WS: Isogeometric Analysis
Module 2: IGA in Electromagnetics

Benjamin Marussig, Mario Mally, Michael Wiesheu and Sebastian Schöps

Overview

Overview

Introduction

IGA with GeoPDEs

Isogeometric Analysis:

- Pros (Yesterday) and Cons/Challenges (Afternoon)

GeoPDEs:

- Dimension independent implementation: the same code is valid for curves, surfaces and volumes (tensor product structure)
- Div- and curl-conforming spline spaces
- Examples for **Poisson**, linear elasticity, advection-diffusion, bilaplacian, Stokes, Navier-Stokes, **Maxwell equations** and Kirchhoff-Love shells
- Implemented in `Matlab/Octave`

R. Vázquez, "A new design for the implementation of isogeometric analysis in Octave and Matlab: GeoPDEs 3.0", Comput. Math. Appl., vol. 72, pp. 523-554, 2016.

Introduction

Your Tutors

- Got into contact with IGA and GeoPDEs during or shortly before Masters thesis
- Focus on modelling of electrical machines with IGA (Mortaring, shape optimization, 2D/3D computations)
- Experienced with integration over boundary parts, torque computation, numerical quadrature and the scalar/vector potential equations in the context of GeoPDEs

Overview

CREATOR
COMPUTATIONAL ELECTRIC MACHINE LABORATORY
CEM

Useful NURBS functions

Function	Action
`nrb4surf`	Create a quadrangle by four points
`nrbruled`	Construct a ruled surface or volume between two NURBS
`nrbcoons`	Construct surface by given boundary NURBS
`nrbtform`	Transform a NURBS by a given transformation matrix (e.g. given by `vecrotz`)
`nrbextract`	Create a NURBS by extracting a boundary curve (or surface) from a surface (or volume)
`nrbinverse`	Find the parametric coordinates of a physical point
...	More functions under `https://octave.sourceforge.io/nurbs/overview.html`

Programming Part – Exercise 1

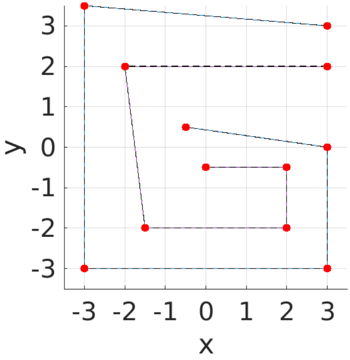


Figure: Initial splines

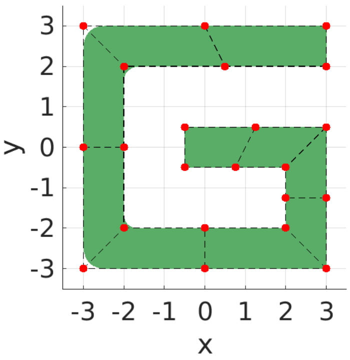


Figure: Model target

Programming Part – Solution 1a

```
Spline1.coefs(2,2) = Spline1.coefs(2,2) - 0.5;
Spline1.coefs(2,5) = Spline1.coefs(2,5) + 0.5;
Spline2.coefs(1,3) = Spline2.coefs(1,3) - 0.5;
Spline2.coefs(1,6) = Spline2.coefs(1,6) - 0.5;
```

Programming Part – Solution 1b

```
Spline1 = nrbdegelev(Spline1, 1);
Spline2 = nrbdegelev(Spline2, 1);
```

Programming Part – Solution 1c

```
Spline1.knots(4) = Spline1.knots(4) - 0.02;
Spline1.knots(5) = Spline1.knots(5) + 0.02;
Spline1.knots(6) = Spline1.knots(6) - 0.02;
Spline1.knots(7) = Spline1.knots(7) + 0.02;

Spline2.knots(4) = Spline2.knots(4) - 0.02;
Spline2.knots(5) = Spline2.knots(5) + 0.02;
Spline2.knots(6) = Spline2.knots(6) - 0.02;
Spline2.knots(7) = Spline2.knots(7) + 0.02;
```

Programming Part – Solution 1d

```
nrbexport(Surface, 'G.txt')
```

Overview

Electrostatics and IGA

Underlying Problem for Capacitor

Electrostatics:

$$\begin{cases} -\nabla \cdot \left[\epsilon \nabla u\right] = f & \text{in} \quad \Omega, \\ u = g_{\mathrm{D}} & \text{on} \quad \Gamma_{\mathrm{D}}, \\ \epsilon \nabla u \cdot \vec{n} = g_{\mathrm{N}} & \text{on} \quad \Gamma_{\mathrm{N}} \end{cases}$$

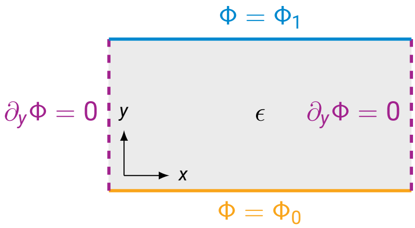
Electrostatics and IGA

Underlying Problem for Capacitor

Electrostatics:

$$
\begin{cases}
-\nabla \cdot \left[\epsilon \nabla u\right] = f & \text{in} \quad \Omega, \\
u = g_{\mathrm{D}} & \text{on} \quad \Gamma_{\mathrm{D}}, \\
\epsilon \nabla u \cdot \vec{n} = g_{\mathrm{N}} & \text{on} \quad \Gamma_{\mathrm{N}}
\end{cases}
$$

E.g. Ideal Capacitor:

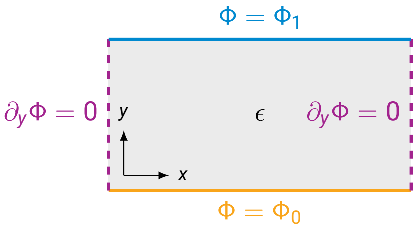


Electrostatics and IGA
Underlying Problem for Capacitor

Electrostatics:

$$\begin{cases} -\nabla\cdot\left[\epsilon\nabla u\right] = f & \text{in} \quad \Omega, \\ u = g_{\mathrm{D}} & \text{on} \quad \Gamma_{\mathrm{D}}, \\ \epsilon\nabla u\cdot\vec{n} = g_{\mathrm{N}} & \text{on} \quad \Gamma_{\mathrm{N}} \end{cases}$$

E.g. Ideal Capacitor:



- Weak formulation:

$$\int_\Omega \epsilon(\nabla u)\cdot(\nabla v)\,\mathrm{d}\vec{x} = \int_\Omega fv\,\mathrm{d}\vec{x} + \int_{\Gamma_{\mathrm{N}}} g_{\mathrm{N}}v\,\mathrm{ds}(\vec{x})$$

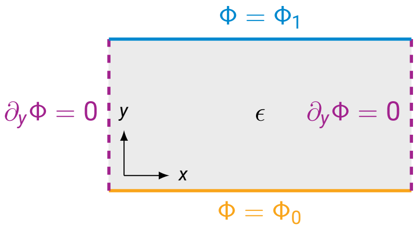
Electrostatics and IGA

Underlying Problem for Capacitor

Electrostatics:

$$\begin{cases} -\nabla \cdot \left[\epsilon \nabla u\right] = f & \text{in} \quad \Omega, \\ u = g_{\mathrm{D}} & \text{on} \quad \Gamma_{\mathrm{D}}, \\ \epsilon \nabla u \cdot \vec{n} = g_{\mathrm{N}} & \text{on} \quad \Gamma_{\mathrm{N}} \end{cases}$$

E.g. Ideal Capacitor:



- Weak formulation:

$$\int_\Omega \epsilon (\nabla u) \cdot (\nabla v) \, \mathrm{d}\vec{x} = \int_\Omega f v \, \mathrm{d}\vec{x} + \int_{\Gamma_{\mathrm{N}}} g_{\mathrm{N}} v \, \mathrm{ds}(\vec{x})$$

- Discretization with basis functions

$$\mathbf{Au} = \ell, \quad u_h(\vec{x}) = \sum_{i=1}^{n} u_i \phi_i(\vec{x})$$

CREATOR COMPUTATIONAL ELECTRIC MACHINE LABORATORY CEM

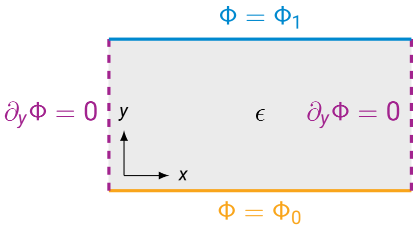
Electrostatics and IGA

Underlying Problem for Capacitor

Electrostatics:

$$\begin{cases} -\nabla \cdot \left[\epsilon \nabla u\right] = f & \text{in} \quad \Omega, \\ u = g_{\mathrm{D}} & \text{on} \quad \Gamma_{\mathrm{D}}, \\ \epsilon \nabla u \cdot \vec{n} = g_{\mathrm{N}} & \text{on} \quad \Gamma_{\mathrm{N}} \end{cases}$$

E.g. Ideal Capacitor:



- Weak formulation:

$$\int_{\Omega} \epsilon (\nabla u) \cdot (\nabla v) \, \mathrm{d}\vec{x} = \int_{\Omega} f v \, \mathrm{d}\vec{x} + \int_{\Gamma_{\mathrm{N}}} g_{\mathrm{N}} v \, \mathrm{ds}(\vec{x})$$

- Discretization with basis functions

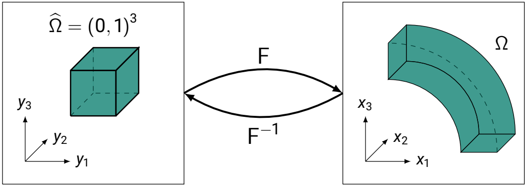
$$\mathbf{A}\mathbf{u} = \ell, \quad u_h(\vec{x}) = \sum_{i=1}^{n} u_i \phi_i(\vec{x})$$

- Homogenization for Dirichlet BC

$$\mathbf{A}_{\mathrm{II}} \mathbf{u}_{\mathrm{I}} = \mathbf{f}_{\mathrm{I}} - \mathbf{A}_{\mathrm{ID}} \mathbf{u}_{\mathrm{D}}, \quad \mathbf{u}_{\mathrm{D}} = \mathbf{M}^{-1} \mathbf{g}_{\mathrm{D}}$$

Electrostatics and IGA

Discretization with IGA



- Reference domain $\widehat{\Omega}$
- Mapping $F : \widehat{\Omega} \to \Omega$
- Often based on B-Splines/NURBS $\widehat{\phi}_i$

- Parameterization of curve, surfaces, volumes is possible
- Tensor product structure for surfaces ($d = 2$) and volumes ($d = 3$)

$$\widehat{\phi}_i(\vec{y}) = \widehat{\phi}_{i_1,\dots,i_d}(\vec{y}) = \widehat{\phi}_{i_1}(y_1) \cdots \widehat{\phi}_{i_d}(y_d)$$

- For IGA we define the basis via pull back (similar to FEM)

$$\phi_i(\vec{x}) = \widehat{\phi}_i\left(F^{-1}(\vec{x})\right)$$

Electrostatics and IGA

Quadrature

- Integration is required for discretization, e.g.

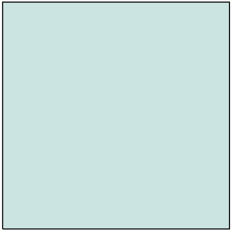
$$A_{ij} = \int_{\Omega} \epsilon(\nabla \phi_i) \cdot (\nabla \phi_j) \, \mathrm{d}\vec{x}$$

Electrostatics and IGA
Quadrature

- Integration is required for discretization, e.g.

$$A_{ij} = \int_{\Omega} \epsilon(\nabla\phi_i) \cdot (\nabla\phi_j)\,\mathrm{d}\vec{x}$$

1. Compute on reference domain
   - Rewrite integral with parameterization



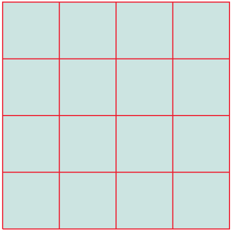
$\widehat{\Omega}$

Electrostatics and IGA
Quadrature

- Integration is required for discretization, e.g.

$$A_{ij} = \int_{\Omega} \epsilon(\nabla \phi_i) \cdot (\nabla \phi_j)\, \mathrm{d}\vec{x}$$

1. Compute on reference domain
   - Rewrite integral with parameterization
2. Mesh for efficient integration
   - Constructed from knot vector



$\widehat{\Omega}$

CREATOR COMPUTATIONAL ELECTRIC MACHINE LABORATORY CEM

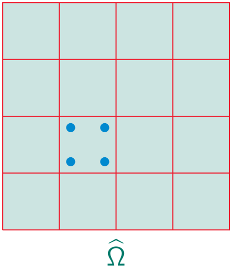
Electrostatics and IGA

Quadrature

- Integration is required for discretization, e.g.

$$A_{ij} = \int_{\Omega} \epsilon(\nabla\phi_i)\cdot(\nabla\phi_j)\,\mathrm{d}\vec{x}$$

1. Compute on reference domain
   - Rewrite integral with parameterization
2. Mesh for efficient integration
   - Constructed from knot vector
3. Quadrature points for elements
   - Exploit tensor product structure



Electrostatics and IGA

Structure of GeoPDE

1. `geometry`: Information on F and its derivatives

```
% Set up geometry structure
geometry  = geo_load(nrb);
```

2. `msh_cartesian`: Information on mesh and quadrature points

```
[knots, zeta] = kntrefine(geometry.nurbs.knots, nsub-1, degree, regularity);
% Set up mesh
rule     = msh_gauss_nodes (nquad);
[qn, qw] = msh_set_quad_nodes (zeta, rule);
msh      = msh_cartesian (zeta, qn, qw, geometry);
```

3. `sp_scalar`: Information on basis functions $\widehat{\phi}_i(\vec{y})$

```
% Set up space
space    = sp_bspline (knots, degree, msh);
```

Electrostatics and IGA

Tensor Product Operators

- Variety of different operators for different problems (with potentially different spaces and meshes)
- We focus just on

$$\int_\Omega \epsilon(\nabla u)\cdot(\nabla v)\,\mathrm{d}\vec{x} = \int_\Omega f v\,\mathrm{d}\vec{x} + \int_{\Gamma_\mathrm{N}} g_\mathrm{N} v\,\mathrm{d}s(\vec{x})$$

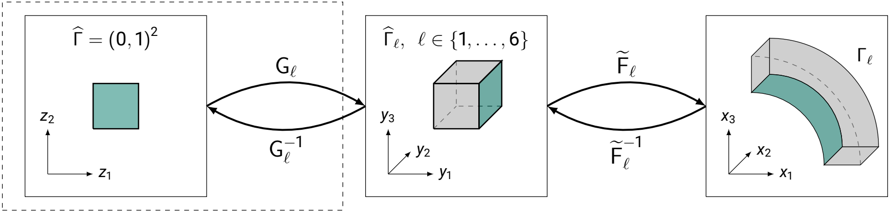
- `tp` version automatically does precomputations

```
% Stiffness matrix
mat = op_gradu_gradv_tp(space1, space2, msh, coeff);
% Compute RHS
rhs = op_f_v_tp (space, msh, coeff)
```


CREATOR COMPUTATIONAL ELECTRIC MACHINE LABORATORY CEM

Electrostatics and IGA

Boundary Conditions



- GeoPDEs automatically constructs boundary objects (msh, space)
- Relation is simple due to tensor product structure and open knot vectors
- Enables treatment of Neumann and Dirichlet boundary conditions

$$\boxed{\int_{\Gamma_\mathrm{N}} g_\mathrm{N} v_h \,\mathrm{d}s(\vec{x})} \qquad \boxed{\int_{\Gamma_\mathrm{D}} u_h v_h \,\mathrm{d}s(\vec{x}) = \int_{\Gamma_\mathrm{D}} g_\mathrm{D} v_h \,\mathrm{d}s(\vec{x})}$$

Electrostatics and IGA

Dirichlet Boundary Conditions

- Points on boundary are not necessarily given or easily computable
- We can compute Dirichlet contribution $\mathbf{u}_{\mathrm{D}}$ weakly with

$$\int_{\Gamma_{\mathrm{D}}} u_h v_h \,\mathrm{d}s(\vec{x}) = \int_{\Gamma_{\mathrm{D}}} g_{\mathrm{D}} v_h \,\mathrm{d}s(\vec{x}), \quad \forall v_h \quad \rightarrow$$

- We obtain L^2-projection $\mathbf{u}_{\mathrm{D}} = \mathbf{M}^{-1}\mathbf{g}_{\mathrm{D}}$
- Provided in GeoPDEs by

```
% L^2 projection
[u_D, dofs_D] = sp_drchlt_l2_proj(space, msh, funD, sidesD);
```

$$\boxed{\mathbf{A}_{\mathrm{II}}\mathbf{u}_{\mathrm{I}} = \mathbf{f}_{\mathrm{I}} - \mathbf{A}_{\mathrm{ID}}\mathbf{u}_{\mathrm{D}}, \;\; \mathbf{u}_{\mathrm{D}} = \mathbf{M}^{-1}\mathbf{g}_{\mathrm{D}}}$$

Programming Part – Exercise 2

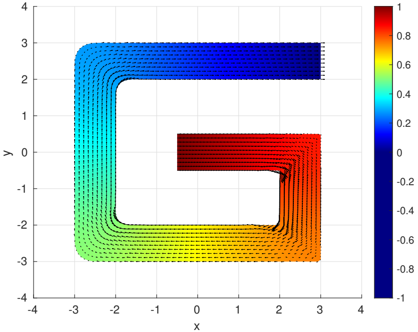


Figure: Toy Problem 1.

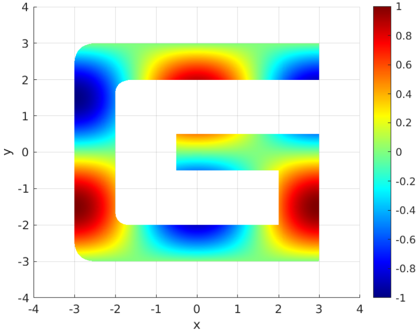


Figure: Toy Problem 2.

Programming Part – Solution 2a

```
stiff_mat = op_gradu_gradv_tp(space, space, msh, matFun);
rhs = op_f_v_tp(space, msh, problem_data.source);
```


CREATOR
COMPUTATIONAL ELECTRIC MACHINE LABORATORY
CEM

Programming Part – Solution 2b

$$\boxed{\mathbf{A}_{\mathrm{II}}\mathbf{u}_{\mathrm{I}} = \mathbf{f}_{\mathrm{I}} - \mathbf{A}_{\mathrm{ID}}\mathbf{u}_{\mathrm{D}} \text{ with } \mathbf{u}_{\mathrm{D}} = \mathbf{M}^{-1}\mathbf{g}_{\mathrm{D}}}$$

```
u = zeros(space.ndof, 1);
[u_D, D] = sp_drchlt_l2_proj(space, msh, funD, D_sides);
I = setdiff(1:space.ndof, D);
rhs(I) = rhs(I) - stiff_mat(I, D)*u_D;
u(I) = stiff_mat(I, I) \ rhs(I);
u(D) = u_D;
```

Programming Part – Solution 2c

```
pts = {linspace(0,1,200), linspace(0,1,10)};
[valu,Fv] = sp_eval(u,space,msh,pts,'value');
[gradu,Fg] = sp_eval(u,space,msh,pts,'gradient');
```

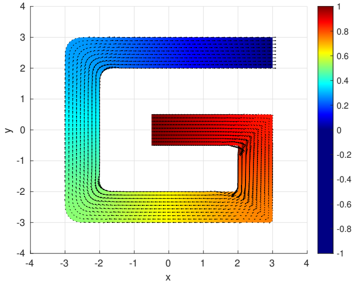


Figure: Toy Problem 1.

CREATOR
COMPUTATIONAL ELECTRIC MACHINE LABORATORY
CEM

Programming Part – Solution 2d

```
[errH1(i,j),errL2(i,j),~,~,~,~] = sp_h1_error(space,msh,u,uex,graduex);
```

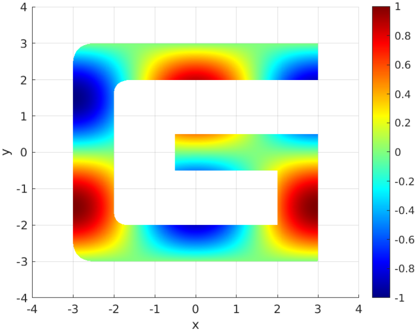


Figure: Toy Problem 2.

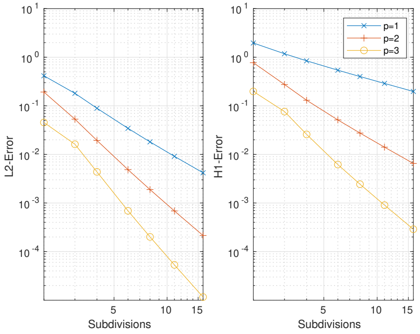


Figure: Error over Subdivisions.

Overview

Programming Part – Exercise 3

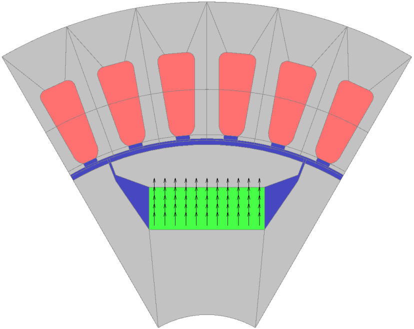


Figure: Motor geometry

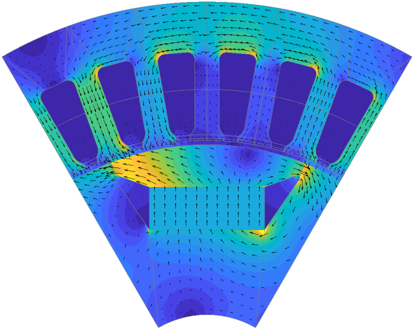


Figure: Magnetic flux density

Programming Part – Solution 3a

```
Motor1.plotGeometry();
Motor1.plotPatchNr();
```

Programming Part – Solution 3b

```
ApplicationCurrent = 10.6;
Angle = 10;

Motor1.setRotationAngle(Angle);
Motor1.setCurrent(ApplicationCurrent, Angle);
Motor1.solveMagneticPotential()
Motor1.plotBResulting();
```

Torque computation

- Various different methods to compute the torque!
- Maxwell stress tensor:
  - Integration of magnetic flux density in the air gap
  - $T = \frac{Lr}{\mu_0} \int_\Gamma B_r B_t \mathrm{d}\Gamma$
  - Unstable in conventional FEM $\rightarrow$ Volume average (Arkkio's method)
- Lagrange multipliers from harmonic mortaring[1]
  - Derived from energy balance
  - Coupling coefficients λ
  - $T \; = L\lambda^T(\alpha)\mathbf{B}'(\alpha)\mathbf{a}_\mathrm{R}(\alpha)$
  - Just a matrix multiplication in the discrete setting!

[1] H. Egger et al. "On torque computation in electric machine simulation by harmonic mortar methods".

Programming Part – Solution 3c

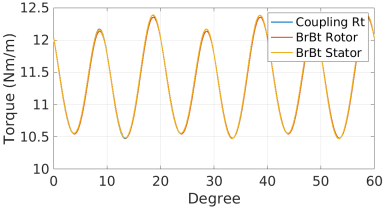
```
Torques = [];
SymMult = 6;
angles = 0:0.25:60;
for angle = angles
    disp(num2str(angle));
    Motor1.setRotationAngle(angle);
    Motor1.setCurrent(ApplicationCurrent, angle);
    Motor1.solveMagneticPotential();
    T1 = SymMult * Motor1.calcTorqueCoupling();
    T2 = SymMult * Motor1.calcTorqueBrBtRotor();
    T3 = SymMult* Motor1.calcTorqueBrBtStator();
    Torques = [Torques, [T1;T2;T3]];
end
```

Programming Part – Solution 3c

```
figure
plot(angles, Torques);
legend("Coupling Rt", "BrBt Rotor", "BrBt Stator")
grid on
xlabel("Degree")
ylabel("Torque (Nm/m)")
```

We see:

Boundary integrals are not such a problem in IGA!



Thank You for Your Attention!