GeoPDEs
An Octave/Matlab software for research on IGA

R. Vázquez

IMATI 'Enrico Magenes', Pavia
Consiglio Nazionale della Ricerca

Joint work with
C. de Falco and A. Reali

Supported by the ERC Starting Grant:
GeoPDEs n. 205004

Motivation

We wanted to share our codes with people interested on IGA.

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

Primary goal: uniform implementation of our different codes.

Motivation

We wanted to share our codes with people interested on IGA.

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

Primary goal: uniform implementation of our different codes.

The result is **GeoPDEs**: open and free software for IGA.

The software is implemented in **Octave**, fully compatible with **Matlab**.

- Very **clear** for **teaching** purposes.
- Easy to modify and to use for **fast prototyping**.
- Follows an **abstract setting** to cover many problems and methods.

Motivation

We wanted to share our codes with people interested on IGA.

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

Primary goal: uniform implementation of our different codes.

The result is **GeoPDEs**: open and free software for IGA.

The software is implemented in **Octave**, fully compatible with **Matlab**.

- Very **clear** for **teaching** purposes.
- Easy to modify and to use for **fast prototyping**.
- Follows an **abstract setting** to cover many problems and methods.

Secondary goal: faster and more efficient implementation.

- Important advances in **GeoPDEs 2.0** (last part of the talk).

General description of the software

GeoPDEs consists of a set of interrelated **packages** for different problems:

- **base**: the main package, with examples for Laplace problem.
- **elasticity**: a simple package for linear elasticity problems.
- **fluid**: Stokes' equations, with different choices for the discrete spaces.
- **maxwell**: Maxwell equations, generalization of edge finite elements.
- **multipatch**: extension to multi-patch defined geometries.

General description of the software

GeoPDEs consists of a set of interrelated **packages** for different problems:

- **base**: the main package, with examples for Laplace problem.
- **elasticity**: a simple package for linear elasticity problems.
- **fluid**: Stokes' equations, with different choices for the discrete spaces.
- **maxwell**: Maxwell equations, generalization of edge finite elements.
- **multipatch**: extension to multi-patch defined geometries.

The **main structures** and functions are defined in the **base** package.

The other packages are based on the structures defined in **base**.

The nomenclature is mostly the same in every package.

General description of the software

GeoPDEs consists of a set of interrelated **packages** for different problems:

- **base**: the main package, with examples for Laplace problem.
- **elasticity**: a simple package for linear elasticity problems.
- **fluid**: Stokes' equations, with different choices for the discrete spaces.
- **maxwell**: Maxwell equations, generalization of edge finite elements.
- **multipatch**: extension to multi-patch defined geometries.

The **main structures** and functions are defined in the **base** package.

The other packages are based on the structures defined in **base**.

The nomenclature is mostly the same in every package.

We define IGA in an **abstract way**, to cover as many cases as possible.

IGA in an abstract framework

- Problem to solve at the continuous level.

Abstract framework

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

Simple example: Poisson equation

$$\int_\Omega \mathbf{grad}\, u \cdot \mathbf{grad}\, v = \int_\Omega f\, v, \quad \forall v \in H_0^1(\Omega).$$

IGA in an abstract framework

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

Abstract framework

$$\mathbf{F} : \widehat{\Omega} \longrightarrow \Omega \subset \mathbb{R}^d, \text{ and } \mathbf{F} \text{ is known and computable.}$$

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 **parameterization** of the physical domain.
- **Discrete** problem and **spaces** in the parametric and physical domain.

Abstract framework

$$a(u_h, v_h) = (f, v_h), \quad \forall v_h \in V_h.$$

Simple example: Poisson equation

$$\int_\Omega \mathbf{grad}\, u_h \cdot \mathbf{grad}\, v_h = \int_\Omega f\, v_h, \quad \forall v_h \in V_h.$$

IGA in an abstract framework

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

Abstract framework

$$V_h = \{v_h : \iota(v_h) = \widehat{v}_h \in \widehat{V}_h\}, \text{ where } \iota \text{ is a pull-back depending on } \mathbf{F},$$

and $\widehat{V}_h = \mathrm{span}\{\widehat{v}_j\}_{j=1}^{N_h}$ is a finite-dimensional and computable space.

Simple example: Poisson equation

$$V_h = \{v_h : v_h \circ \mathbf{F} = \widehat{v}_h \in \widehat{V}_h\},$$

with $\widehat{V}_h = \mathrm{span}\{R_j\}_{i=1}^{N_h}$ a space of NURBS.

IGA in an abstract framework

- Problem to solve at the continuous level.
- Parametric domain and **parameterization** 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

Trial function $u_h = \sum_{i=1}^{N_h} \alpha_i v_i$, and test again every v_j, to get

$$\sum_{i=1}^{N_h} \alpha_i a(v_i, v_j) = (f, v_j),\ j = 1, \ldots, N_h, \quad \text{or} \quad \sum_{i=1}^{N_h} A_{ji} \alpha_i = b_j.$$

Simple example: Poisson equation

Trial function $u_h = \sum_{i=1}^{N_h} \alpha_i R_i$, and test functions R_j:

$$\sum_{i=1}^{N_h} \alpha_i \int_\Omega \mathbf{grad}\, R_i \cdot \mathbf{grad}\, R_j = \int_\Omega f\, R_j, \quad j = 1, \ldots, N_h.$$

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**: $\{(\widehat{\mathbf{x}}_{\ell,k}, w_{\ell,k})\}_{\ell=1}^{n_k}$.

$\int_\Omega \phi(\mathbf{x}) = \sum_{k=1}^{N_e} \int_{\widehat{K}_k} \phi(\mathbf{F}(\widehat{\mathbf{x}})) |\det(D\mathbf{F}(\widehat{\mathbf{x}}))|$

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**: $\{(\widehat{\mathbf{x}}_{\ell,k}, w_{\ell,k})\}_{\ell=1}^{n_k}$.

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

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**: $\{(\widehat{\mathbf{x}}_{\ell,k}, w_{\ell,k})\}_{\ell=1}^{n_k}$.

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

Using the quadrature rule, the **stiffness matrix** is computed as

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

And recall that $\mathbf{grad}\, v_i(\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k})) = D\mathbf{F}^{-T}\, \widehat{\mathbf{grad}}\, \widehat{v}_i(\widehat{\mathbf{x}}_{\ell,k})$.

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**: $\{(\widehat{\mathbf{x}}_{\ell,k}, w_{\ell,k})\}_{\ell=1}^{n_k}$.

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

Using the quadrature rule, the **stiffness matrix** is computed as

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

And recall that $\mathbf{grad}\, v_i(\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k})) = D\mathbf{F}^{-T}\, \widehat{\mathbf{grad}}\, \widehat{v}_i(\widehat{\mathbf{x}}_{\ell,k})$.

Summarizing, what we need is

- 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.

The main structures of GeoPDEs

GeoPDEs has been implemented following the abstract framework.

The code is based on **three** main **structures**:

- **Geometry**: the parameterization $\mathbf{F}$ and its derivatives.
- **Mesh**: the partition of the domain and the **quadrature rule**.
- **Space**: the **shape functions** of the discrete space V_h.

The main structures of GeoPDEs

GeoPDEs has been implemented following the abstract framework.

The code is based on **three** main **structures**:

- **Geometry**: the parameterization $\mathbf{F}$ and its derivatives.
- **Mesh**: the partition of the domain and the **quadrature rule**.
- **Space**: the **shape functions** of the discrete space V_h.

Everything is **precomputed** (v.1). Easy to understand and to debug.

As a consequence, the computation of the matrices is very **clear**.

The structures can be used in **different applications** with minor changes.

The parameterization: geometry structure

Computation of the **parameterization** $\mathbf{F}$ and its derivatives.

- **map**: function handle to compute $\mathbf{F}$ at given points in $\widehat{\Omega}$.
- **map_der**: function handle to compute $D\mathbf{F}$, the derivatives of $\mathbf{F}$.

The fields contain the handles to evaluate $\mathbf{F}$, not the values of $\mathbf{F}$.

The parameterization: geometry structure

Computation of the **parameterization** $\mathbf{F}$ and its derivatives.

- **map**: function handle to compute $\mathbf{F}$ at given points in $\widehat{\Omega}$.
- **map_der**: function handle to compute $D\mathbf{F}$, the derivatives of $\mathbf{F}$.

The fields contain the handles to evaluate $\mathbf{F}$, not the values of $\mathbf{F}$.

For NURBS and B-splines, we make use of the **NURBS toolbox**.

- Based on standard NURBS algorithms (see, e.g., the NURBS book).
- Useful for simple **geometry manipulation** (revolution, extrusion,...)
- It is also used in GeoPDEs for **function evaluation**.

The parameterization: geometry structure

Computation of the **parameterization** $\mathbf{F}$ and its derivatives.

- **map**: function handle to compute $\mathbf{F}$ at given points in $\widehat{\Omega}$.
- **map_der**: function handle to compute $D\mathbf{F}$, the derivatives of $\mathbf{F}$.

The fields contain the handles to evaluate $\mathbf{F}$, not the values of $\mathbf{F}$.

For NURBS and B-splines, we make use of the **NURBS toolbox**.

- Based on standard NURBS algorithms (see, e.g., the NURBS book).
- Useful for simple **geometry manipulation** (revolution, extrusion,...)
- It is also used in GeoPDEs for **function evaluation**.

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

- Necessary for **non-isoparametric** discretizations.
- Geometry evaluations can be made in the **coarsest** given **geometry**.

The quadrature rule: mesh structure

Contains information on the **partition** of the domain, $\widehat{\Omega} = \cup_{k=1}^{N_e} \widehat{K}_k$,

and the **quadrature rule** $\{(\widehat{\mathbf{x}}_{\ell,k}, w_{\ell,k})\}_{\ell=1}^{n_k}$.

The quadrature rule: mesh structure

Contains information on the **partition** of the domain, $\widehat{\Omega} = \cup_{k=1}^{N_e} \widehat{K}_k$,

and the **quadrature rule** $\{(\widehat{\mathbf{x}}_{\ell,k}, w_{\ell,k})\}_{\ell=1}^{n_k}$.

Remember the expression for the entries of the stiffness matrix

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

- **nel**: N_e, number of elements of the partition.
- **nqn**: n_k, number of quadrature nodes per element.
- **quad_nodes**: $\widehat{\mathbf{x}}_{\ell,k}$, quadrature nodes in $\widehat{\Omega}$.
- **quad_weights**: $w_{\ell,k}$, quadrature weights.

The quadrature rule: mesh structure

Contains information on the **partition** of the domain, $\widehat{\Omega} = \cup_{k=1}^{N_e} \widehat{K}_k$,

and the **quadrature rule** $\{(\widehat{\mathbf{x}}_{\ell,k}, w_{\ell,k})\}_{\ell=1}^{n_k}$.

Remember the expression for the entries of the stiffness matrix

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

- **nel**: N_e, number of elements of the partition.
- **nqn**: n_k, number of quadrature nodes per element.
- **quad_nodes**: $\widehat{\mathbf{x}}_{\ell,k}$, quadrature nodes in $\widehat{\Omega}$.
- **quad_weights**: $w_{\ell,k}$, quadrature weights.

- **geo_map**: $\mathbf{x}_{\ell,k} = \mathbf{F}(\widehat{\mathbf{x}}_{\ell,k})$, quadrature nodes in Ω.
- **geo_map_jac**: $D\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k})$, Jacobian matrix evaluated at **quad_nodes**.
- **jacdet**: $|\det(D\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k}))|$, absolute value of the Jacobian.

The discrete space: space structure

Information on the basis functions of the **discrete space**

$$V_h = \mathrm{span}\{v_i\}_{i=1}^{N_h},$$

and their **values** at the **quadrature points**.

The discrete space: space structure

Information on the basis functions of the **discrete space**

$$V_h = \mathrm{span}\{v_i\}_{i=1}^{N_h},$$

and their **values** at the **quadrature points**.

- **ndof**: N_h, total number of degrees of freedom.
- **nsh**: number of non-vanishing functions in each "element".
- **connectivity** (IEN): global indices of non-vanishing basis functions.

The discrete space: space structure

Information on the basis functions of the **discrete space**

$$V_h = \mathrm{span}\{v_i\}_{i=1}^{N_h},$$

and their **values** at the **quadrature points**.

- **ndof**: N_h, total number of degrees of freedom.
- **nsh**: number of non-vanishing functions in each "element".
- **connectivity** (IEN): global indices of non-vanishing basis functions.
- **shape_functions**: $v_i(\mathbf{x}_{\ell,k})$, shape functions evaluated at the quadrature points.
- **shape_function_gradients**: $\mathbf{grad}\, v_i(\mathbf{x}_{\ell,k})$, gradients of the shape functions evaluated at the quadrature points.

For NURBS and splines, they are computed with the NURBS toolbox.

Other fields may be necessary (curl, divergence, Laplacian...)

A simple example on how to use GeoPDEs

```
geometry = geo_load('ring_refined.mat');
knots = geometry.nurbs.knots;
```

- Create the **geometry** structure, from a NURBS toolbox file.

A simple example on how to use GeoPDEs

```
geometry = geo_load('ring_refined.mat');
knots = geometry.nurbs.knots;
[qn, qw] = msh_set_quad_nodes(knots,msh_gauss_nodes(ngauss));
msh = msh_2d (knots, qn, qw, geometry);
```

- Create the **geometry** structure, from a NURBS toolbox file.
- Create the **mesh** structure in the parametric domain.
- Map the **mesh** structure to the physical domain, using **geometry**.

A simple example on how to use GeoPDEs

```
geometry = geo_load('ring_refined.mat');
knots = geometry.nurbs.knots;
[qn, qw] = msh_set_quad_nodes(knots,msh_gauss_nodes(ngauss));
msh = msh_2d (knots, qn, qw, geometry);
space = sp_nurbs_2d (geometry.nurbs, msh);
```

- Create the **geometry** structure, from a NURBS toolbox file.
- Create the **mesh** structure in the parametric domain.
- Map the **mesh** structure to the physical domain, using **geometry**.
- Construct the **space** structure (the knots are stored in **geometry**).

A simple example on how to use GeoPDEs

```
geometry = geo_load('ring_refined.mat');
knots = geometry.nurbs.knots;
[qn, qw] = msh_set_quad_nodes(knots,msh_gauss_nodes(ngauss));
msh = msh_2d (knots, qn, qw, geometry);
space = sp_nurbs_2d (geometry.nurbs, msh);
[x, y] = deal (msh.geo_map(1,:,:), msh.geo_map(2,:,:));
mat = op_gradu_gradv (space, msh);
rhs = op_f_v (space, msh, rhs_fun(x, y));
```

- Create the **geometry** structure, from a NURBS toolbox file.
- Create the **mesh** structure in the parametric domain.
- Map the **mesh** structure to the physical domain, using **geometry**.
- Construct the **space** structure (the knots are stored in **geometry**).
- Build the matrix and right-hand side.

Matrices and vector construction

Remember the expression for the entries of the stiffness matrix
$A_{ij} \simeq \sum_{k=1}^{N_e} \sum_{\ell=1}^{n_k} w_{\ell,k} \, \mathbf{grad} \, v_j(\mathbf{F}(\mathbf{x}_{\ell,k})) \cdot \mathbf{grad} \, v_i(\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k})) \, |\det(D\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k}))|$

- Everything is **precomputed** (v.1) in the previous structures.
- To construct the matrices, it is enough to correctly gather the information.
- The computation of the matrices is **simple**, and identical to FEM.

Matrices and vector construction

Remember the expression for the entries of the stiffness matrix
$A_{ij} \simeq \sum_{k=1}^{N_e} \sum_{\ell=1}^{n_k} w_{\ell,k} \, \mathbf{grad} \, v_j(\mathbf{F}(\mathbf{x}_{\ell,k})) \cdot \mathbf{grad} \, v_i(\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k})) \, |\det(D\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k}))|$

- Everything is **precomputed** (v.1) in the previous structures.
- To construct the matrices, it is enough to correctly gather the information.
- The computation of the matrices is **simple**, and identical to FEM.

Let me show an example.

Matrices and vector construction

Remember the expression for the entries of the stiffness matrix
$A_{ij} \simeq \sum_{k=1}^{N_e} \sum_{\ell=1}^{n_k} w_{\ell,k} \, \mathbf{grad} \, v_j(\mathbf{F}(\mathbf{x}_{\ell,k})) \cdot \mathbf{grad} \, v_i(\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k})) \, |\det(D\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k}))|$

```
function mat = op_gradu_gradv (space, msh)
for iel = 1:msh.nel
  mat_loc = zeros (space.nsh(iel), space.nsh(iel));
  for idof = 1:space.nsh(iel)
    ishp = space.shape_function_gradients(:,:,idof,iel);
    for jdof = 1:space.nsh(iel)
      jshp = space.shape_function_gradients(:,:,jdof,iel);
      for inode = 1:msh.nqn
        mat_loc(idof,jdof) += ishp(:,inode).*jshp(:,inode) *
         msh.jacdet(inode,iel) * msh.quad_weights(inode,iel);
      endfor %inode
    endfor %jdof
  endfor %idof
  mat(space.connect(:,iel), space.connect(:,iel)) += mat_loc;
endfor %iel
```

Matrices and vector construction

Remember the expression for the entries of the stiffness matrix
$A_{ij} \simeq \sum_{k=1}^{N_e} \sum_{\ell=1}^{n_k} w_{\ell,k} \, \mathbf{grad} \, v_j(\mathbf{F}(\mathbf{x}_{\ell,k})) \cdot \mathbf{grad} \, v_i(\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k})) \, | \, \det(D\mathbf{F}(\widehat{\mathbf{x}}_{\ell,k}))|$

- Everything is **precomputed** (v.1) in the previous structures.
- To construct the matrices, it is enough to correctly gather the information.
- The computation of the matrices is **simple**, and identical to FEM.

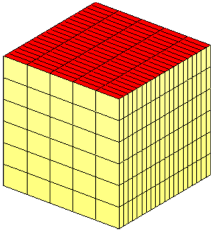
Actually, the efficient Matlab implementation is quite different.

For NURBS, we can also take advantage of the tensor product structure.

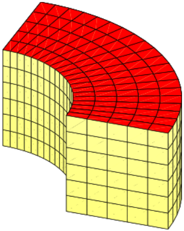
Boundary conditions: the boundary substructures

The structures **mesh** and **space** are completed with the field **boundary**.

These are mesh and space **substructures** of dimension $N-1$.



$\xrightarrow{\mathbf{F}}$



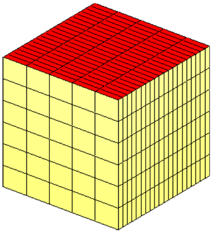
Boundary conditions: the boundary substructures

The structures **mesh** and **space** are completed with the field **boundary**.

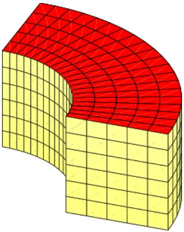
These are mesh and space **substructures** of dimension $N-1$.

The boundary structures have some particularities:

- **jacdet** contains the area element of the boundary parameterization.
- The **space** structure uses a local numbering for each boundary.
- A new field, **dofs**, is added to recover the global numbering.



$\xrightarrow{\mathbf{F}}$



Boundary conditions: the boundary substructures

For **Neumann** conditions, $\frac{\partial u}{\partial n} = g$ on Γ_N, we must compute $\int_{\Gamma_N} g v_j$.

- The integral is computed in the same manner as for bulk forces.
- It is assembled into the global r.h.s. using the field **dofs**.

```
x = msh.boundary.geo_map(1,:,:);
y = msh.boundary.geo_map(2,:,:);
rhs_bnd = op_f_v(space.boundary,msh.boundary,g(x,y));
rhs(space.boundary.dofs) += rhs_bnd;
```

Boundary conditions: the boundary substructures

For **Neumann** conditions, $\frac{\partial u}{\partial n} = g$ on Γ_N, we must compute $\int_{\Gamma_N} g v_j$.

- The integral is computed in the same manner as for bulk forces.
- It is assembled into the global r.h.s. using the field **dofs**.

```
x = msh.boundary.geo_map(1,:,:);
y = msh.boundary.geo_map(2,:,:);
rhs_bnd = op_f_v(space.boundary,msh.boundary,g(x,y));
rhs(space.boundary.dofs) += rhs_bnd;
```

For **Dirichlet** conditions, $u = h$ on Γ_D, we must assign the d.o.f. in **boundary.dofs**.

- The needed information should already be in the **boundary** structures.
- As an example we have included the least squares best fit, i.e.

$$\int_{\Gamma_D} uv = \int_{\Gamma_D} hv \quad \forall v.$$

A simple example on how to use GeoPDEs

We have computed all the **structures** and the **linear system**.

```
geometry = geo_load('ring_refined.mat');
knots = geometry.nurbs.knots;
[qn, qw] = msh_set_quad_nodes(knots,msh_gauss_nodes(ngauss));
msh = msh_2d (knots, qn, qw, geometry);
space = sp_nurbs_2d (geometry.nurbs, msh);
[x, y] = deal (msh.geo_map(1,:,:), msh.geo_map(2,:,:));
mat = op_gradu_gradv (space, msh);
rhs = op_f_v (space, msh, rhs_fun(x, y));
```

A simple example on how to use GeoPDEs

Apply **boundary conditions** and **solve** the linear system.

```
geometry = geo_load('ring_refined.mat');
knots = geometry.nurbs.knots;
[qn, qw] = msh_set_quad_nodes(knots,msh_gauss_nodes(ngauss));
msh = msh_2d (knots, qn, qw, geometry);
space = sp_nurbs_2d (geometry.nurbs, msh);
[x, y] = deal (msh.geo_map(1,:,:), msh.geo_map(2,:,:));
mat = op_gradu_gradv (space, msh);
rhs = op_f_v (space, msh, rhs_fun(x, y));
drchlt_dofs = unique ([space.boundary(:).dofs]);
int_dofs = setdiff (1:space.ndof, drchlt_dofs);
u(drchlt_dofs) = 0;
u(int_dofs) = mat(int_dofs, int_dofs) \ rhs(int_dofs);
```

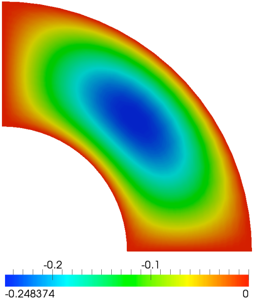
A simple example on how to use GeoPDEs

We end up with some **postprocessing**.

```
geometry = geo_load('ring_refined.mat');
knots = geometry.nurbs.knots;
[qn, qw] = msh_set_quad_nodes(knots, msh_gauss_nodes(ngauss));
msh = msh_2d (knots, qn, qw, geometry);
space = sp_nurbs_2d (geometry.nurbs, msh);
[x, y] = deal (msh.geo_map(1,:,:), msh.geo_map(2,:,:));
mat = op_gradu_gradv (space, msh);
rhs = op_f_v (space, msh, rhs_fun(x, y));
drchlt_dofs = unique ([space.boundary(:).dofs]);
int_dofs = setdiff (1:space.ndof, drchlt_dofs);
u(drchlt_dofs) = 0;
u(int_dofs) = mat(int_dofs, int_dofs) \ rhs(int_dofs);
sp_to_vtk (u, space, geometry, [20 20], filename, 'u');
err = sp_l2_error (space, msh, u, exact_solution(x, y));
```

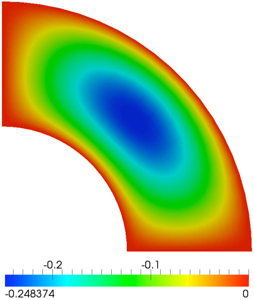
A simple example on how to use GeoPDEs

And the final result is something like this.



A simple example on how to use GeoPDEs

And the final result is something like this.



The package contains **several** simple **examples**:

h-,*p*-,*k*-refinement, 2D and 3D, B-splines and NURBS...

The structures can be **easily extended** to solve more **complex problems**.

Linear elasticity problems: the elasticity package

Let us see how to solve the following **linear elasticity** problem:

Find $\mathbf{u} \in V = (H^1_{0,\Gamma_D}(\Omega))^3$ *such that*

$$\int_\Omega (2\mu\,\varepsilon(\mathbf{u}) : \varepsilon(\mathbf{v}) + \lambda\,\mathrm{div}(\mathbf{u})\,\mathrm{div}(\mathbf{v})) = \int_\Omega \mathbf{f}\cdot\mathbf{v} + \int_{\Gamma_N} \mathbf{g}\cdot\mathbf{v} \quad \forall \mathbf{v} \in V,$$

Linear elasticity problems: the elasticity package

Let us see how to solve the following **linear elasticity** problem:

Find $\mathbf{u} \in V = (H^1_{0,\Gamma_D}(\Omega))^3$ *such that*

$$\int_\Omega (2\mu\, \varepsilon(\mathbf{u}) : \varepsilon(\mathbf{v}) + \lambda\, \mathrm{div}(\mathbf{u})\, \mathrm{div}(\mathbf{v})) = \int_\Omega \mathbf{f} \cdot \mathbf{v} + \int_{\Gamma_N} \mathbf{g} \cdot \mathbf{v} \quad \forall \mathbf{v} \in V,$$

- The **geometry** and **mesh** are described as in the previous example.
- The basis functions in the **space** structure are now vector-valued.
- A specific **operator** for this problem must be defined.
- Imposing the boundary conditions is similar to the previous problem.

Definition of the vectorial space

We first define one **space** structure for **each component**.

From these we define the vector-valued **space** structure for our problem.

```
spx = spy = spz = sp_nurbs_3d (geometry, msh);
space = sp_vector_3d (spx, spy, spz, msh);
```

This command computes the new vector-valued **space** and the numbering.

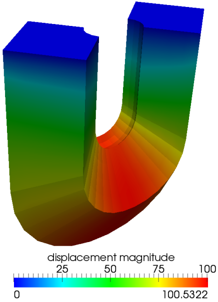
Definition of the vectorial space

We first define one **space** structure for **each component**.

From these we define the vector-valued **space** structure for our problem.

```
spx = spy = spz = sp_nurbs_3d (geometry, msh);
space = sp_vector_3d (spx, spy, spz, msh);
```

This command computes the new vector-valued **space** and the numbering.

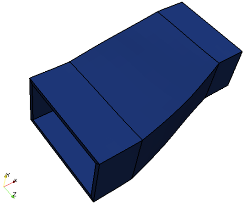


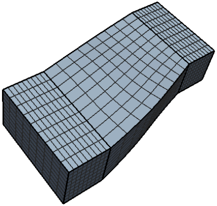
The horseshoe is a courtesy of T.J.R. Hughes' and his group

Electromagnetism: the Maxwell package

An example in electromagnetism:

- Metallic WR-2300 waveguide with a mechanical deformation.
- Consider only the TE_{10} mode at $\simeq 0.35$ GHz.
- Compute relative amplitudes of the transmitted and reflected waves.

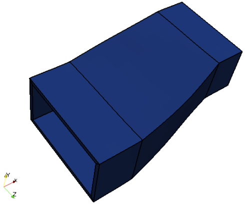


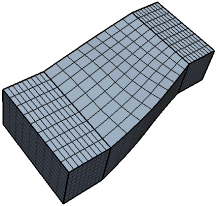


Electromagnetism: the Maxwell package

An example in electromagnetism:

- Metallic WR-2300 waveguide with a mechanical deformation.
- Consider only the TE_{10} mode at $\simeq 0.35$ GHz.
- Compute relative amplitudes of the transmitted and reflected waves.





Time-harmonic Maxwell equations in the interior of the waveguide.

Forward (known) and reverse (unknown) traveling waves at Γ_i.

Forward (unknown) traveling wave at Γ_o.

Perfect electrical conductor ($\mathbf{E} \times \mathbf{n} = \mathbf{0}$) on the other boundaries.

Electromagnetism: the Maxwell package

Find $\mathbf{E} \in \mathbf{H}_{0,\Gamma_D}(\mathbf{curl};\Omega)$ *, and* $\alpha_i^r, \alpha_o^f \in \mathbb{C}$ *such that*

$$\int_\Omega (\frac{1}{\mu}\mathbf{curl}\,\mathbf{E} \cdot \mathbf{curl}\,\overline{\mathbf{G}} - \omega^2\varepsilon \mathbf{E} \cdot \overline{\mathbf{G}}) +$$

$$\mathrm{i}\omega(\alpha_i^r \int_{\Gamma_i} \mathbf{H}_\tau^{10} \cdot \overline{\mathbf{G}}_\tau + \alpha_o^f \int_{\Gamma_o} \mathbf{H}_\tau^{10} \cdot \overline{\mathbf{G}}_\tau) = \mathrm{i}\omega\alpha_i^f \int_{\Gamma_i} \mathbf{H}_\tau^{10} \cdot \overline{\mathbf{G}}_\tau,$$

$$\mathrm{i}\omega(\int_{\Gamma_i} \mathbf{E}_\tau \cdot \mathbf{H}_\tau^{10} - \alpha_i^r \int_{\Gamma_i} \mathbf{E}_\tau^{10} \cdot \mathbf{H}_\tau^{10}) = \mathrm{i}\omega\alpha_i^f \int_{\Gamma_i} \mathbf{E}_\tau^{10} \cdot \mathbf{H}_\tau^{10},$$

$$\mathrm{i}\omega(\int_{\Gamma_o} \mathbf{E}_\tau \cdot \mathbf{H}_\tau^{10} - \alpha_o^f \int_{\Gamma_o} \mathbf{E}_\tau^{10} \cdot \mathbf{H}_\tau^{10}) = 0.$$

- The **geometry** and **mesh** are described as in the previous examples.
- We need an operator to compute the **curl-curl** matrix.

Electromagnetism: the Maxwell package

Find $\mathbf{E} \in \mathbf{H}_{0,\Gamma_D}(\mathbf{curl};\Omega)$ *, and* $\alpha_i^r, \alpha_o^f \in \mathbb{C}$ *such that*

$$\int_\Omega (\tfrac{1}{\mu}\mathbf{curl}\,\mathbf{E}\cdot\mathbf{curl}\,\overline{\mathbf{G}} - \omega^2\varepsilon\mathbf{E}\cdot\overline{\mathbf{G}}) +$$

$$\mathrm{i}\omega(\alpha_i^r\int_{\Gamma_i}\mathbf{H}_\tau^{10}\cdot\overline{\mathbf{G}}_\tau + \alpha_o^f\int_{\Gamma_o}\mathbf{H}_\tau^{10}\cdot\overline{\mathbf{G}}_\tau) = \mathrm{i}\omega\alpha_i^f\int_{\Gamma_i}\mathbf{H}_\tau^{10}\cdot\overline{\mathbf{G}}_\tau,$$

$$\mathrm{i}\omega(\int_{\Gamma_i}\mathbf{E}_\tau\cdot\mathbf{H}_\tau^{10} - \alpha_i^r\int_{\Gamma_i}\mathbf{E}_\tau^{10}\cdot\mathbf{H}_\tau^{10}) = \mathrm{i}\omega\alpha_i^f\int_{\Gamma_i}\mathbf{E}_\tau^{10}\cdot\mathbf{H}_\tau^{10},$$

$$\mathrm{i}\omega(\int_{\Gamma_o}\mathbf{E}_\tau\cdot\mathbf{H}_\tau^{10} - \alpha_o^f\int_{\Gamma_o}\mathbf{E}_\tau^{10}\cdot\mathbf{H}_\tau^{10}) = 0.$$

- The **geometry** and **mesh** are described as in the previous examples.
- We need an operator to compute the **curl-curl** matrix.
- Vector-valued **space** with a curl-conserving transform.

The shape functions are given by $\mathbf{E}\circ\mathbf{F} = D\mathbf{F}^{-\top}\widehat{\mathbf{E}}$.

And $(\mathbf{curl}\,\mathbf{E})\circ\mathbf{F} = \frac{1}{\det(D\mathbf{F})}D\mathbf{F}(\widehat{\mathbf{curl}}\,\widehat{\mathbf{E}})$.

Electromagnetism: the Maxwell package

Find $\mathbf{E} \in \mathbf{H}_{0,\Gamma_D}(\mathbf{curl};\Omega)$ *, and* $\alpha_i^r, \alpha_o^f \in \mathbb{C}$ *such that*

$$\int_\Omega (\tfrac{1}{\mu}\mathbf{curl}\,\mathbf{E} \cdot \mathbf{curl}\,\overline{\mathbf{G}} - \omega^2 \varepsilon \mathbf{E} \cdot \overline{\mathbf{G}}) +$$

$$\mathrm{i}\omega(\alpha_i^r \int_{\Gamma_i} \mathbf{H}_\tau^{10} \cdot \overline{\mathbf{G}}_\tau + \alpha_o^f \int_{\Gamma_o} \mathbf{H}_\tau^{10} \cdot \overline{\mathbf{G}}_\tau) = \mathrm{i}\omega\alpha_i^f \int_{\Gamma_i} \mathbf{H}_\tau^{10} \cdot \overline{\mathbf{G}}_\tau,$$

$$\mathrm{i}\omega(\int_{\Gamma_i} \mathbf{E}_\tau \cdot \mathbf{H}_\tau^{10} - \alpha_i^r \int_{\Gamma_i} \mathbf{E}_\tau^{10} \cdot \mathbf{H}_\tau^{10}) = \mathrm{i}\omega\alpha_i^f \int_{\Gamma_i} \mathbf{E}_\tau^{10} \cdot \mathbf{H}_\tau^{10},$$

$$\mathrm{i}\omega(\int_{\Gamma_o} \mathbf{E}_\tau \cdot \mathbf{H}_\tau^{10} - \alpha_o^f \int_{\Gamma_o} \mathbf{E}_\tau^{10} \cdot \mathbf{H}_\tau^{10}) = 0.$$

- The **geometry** and **mesh** are described as in the previous examples.
- We need an operator to compute the **curl**-**curl** matrix.
- Vector-valued **space** with a curl-conserving transform.
- For the boundaries, we only store the tangential components.
- Operators to compute the integrals on the boundary are also needed.

Electromagnetism: the Maxwell package

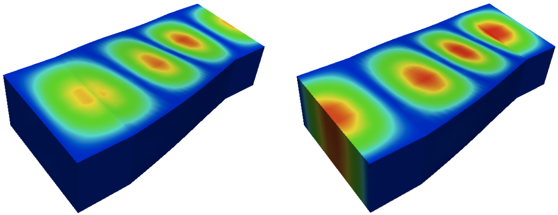
Find $\mathbf{E} \in \mathbf{H}_{0,\Gamma_D}(\mathbf{curl};\Omega)$ *, and* $\alpha_i^r, \alpha_o^f \in \mathbb{C}$ *such that*

$$\int_\Omega (\tfrac{1}{\mu}\mathbf{curl}\,\mathbf{E} \cdot \mathbf{curl}\,\overline{\mathbf{G}} - \omega^2 \varepsilon \mathbf{E} \cdot \overline{\mathbf{G}}) +$$

$$\mathrm{i}\omega(\alpha_i^r \int_{\Gamma_i} \mathbf{H}_\tau^{10} \cdot \overline{\mathbf{G}}_\tau + \alpha_o^f \int_{\Gamma_o} \mathbf{H}_\tau^{10} \cdot \overline{\mathbf{G}}_\tau) = \mathrm{i}\omega\alpha_i^f \int_{\Gamma_i} \mathbf{H}_\tau^{10} \cdot \overline{\mathbf{G}}_\tau,$$

$$\mathrm{i}\omega(\int_{\Gamma_i} \mathbf{E}_\tau \cdot \mathbf{H}_\tau^{10} - \alpha_i^r \int_{\Gamma_i} \mathbf{E}_\tau^{10} \cdot \mathbf{H}_\tau^{10}) = \mathrm{i}\omega\alpha_i^f \int_{\Gamma_i} \mathbf{E}_\tau^{10} \cdot \mathbf{H}_\tau^{10},$$

$$\mathrm{i}\omega(\int_{\Gamma_o} \mathbf{E}_\tau \cdot \mathbf{H}_\tau^{10} - \alpha_o^f \int_{\Gamma_o} \mathbf{E}_\tau^{10} \cdot \mathbf{H}_\tau^{10}) = 0.$$



(a) Real part (b) Imaginary part

GeoPDEs 2.0

Clarity was the primary goal in the first version of **GeoPDEs**.

- All the fields were **precomputed** (high memory consumption).
- Clear but **slow** computation of the matrices.

GeoPDEs 2.0

Clarity was the primary goal in the first version of **GeoPDEs**.

- All the fields were **precomputed** (high memory consumption).
- Clear but **slow** computation of the matrices.

Efficiency became an important issue in **GeoPDEs 2.0**.

- Faster version of matrix computations.
  - Vectorization of some loops.
  - Better use of sparse matrices in Matlab.

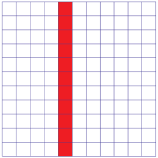
GeoPDEs 2.0

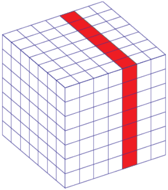
Clarity was the primary goal in the first version of **GeoPDEs**.

- All the fields were **precomputed** (high memory consumption).
- Clear but **slow** computation of the matrices.

Efficiency became an important issue in **GeoPDEs 2.0**.

- Faster version of matrix computations.
- For NURBS, we take advantage of the tensor product structure.
  - Only the 1D functions and derivatives are precomputed.
  - The 2D/3D fields are computed one column at a time.





GeoPDEs 2.0

Clarity was the primary goal in the first version of **GeoPDEs**.

- All the fields were **precomputed** (high memory consumption).
- Clear but **slow** computation of the matrices.

Efficiency became an important issue in **GeoPDEs 2.0**.

- Faster version of matrix computations.
- For NURBS, we take advantage of the tensor product structure.
  - Only the 1D functions and derivatives are precomputed.
  - The 2D/3D fields are computed one column at a time.

```
for iel = 1:msh.nel_dir(1)
  msh_col = msh_evaluate_col (msh, iel);
  sp_col = sp_evaluate_col (space, msh_col);
  A = A + op_gradu_gradv (sp_col, msh_col);
end
```

The “column” structures contain the same fields we have seen before.

Conclusions

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

- Very useful for **teaching** purposes and for new researchers.
- It can serve as a **rapid prototyping** tool to test new ideas.
- Several **packages** already released to solve different problems.
- Many **examples** and a **short guide** with detailed explanations.

Conclusions

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

- Very useful for **teaching** purposes and for new researchers.
- It can serve as a **rapid prototyping** tool to test new ideas.
- Several **packages** already released to solve different problems.
- Many **examples** and a **short guide** with detailed explanations.

In the last 4 months, around 300 downloads of the **base** package.

Conclusions

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

- Very useful for **teaching** purposes and for new researchers.
- It can serve as a **rapid prototyping** tool to test new ideas.
- Several **packages** already released to solve different problems.
- Many **examples** and a **short guide** with detailed explanations.

In the last 4 months, around 300 downloads of the **base** package.

Contributions are welcome, to improve the code or to show new methods.

Conclusions

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

- Very useful for **teaching** purposes and for new researchers.
- It can serve as a **rapid prototyping** tool to test new ideas.
- Several **packages** already released to solve different problems.
- Many **examples** and a **short guide** with detailed explanations.

In the last 4 months, around 300 downloads of the **base** package.

Contributions are welcome, to improve the code or to show new methods.

GeoPDEs tutorial today at 17:00.

Conclusions

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

- Very useful for **teaching** purposes and for new researchers.
- It can serve as a **rapid prototyping** tool to test new ideas.
- Several **packages** already released to solve different problems.
- Many **examples** and a **short guide** with detailed explanations.

In the last 4 months, around 300 downloads of the **base** package.

Contributions are welcome, to improve the code or to show new methods.

GeoPDEs tutorial today at 17:00.

Software download and information

http://geopdes.sourceforge.net

Thanks for your attention!