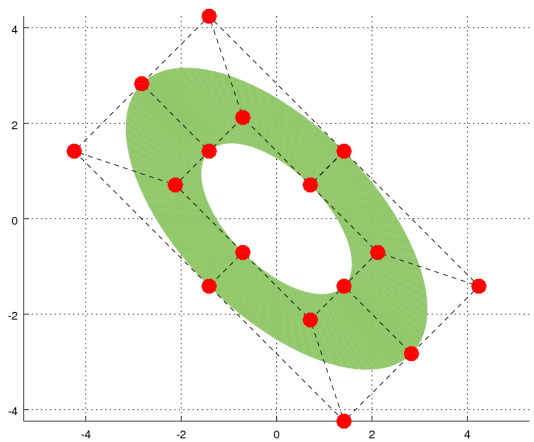


Transformations

Translation, rotation and scaling are applied with a transformation matrix.

`nrbtform`: modify a NURBS structure using a transformation matrix.

The matrix can be constructed using **`vectrans`**, **`vecrot`**, **`vecscale`**.

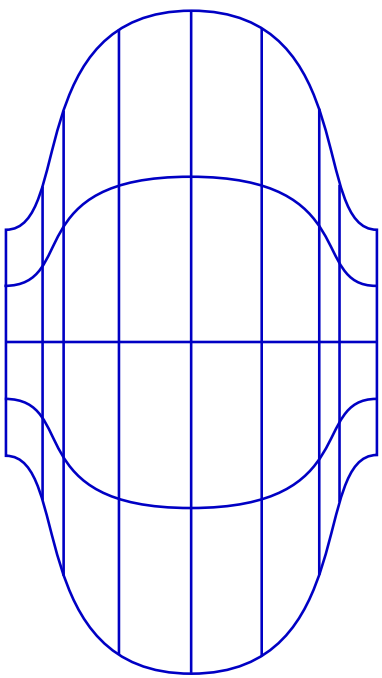


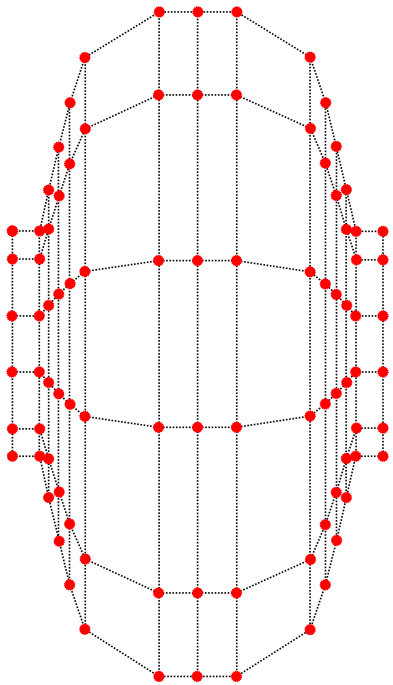

```
crv = nrbline ([1 0], [2 0]);
srf = nrbrevolve (crv, [0 0 0], [0 0 1], 2*pi);

srf = nrbtform (srf, vecscale ([1 2 1]));
srf = nrbtform (srf, vecrot (pi/4, [0 0 1]))
```

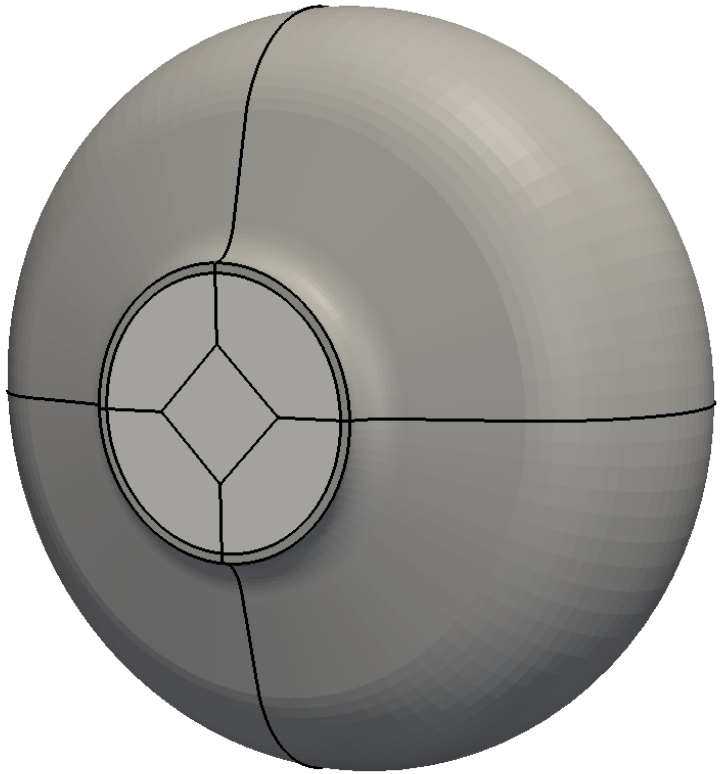
Other Useful Functions

- **`nrbruled`**: construct a ruled surface given two NURBS curves.
- **`nrbcoons`**: construct a Coons surface given four NURBS curves that define the boundary.
- **`nrbextract`**: collect the four boundary curves of a NURBS surface, or collect the six boundary surfaces of a NURBS volume. The output is an array of NURBS structures.
- **`nrbreverse`**: reverse the evaluation direction in a chosen dimension.
- **`nrbtransp`**: transpose a surface, changing the two directions.
- **`nrbpermute`**: generalisation of the previous function to volumes.
- **`nrbexport`**: save NURBS to a text file.
- **`nrb2iges`**: save NURBS to a text file compliant with the IGES format.

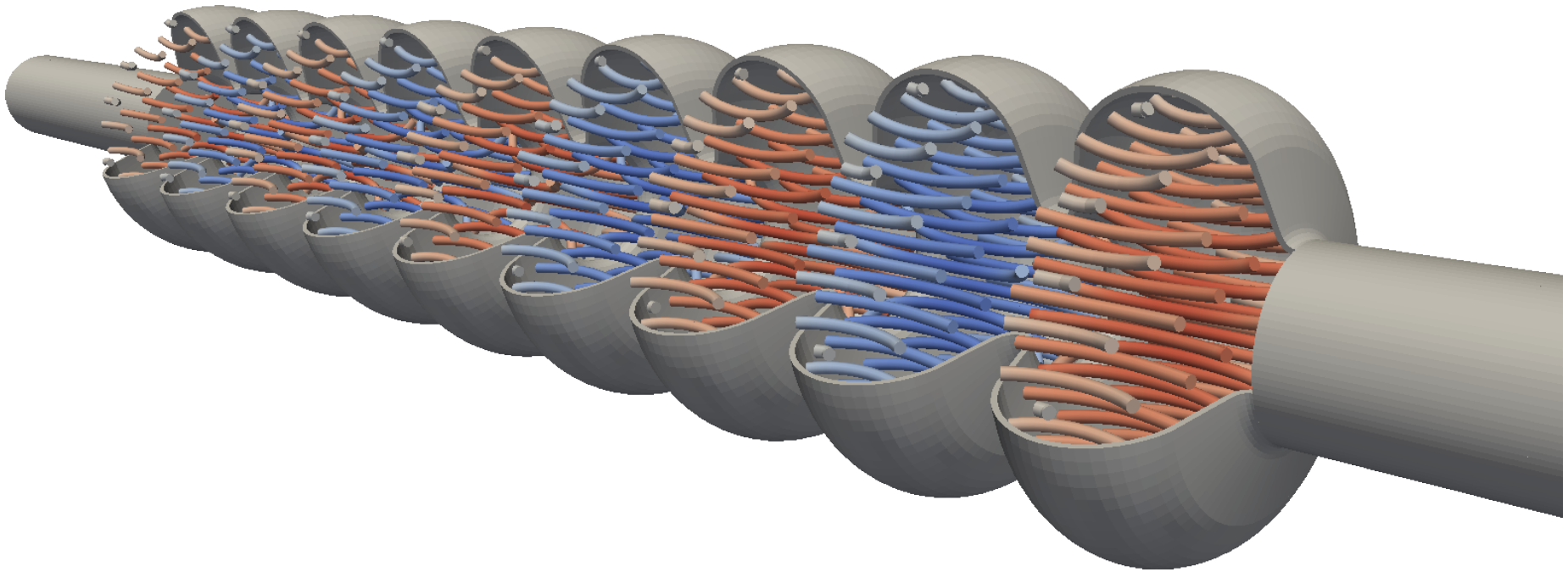
A Complicated Example



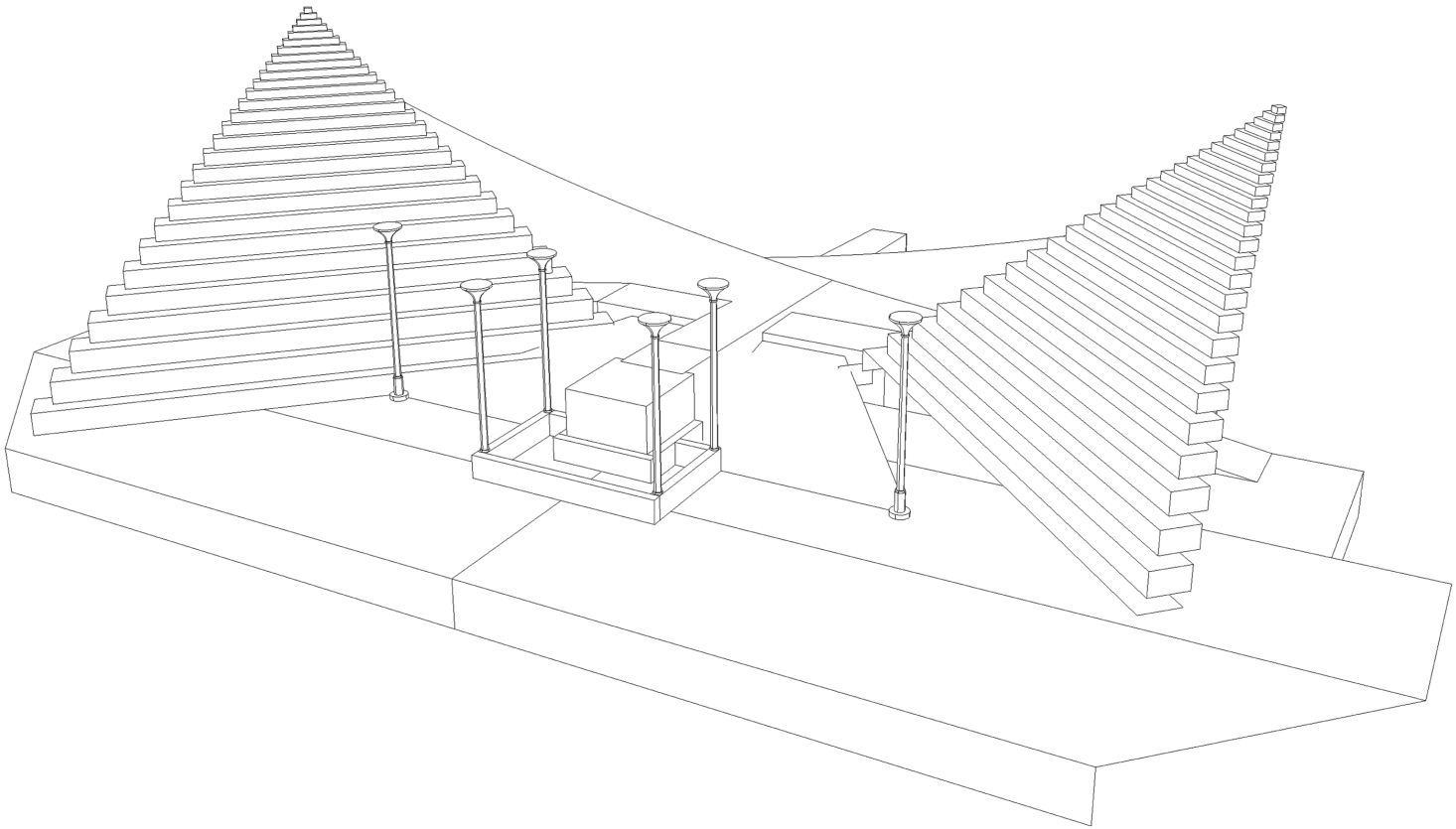




A Complicated Example



A Complicated Example



NURBS Toolbox: Conclusions

- It serves for basis function evaluation and geometry manipulation.
- It supports curves, surfaces and volumes.
- All the functions are well documented. Read the help if needed and check out The NURBS book.
- The NURBS structure can be used directly in GeoPDEs.
- Remember: it works with homogeneous (weighted) coordinates.

A list of all the functions in the toolbox is available online.

`https://octave.sourceforge.io/nurbs/`
`http://rafavzqz.github.io/geopdes/`

L. Piegl and W. Tiller. *The NURBS book*. Springer Science & Business Media, 2012.

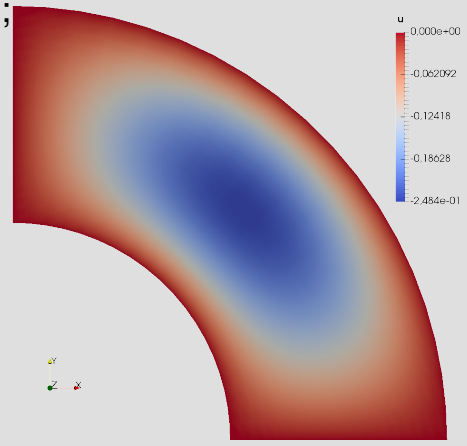
Outline

A Simple Example: $\Delta u = f$ on a Quarter Ring

```
geo = geo_load ('ring_refined.mat');

knots = geo.nurbs.knots;
[qn, qw] = msh_set_quad_nodes (knots, msh_gauss_nodes (geo.nurbs.order));
msh = msh_cartesian (knots, qn, qw, geo);

space  = sp_nurbs (geo.nurbs, msh);

mat = op_gradu_gradv_tp (space, space, msh);
rhs = op_f_v_tp (space, msh,
    @(x, y) (8-9*sqrt(x.^2+y.^2)).*sin(2*atan(y./x))./(x.^2+y.^2));

drchlt_dofs = [];
for iside = 1:4
  drchlt_dofs = union (drchlt_dofs, space.boundary(iside).dofs);
end
int_dofs = setdiff (1:space.ndof, drchlt_dofs);

u = zeros (space.ndof, 1);
u(int_dofs) = mat(int_dofs, int_dofs) \ rhs(int_dofs);

sp_to_vtk (u, space, geo, [20 20], 'laplace_solution.vts', 'u')
err = sp_l2_error (space, msh, u,
    @(x,y)(x.^2+y.^2-3*sqrt(x.^2+y.^2)+2).*sin(2.*atan(y./x)))
```

A Simple Example: $\Delta u = f$ on a Quarter Ring

```
geo = geo_load ('ring_refined.mat');

knots = geo.nurbs.knots;
[qn, qw] = msh_set_quad_nodes (knots, msh_gauss_nodes (geo.nurbs.order));
msh = msh_cartesian (knots, qn, qw, geo);

space  = sp_nurbs (geo.nurbs, msh);

mat = op_gradu_gradv_tp (space, space, msh);
rhs = op_f_v_tp (space, msh,
    @(x, y) (8-9*sqrt(x.^2+y.^2)).*sin(2*atan(y./x))./(x.^2+y.^2));

drchlt_dofs = [];
for iside = 1:4
  drchlt_dofs = union (drchlt_dofs, space.boundary(iside).dofs);
end
int_dofs = setdiff (1:space.ndof, drchlt_dofs);

u = zeros (space.ndof, 1);
u(int_dofs) = mat(int_dofs, int_dofs) \ rhs(int_dofs);

sp_to_vtk (u, space, geo, [20 20], 'laplace_solution.vts', 'u')
err = sp_l2_error (space, msh, u,
    @(x,y)(x.^2+y.^2-3*sqrt(x.^2+y.^2)+2).*sin(2.*atan(y./x)))
```



IGA in an Abstract Framework

- Problem to solve at the continuous level.

Abstract Framework:

$$a(u, v) = (f, v) \qquad \forall v \in V$$

Simple Example: Poisson Equation

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f \cdot v \qquad \forall v \in H_0^1(\Omega)$$

IGA in an Abstract Framework

- Problem to solve at the continuous level.
- Parametric domain and parametrisation of the physical domain.

Abstract Framework:

$$\mathbf{F} : \widehat{\Omega} \to \Omega, \ \mathbf{F} \text{ known and computable}$$

Simple Example: Poisson Equation

$$\widehat{\Omega} = [0, 1]^d \text{ and } \mathbf{F} \text{ is a NURBS}$$

IGA in an Abstract Framework

- Problem to solve at the continuous level.
- Parametric domain and parametrisation of the physical domain.
- Discrete problem and spaces in the parametric and physical domain.

Abstract Framework:

$V_h = \{v_h : \iota(v_h) = \widehat{v}_h \in \widehat{V}_h\}$, where ι is a pull-back depending on $\mathbf{F}$ and $\widehat{V}_h = \text{span}\{\widehat{v}_j\}_{j=1}^{N_h}$ is a finite-dimensional and computable space

Simple Example: Poisson Equation

$$V_h = \{v_h : v_h \circ \mathbf{F} = \widehat{v}_h \in \widehat{V}_h\}, \text{ with } \widehat{V}_h = \text{span}\{R_j\}_{j=1}^{N_h} \text{ a space of NURBS}$$

IGA in an Abstract Framework

- Problem to solve at the continuous level.
- Parametric domain and parametrisation of the physical domain.
- Discrete problem and spaces in the parametric and physical domain.
- Construct and solve a linear system to find the discrete solution.

Abstract Framework:

$$u_h = \sum_{i=1}^{N_h} \alpha_i v_i \qquad \sum_{i=1}^{N_h} \alpha_i a(v_i, v_j) = (f, v_j),\ j = 1, \dots, N_h$$

Simple Example: Poisson Equation

$$u_h = \sum_{i=1}^{N_h} \alpha_i R_i \qquad \sum_{i=1}^{N_h} \alpha_i \int_\Omega \nabla R_i \cdot \nabla R_j = \int_\Omega f R_j,\ j = 1, \dots, N_h$$

IGA in an Abstract Framework

To numerically compute the integrals, we define a partition $\widehat{\Omega} = \cup_{k=1}^{N_e} \widehat{K}_k$, and on each element $\widehat{K}_k$ a quadrature rule $\{(\widehat{\mathbf{x}}_{l,k}, w_{l,k})\}_{l=1}^{n_k}$

$$\int_{\Omega} \phi(\mathbf{x}) \approx \sum_{k=1}^{N_e} \sum_{l=1}^{n_k} w_{l,k} \phi(\mathbf{F}(\widehat{\mathbf{x}}_{l,k})) \left|\det(D\mathbf{F}(\widehat{\mathbf{x}}_{l,k}))\right|$$

IGA in an Abstract Framework

To numerically compute the integrals, we define a partition $\widehat{\Omega} = \cup_{k=1}^{N_e} \widehat{K}_k$, and on each element $\widehat{K}_k$ a quadrature rule $\{(\widehat{\mathbf{x}}_{l,k}, w_{l,k})\}_{l=1}^{n_k}$

$$\int_{\Omega} \phi(\mathbf{x}) \approx \sum_{k=1}^{N_e} \sum_{l=1}^{n_k} w_{l,k} \phi(\mathbf{F}(\widehat{\mathbf{x}}_{l,k})) |\det(D\mathbf{F}(\widehat{\mathbf{x}}_{l,k}))|$$

We compute the stiffness matrix as

$$A_{ij} \approx \sum_{k=1}^{N_e} \sum_{l=1}^{n_k} w_{l,k} \nabla v_j(\mathbf{F}(\widehat{\mathbf{x}}_{l,k})) \cdot \nabla v_i(\mathbf{F}(\widehat{\mathbf{x}}_{l,k})) |\det(D\mathbf{F}(\widehat{\mathbf{x}}_{l,k}))|$$

and recall that $\nabla v(\mathbf{F}(\widehat{\mathbf{x}}_{l,k})) = D\mathbf{F}^{-\top} \widehat{\nabla} \widehat{v}_i(\widehat{\mathbf{x}}_{l,k}))$.

IGA in an Abstract Framework

Summarising, the ingredients we need are:

- A partition of $\widehat{\Omega}$ and a quadrature rule (nodes and weights)
- The evaluation of **F** and the Jacobian $D\mathbf{F}$ at the quadrature points
- Value of the shape functions at the "mapped" quadrature points.
- A routine to put everything together and compute the matrices.

GeoPDEs: Main Structure

GeoPDEs has been implemented following the abstract framework.

The code is based on three main structures:

- Geometry: the parametrization **F** and its derivatives.
- Mesh: the partition of the domain and the quadrature rule.
- Space: the shape functions of the discrete space V_h.

Implementation is based on Matblab/Octave classes.

Only 1D information is stored to reduce memory consumption. Matrix assembly routines exploit tensor product structure to be efficient.

`geometry` Structure

`geo_load`: reads a NURBS geometry and returns a geometry structure.

Computation of the parametrisation **F** and its derivatives.

- **`map`**: function handle to compute **F** at given points in $\widehat{\Omega}$
- **`map_der`**: function handle to compute the Jacobian of **F**
- **`map_der2`**: (optional) function handle to compute the 2^{nd} derivatives of **F**

The NURBS Toolbox is used for the evaluation.

The computation of the geometry is separated from the shape functions.

- Necessary for non-isoparametric discretisations
- Geometry evaluations can be made in the coarsest given geometry

`msh_cartesian` Class

```
knots = geo.nurbs.knots;
[qn, qw] = msh_set_quad_nodes (knots, msh_gauss_nodes (geo.nurbs.order));
msh = msh_cartesian (knots, qn, qw, geo);
```

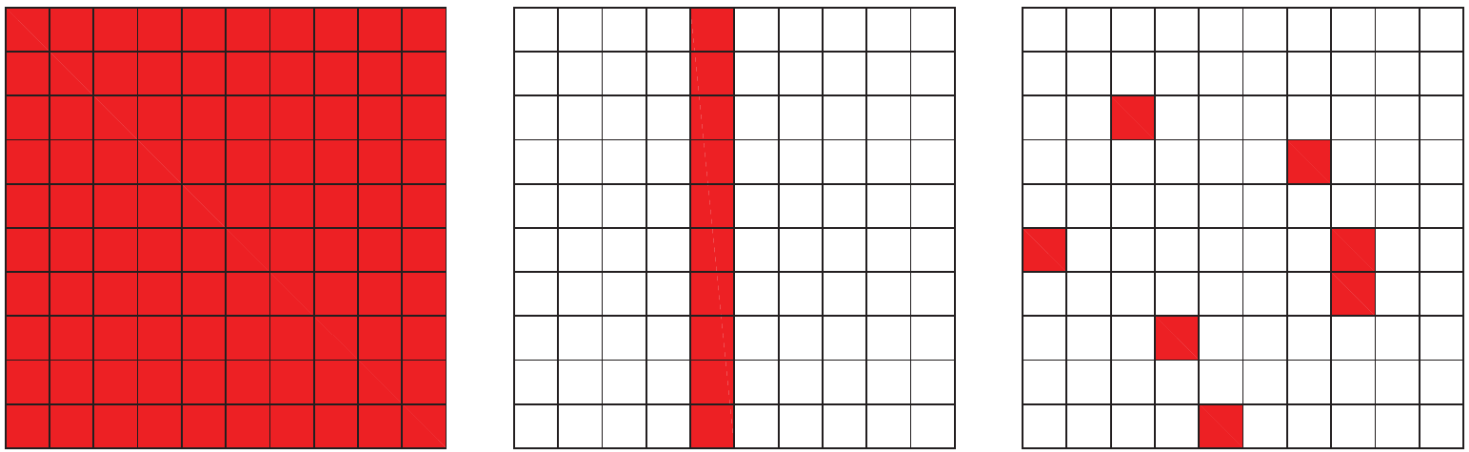
- Collect the knots of the geometry
- Set the quadrature nodes and weights in each parametric direction
- Call the **`msh_cartesian`** constructor

`msh_cartesian` Class Properties

Name	Type	Size	Description
`ndim`	scalar	1×1	Dimension of the parametric domain
`rdim`	scalar	1×1	Dimension of the physical space in which the domain is embedded
`nel`	scalar	1×1	Number of elements of the mesh
`nel_dir`	array	1×ndim	Number of elements in each parametric direction
`nqn`	scalar	1×1	Number of quadrature points per element
`nqn_dir`	array	1×ndim	Number of quadrature points per element, in each parametric direction
`qn`	cell array	1×ndim (nqn_dir×nel_dir)	Coordinates of the quadrature nodes in each parametric direction
`qw`	cell array	1×ndim (nqn_dir×nel_dir)	Quadrature weights in each parametric direction
`breaks`	cell array	1×ndim	Breakpoints that determine the elements, in each parametric direction
`boundary`	**`msh_cartesian`** array	1×(2*ndim)	A mesh object for each boundary side

`msh_cartesian` Methods

- **`msh_precompute`**: compute the quadrature rule, the parametrization **F** and its derivatives, for all quadrature points and all elements in the grid.
- **`msh_evaluate_col`**: compute the same quantities in one "column" of the mesh, i.e., fixing the element number in the first parametric direction.
- **`msh_evaluate_element_list`**: compute the same quantities in a given list of elements.



The output of these methods is a structure (see **`ex07_mesh.m`**).

`sp_scalar` Class

The `nurbs` field in the geometry structure already contains the necessary information to define the space in the parametric domain: the knot vector, the degree and the weights.

The **`sp_scalar`** class provides two constructors: **`sp_nurbs`** and **`sp_bspline`**.

```
1 space  = sp_nurbs (geo.nurbs, msh);
```

NURBS weights are internally handled.

Only univariate information are stored.

`sp_scalar` Class Properties

Name	Type	Size	Description
`space type`	string		Either spline or NURBS
`knots`	cell array	1×ndim	Knot vector in each parametric direction
`degree`	matrix	1×ndim	Degree in each parametric direction
`weights`	NDArray	ndof dir	Weight associated to each basis function, (only NURBS spaces)
`ndof`	scalar	1 × 1	Total number of degrees of freedom
`ndof_dir`	matrix	1×ndim	Number of degrees of freedom in each parametric direction
`nsh_max`	scalar	1 × 1	Maximum number of non-vanishing functions per element
`nsh_dir`	matrix	1×ndim	Maximum number of non-vanishing functions per element in each parametric direction
`sp_univ`	space struct	1×ndim	Univariate spline spaces in each parametric direction, with basis functions evaluated at points given by **`msh.qn`**
`transform`	string		Either *grad-preserving* or *integral-preserving*
`constructor`	function handle		Function handle to generate the same discrete space in a different Cartesian grid
`boundary`	**`sp_scalar`** array	1×(2*ndim)	A space object for each boundary side. Empty for *integral-preserving* transformation
Properties for boundary spaces			
`dofs`	array	1×ndof	Global numbering of the functions on each boundary. Empty for *integral-preserving* transformation.

`sp_scalar` Class Properties

- Due to the weights, the multivariate NURBS space is not the tensor product of univariate NURBS spaces. Thus, we store univariate spline spaces (**`sp_univ`**), plus the coefficients w_i (**`weights`**).

- It is possible to define also a mapping that preserves integrals

$$V_h = \left\{ v_h = \frac{1}{|D\mathbf{F}|} \left(\widehat{v}_h \circ \mathbf{F}^{-1} \right), \ \widehat{v}_h \in \widehat{V}_h \right\}$$

- Analogous methods to the mesh class.

Matrix and Vector Assembly

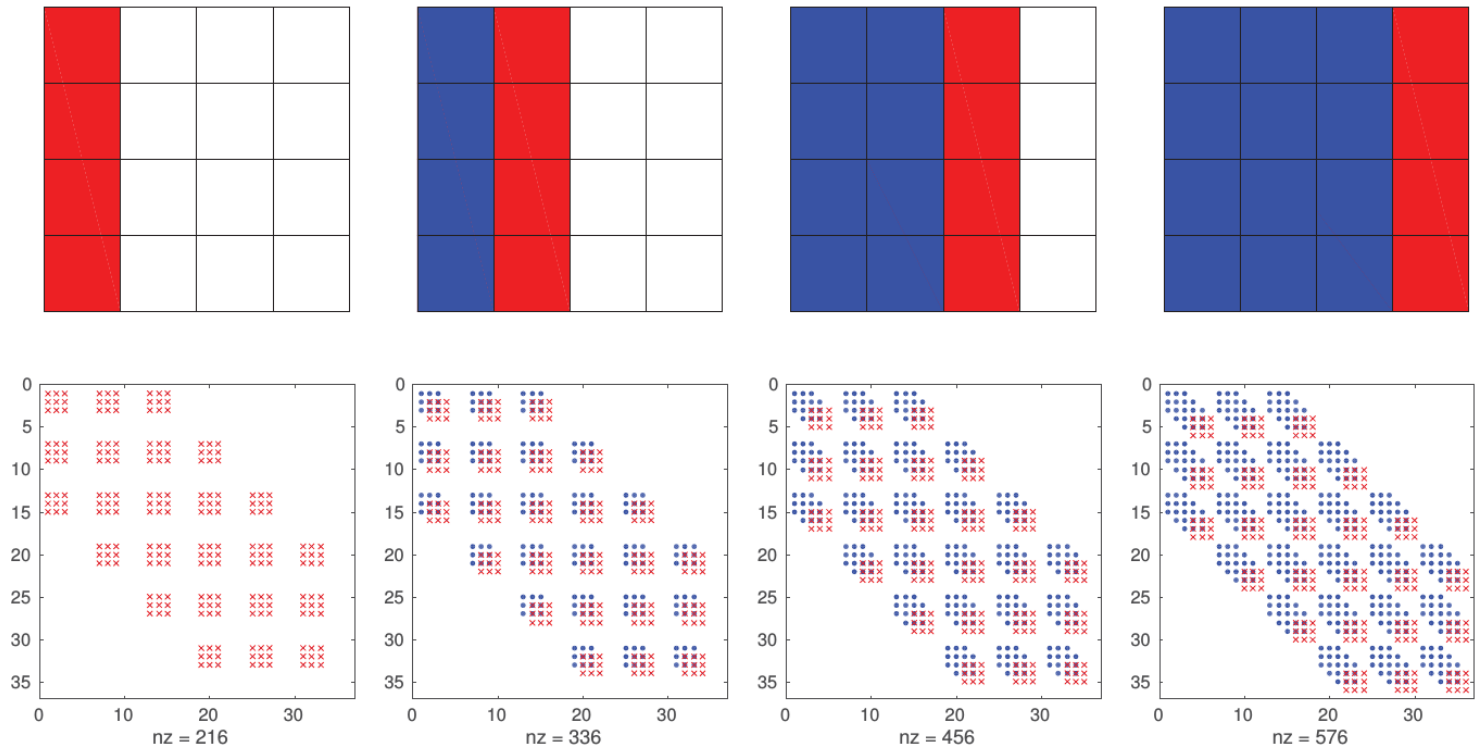
```
1 mat = op_gradu_gradv_tp (space, space, msh);
  rhs = op_f_v_tp (space, msh,
3     @(x, y) (8-9*sqrt(x.^2+y.^2)).*sin(2*atan(y./x))./(x.^2+y.^2));
```

Matrix and Vector Assembly

```
mat = op_gradu_gradv_tp (space, space, msh);
rhs = op_f_v_tp (space, msh,
    @(x, y) (8-9*sqrt(x.^2+y.^2)).*sin(2*atan(y./x))./(x.^2+y.^2));
```

```
function A = op_gradu_gradv_tp (space1, space2, msh, coeff)
A = sparse (space2.ndof, space1.ndof);
for iel = 1:msh.nel_dir(1)
  msh_col = msh_evaluate_col (msh, iel);
  sp1_col = sp_evaluate_col (space1, msh_col, 'value', false, 'gradient', true);
  sp2_col = sp_evaluate_col (space2, msh_col, 'value', false, 'gradient', true);
  for idim = 1:msh.rdim
    x{idim} = reshape (msh_col.geo_map(idim,:,:), msh_col.nqn, msh_col.nel);
  end
  A = A + op_gradu_gradv (sp1_col, sp2_col, msh_col, coeff (x{:}));
end
```

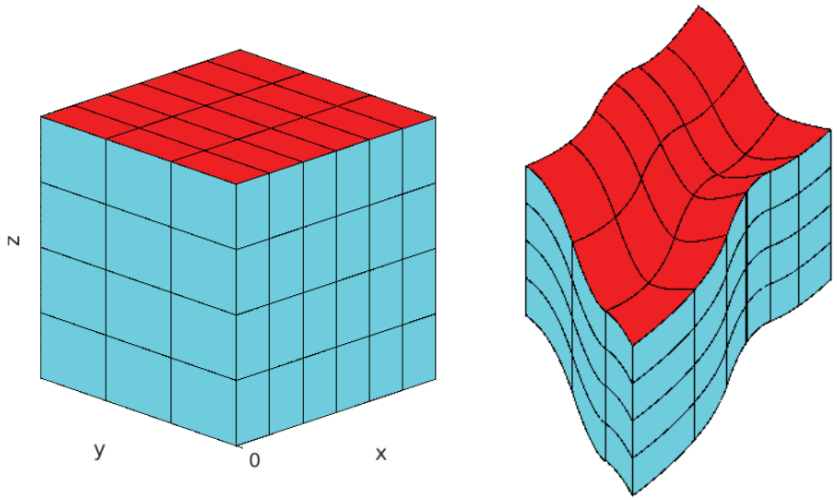
Matrix and Vector Assembly



- Low level assembly in `oct` files (C++) for efficiency
- To improve efficiency it would be possible to:
  - Assemble matrices and rhs at the same time
  - Exploit symmetry (if trial and test space are the same)

Boundary Objects

- $\mathbf{F}_\Gamma : \widehat{\Gamma} \to \Gamma$ restriction of $\mathbf{F}$ to one side
- Restriction of the space on the boundary, i.e. trace, by considering the relevant 1D knot vectors



- **`msh_cartesian.boundary`** and **`sp_scalar.boundary`** are arrays of objects of the same class
- **`sp_scalar.boundary.dofs`**: local-to-global numbering

N.B. Boundary objects contain only boundary info. Quantities requiring volumetric information (e.g. normal derivatives) cannot be computed using only boundary structures.

Imposing Boundary Conditions

Non-Homogeneous Neumann Boundary Condition

$$g_i = \int_{\Gamma_N} g_N R_i$$

We exploit the same operators on the boundary objects:

```
for side = nmnn_side
  dofs = space.boundary(side).dofs;
  rhs(dofs) = rhs(dofs) + op_f_v_tp (space.boundary(side), msh.boundary(side), g);
end
```

Imposing Boundary Conditions

Non-Homogeneous Dirichlet Boundary Condition

Lifting $\tilde{u}_h$ such that $\tilde{u}_h|_{\Gamma_D} = g_D$ (ends up on the rhs).

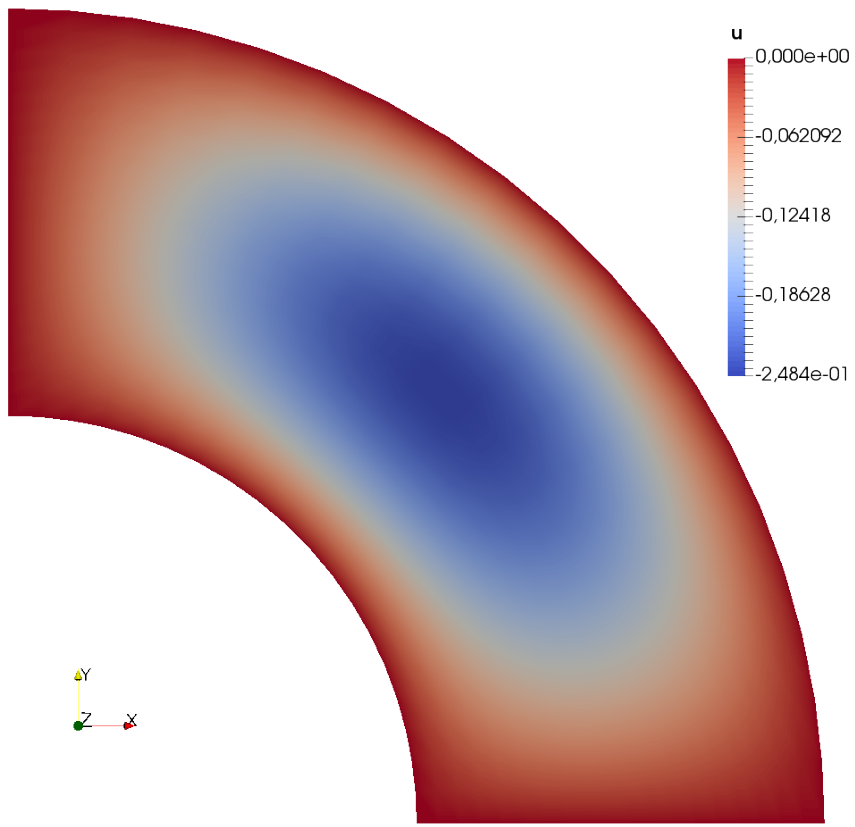
We use L^2 projection to impose the condition weakly, i.e. find $\tilde{u}_h \in V_{h,\Gamma_D}$ s.t.

$$\int_{\Gamma_N} \tilde{u}_h v_h = \int_{\Gamma_N} g_D v_h \qquad \forall v_h \in V_{h,\Gamma_D}$$

```
u = zeros (space.ndof, 1);
drchlt_dofs = [];
for side = drchlt_sides
  side_dofs = space.boundary(side).dofs;
  drchlt_dofs = union (drchlt_dofs, side_dofs);
  M(side_dofs, side_dofs) = M(side_dofs, side_dofs)
    + op_uv_tp (space.boundary(side), space.boundary(side), msh.boundary(side),
     f_one) ;
  rhs_drchlt(side_dofs) = rhs_drchlt(side_dofs)
    + op_fv_tp (space.boundary(side), msh.boundary(side), g);
end
u(drchlt_dofs) = M(drchlt_dofs,drchlt_dofs) \ rhs_drchlt(drchlt_dofs);
int dofs = setdiff (1:space.ndof, drchlt_dofs);
rhs(int_dofs) = rhs(int_dofs) - mat(int_dofs,drchlt_dofs) * u(drchlt_dofs);
```

Solve the Linear System

```
1 u(int_dofs) = mat(int_dofs,int_dofs) \ rhs(int_dofs);
```



Vector Field Spaces

- Define a NURBS space for each component in the reference domain:

$$\left\{ \begin{bmatrix} \widehat{N}_i \\ 0 \end{bmatrix} \right\}_{i=1}^{N} \bigcup \left\{ \begin{bmatrix} 0 \\ \widehat{N}_i \end{bmatrix} \right\}_{i=1}^{N}$$

- Map to the physical space with an appropriate transformation

$$\mathbf{v}_h = \widehat{v}_h \circ \mathbf{F}^{-1} \qquad \text{grad-preserving}$$

$$\mathbf{v}_h = \left(D\mathbf{F}^{+}\right)^{\top} \left(\widehat{\mathbf{v}}_h \circ \mathbf{F}^{-1}\right) \qquad \text{curl-preserving}$$

$$\mathbf{v}_h = \frac{1}{|D\mathbf{F}|} D\mathbf{F} \left(\widehat{\mathbf{v}}_h \circ \mathbf{F}^{-1}\right) \qquad \text{div-preserving}$$

`sp_vector` Class

```
space_scalar = sp_nurbs (geometry.nurbs, msh);
for idim = 1:msh.rdim
  scalar_spaces{idim} = space_scalar;
end
space = sp_vector (scalar_spaces, msh, 'grad-preserving');
```

- Properties and methods analogous to the scalar case
- Boundary spaces obtained by the natural trace operators induced by each transformation:
  - grad-preserving: $\gamma \mathbf{v} = \mathbf{v}$
  - curl-preserving: $\gamma_{\mathbf{t}} \mathbf{v} = \mathbf{n} \times (\mathbf{v} \times \mathbf{n})$. Boundary is a curl-preserving **`sp_vector`** space with only tangential components
  - div-preserving: $\gamma_{\mathbf{n}} \mathbf{v} = \mathbf{v} \cdot \mathbf{n}$. Boundary its a **`sp_scalar`** class with integral preserving transformation

Maxwell Eigenvalue Problem

Find $\mathbf{E}_h \in V_{0,h}$ and $\lambda_h \in \mathbb{R}$ s.t.

$$\int_\Omega \nabla \times \mathbf{E}_h \cdot \nabla \times \mathbf{v}_h = \lambda_h \varepsilon_0 \mu_0 \int_\Omega \mathbf{E}_h \cdot \mathbf{v}_h \qquad \forall \mathbf{v}_h \in V_{0,h}$$

After discretisation we obtain a generalised eigenvalue problem:

$$\mathbf{Ke} = \lambda \mathbf{Me}$$

with **e** the electric field DoFs, **K** and **M** the curl-curl and mass matrices

Maxwell Eigenvalue Problem

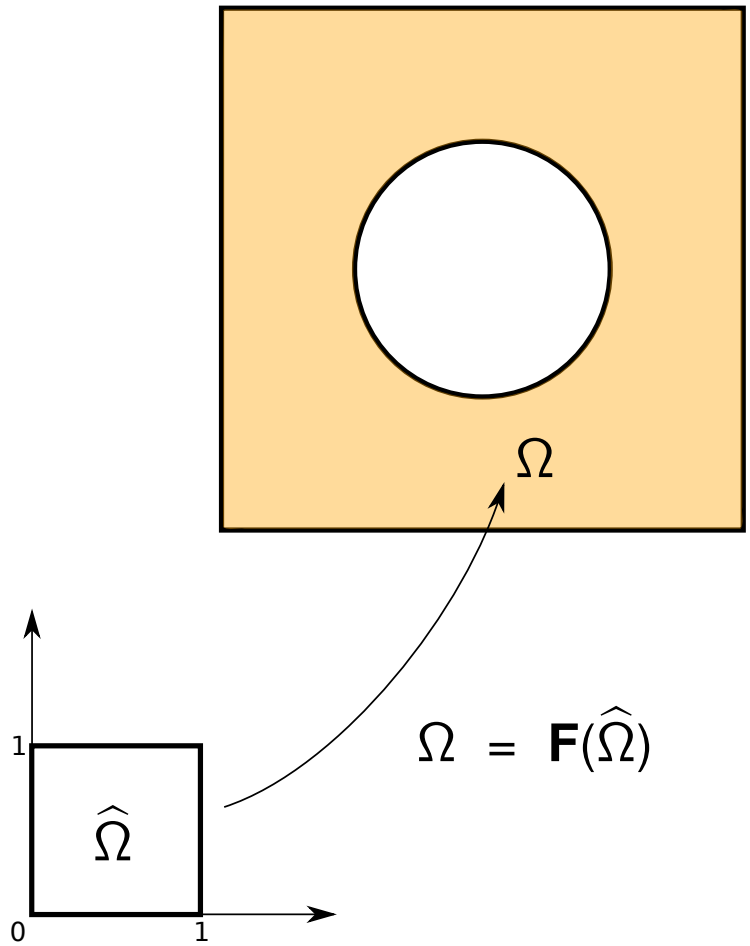
```
mu0  = 1;
eps0 = 1;
degree     = [2 2 2];
regularity = degree - 1;
nsub       = [5 5 5];
nquad      = degree + 1;
% Create a Unit Cube and load the Geometry
cube = nrbextrude (nrb4surf ([0 0], [0 1], [1 0], [1 1]), [0 0 1]);
geo = geo_load (cube);
% Define Hcurl conforming knot vectors
[knots, zeta] = kntrefine (geo.nurbs.knots, nsub, degree, regularity);
[knots_hcurl, degree_hcurl] = knt_derham (knots, degree, 'Hcurl');
% Construct the Mesh
rule = msh_gauss_nodes (nquad);
[qn, qw] = msh_set_quad_nodes (zeta, rule);
msh = msh_cartesian (zeta, qn, qw, geo);
% Construct the Space
scalar_spaces = cell (msh.ndim, 1);
for idim = 1:msh.ndim
  scalar_spaces{idim} = sp_bspline (knots_hcurl{idim}, degree_hcurl{idim}, msh);
end
space = sp_vector (scalar_spaces, msh, 'curl-preserving');
```

Maxwell Eigenvalue Problem

```
% Assemble the matrices
K = op_curlu_curlv_tp (space, space, msh);
M = op_u_v_tp (space, space, msh, @(x,y) mu0*eps0*ones (size(x)));

% Apply PEC Boundary Conditions
drchlt_dofs = [];
for iside = 1:numel (space.boundary)
  drchlt_dofs = union (drchlt_dofs, space.boundary(iside).dofs);
end
int_dofs = setdiff (1:space.ndof, drchlt_dofs);

% Solve the Eigenvalue Problem
eigf = zeros (space.ndof, numel(int_dofs));
[eigf(int_dofs, :), lambda] = eig (full (K(int_dofs, int_dofs)), full (M(int_dofs,
     int_dofs)));
lambda = diag (lambda);
nzeros = numel (find (lambda < 1e-10));

fprintf('First Computed Eigenvalues:\n')
disp (lambda(nzeros+1:nzeros+17))
```

Multipatch Setting

Not all geometries can be represented as a regular transformation of $[0, 1]^d$

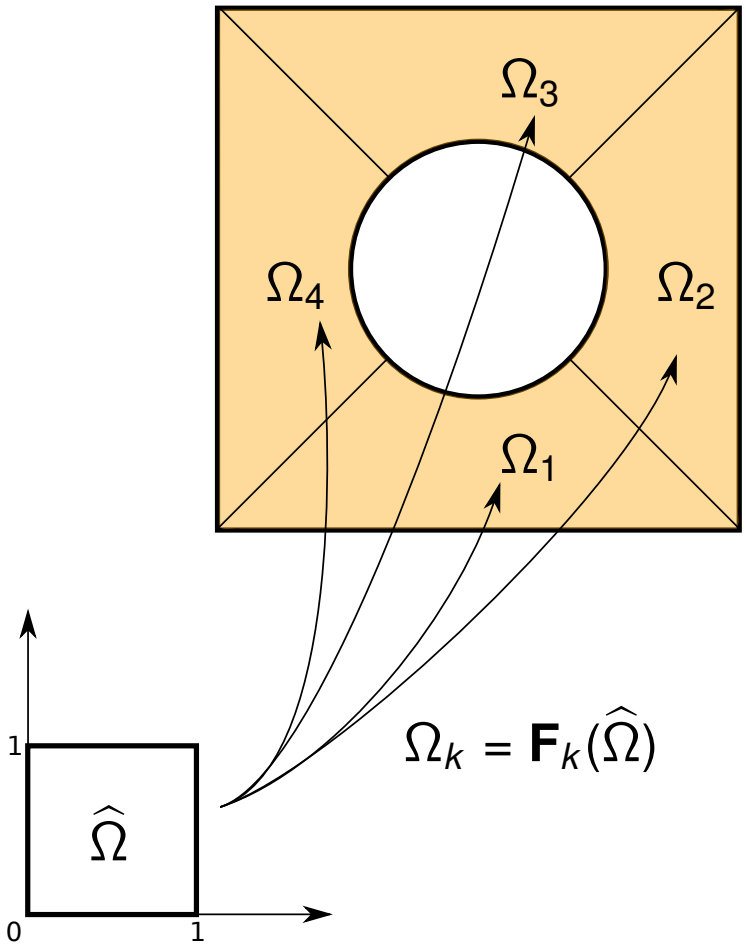


Multipatch Setting

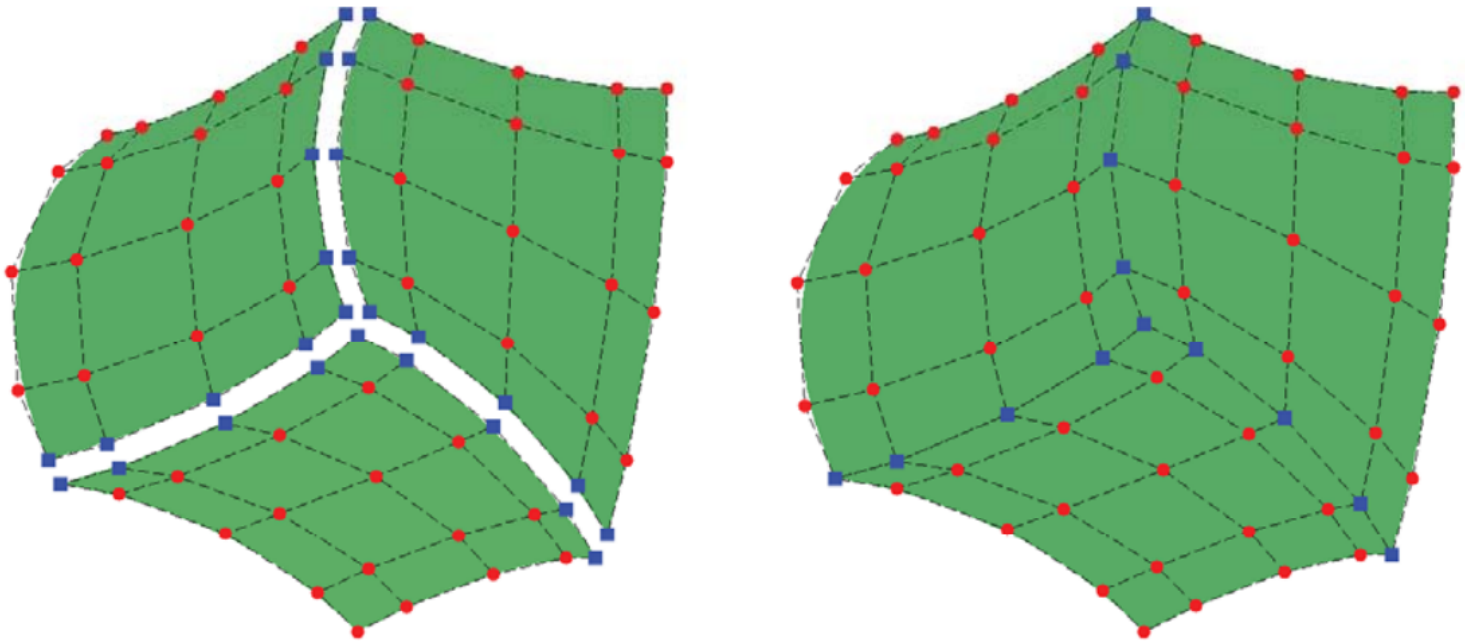
Not all geometries can be represented as a regular transformation of $[0,1]^d$

Multipatch:

- Subdivide the domain in k hexahedral patches
- Each $\Omega_k = \mathbf{F}_k(\widehat{\Omega})$
- Matching discretisation at the interfaces
- One-to-one matching of DoFs

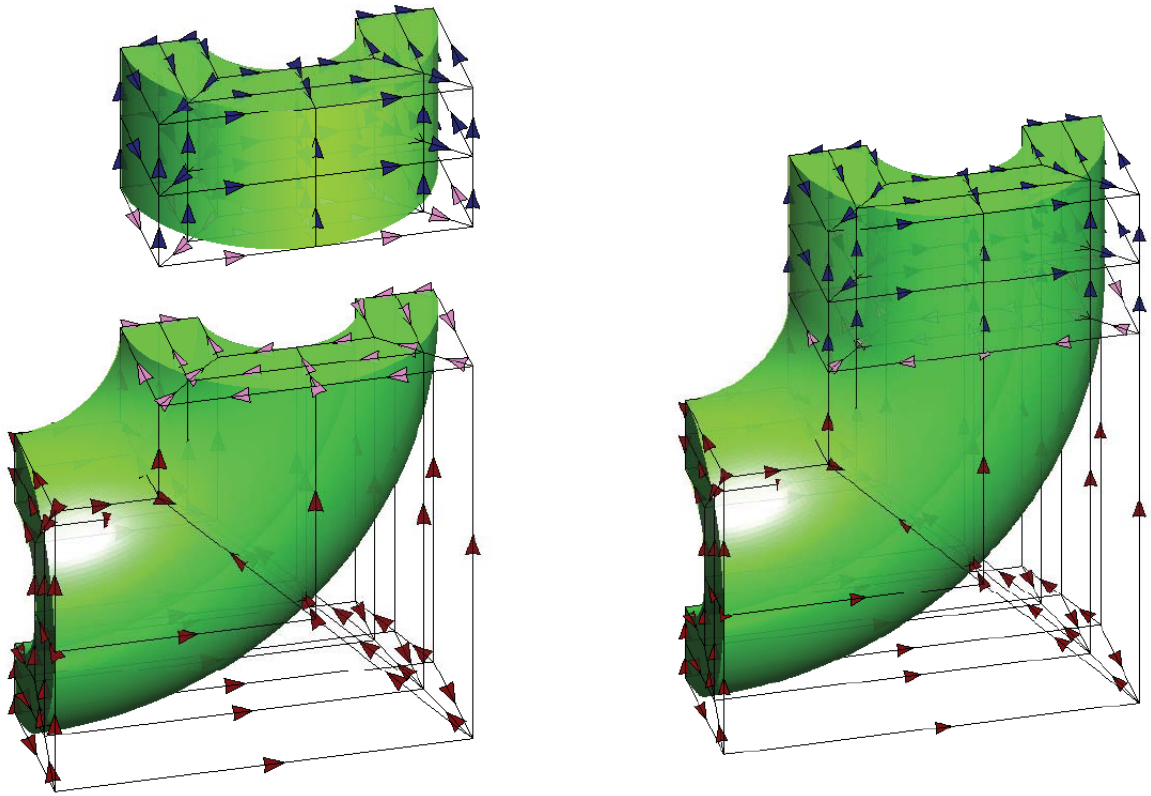


Multipatch Setting - Scalar Spaces



- Conforming meshes across patches
- One-to-one correspondance of boundary functions
- Enforce continuity $V_h = \{v \in C^0 : v|_{\Omega_i} \in V_h^{(i)},\ i = 0, \dots, N_{\text{ptc}}\}$ to obtain a discrete subspace of $H^1(\Omega)$
- Matching is enforced using a local-to-global indexing

Multipatch Setting - Vector Spaces



- Glue only tangential components for $H(\mathrm{curl}, \Omega)$
- Glue only normal components for $H(\mathrm{div}, \Omega)$
- Orientation needs to be taken into account

Multipatch in GeoPDEs

- **`nrbmultipatch`**: given an array of NURBS entities, computes the information about interfaces and boundaries

```
[interfaces, boundary] = nrbmultipatch (nurbs);
```

- **`msh_multipatch`**: analogous to the single patch case. It stores
  - number of patches
  - total number of elements
  - meshes associated to each patch (**`msh_cartesian`** objects)
- **`sp_multipatch`**: analogous to the single patch case. It stores
  - total number of basis functions
  - space objects for each patch
  - **`gnum`** cell array, relates the local numbering to the global one
  - **`dofs_ornt`** contains the orientation information

```
msh   = msh_multipatch (local_meshes, boundaries);
space = sp_multipatch (local_spaces, msh, interfaces, boundary_interfaces);
```

- Boundary of multipatch entities are multipatch entities

Available Operators

- Methods present in both **sp_scalar** and **sp_vector**:
  op_f_v_tp , **op_gradu_gradv_tp**, **op_u_v_tp**
- Methods only present in the class **sp_scalar**:
  op_gradgradu_gradgradv_tp, **op_laplaceu_laplacev_tp**, **op_mat_stab_SUPG_tp**, **op_rhs_stab_SUPG_tp**, **op_vel_dot_gradu_v_tp**
- Methods only present in the class **sp_vector**:
  op_curlu_curlv_tp , **op_curlu_v_tp** , **op_div_v_q_tp**, **op_su_ev_tp**, **op_v_gradp_tp**
- Methods present in the class **sp_multipatch**:
  op_f_v_mp, **op_gradu_gradv_mp**, **op_u_v_mp**, **op_curlu_curlv_mp**, **op_div_v_q_mp**, **op_su_ev_mp**, **op_v_gradp_mp**

Conclusions

GeoPDEs is an open source and free Matlab implementation of IGA.

- Particularly useful for teaching purposes and for new researchers
- Can serve as a rapid prototyping tool to test new ideas
- Different problems are already implemented
- Many examples and a technical report with detailed explanations

As of v.3.1.0 Hierarchical refinement is available!

Software download and information: http://rafavzqz.github.io/geopdes/

Mailing list: geopdes-users@lists.sourceforge.net

Useful References


A. Buffa, J. Rivas, G. Sangalli, and R. Vázquez, *Isogeometric discrete differential forms in three dimensions*, SIAM Journal on Numerical Analysis **49** (2011), no. 2, 818–844.


J.A. Cottrell, T.J.R. Hughes, and Y. Bazilevs, *Isogeometric analysis: Toward integration of CAD and FEA*, Wiley, 2009.


C. de Falco, A. Reali, and R. Vázquez, *GeoPDEs: A research tool for isogeometric analysis of PDEs*, Advances in Engineering Software **42** (2011), 1020—-1034.


L. Piegl and W. Tiller, *The NURBS book*, 2 ed., Springer, 1997.


R. Vázquez, *A new design for the implementation of isogeometric analysis in octave and matlab: Geopdes 3.0*, Computers & Mathematics with Applications **72** (2016), no. 3, 523–554.