

An Introduction to Finite Element Modelling with ONELAB

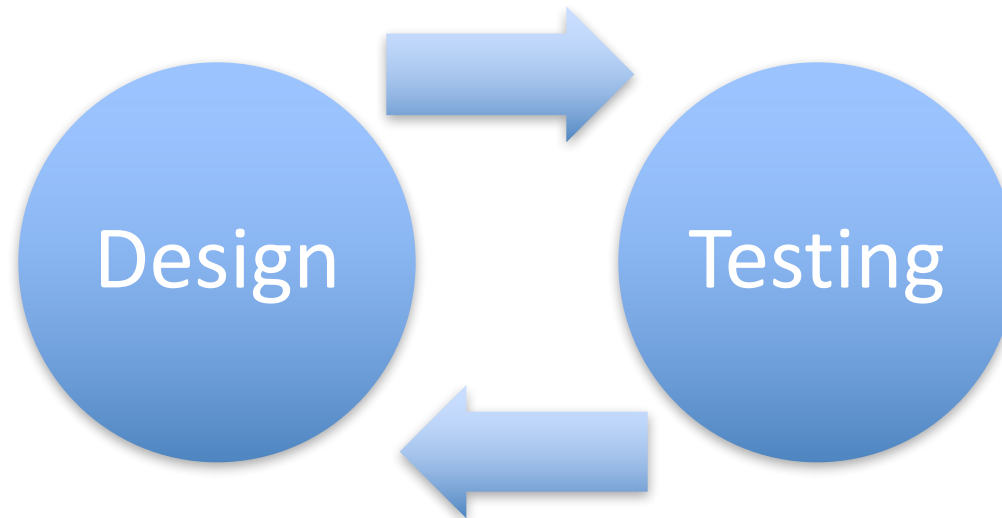
C. Geuzaine, R. Sabariego, J. Gyselinck, F. Henrotte, E. Kuci

BEST Summer Course – University of Liège, Belgium

August 22-26 2016

About modern product development

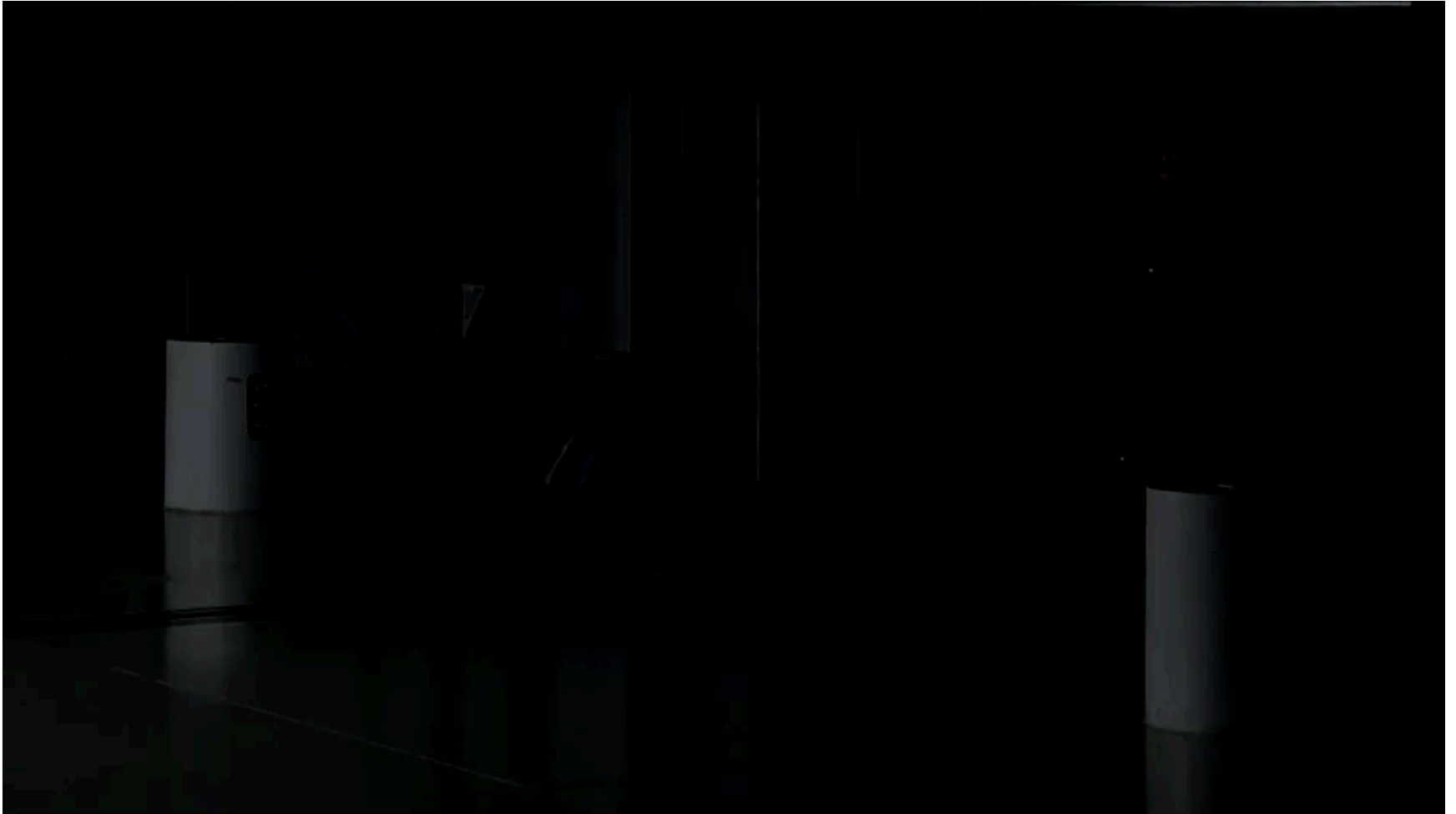
Development cycle



How is testing performed?

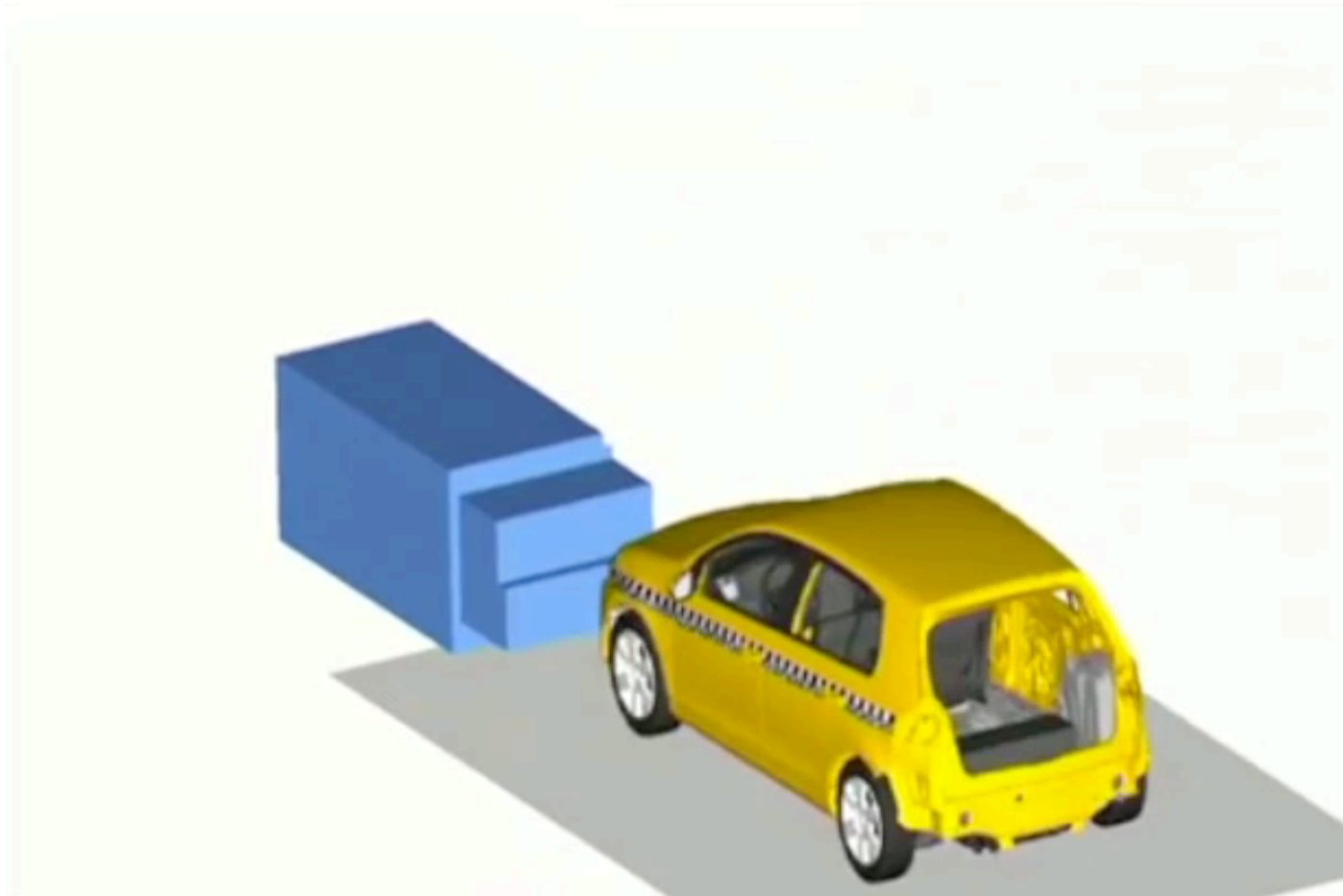
Physical vs. Numerical

Test of a car – The physical crash test



Costly (vehicle 250k€, dummy 50-500k€) and time-consuming

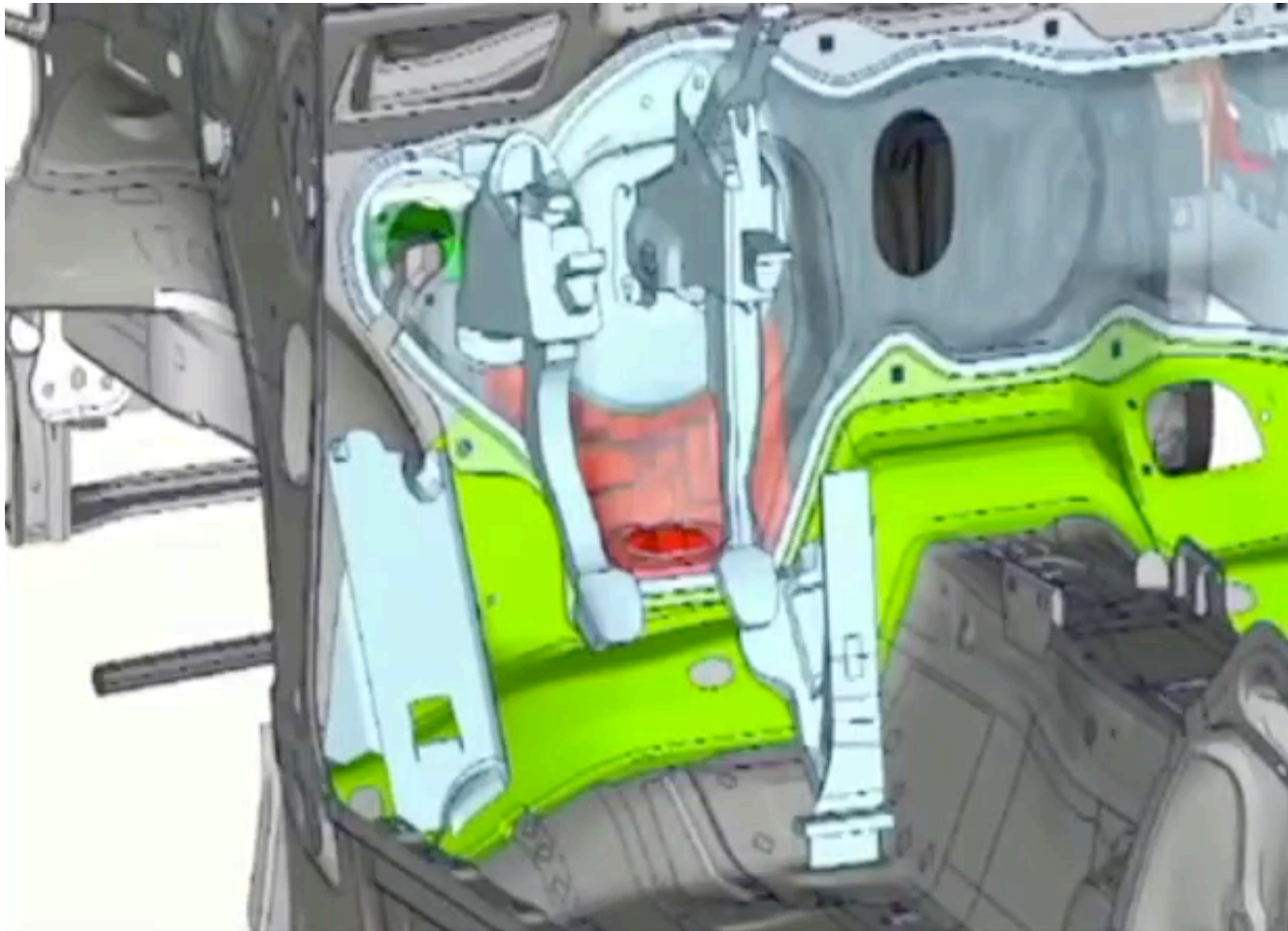
Test of a car – Numerical simulations



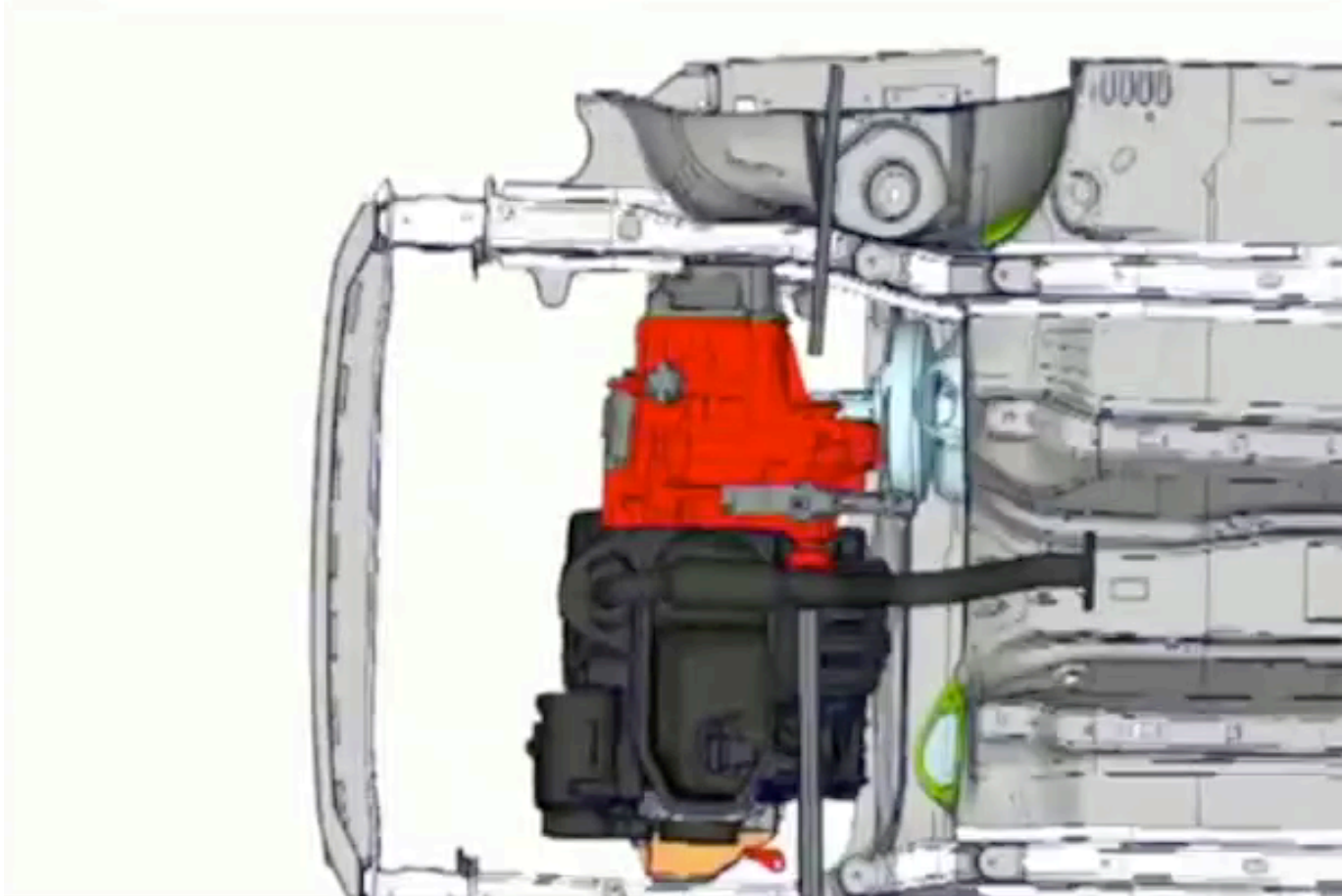
Test of a car – Numerical simulations



Test of a car – Numerical simulations



Test of a car – Numerical simulations



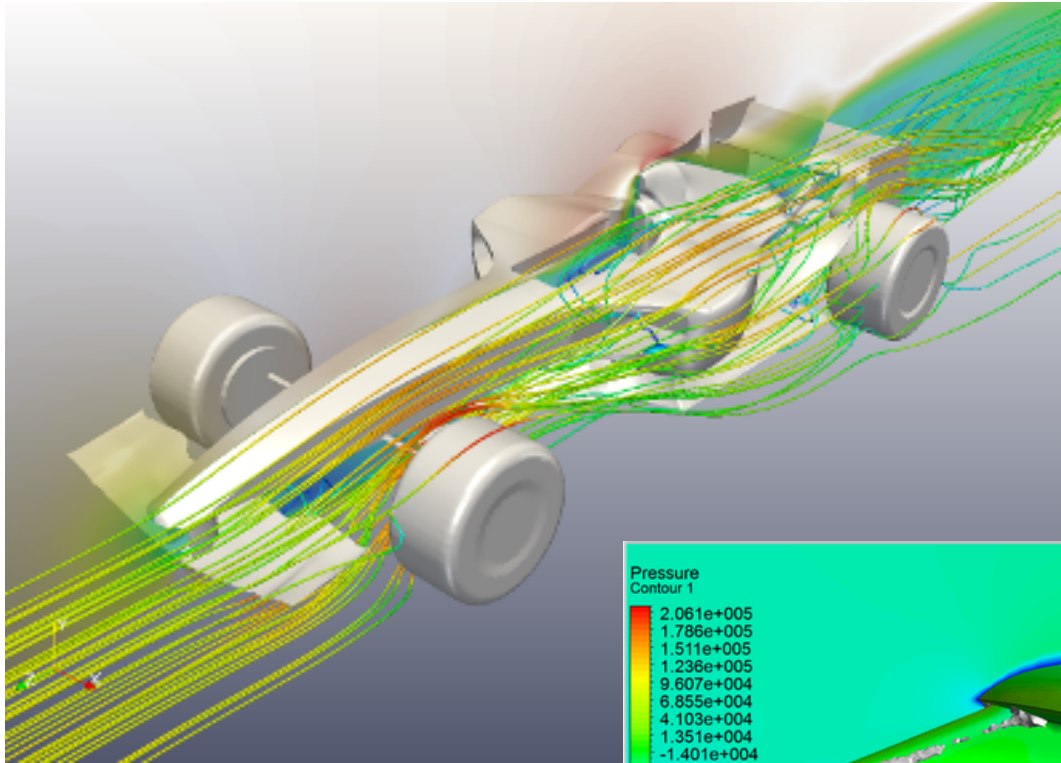
Test of a car – Numerical simulations

- Nowadays, typically more than 100 numerical tests are performed for 1 physical test

Thanks to the finite element method
(and related numerical techniques)

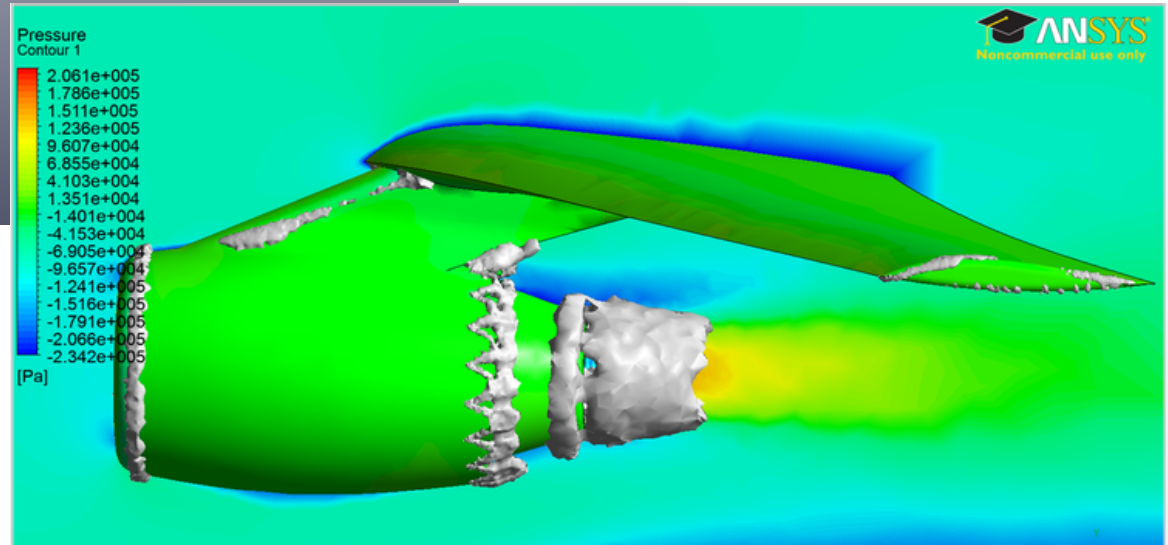
- Both testing strategies are complementary

Finite element method – Domains of application

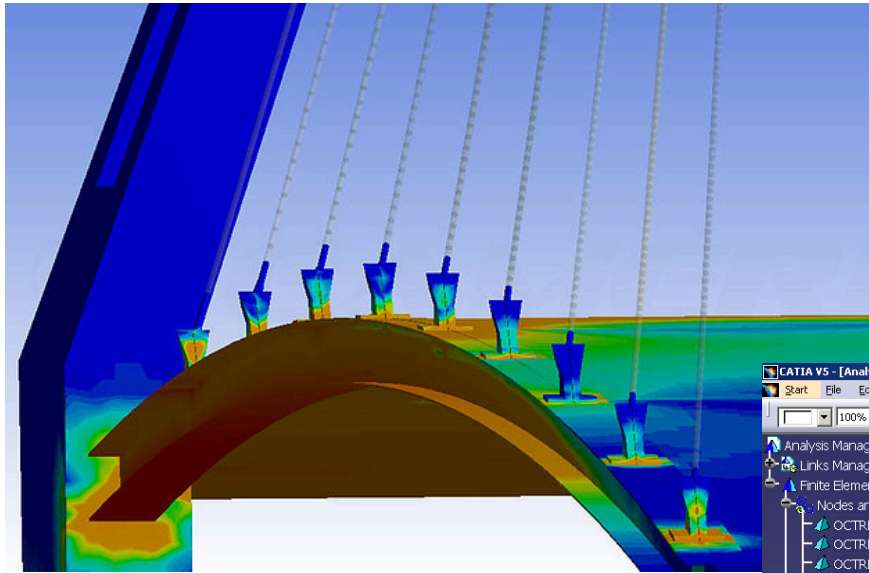


Aerodynamics

Plane engine flow

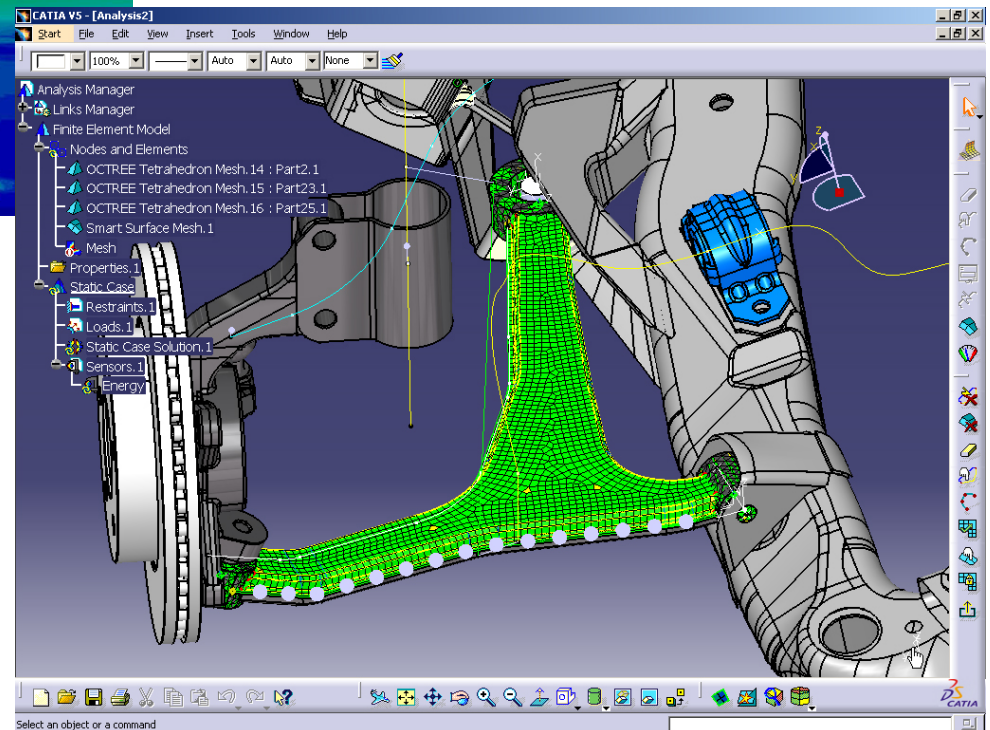


Finite element method – Domains of application



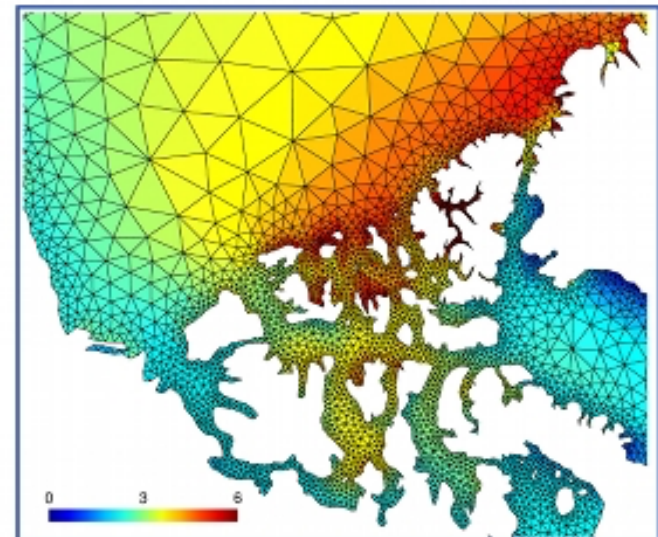
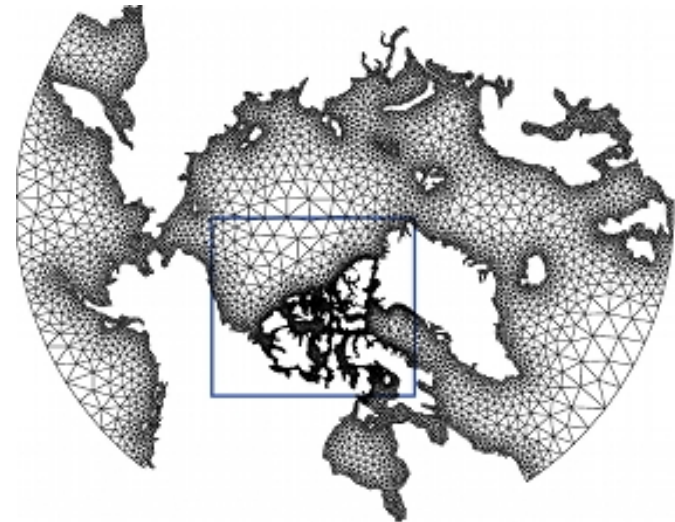
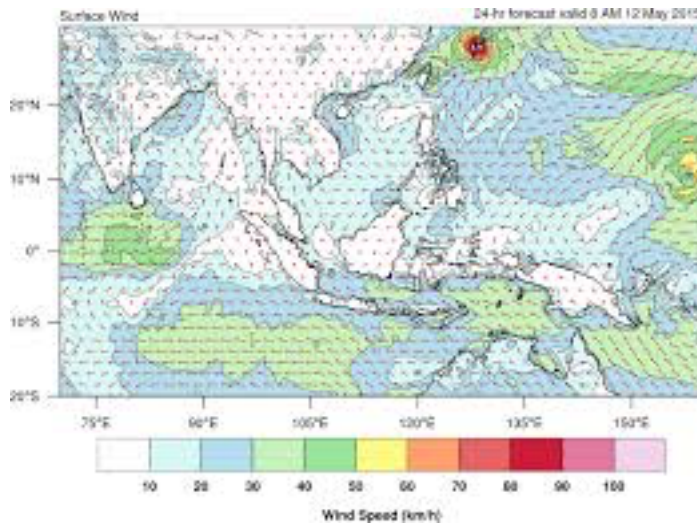
Structural analysis

Mechanical design



Finite element method – Domains of application

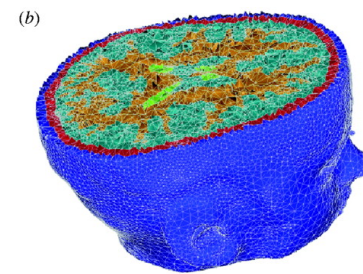
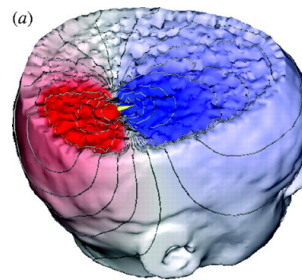
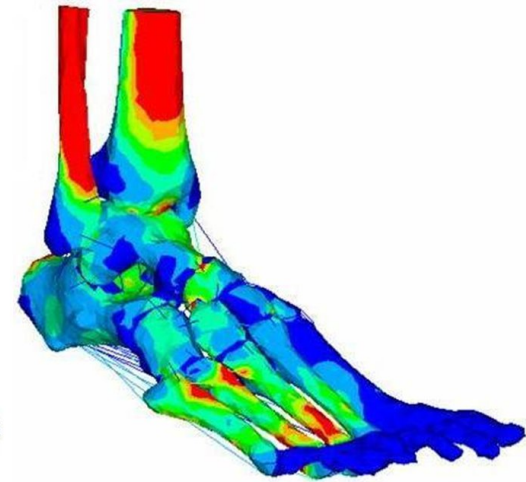
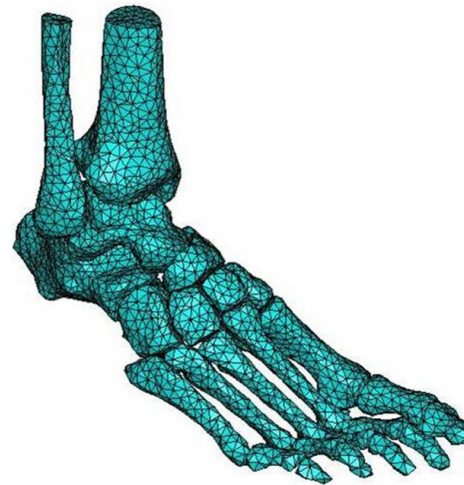
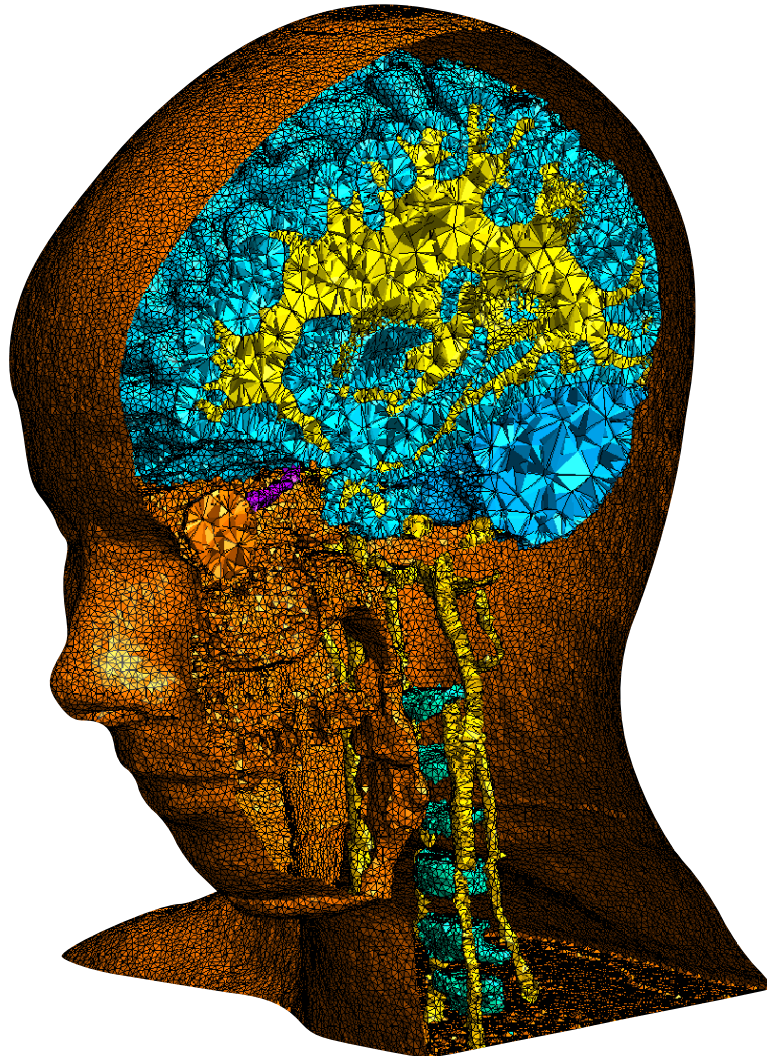
Weather forecasting



Geophysical modelling

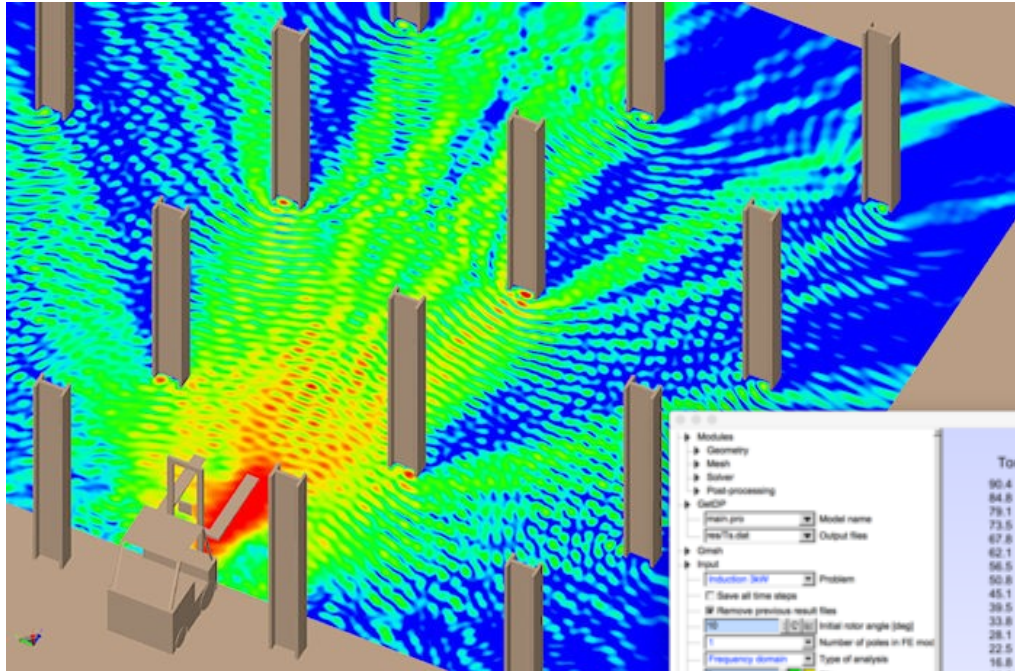
Finite element method – Domains of application

Biomedical engineering

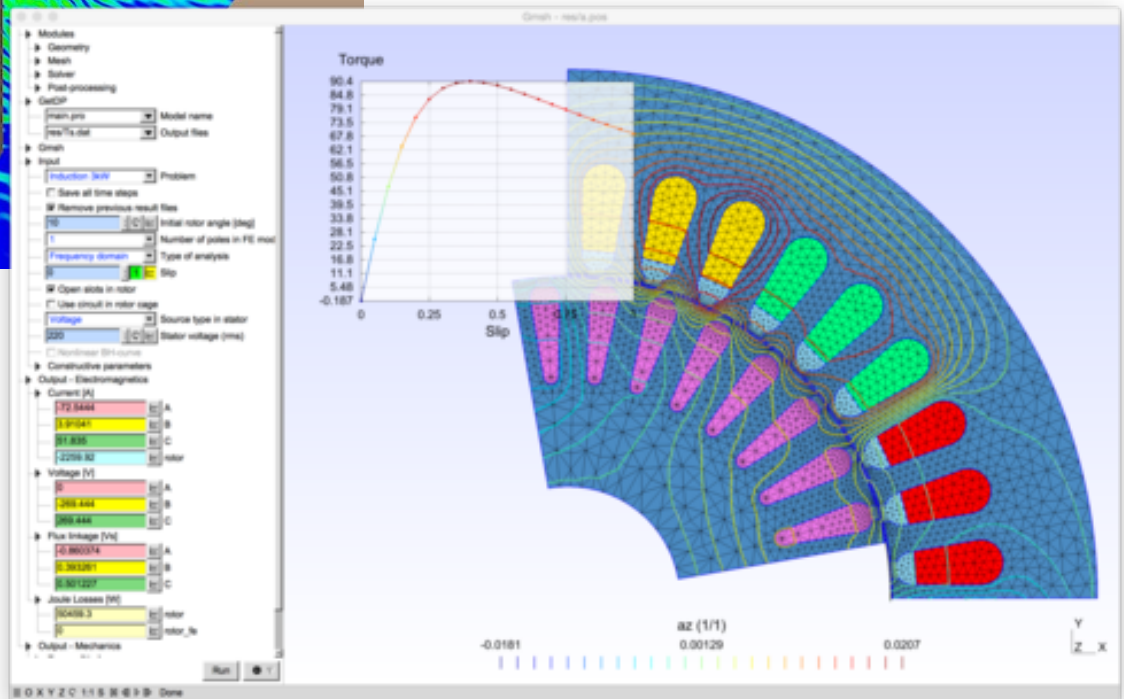


Health

Finite element method – Domains of application

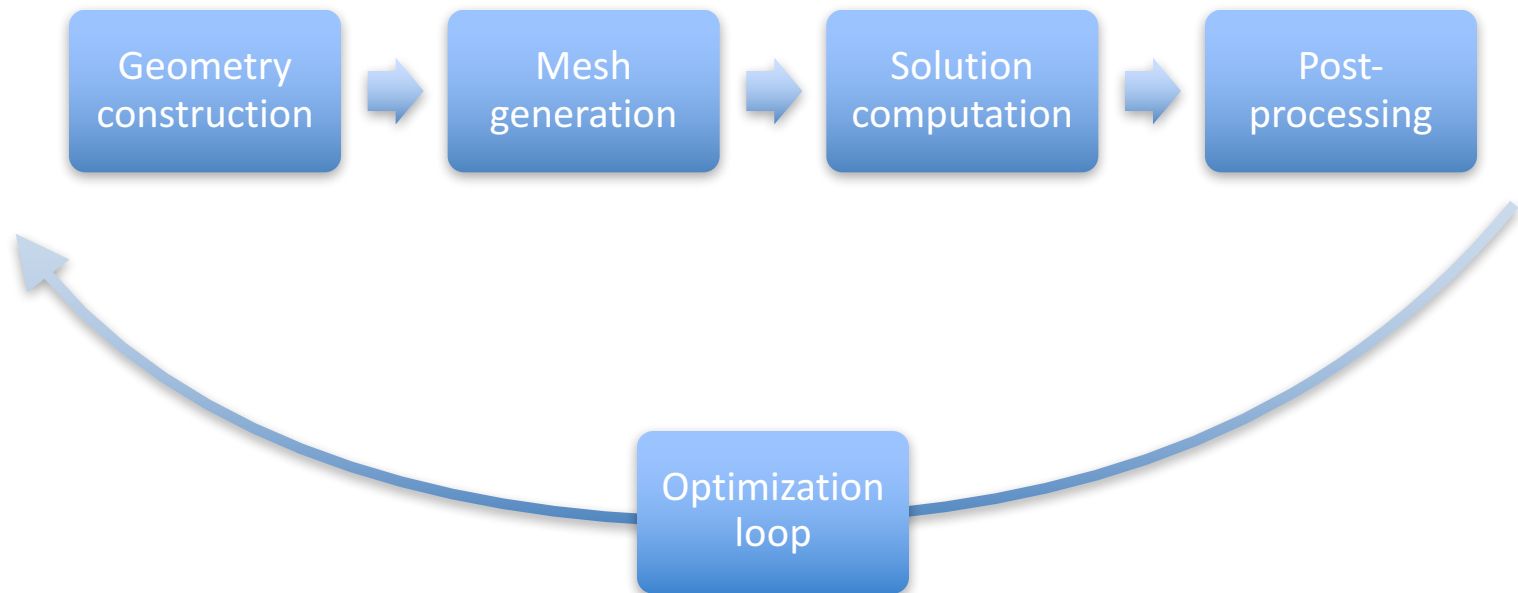


Electromagnetics



Energy

Steps of a finite element analysis



Steps of a finite element analysis

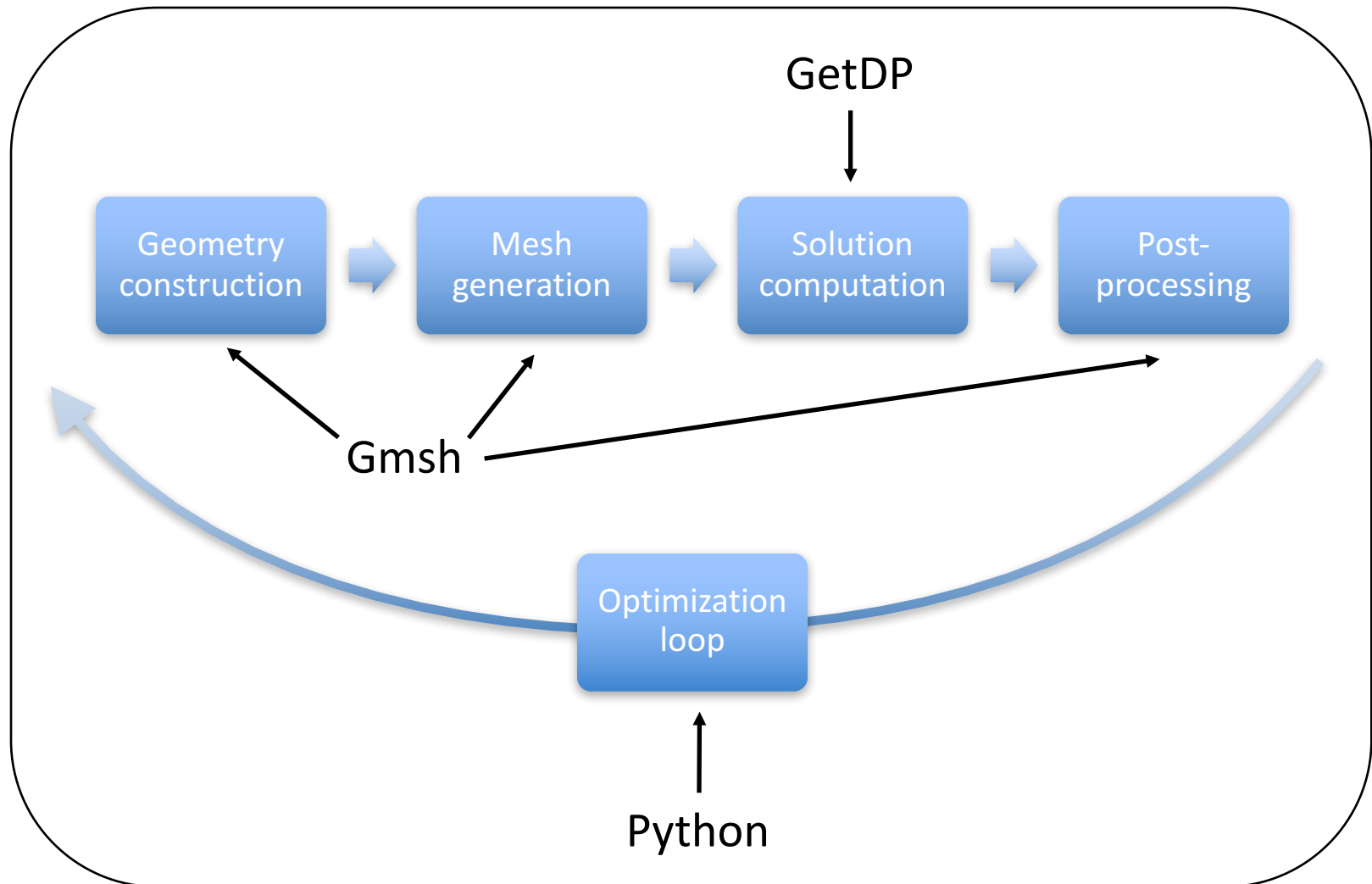
- Many commercial software packages are available, that perform all the steps:
 - ANSYS, Abaqus/CATIA/SIMULIA, Siemens NX/Nastran/SAMCEF, Altair, ADINA, COMSOL, LS-DYNA, ...
- Several open source packages are also available, but most only implement one step:
 - Code_Aster, deal.II, CalculiX, Elmer, FEniCS, FreeFem++, OpenFOAM, GetDP, Gmsh, Paraview, ...
- This week: introduction to finite element modelling with ONELAB (<http://onelab.info>)

ONELAB

- Open source, lightweight interface to finite element software
- The default ONELAB software bundle contains
 - the mesh generator Gmsh (<http://gmsh.info>)
 - the finite element solver GetDP (<http://getdp.info>)
- Many other codes (free or not) can be easily interfaced as well
- Download the software bundle now for Windows, Linux, Mac, Android or iOS from <http://onelab.info>

Steps of a finite element analysis with ONELAB

ONELAB



This week – Objective of the course

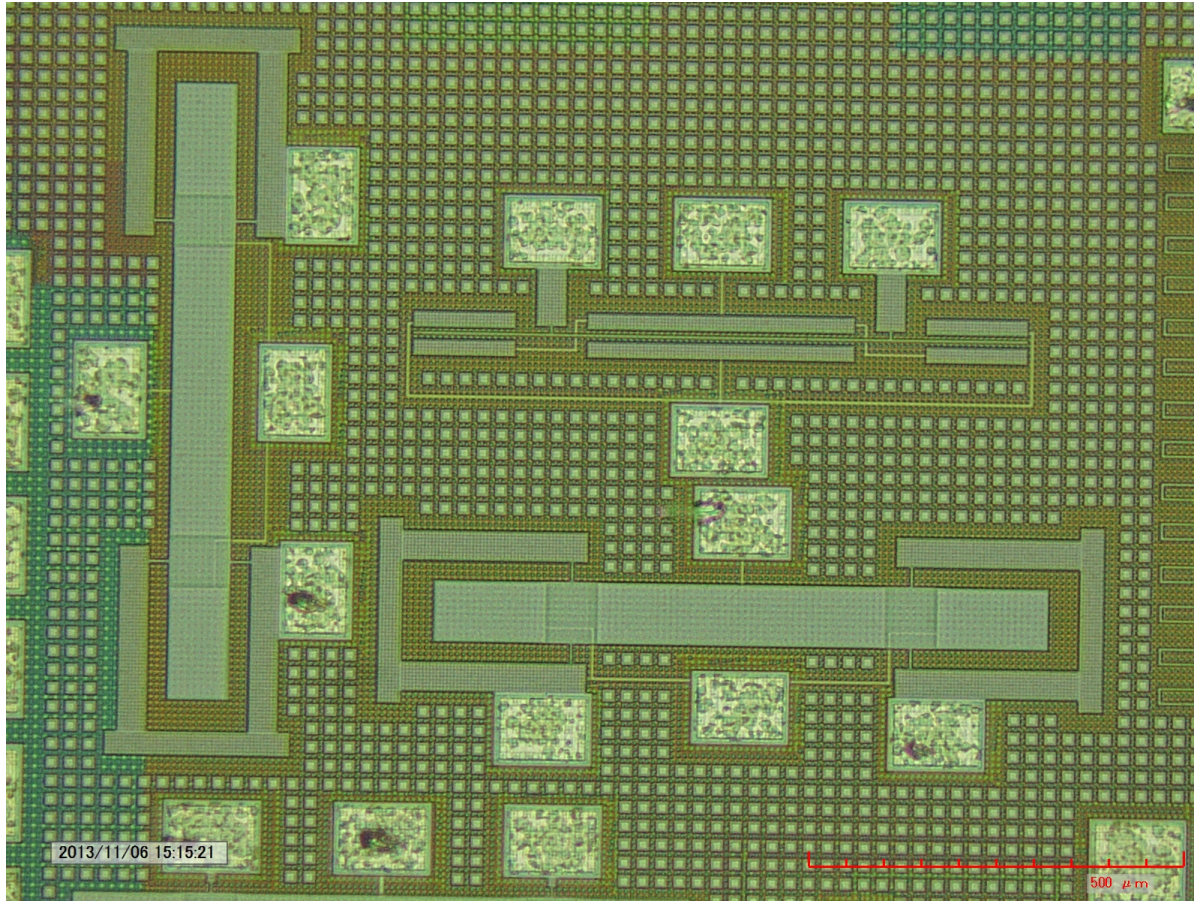
As an introduction to finite element modelling, we would like you to

- understand,
- model,
- simulate, and
- optimize

a practical device with ONELAB, and learn the basics of the finite element method along the way

The device we chose is a Micro Electro-Mechanical System (MEMS): a Xylophone Bar Magnetometer (XBM)

This week – Objective of the course



Courtesy
of IMEC

Xylophone Bar Magnetometer (XBM)
Micro Electro-Mechanical System (MEMS)

This week – Objective of the course

- When an alternating current flows through it, the magnetometer starts to vibrate if it is placed in a magnetic field; the amplitude of the vibration can be used to measure the magnetic field
- To increase sensitivity, the frequency of the current is tuned to excite a resonant mode of the xylophone-type structure
- Modelling should take into account the multi-physics coupling (electromagnetic, mechanical and thermal) and multiple time states (static, transient, harmonic, eigenmodes)

This week – Schedule

- Monday 10h00-12h00 & 14h00-17h00
 - Basics of the finite element method
- Tuesday 10h00-12h00 & 14h00-17h30
 - Electromagnetics and heat transfer
- Wednesday 10h00-12h00 & 14h00-17h00
 - Mechanics
- Thursday 10h00-12h00 & 14h00-17h30
 - Optimization
- Friday 10h00-13h00
 - Siemens LMS Samtech visit

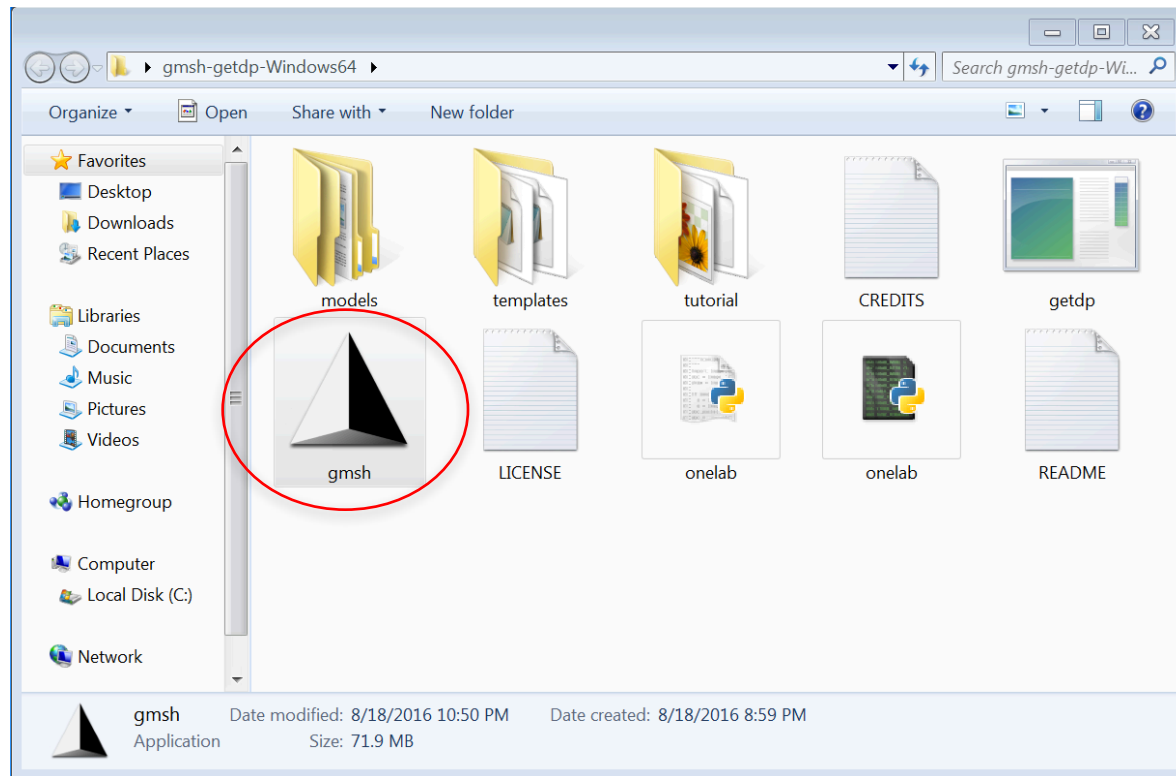
Geometry construction

Geometry construction with Gmsh

- Gmsh uses a Boundary REPresentation (BREP) approach to describe geometrical models:
 - volumes are bounded by surfaces
 - surfaces are bounded by curves
 - curves are bounded by points
- Models can be
 - built interactively using the graphical user interface (which logs all commands in a “.geo” file)
 - scripted directly in “.geo” files
 - imported from external CAD packages

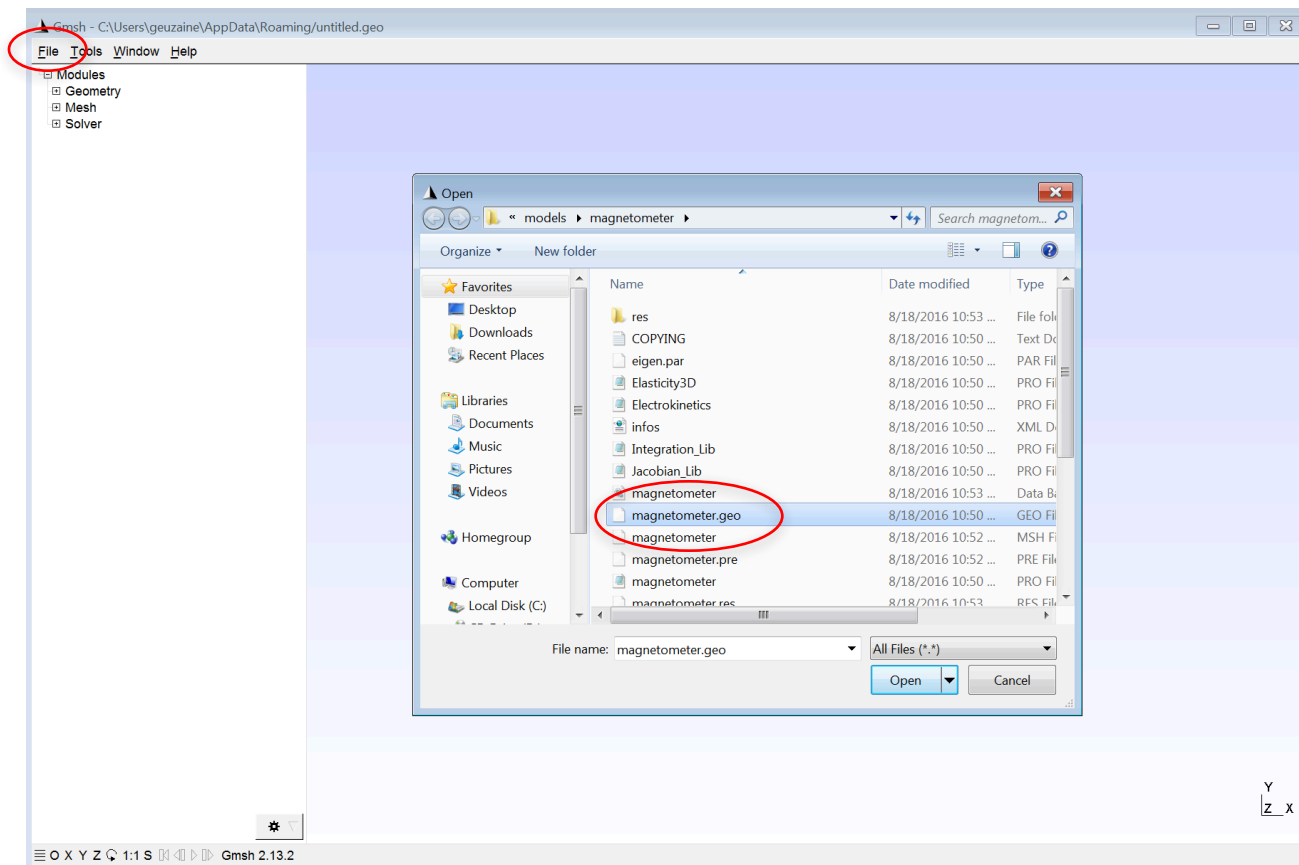
Geometry construction with Gmsh

- To launch Gmsh, double-click on the  icon



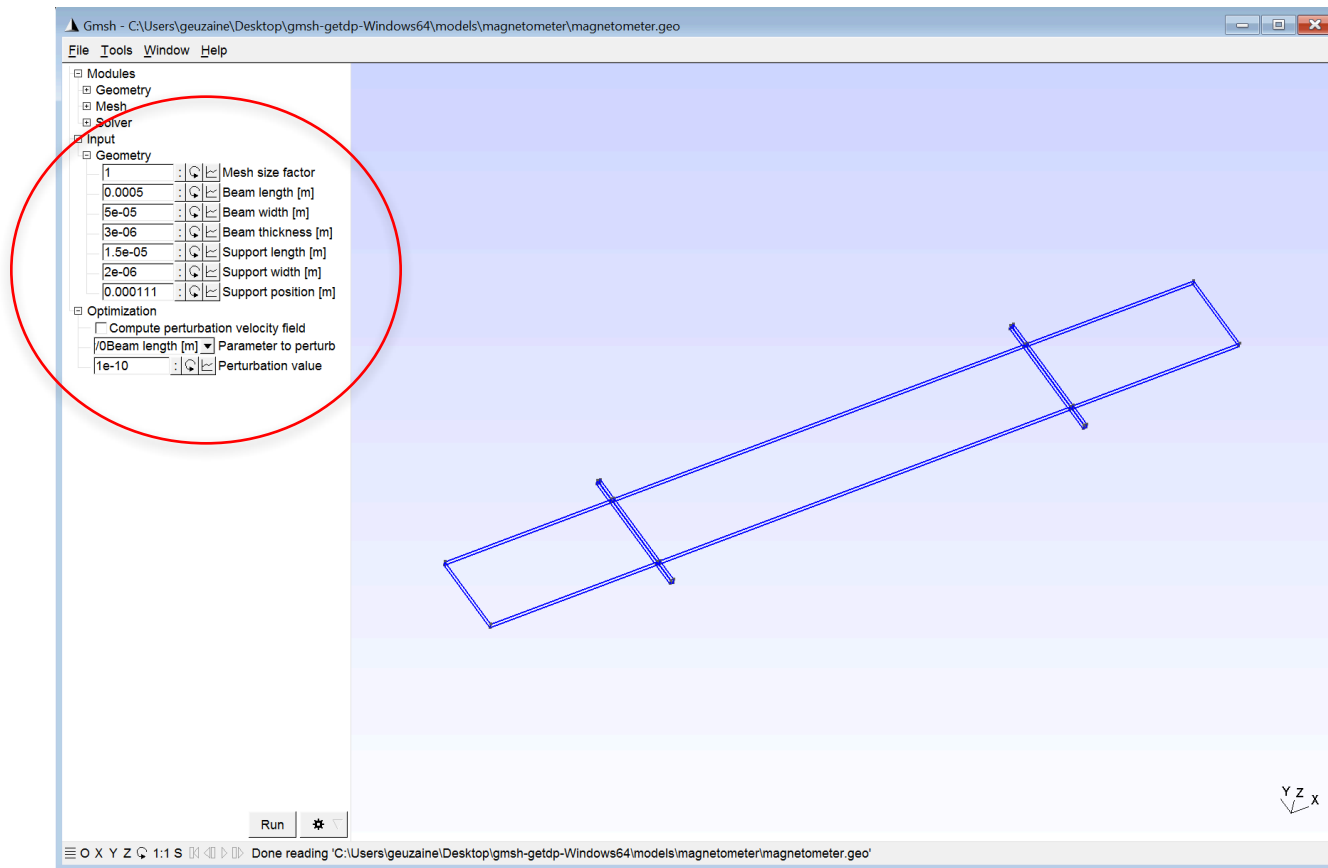
Geometry construction with Gmsh

- To open an existing geometry, use “File->Open” and select “models/magnetometer/magnetometer.geo”



Geometry construction with Gmsh

- Model parameters are on the left, below the “Geometry”, “Mesh” and “Solver” modules

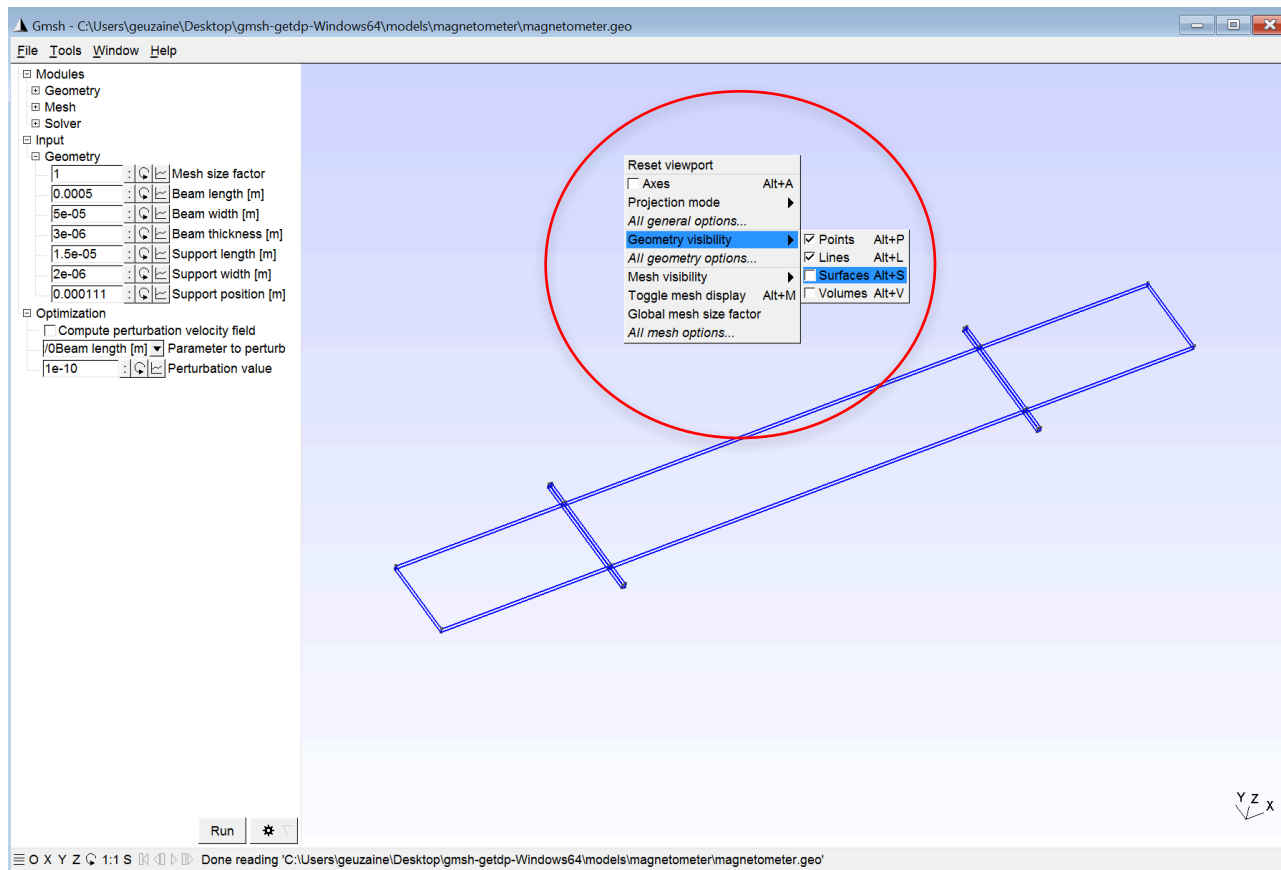


Geometry construction with Gmsh

- To change a parameter, simply enter a value or click and slide the mouse across an input field
- In the graphic window:
 - The left mouse button rotates the model and selects objects
 - The middle mouse button (or the mouse wheel, or a two finger zoom gesture) zooms in or out
 - The right mouse button pans (translates)
 - For two- or single-button manipulation:
 - Middle click = Shift+left click
 - Right click = Alt+left click

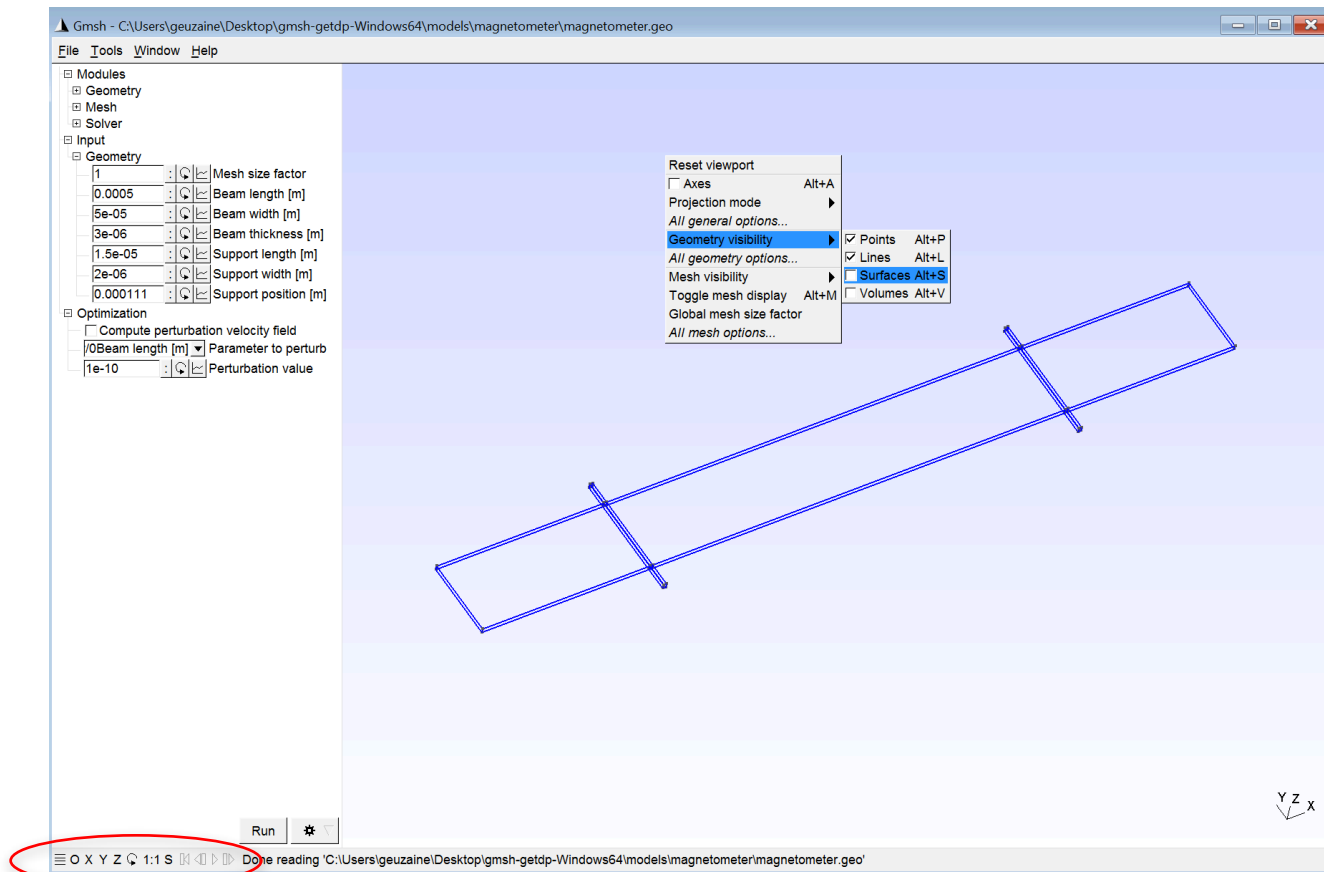
Geometry construction with Gmsh

- Common options can be accessed by double-clicking in the graphic window



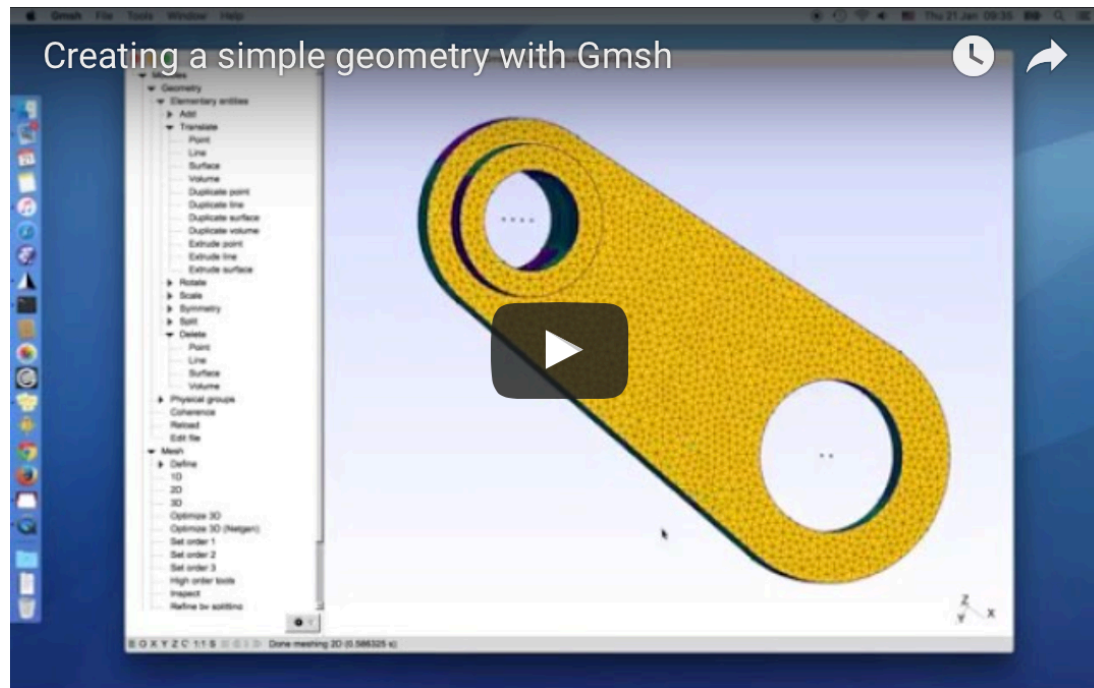
Geometry construction with Gmsh

- Preset viewpoints and rotations can also be accessed in the status bar



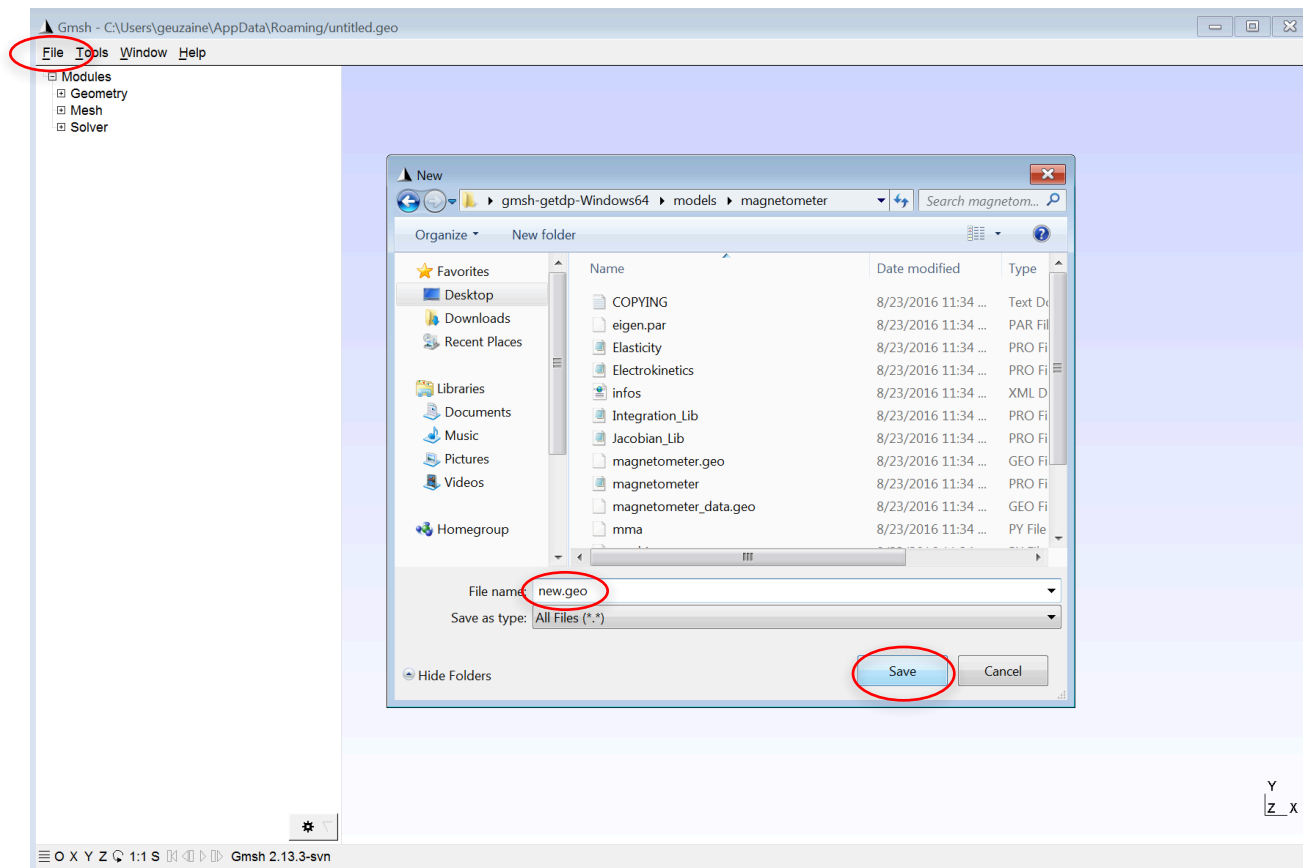
Geometry construction with Gmsh

- A video tutorial showing how to create geometries interactively is available here:
<http://youtu.be/nkuawZkiu1w>



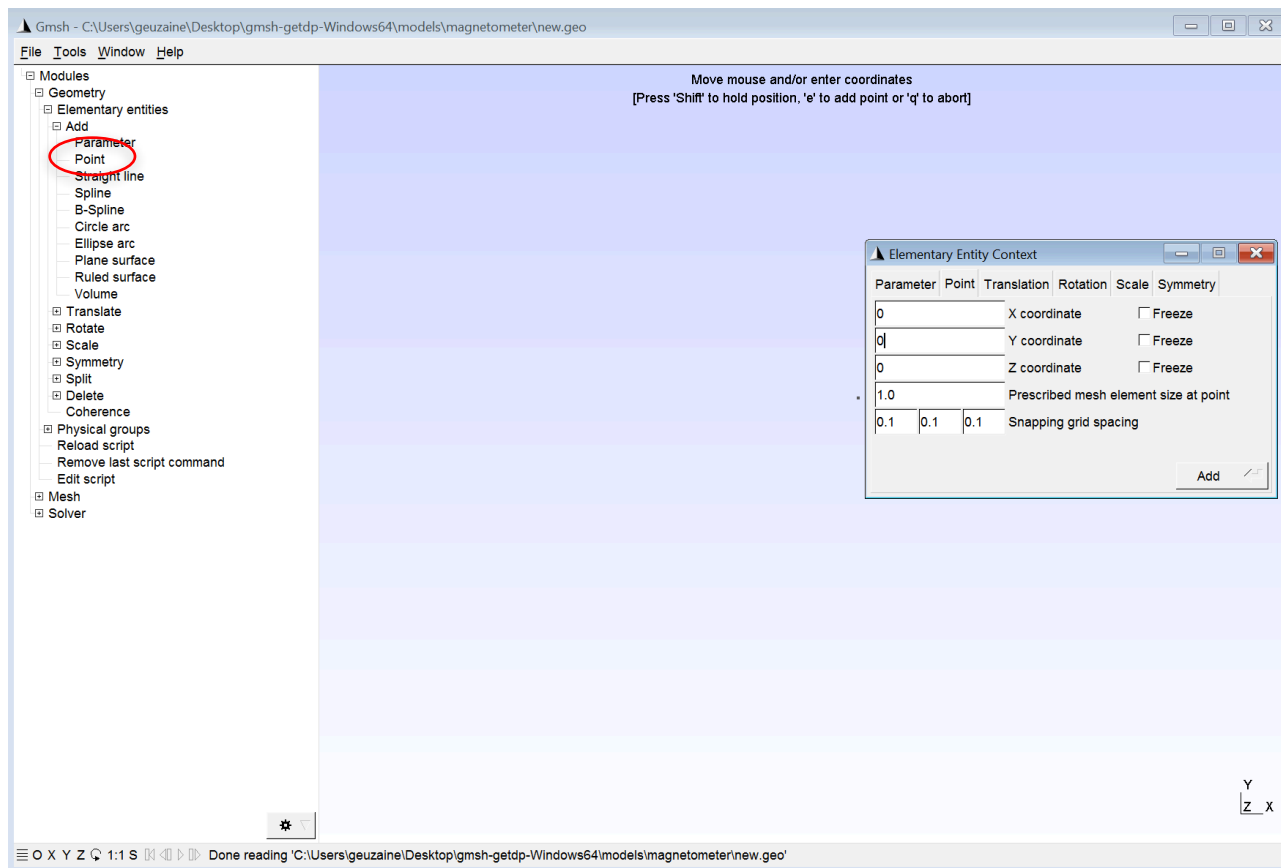
Geometry construction with Gmsh

- Let us create a magnetometer model from scratch: select “File->New” and choose a name (“new.geo”)



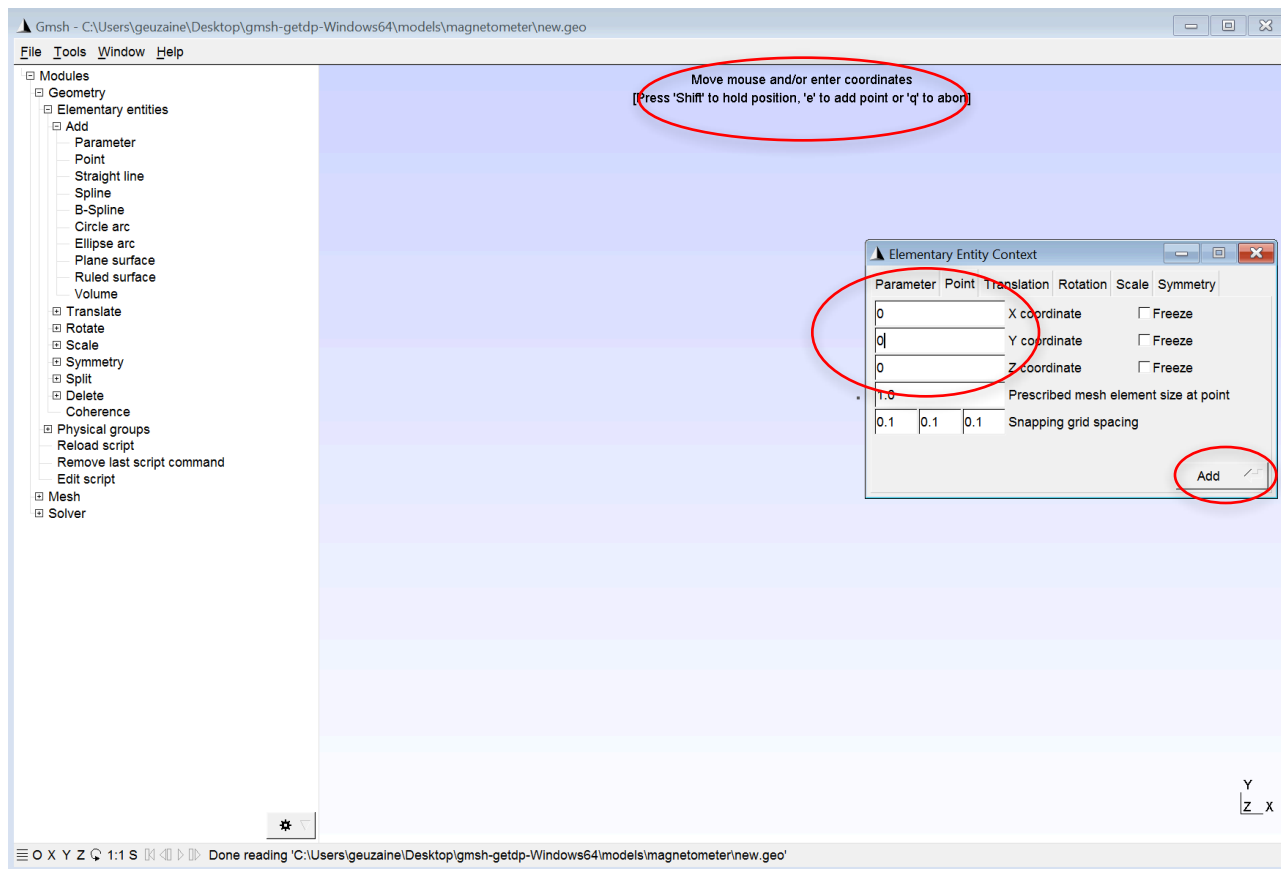
Geometry construction with Gmsh

- Select “Modules->Geometry->Elementary entities->Add->Point” to create a point



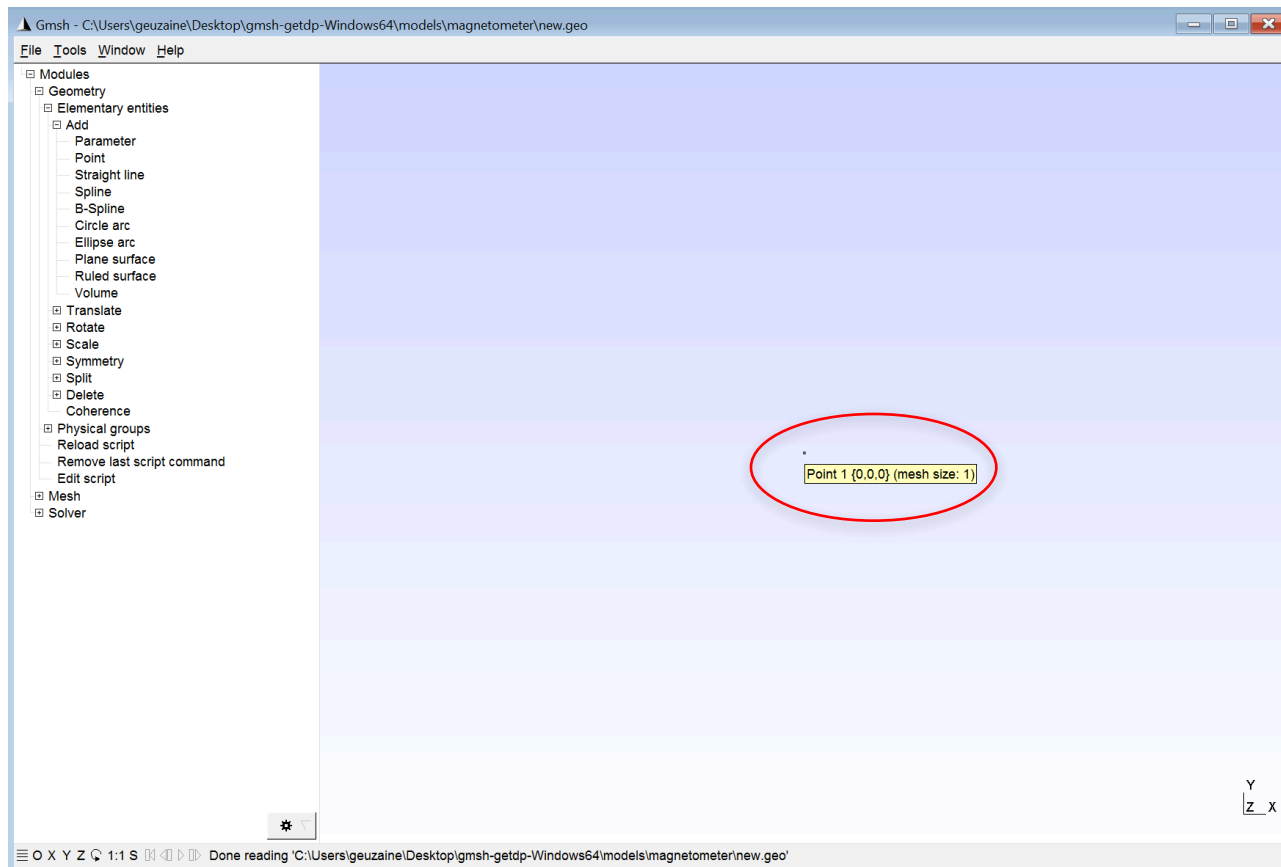
Geometry construction with Gmsh

- Enter coordinates (0,0,0) and press “Add” (or choose the position with the mouse and press “e”)



Geometry construction with Gmsh

- Press “q” when you are done; placing the mouse on the point will show its attributes



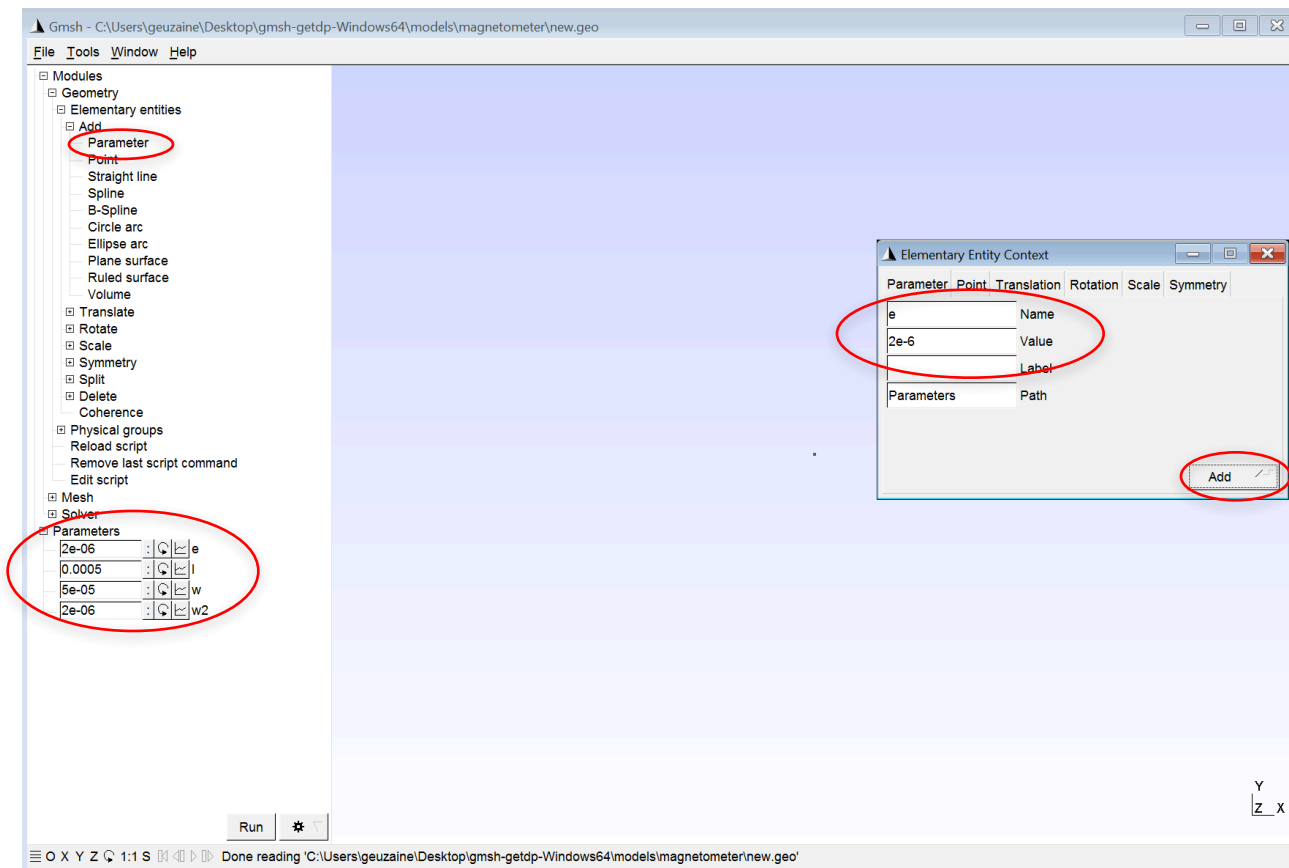
Geometry construction with Gmsh

If anything goes wrong, don't worry:

- Select “Modules->Geometry->Remove last script command”
 - This removes the last command that was added to the script “new.geo” and reloads the updated script
- Or select “Modules->Geometry->Edit script”
 - This opens the script in a text editor, where you can remove the erroneous commands or fine-tune the commands that were inserted
 - Then save the file, close the text editor and select Select “Modules->Geometry->Reload”

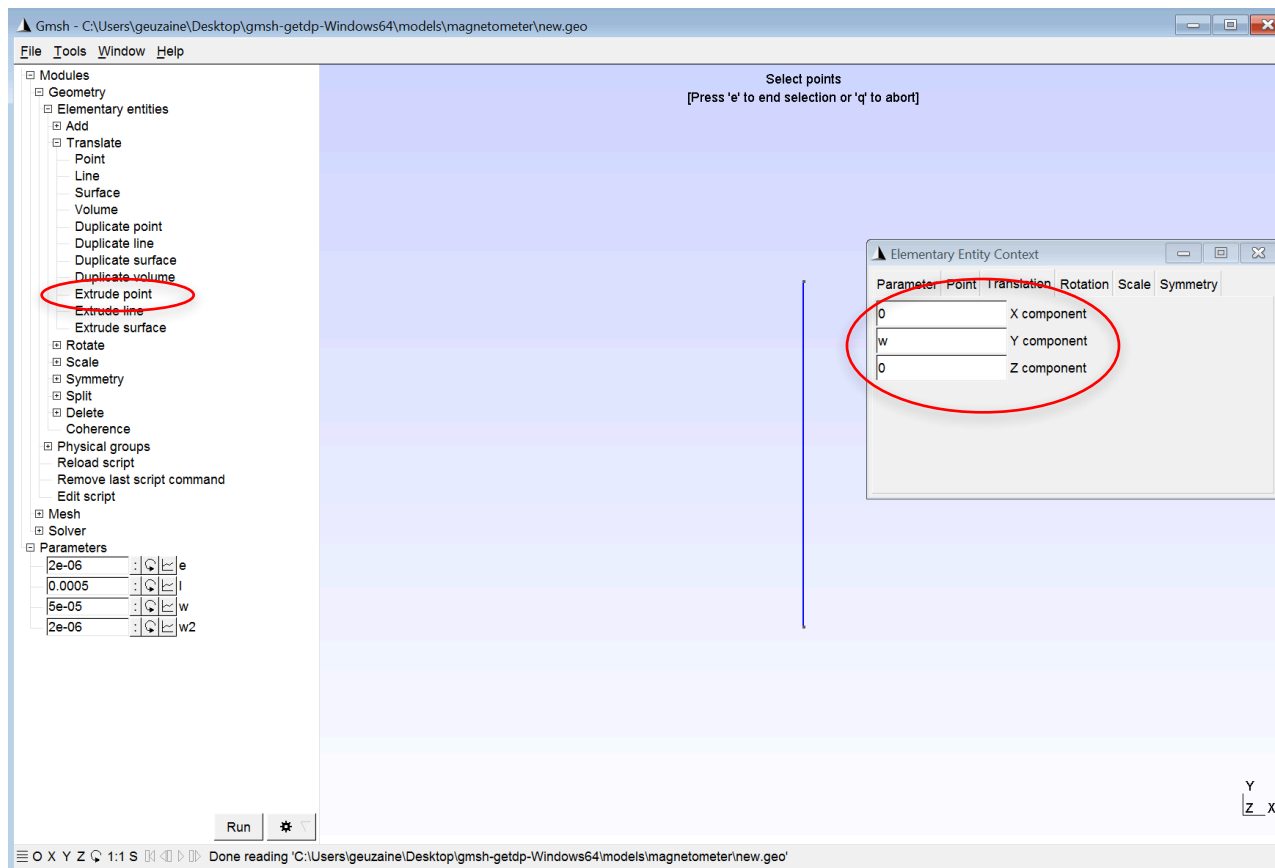
Geometry construction with Gmsh

- Define some variables by selecting “Add->Parameter”:
 $w=50e-6$, $w2=2e-6$, $l=500e-6$, $e=2e-6$



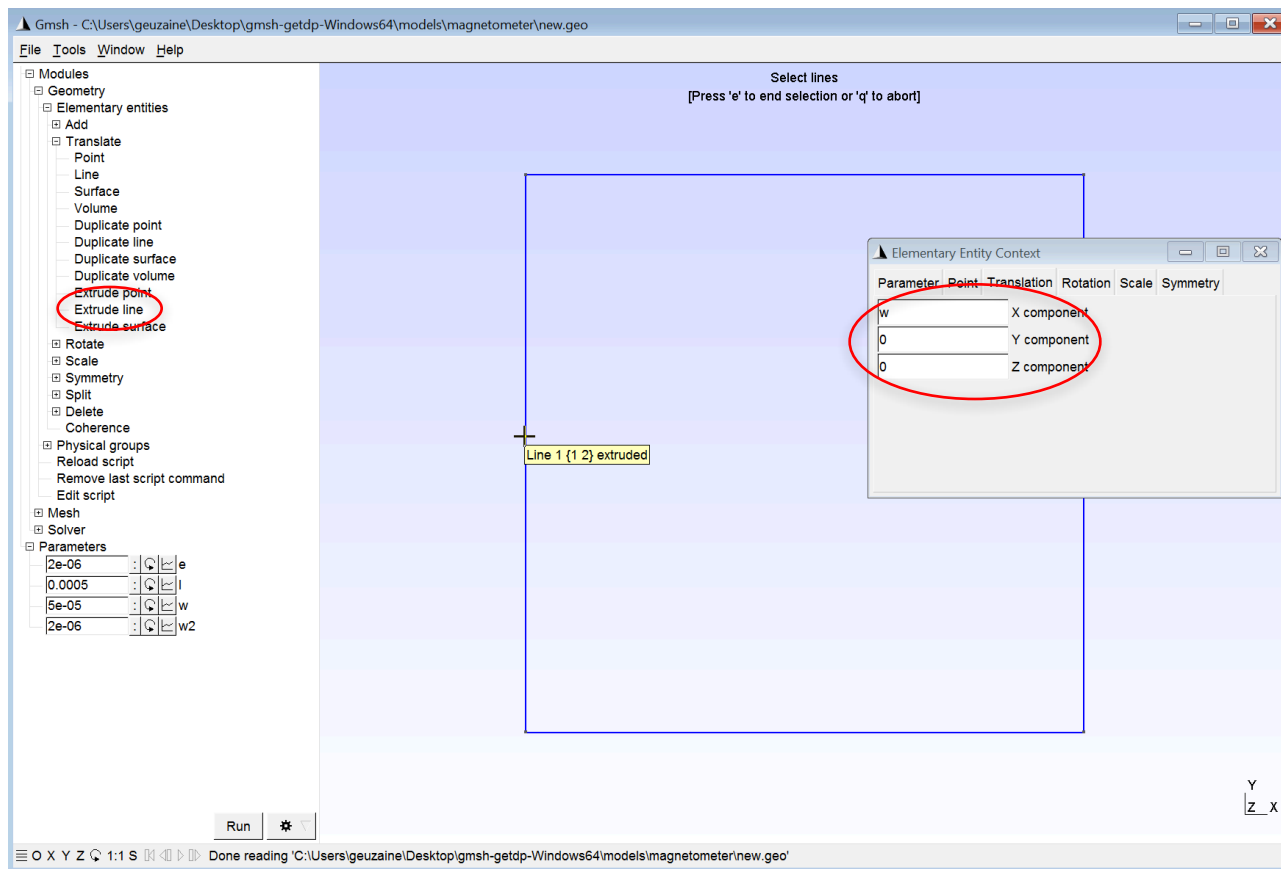
Geometry construction with Gmsh

- Create a line of length “w” by extruding the point with “Translate->Extrude point”; then press “e”



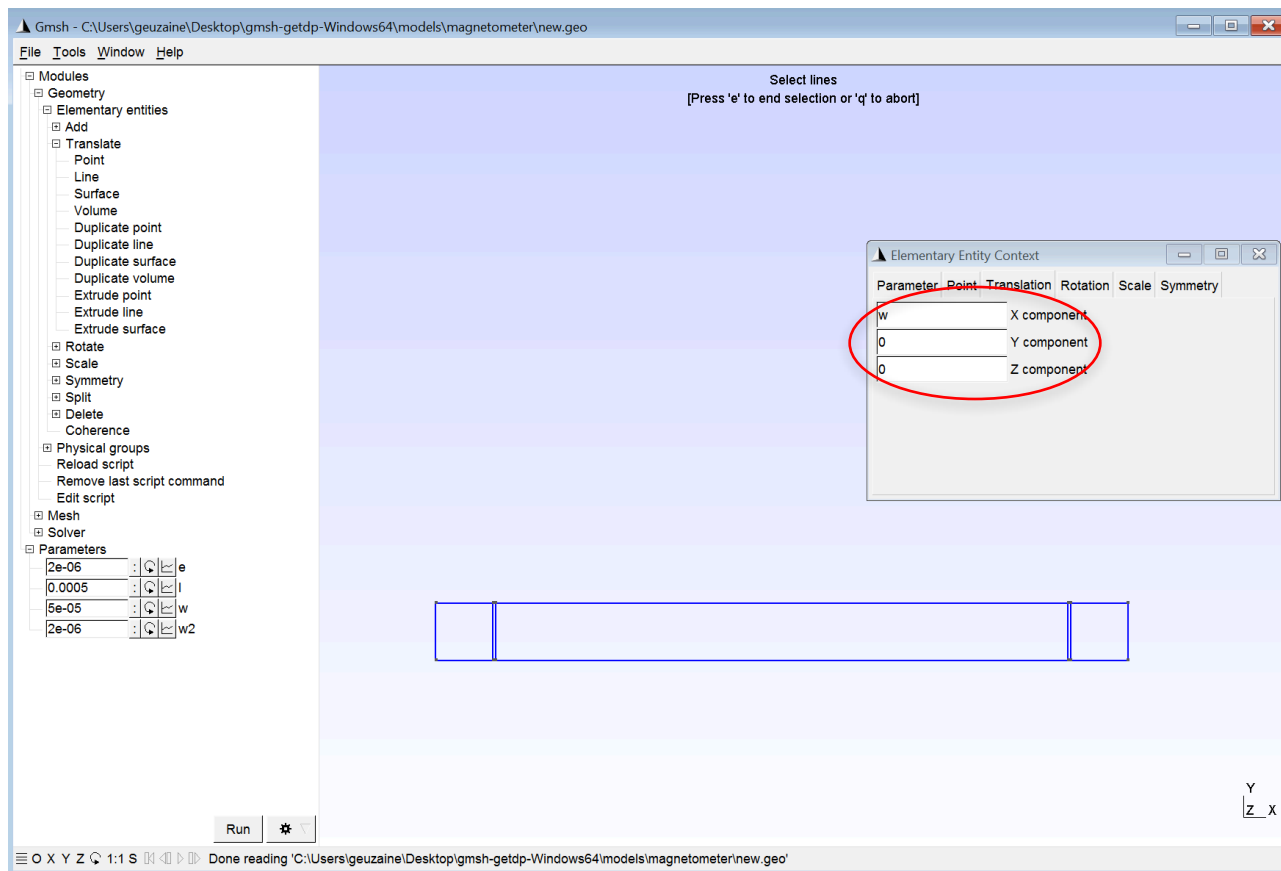
Geometry construction with Gmsh

- This is the left side of the magnetometer; Create a surface of width “w” with “Translate->Extrude line”



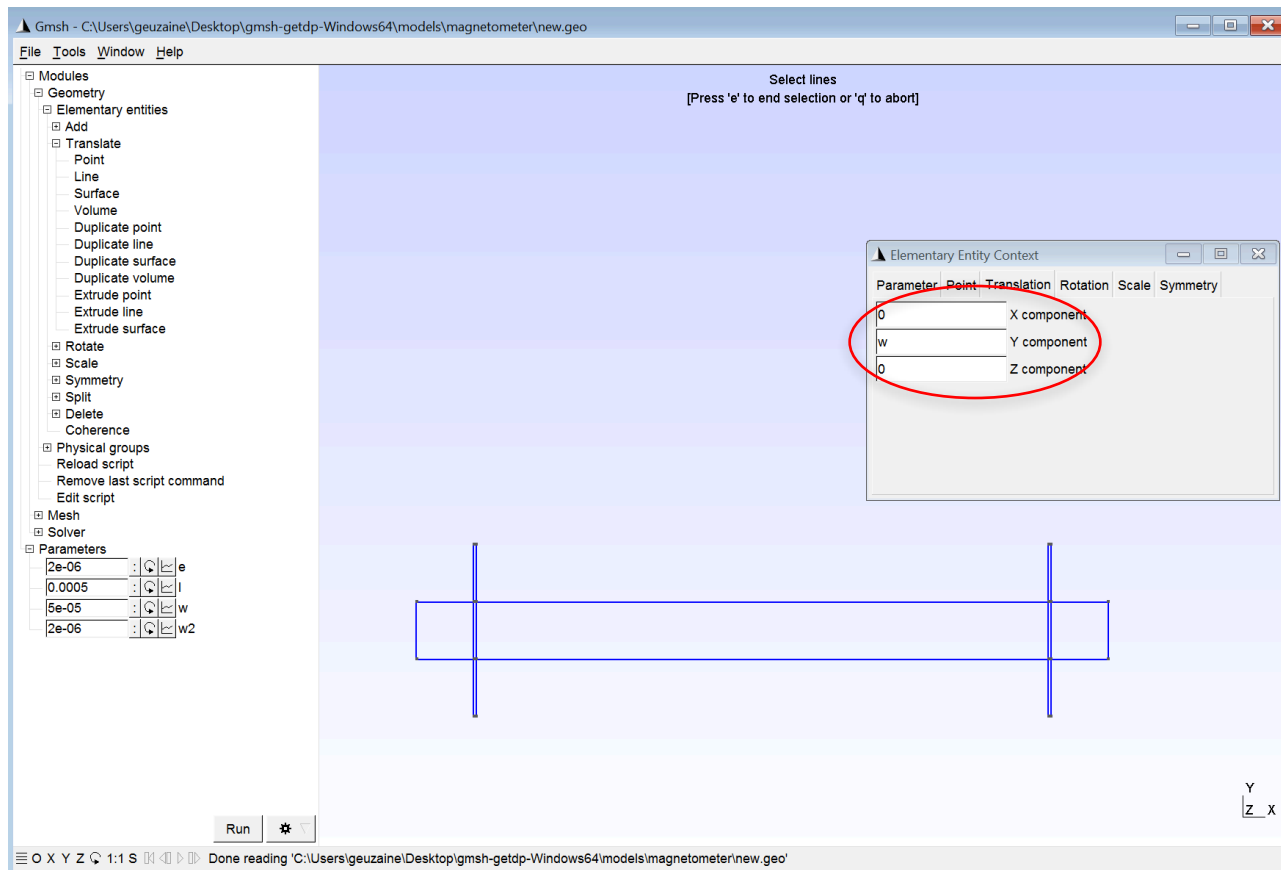
Geometry construction with Gmsh

- Proceed along axis “X” with successive line extrusions of length “w2”, “l”, “w2” and “w”



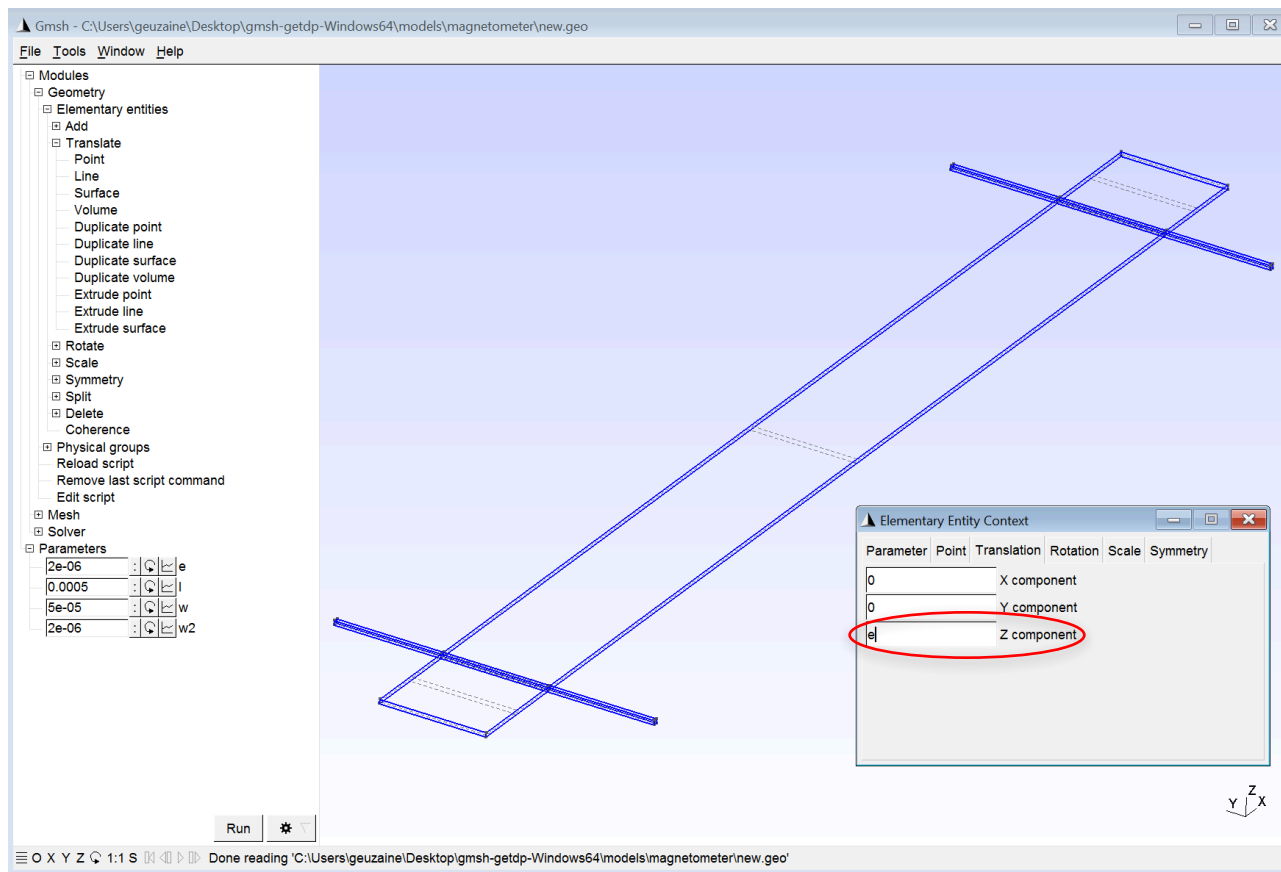
Geometry construction with Gmsh

- Create the “legs” by extruding by “w” and “-w” along axis “Y”



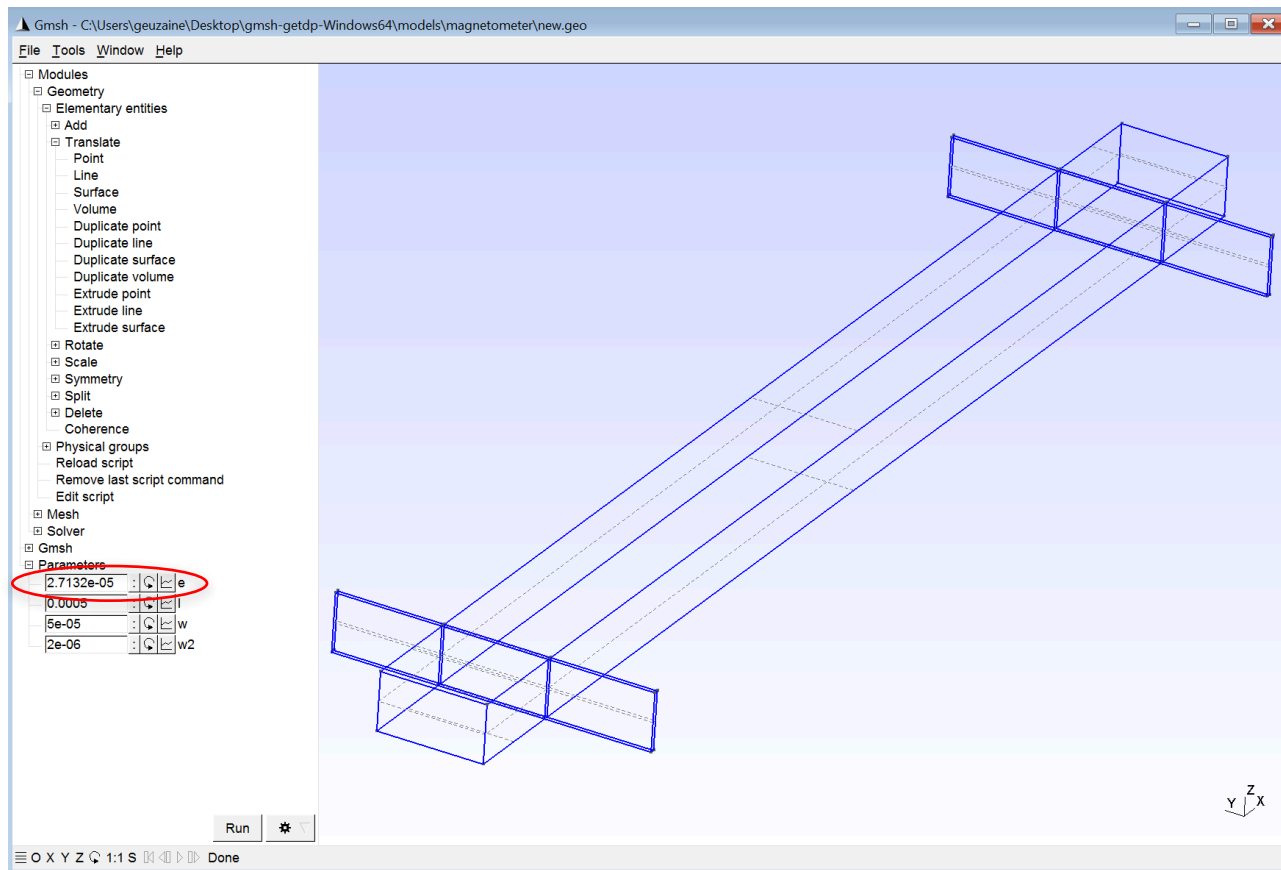
Geometry construction with Gmsh

- Create the volume by extruding all the surfaces (“Ctrl+left click” to select a region) along the “Z” axis



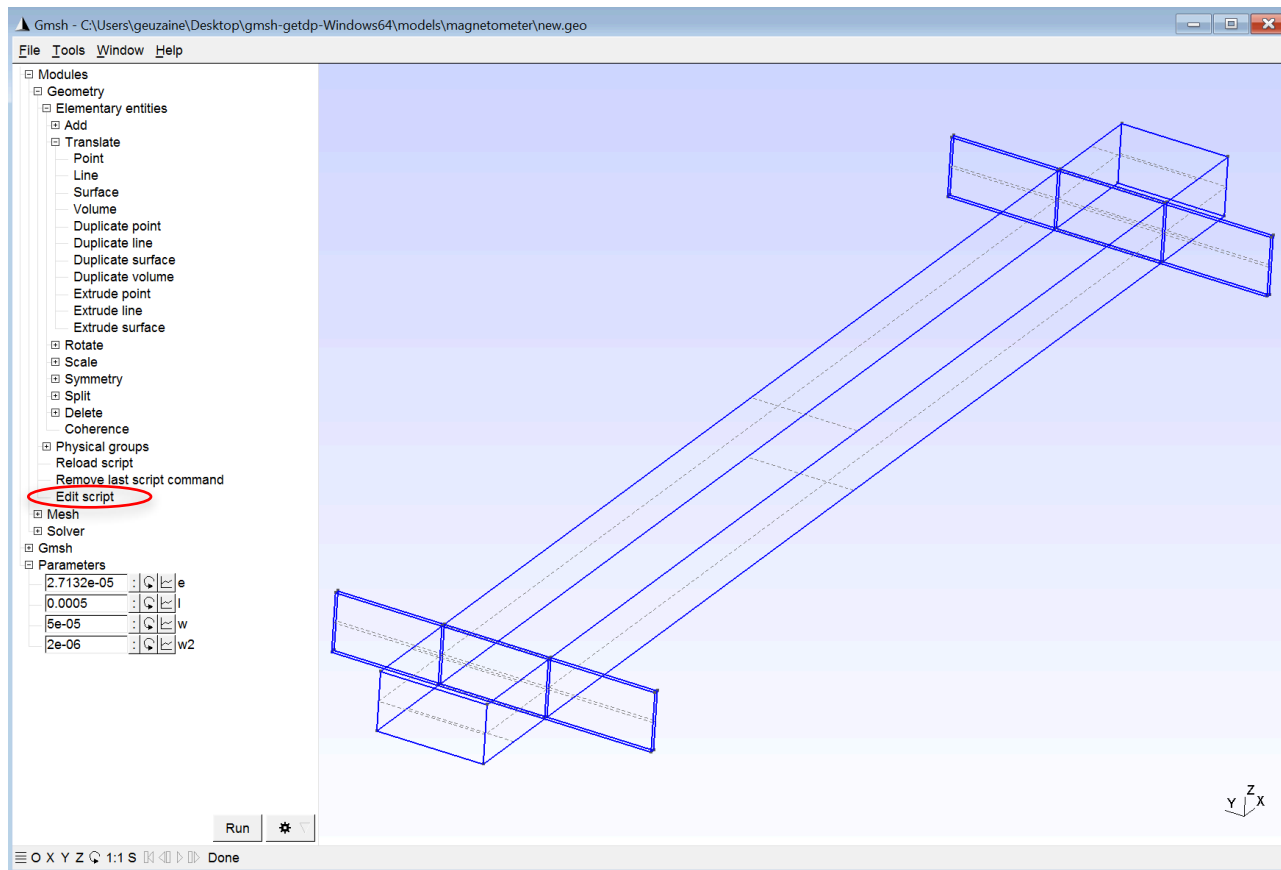
Geometry construction with Gmsh

- Note that the parameters are “live”: you can change them in the left menu and see the result



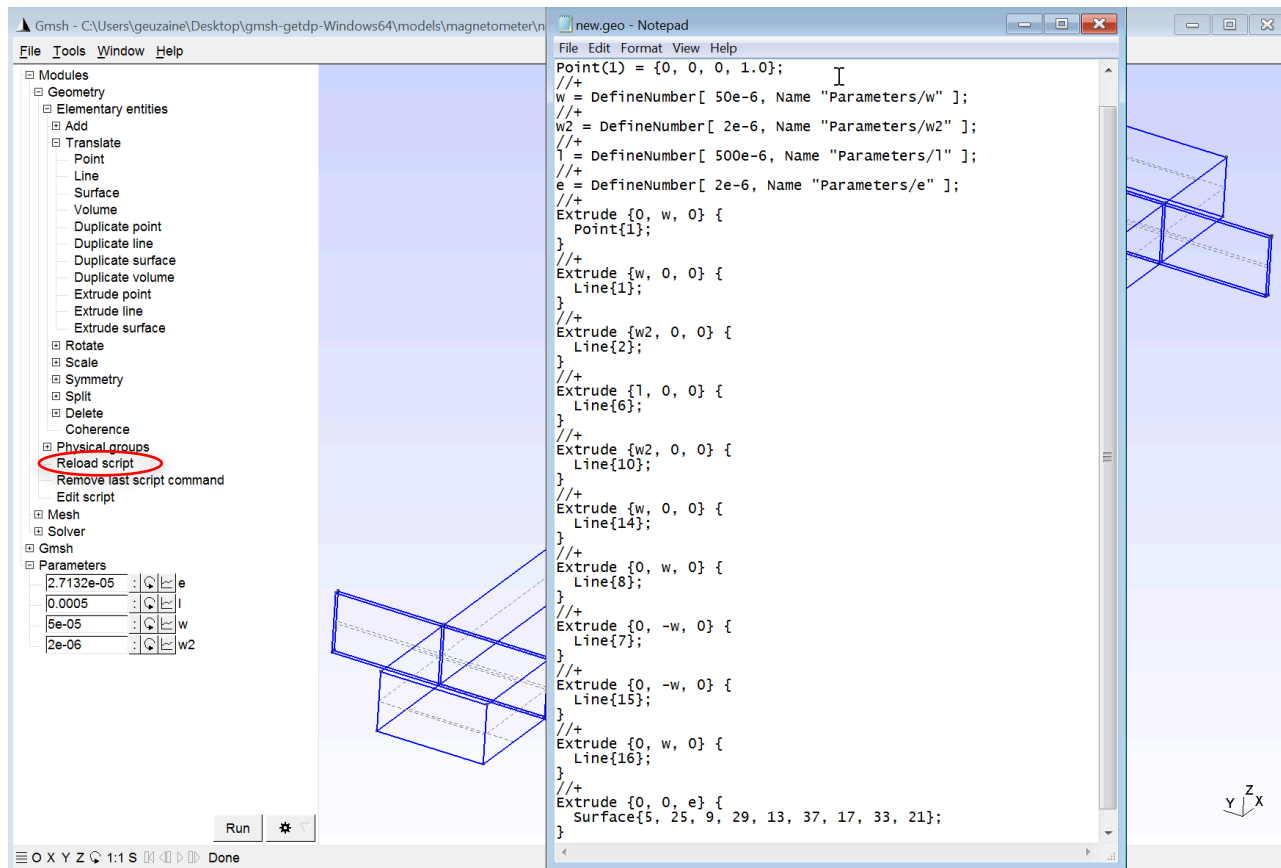
Geometry construction with Gmsh

- Select “Modules->Geometry->Edit script” to see the transcription of our work so far



Geometry construction with Gmsh

- You can edit this file, save it and press “Modules->Geometry->Reload script” to reload it



Geometry construction with Gmsh

- The main commands we have used are
 - `Point(1) = {0, 0, 0, 1.0};`
 - This creates a point with identification number “1”, at Cartesian coordinates “(0,0,0)” and with a target mesh size of “1.0” (the size of the finite elements that will be created close to this point)
 - `w = DefineNumber[50e-6, Name "Parameters/w"];`
 - This creates a variable “w” with default value “50e-6” and ONELAB name “Parameters/w”. ONELAB names are provided as paths, which correspond to the tree hierarchy in the graphical user interface.

Geometry construction with Gmsh

- Extrude {0, w, 0} { Point{1}; }
 - This extrudes the point “1” and creates a line of length “w” in along the “Y” axis
- Extrude {w2, 0, 0} { Line{2}; }
 - This extrudes the line “2” by “w2” along the “X” axis
- Extrude {0, 0, e} { Surface{5, 25, 9, 29, ...};}
 - This extrudes the listed surfaces along the “Z” axis to create a volume

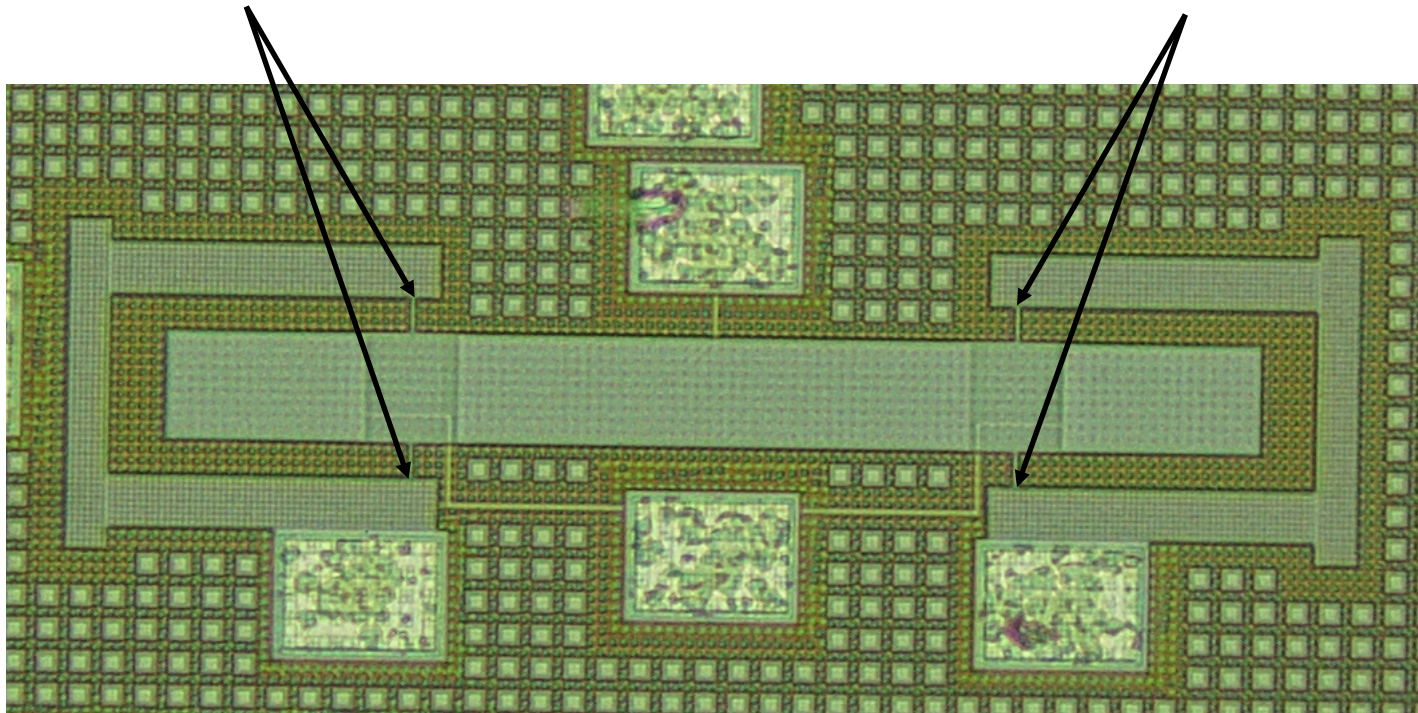
Geometry construction with Gmsh

- The geometry is almost ready to be meshed and used for computing a solution: we just need to group elements in “physical groups”
- We will need one physical group for each region that must be distinguished in the solution process: for example, for each material and for each boundary condition
- For the magnetometer, we define a single “physical volume” (single material) grouping all the elementary volumes and two “physical surfaces” – one for each electrode, where the structure is also clamped

Geometry construction with Gmsh

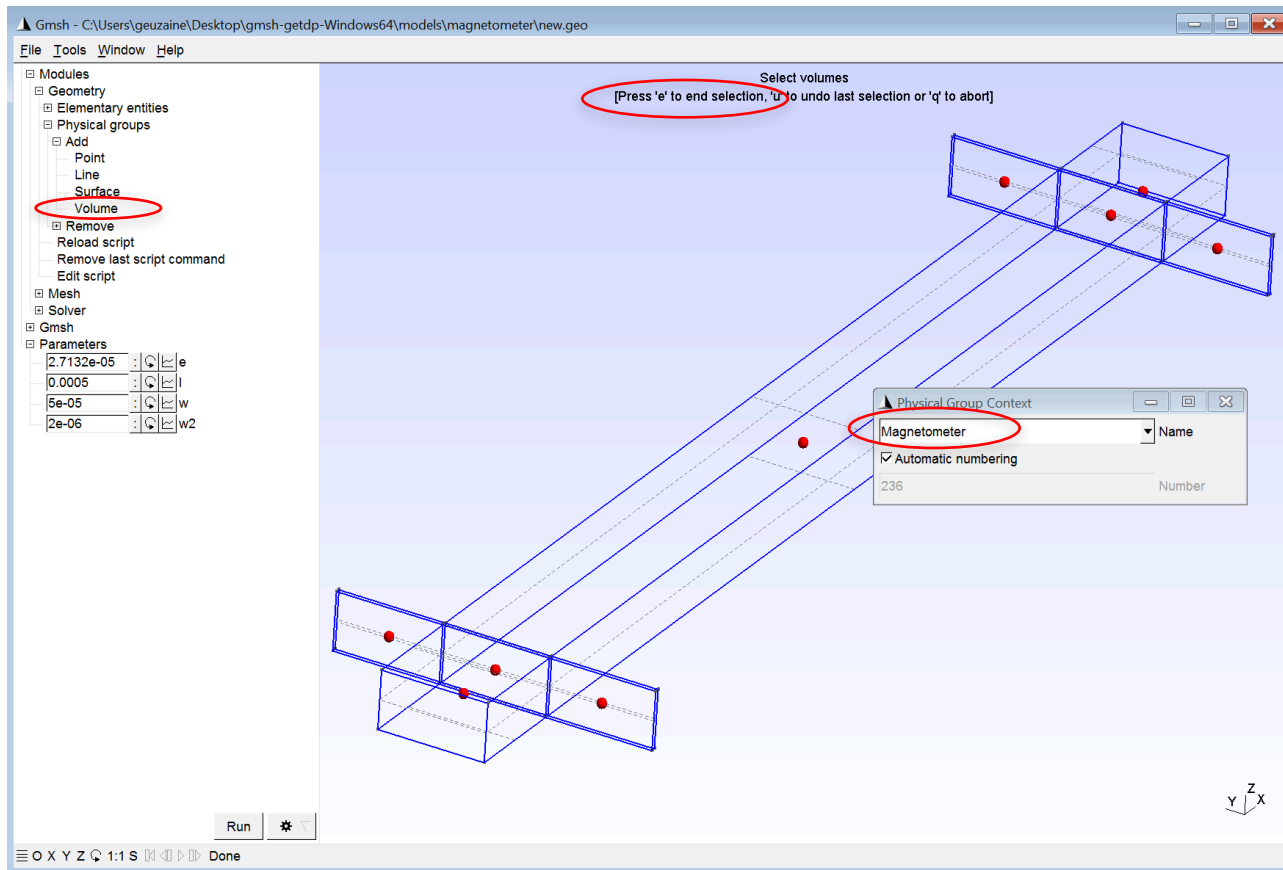
Left electrodes (clamped)

Right electrodes (clamped)



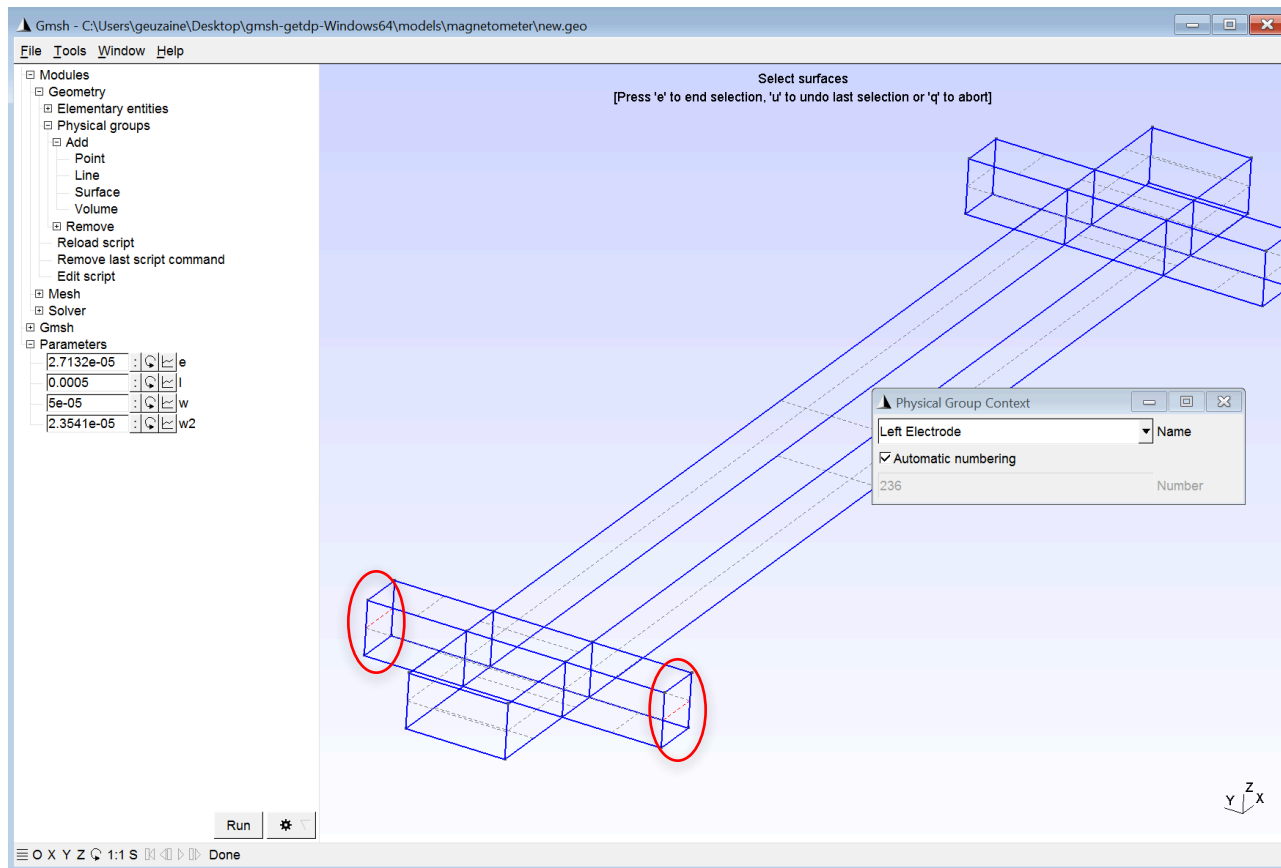
Geometry construction with Gmsh

- Click on “Physical groups->Add->Volume”, select all volumes, choose a name and press “e”



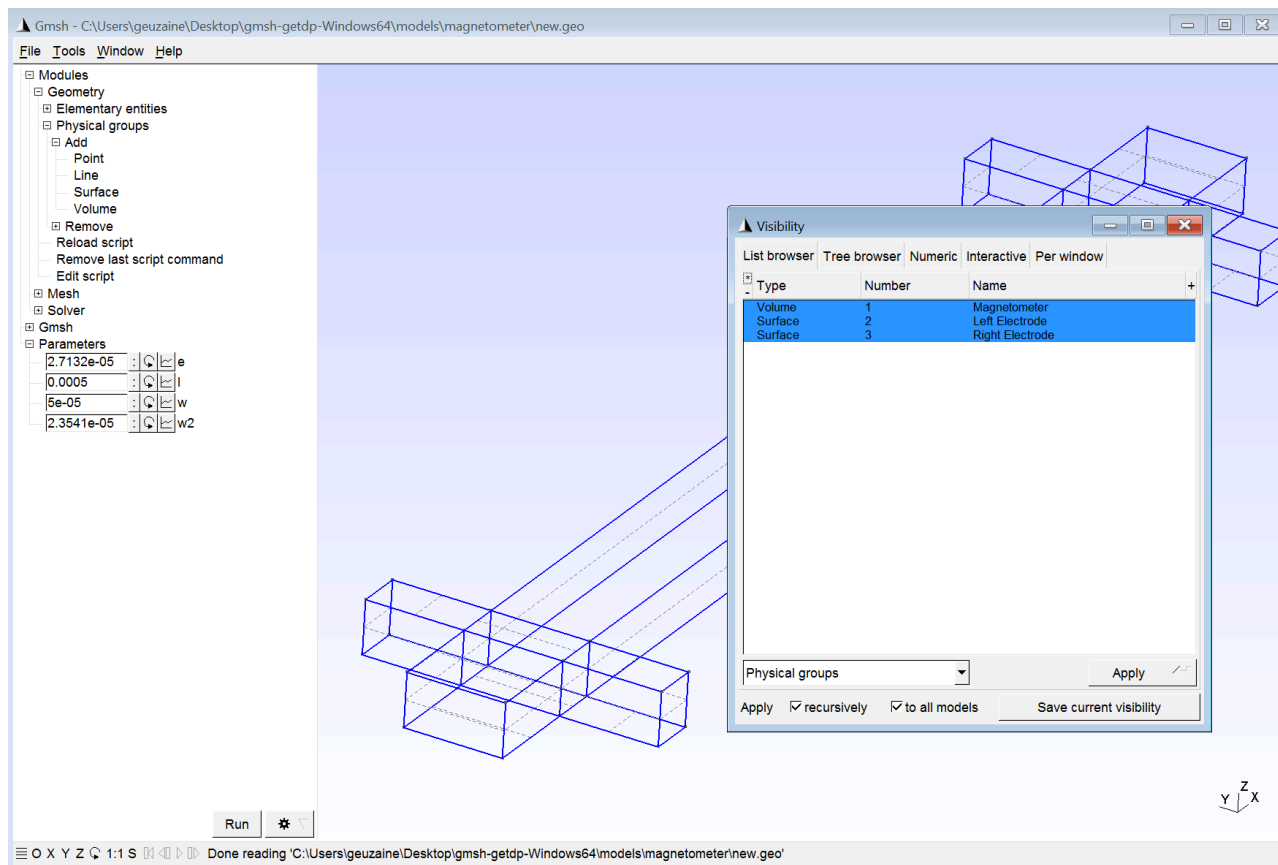
Geometry construction with Gmsh

- Click on “Physical groups->Add->Surface”, select the electrodes, choose names and press “e”



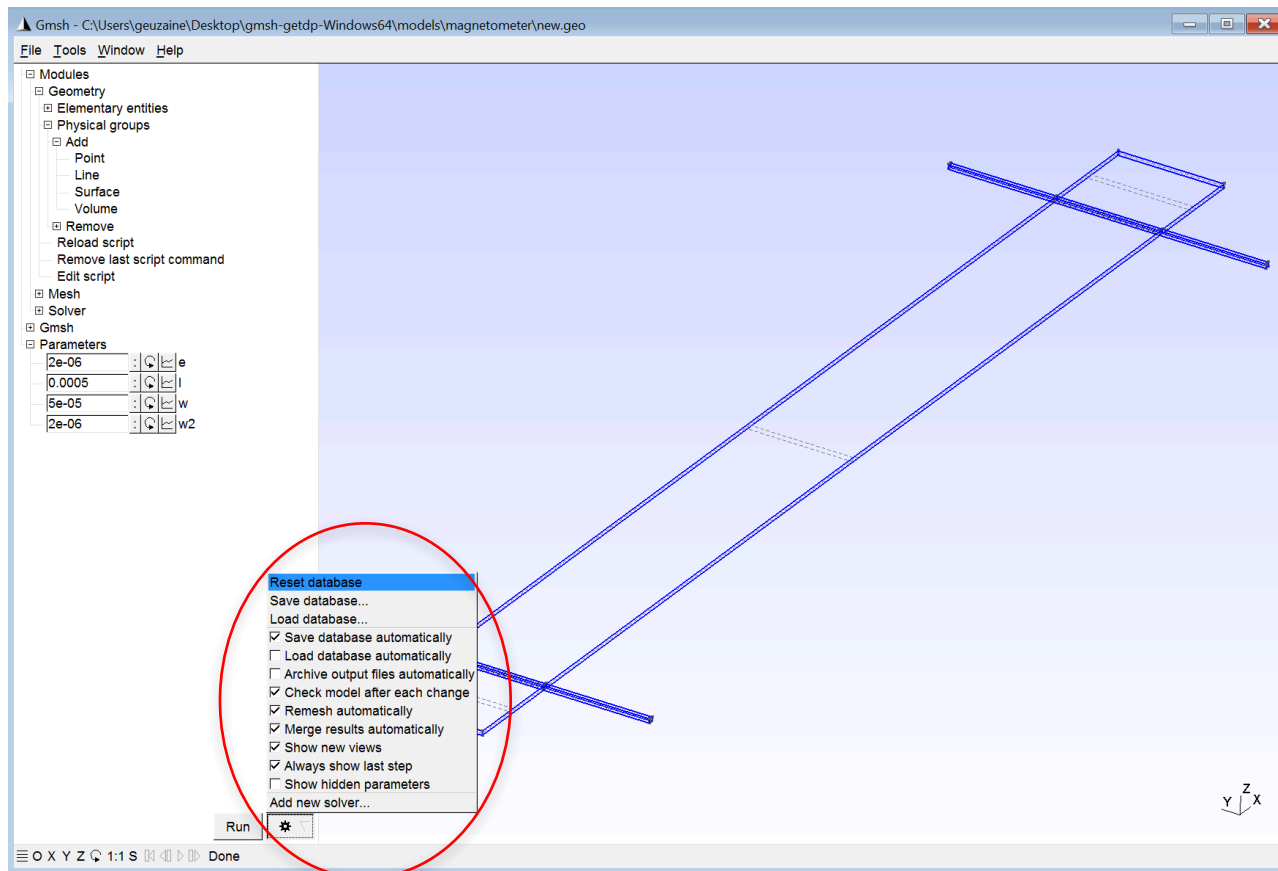
Geometry construction with Gmsh

- You can inspect all entities (elementary and physical) by selecting Tools->Visibility



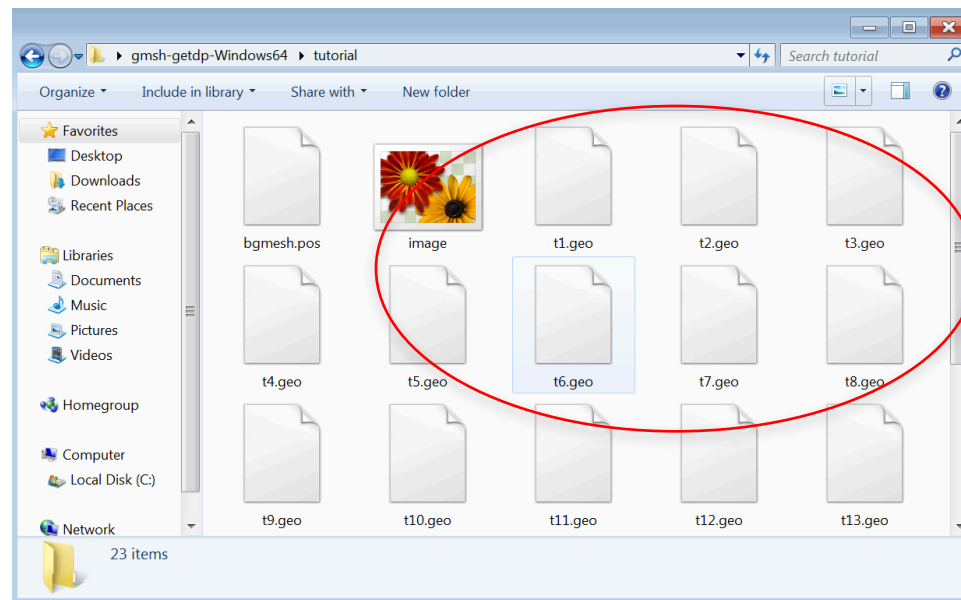
Geometry construction with Gmsh

- To reset all ONELAB variables to their default value, select “Reset database” in the gear menu



Geometry construction with Gmsh

- Many more commands exist in Gmsh: examine in more detail the first tutorials provided in the distribution (in the “tutorial” directory):

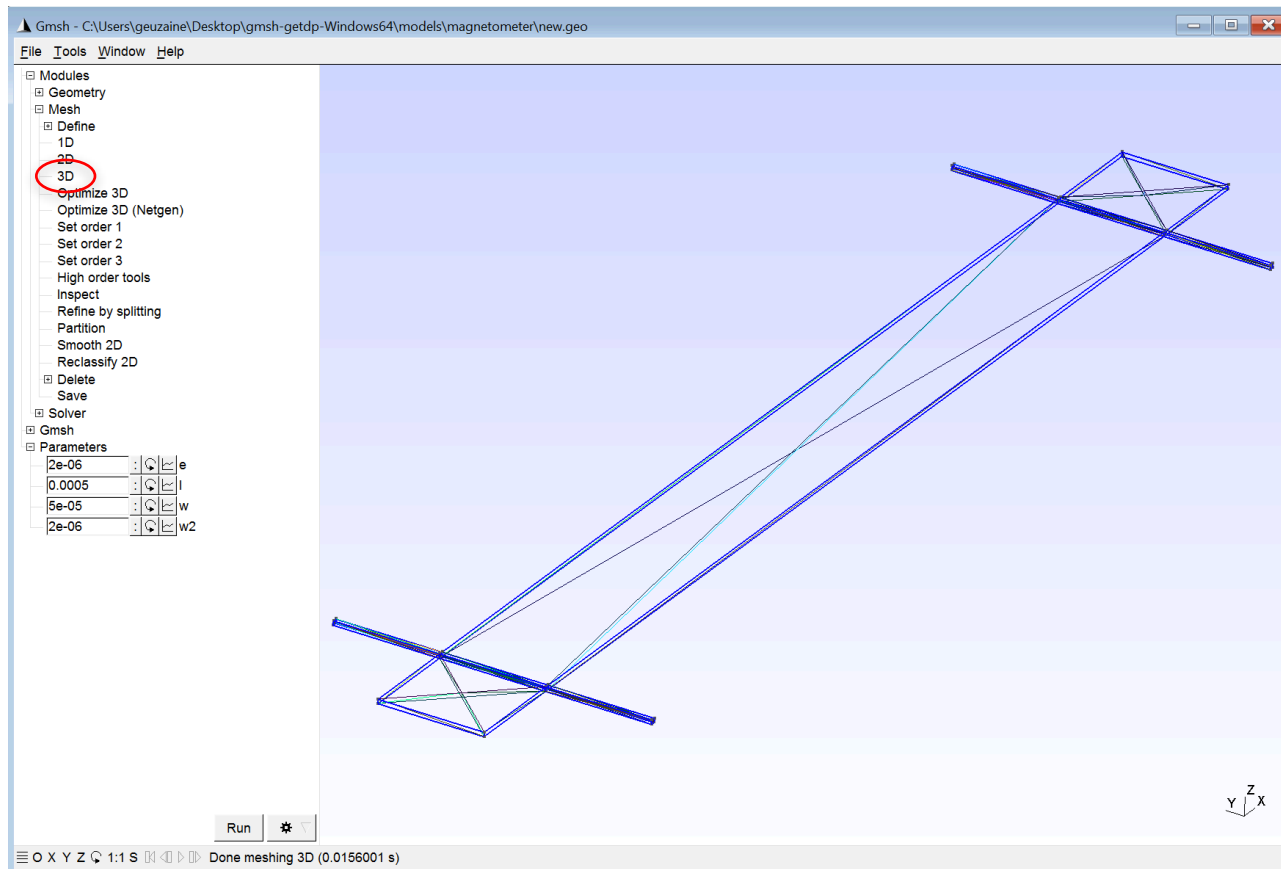


- The complete documentation is available on <http://gmsh.info>

Mesh generation

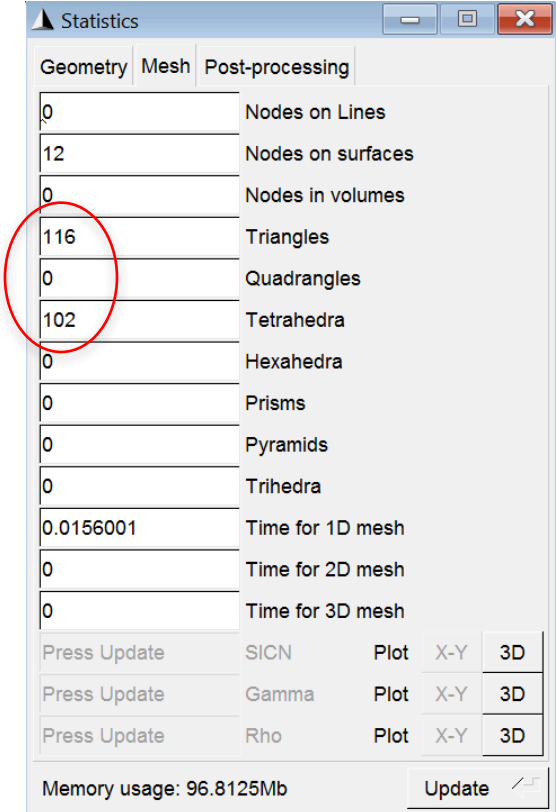
Mesh generation with Gmsh

- To generate a finite element mesh, select “Modules->Mesh->3D”



Mesh generation with Gmsh

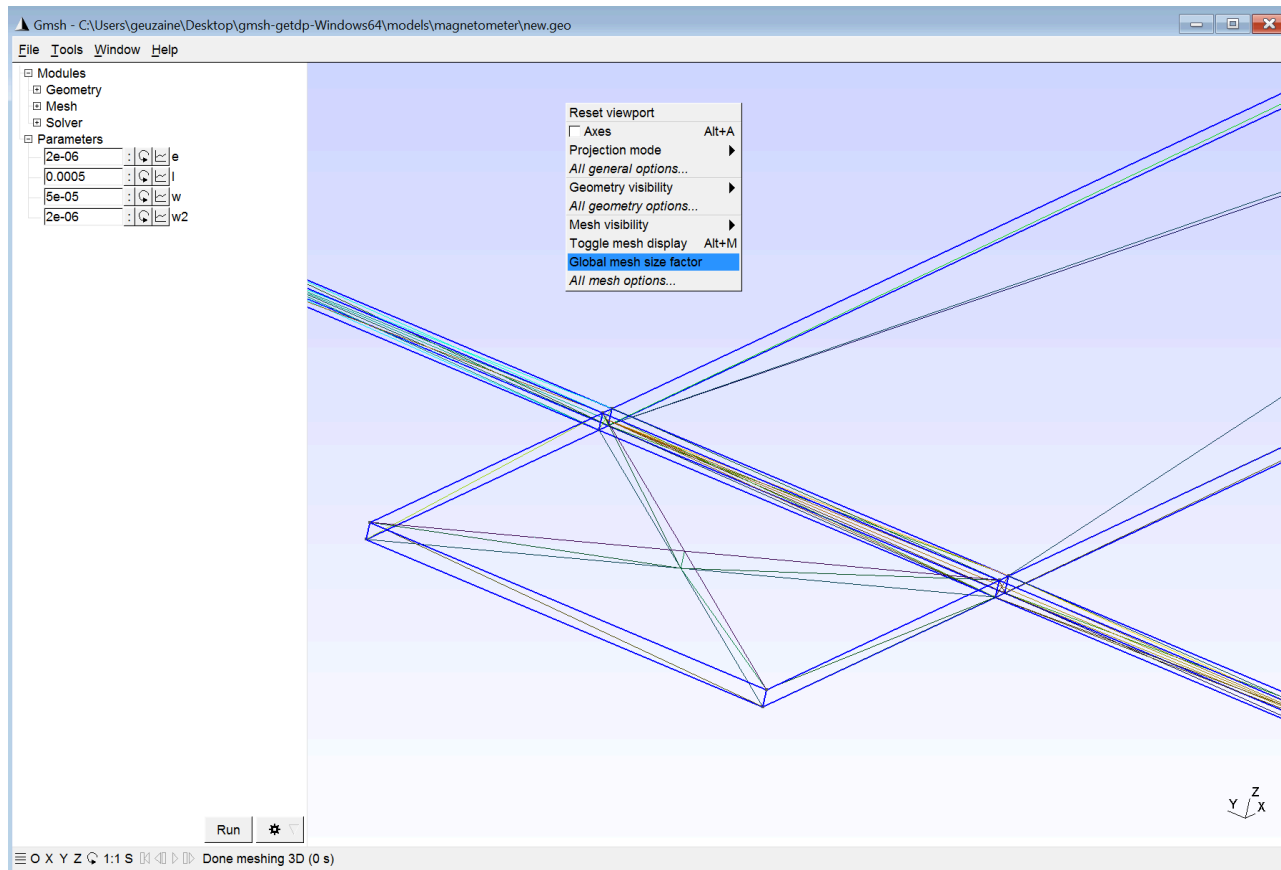
- The resulting mesh is
 - as coarse as possible (we set a target mesh size of “1.0” on point 1, for a structure of about $6e-4$; notice that there are no units at this stage)
 - made of tetrahedra in the volume, triangles on surfaces and line segments on curves (select “Tools->Statistics” to see the details)



Geometry	Mesh	Post-processing
0		Nodes on Lines
12		Nodes on surfaces
0		Nodes in volumes
116		Triangles
0		Quadrangles
102		Tetrahedra
0		Hexahedra
0		Prisms
0		Pyramids
0		Trihedra
0.0156001		Time for 1D mesh
0		Time for 2D mesh
0		Time for 3D mesh
Press Update	SICN	Plot X-Y 3D
Press Update	Gamma	Plot X-Y 3D
Press Update	Rho	Plot X-Y 3D
Memory usage: 96.8125Mb		Update

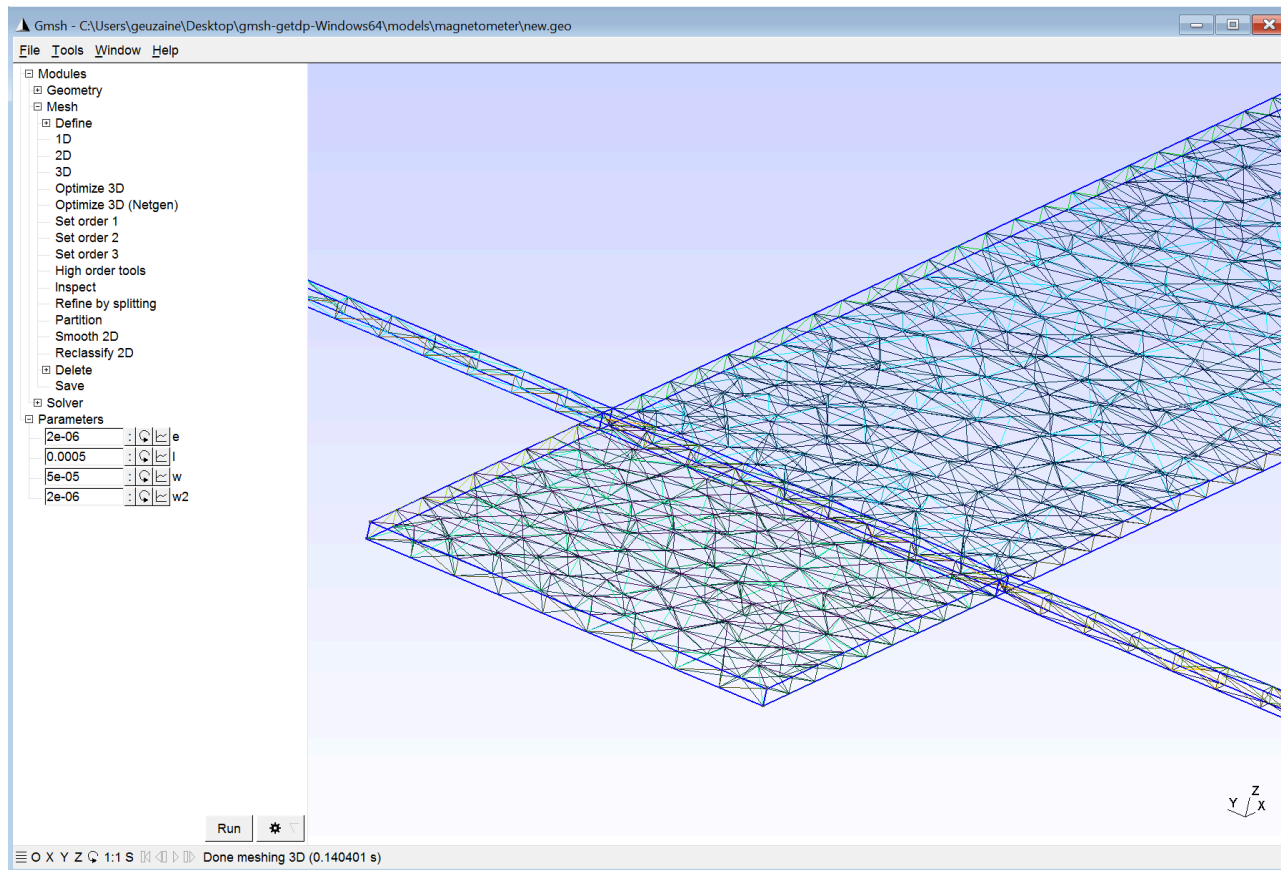
Mesh generation with Gmsh

- Double-click in the graphic window and set the “Global mesh size factor” to $1e-2$



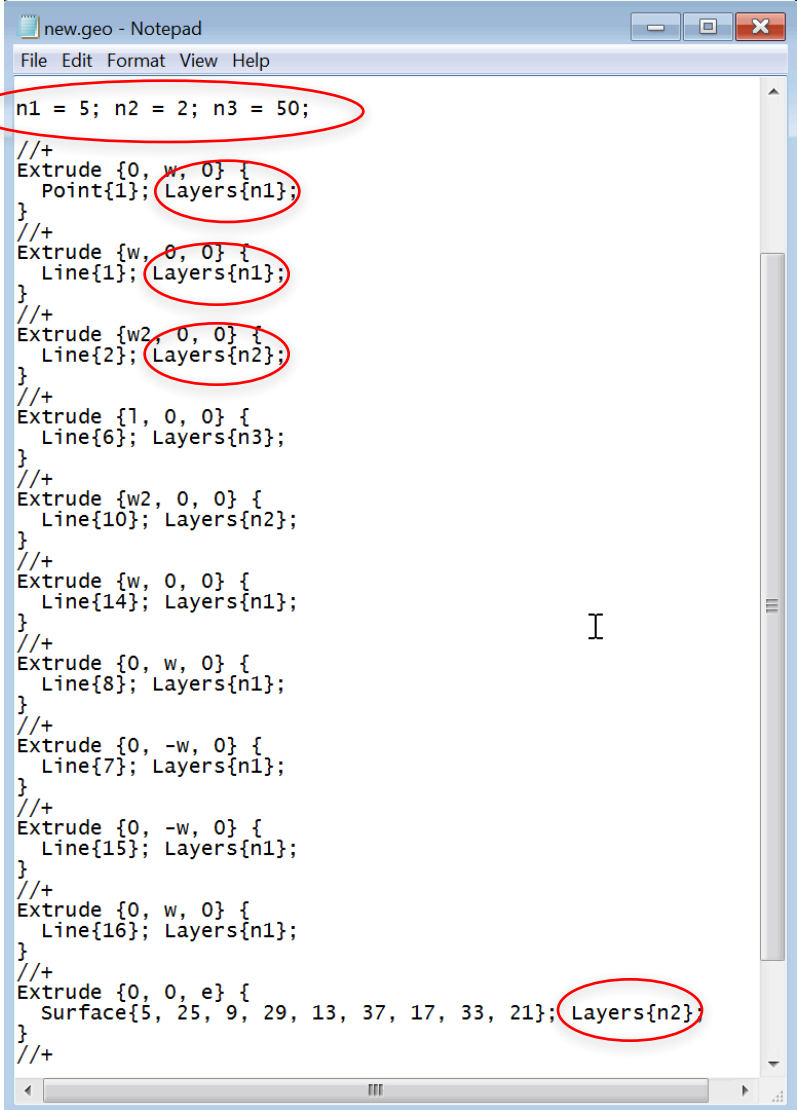
Mesh generation with Gmsh

- Press “1D”, “2D” and “3D” in “Modules->Mesh” to remesh the curves, surfaces and volumes



Mesh generation with Gmsh

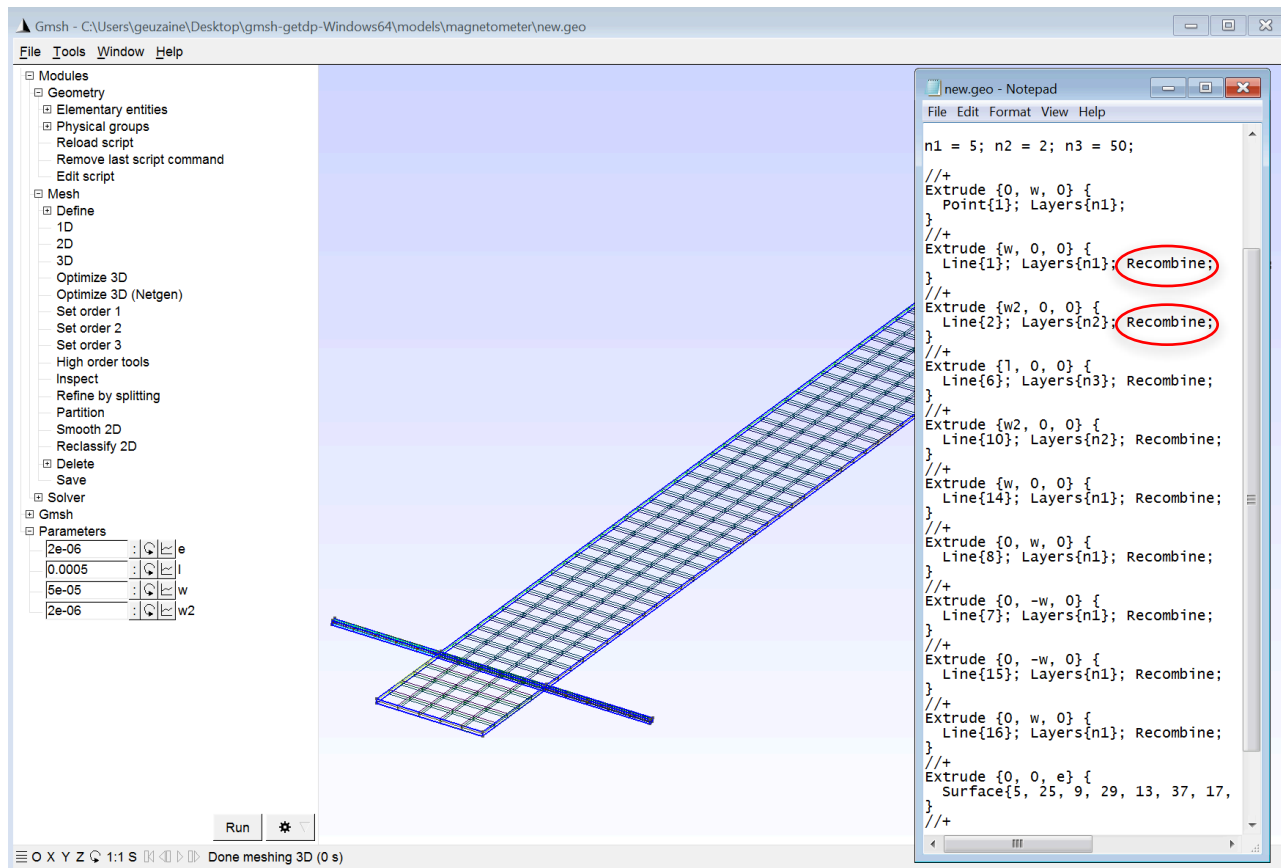
- For a thin structure like the magnetometer, a regular (“structured”) mesh would be more suitable
- Edit the “new.geo” file and modify the Extrude commands by adding “Layers” options to specify explicitly the number of layers of elements to be generated in the extrusion



```
new.geo - Notepad
File Edit Format View Help
n1 = 5; n2 = 2; n3 = 50;
//+
Extrude {0, w, 0} {
  Point{1}; Layers{n1};
}
//+
Extrude {w, 0, 0} {
  Line{1}; Layers{n1};
}
//+
Extrude {w2, 0, 0} {
  Line{2}; Layers{n2};
}
//+
Extrude {1, 0, 0} {
  Line{6}; Layers{n3};
}
//+
Extrude {w2, 0, 0} {
  Line{10}; Layers{n2};
}
//+
Extrude {w, 0, 0} {
  Line{14}; Layers{n1};
}
//+
Extrude {0, w, 0} {
  Line{8}; Layers{n1};
}
//+
Extrude {0, -w, 0} {
  Line{7}; Layers{n1};
}
//+
Extrude {0, -w, 0} {
  Line{15}; Layers{n1};
}
//+
Extrude {0, w, 0} {
  Line{16}; Layers{n1};
}
//+
Extrude {0, 0, e} {
  Surface{5, 25, 9, 29, 13, 37, 17, 33, 21}; Layers{n2};
}
//+
```

Mesh generation with Gmsh

- Adding “Recombine;” will recombine all triangles into quadrangles and all tetrahedra into hexahedra



Mesh generation with Gmsh

The final “new.geo” script after removing the comments:

```
Point(1) = {0, 0, 0, 1.0};
w = DefineNumber[ 50e-6, Name "Parameters/w" ];
w2 = DefineNumber[ 2e-6, Name "Parameters/w2" ];
l = DefineNumber[ 500e-6, Name "Parameters/l" ];
e = DefineNumber[ 2e-6, Name "Parameters/e" ];
n1 = 5; n2 = 2; n3 = 50;
Extrude {0, w, 0} { Point{1}; Layers{n1};}
Extrude {w, 0, 0} { Line{1}; Layers{n1}; Recombine;}
Extrude {w2, 0, 0} { Line{2}; Layers{n2}; Recombine;}
Extrude {l, 0, 0} { Line{6}; Layers{n3}; Recombine;}
Extrude {w2, 0, 0} { Line{10}; Layers{n2}; Recombine;}
Extrude {w, 0, 0} { Line{14}; Layers{n1}; Recombine;}
Extrude {0, w, 0} { Line{8}; Layers{n1}; Recombine;}
Extrude {0, -w, 0} { Line{7}; Layers{n1}; Recombine;}
Extrude {0, -w, 0} { Line{15}; Layers{n1}; Recombine;}
Extrude {0, w, 0} { Line{16}; Layers{n1}; Recombine;}
Extrude {0, 0, e} { Surface{5, 25, 9, 29, 13, 37, 17, 33, 21}; Layers{n2};
                  Recombine;}
Physical Volume("Magnetometer") = {2, 1, 3, 4, 5, 6, 7, 9, 8};
Physical Surface("Left Electrode") = {76, 120};
Physical Surface("Right Electrode") = {164, 208};
```

Mesh generation with Gmsh

- To save the mesh to disk in the default mesh format, select “Modules->Mesh->Save”
- The “File->Save As” menu has many more options to save the mesh in various formats suitable for different solvers, or to export images. Simply choosing a file name with the right extension (e.g. “image.png” for a PNG image) will select the appropriate file format
- You can mesh without the graphical user interface, by opening a terminal and typing: “gmsh new.geo -3”

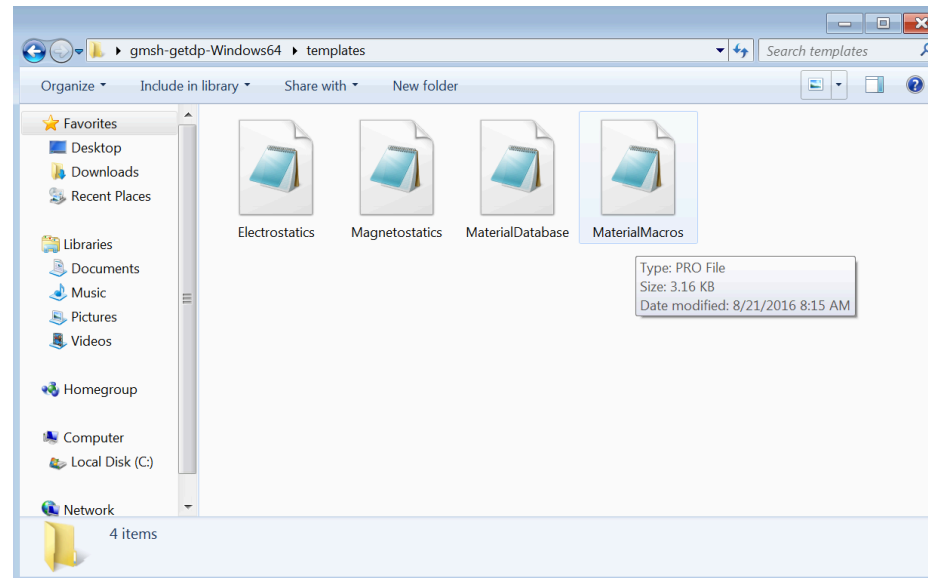
Solution computation

GetDP solver

- GetDP is a rather general finite element solver, whose main feature is the closeness between the input data defining discrete problems (written in plain text “.pro” files) and the symbolic mathematical expressions of these problems
- ONELAB uses GetDP both
 - as a “black-box” solver, if a template problem definition is available
 - as a fully customizable solver, when the physical problem definition is written from scratch

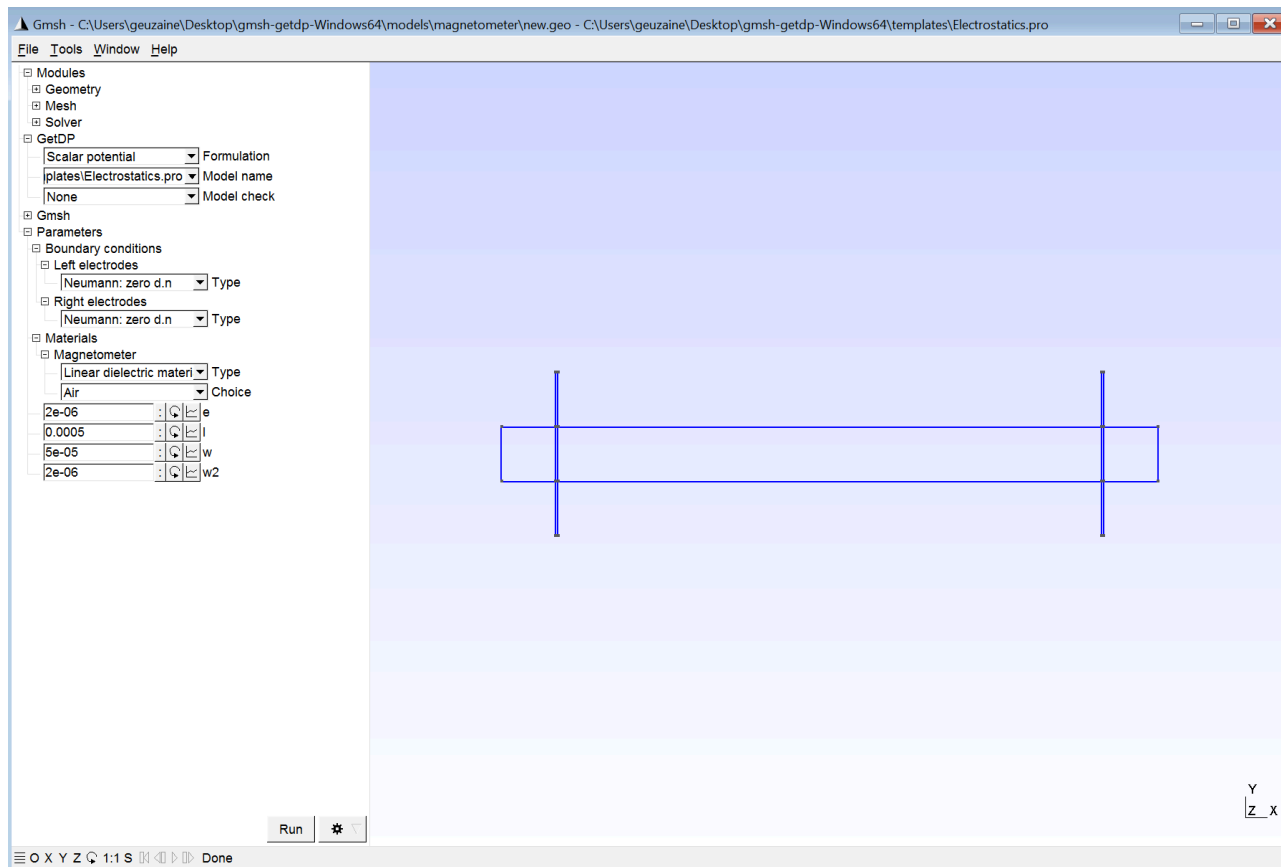
GetDP solver – Using a template

- If a template for the given physical problem already exists, simply merge it along with the geometrical description with “File->Merge”
- Simple templates are provided in the “templates” directory



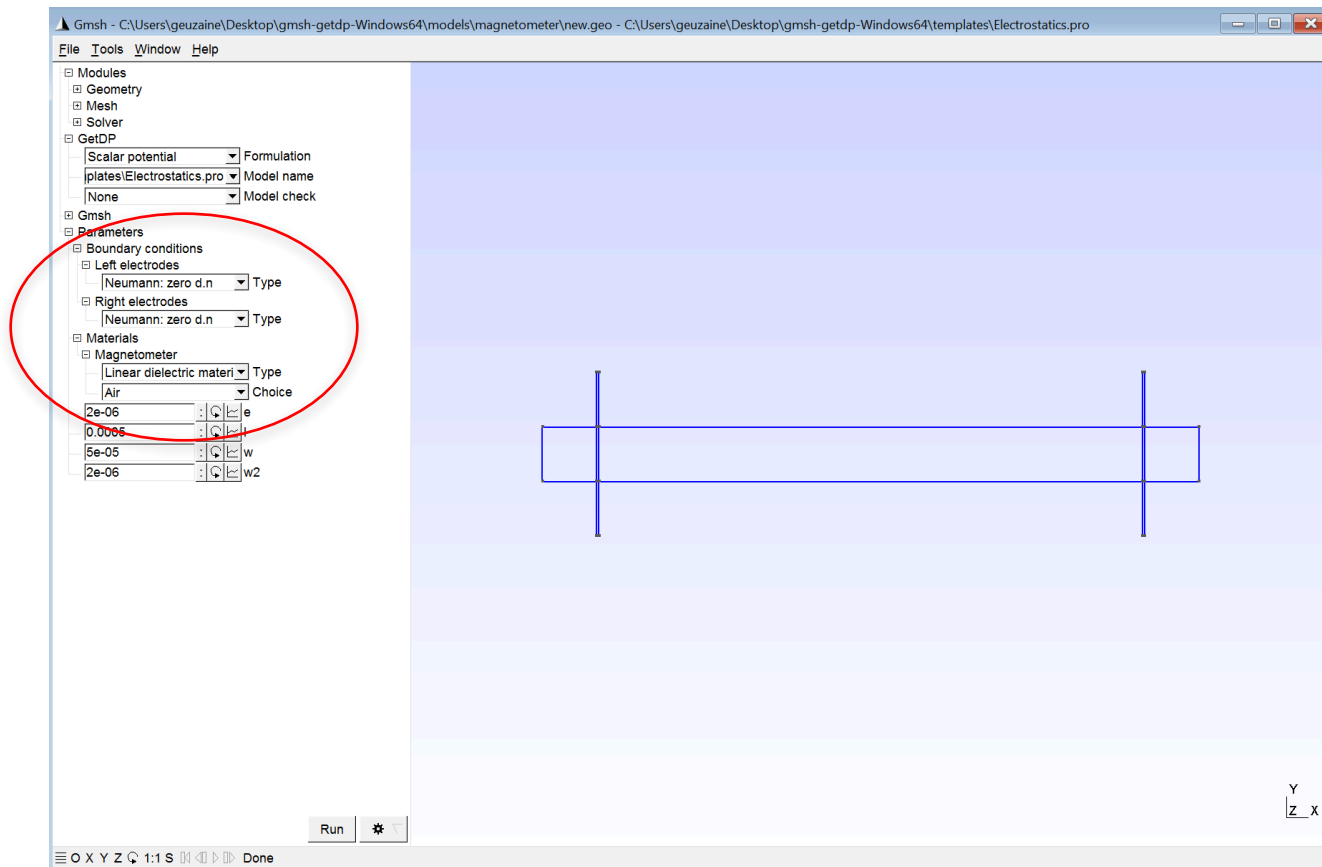
GetDP solver – Using a template

- “File->Merge” the “Electrostatics.pro” template with the magnetometer geometry



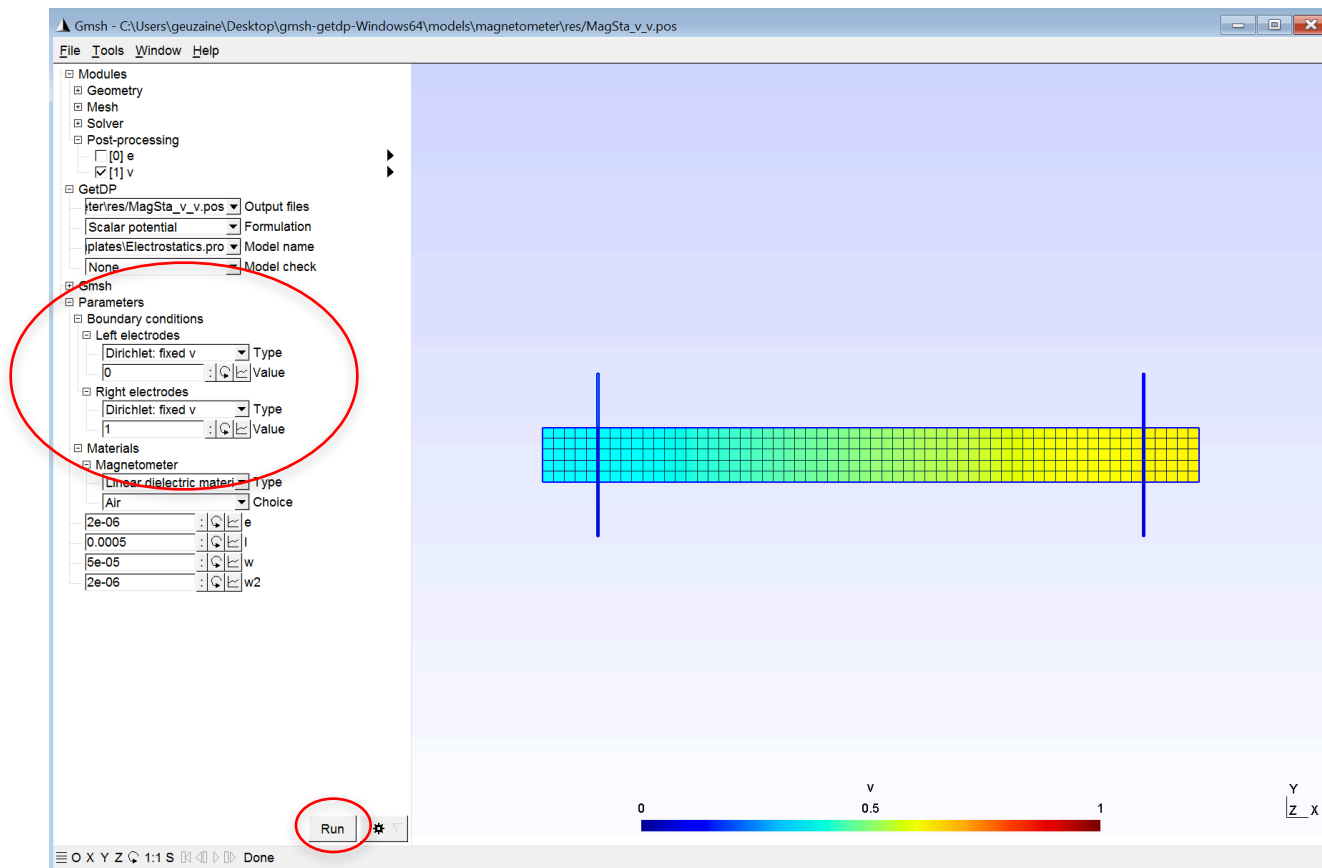
GetDP solver – Using a template

- Materials and boundary conditions can be specified interactively on the available physical groups



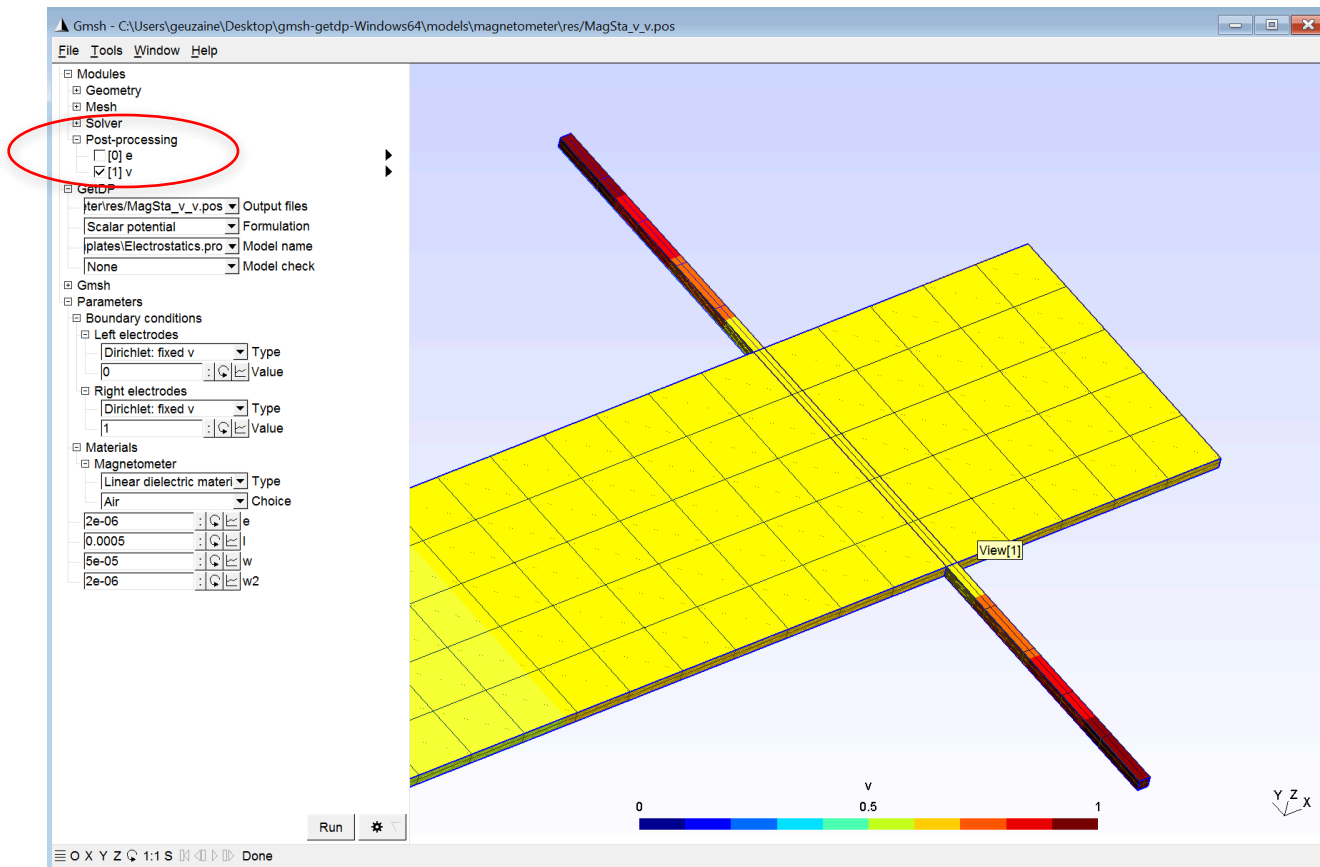
GetDP solver – Using a template

- Once the parameters are chosen, press “Run” to compute the solution



GetDP solver – Using a template

- We will see later how to exploit the results in the post-processing module



GetDP solver – Constructing your own model

- In order to construct your own physical model, you need to start with the mathematical description of the physical problem as partial differential equations
- The finite element method allows to solve such partial differential equations in so-called “weak form”, after discretization on a mesh
- The next slides provide a short summary of the general approach – you will study details for electromagnetics, heat transfer and mechanics on Tuesday and Wednesday

GetDP solver – Constructing your own model

- Let us assume that the physical problem we want to solve is modelled by the following partial differential equation on the open set $\Omega \subset R^3$, with homogeneous boundary conditions on the boundary $\partial\Omega$:

$$\begin{aligned}\operatorname{div} \alpha(\mathbf{x}) \operatorname{grad} u(\mathbf{x}) &= f(\mathbf{x}), & \mathbf{x} \in \Omega \\ u(\mathbf{x}) &= 0, & \mathbf{x} \in \partial\Omega\end{aligned}$$

- Given $\alpha(\mathbf{x}) > 0$ and $f(\mathbf{x})$, we want to find $u(\mathbf{x})$ everywhere in Ω

GetDP solver – Constructing your own model

- For general Ω , $\alpha(\mathbf{x})$ and $f(\mathbf{x})$ there is no closed form solution for $u(\mathbf{x})$
- The main idea behind the finite element method is to look for the solution $u(\mathbf{x})$ such that the partial differential equation is satisfied when tested against well-chosen test functions
- Mathematically, we look for $u(\mathbf{x})$, with $u(\mathbf{x}) = 0$ on $\partial\Omega$, such that

$$\int_{\Omega} (\operatorname{div} \alpha(\mathbf{x}) \operatorname{grad} u(\mathbf{x})) u'(\mathbf{x}) d\Omega = \int_{\Omega} f(\mathbf{x}) u'(\mathbf{x}) d\Omega$$

holds for all appropriately chosen functions $u'(\mathbf{x})$

GetDP solver – Constructing your own model

- The differential operators `div` and `grad` should be understood in the distributional sense (think about $\alpha(\mathbf{x})$ being discontinuous)
- Integrating by parts, the problem becomes: find $u(\mathbf{x})$ such that

$$-\int_{\Omega} \alpha(\mathbf{x}) \mathbf{grad} u(\mathbf{x}) \cdot \mathbf{grad} u'(\mathbf{x}) d\Omega = \int_{\Omega} f(\mathbf{x}) u'(\mathbf{x}) d\Omega$$

holds for all $u'(\mathbf{x})$

- The finite element methods consists in approximating $u(\mathbf{x})$ by $u_h(\mathbf{x})$ in a finite dimensional function space

GetDP solver – Constructing your own model

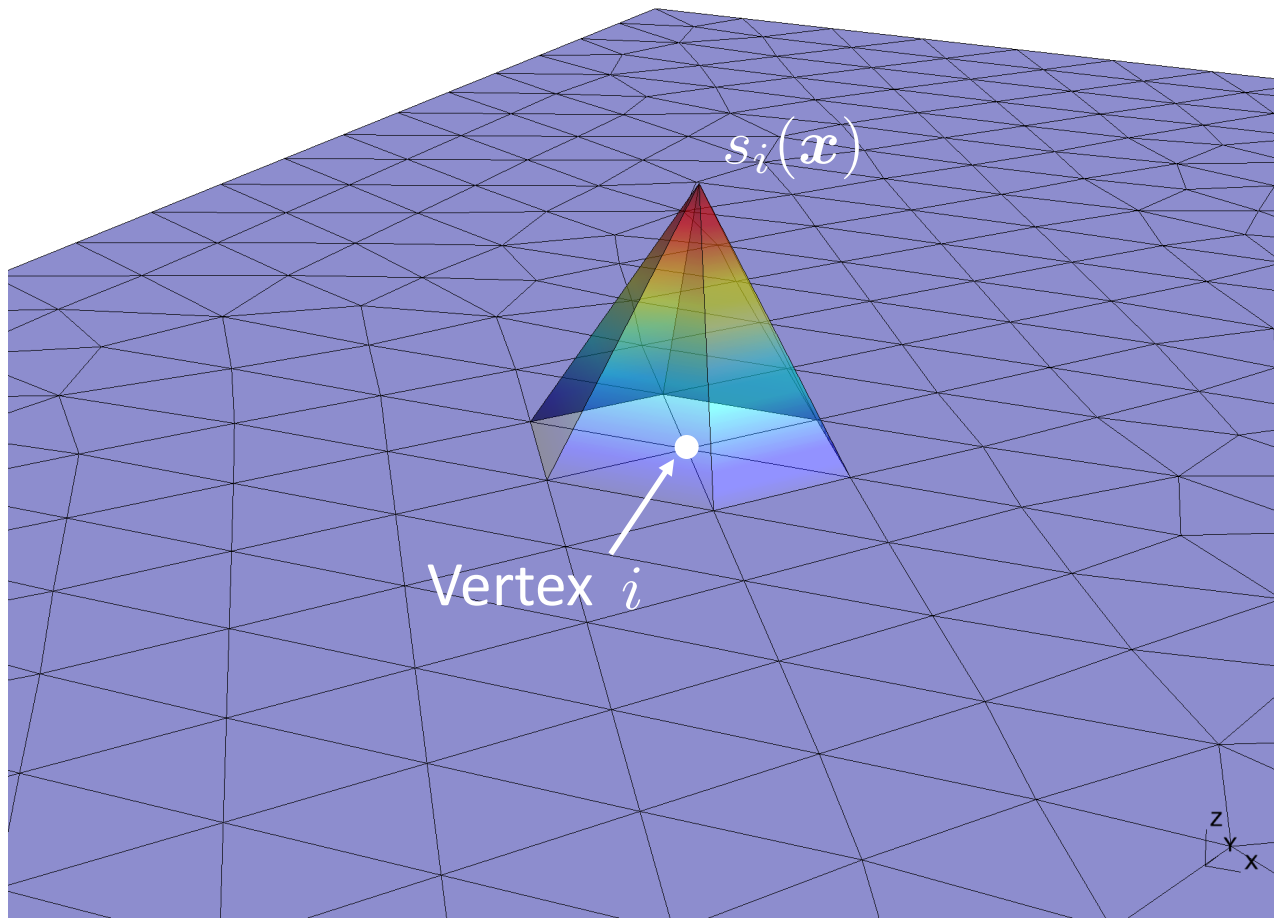
- For classical finite elements, this means taking $u_h(\mathbf{x})$ and the test functions in

$$F^0(\Omega) = \text{span}\{s_1(\mathbf{x}), s_2(\mathbf{x}), \dots, s_N(\mathbf{x})\}$$

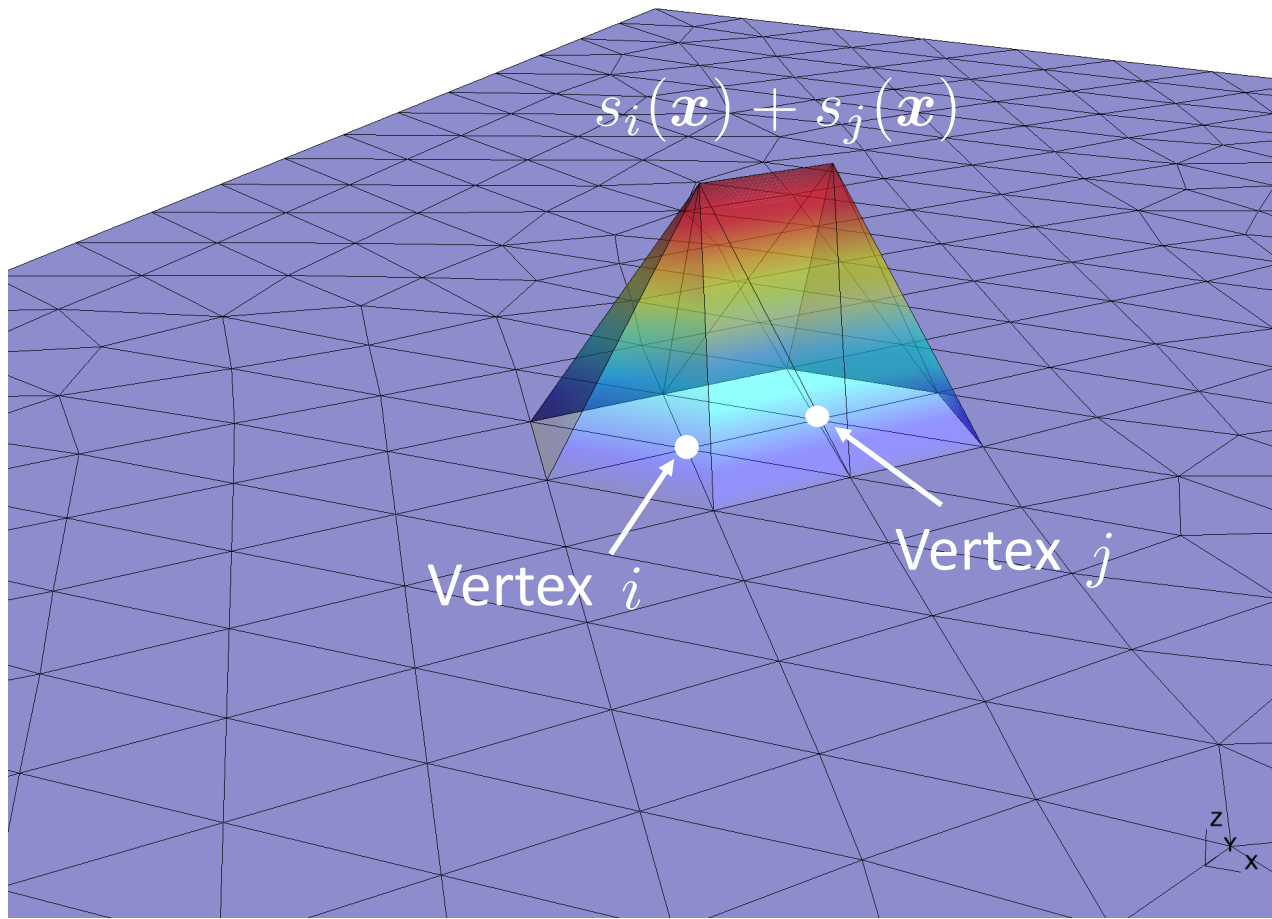
where the basis functions $s_i(\mathbf{x})$, $i = 1, \dots, N$ are associated with the N vertices of the mesh

- Lowest order basis functions are linear on line, triangle and tetrahedral elements, bilinear on quadrangles and trilinear on hexahedra; their value goes from 1 at the associated vertex, to 0 on the other vertices

GetDP solver – Constructing your own model



GetDP solver – Constructing your own model



GetDP solver – Constructing your own model

- Injecting $u(\mathbf{x}) \approx u_h(\mathbf{x}) = \sum_{i=1}^N u_i s_i(\mathbf{x})$ in the weak form

$$- \int_{\Omega} \alpha(\mathbf{x}) \mathbf{grad} u(\mathbf{x}) \cdot \mathbf{grad} u'(\mathbf{x}) d\Omega = \int_{\Omega} f(\mathbf{x}) u'(\mathbf{x}) d\Omega$$

and taking for $u'(\mathbf{x}) : s_i(\mathbf{x}), i = 1, \dots, N$ leads to a system of linear equations, for $j = 1, \dots, N$:

$$- \sum_{i=1}^N u_i \int_{\Omega} \alpha(\mathbf{x}) \mathbf{grad} s_i(\mathbf{x}) \cdot \mathbf{grad} s_j(\mathbf{x}) d\Omega = \int_{\Omega} f(\mathbf{x}) s_j(\mathbf{x}) d\Omega$$

GetDP solver – Constructing your own model

- In matrix form, we have

$$\mathbf{A}\mathbf{u} = \mathbf{f}$$

with $\mathbf{A}_{ij} = - \int_{\Omega} \alpha(\mathbf{x}) \mathbf{grad} s_i(\mathbf{x}) \cdot \mathbf{grad} s_j(\mathbf{x}) d\Omega$

$$\mathbf{u} = [u_1, u_2, \dots, u_N]^T$$

$$\mathbf{f} = \left[\int_{\Omega} f(\mathbf{x}) s_1(\mathbf{x}) d\Omega, \dots, \int_{\Omega} f(\mathbf{x}) s_N(\mathbf{x}) d\Omega \right]^T$$

This is the system solved by the computer, e.g. using LU factorization

- The matrix \mathbf{A} is very sparse, since basis functions associated with distant nodes have disjoint support – leading to many zero integrals

GetDP solver – Constructing your own model

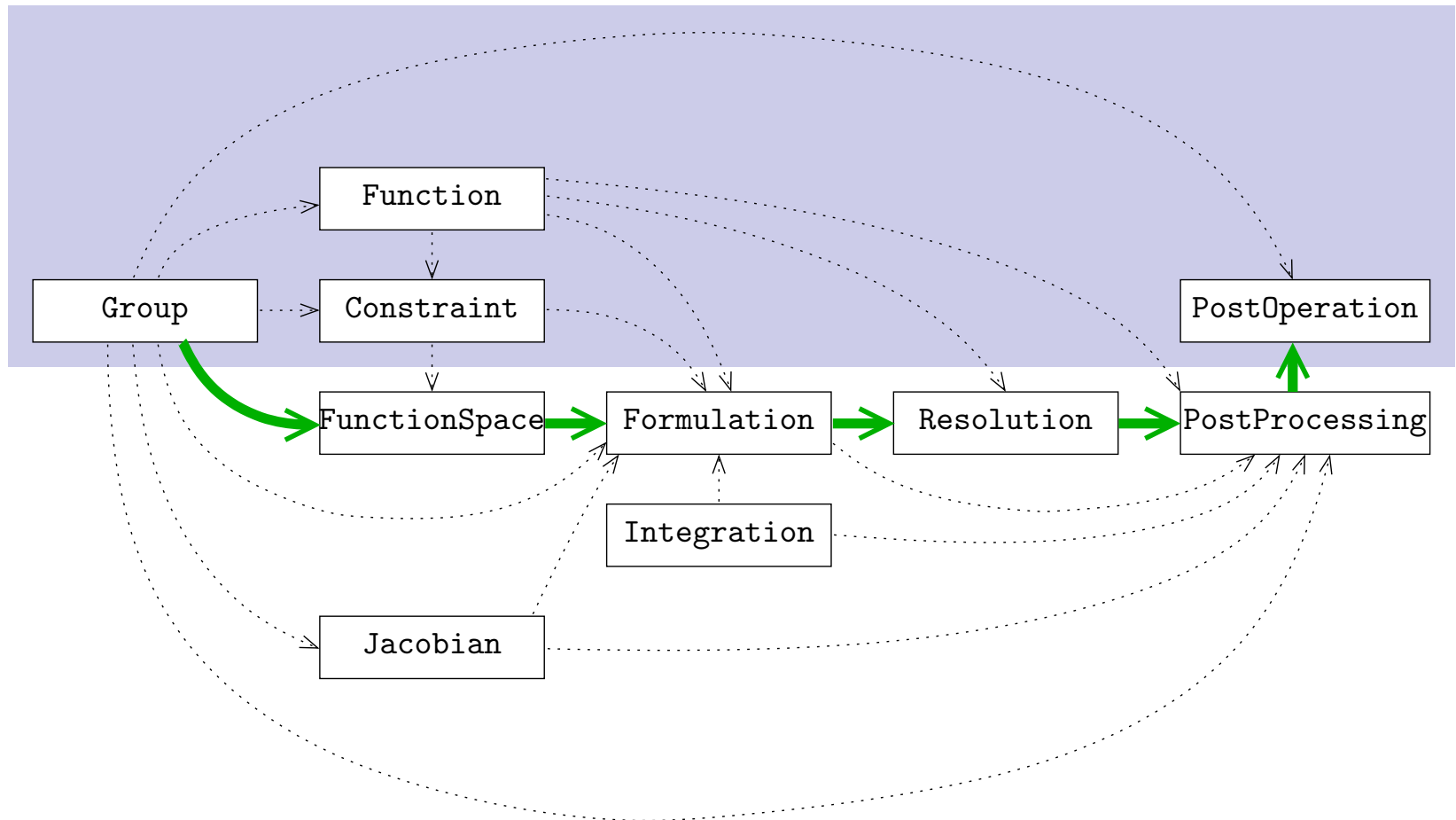
- The implementation in GetDP “.pro” files reflects this problem definition structure:
 - Group{ } defines the domains (i.e. Ω and $\partial\Omega$)
 - Function{ } defines the functions (i.e. $\alpha(\mathbf{x})$ and $f(\mathbf{x})$)
 - FunctionSpace{ } and Constraint{ } define the function space (i.e. $F^0(\Omega)$) and associated constraints ($u(\mathbf{x}) = 0$ on $\partial\Omega$)
 - Formulation{ } defines the weak form

GetDP solver – Constructing your own model

- Additional objects define
 - Integration methods: `Integration{ }`
 - Geometrical transformation methods: `Jacobian{ }`
 - Resolution steps: `Resolution{ }`
 - Post-processing specifications: `PostProcessing{ }` and `PostOperation{ }`

GetDP solver – Constructing your own model

10 objects Top: particular to a particular test-case
Bottom: generic for a physical/mathematical model



GetDP solver – Constructing your own model

1. Group

- Regions (physical groups)
- “Functions” on regions (nodes, edges, edges of tree, ...)

```
Group{  
  Air = Region[1]; //simple group (linked with the mesh)  
  Core = Region[2];  
  Gamma = Region[{3,4}];  
  
  Omega = Region[{Air, Core}]; //combining groups  
  
  nodes = NodesOf[Omega]; //function group  
  edgesOfSpanningTree = EdgesOfTreeIn[Omega, StartinOn Gamma];  
}
```


GetDP solver – Constructing your own model

2. Function

- Piecewise definition (on groups)
- Space/time dependent, with arguments
- For physical characteristics, sources, constraints, ...

```
Function{  
  f = 50; //constants  
  mu0 = 4.e-7 * Pi;  
  
  mu[Air] = mu0; //piecewise definition  
  mu[Core] = mu0 + 1/(100 + 100 * ($1)^6); //argument ($1)  
  
  TimeFct[] = Cos[2*Pi*f*$Time] * Exp[-$Time/0.01]; //current value  
}
```

GetDP solver – Constructing your own model

3. FunctionSpace

- List of basis functions associated with nodes, edges, faces or volumes, of various orders
- Local or nonlocal (fluxes, circulations, ...)

```
FunctionSpace{
  { Name H1; Type Form0; //discrete function space for H1_h
    BasisFunction {
      { Name wi; NameOfCoef fi; Function BF_Node; //‘‘P1 finite elements’’
        Support Omega; Entity NodesOf[All]; }
      }
    Constraint {
      { NameOfCoef fi; EntityType NodesOf; NameOfConstraint Dirichlet; }
      }
  }
}
```

GetDP solver – Constructing your own model

3. FunctionSpace

//higher-order version

```
FunctionSpace{
  { Name H1; Type Form0;
    BasisFunction {
      { Name wi; NameOfCoef fi; Function BF_Node; //order 1
        Support Omega; Entity NodesOf[All]; }
      { Name wi2; NameOfCoef fi2; Function BF_Node_2E; //order 2
        Support Omega; Entity EdgesOf[All]; }
    }
    Constraint {
      { NameOfCoef fi; EntityType NodesOf; NameOfConstraint Dirichlet; }
      { NameOfCoef fi2; EntityType EdgesOf; NameOfConstraint Dirichlet2; }
    }
  }
}
```

GetDP solver – Constructing your own model

4. Constraint

- Constraints on FunctionSpace coefficients (boundary, periodicity, initial and gauge conditions, ...)
- Topology of circuits with lumped elements

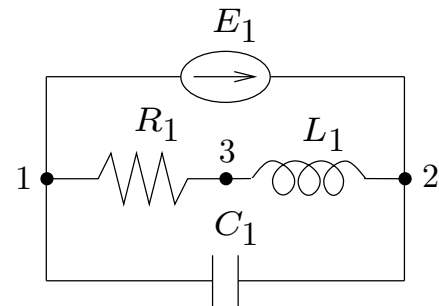
```
Constraint{  
  { Name Dirichlet; Type Assign; //boundary conditions  
    Case {  
      { Region Surface0; Value 0; }  
      { Region Surface1; Value 1; }  
    }  
  }  
}
```

GetDP solver – Constructing your own model

4. Constraint

```
Constraint{  
  //time-dependent or harmonic constraints  
  { Name Current; Type Assign;  
    Case {  
      { Region CurrentLoop; Value 1000; TimeFunction TimeFct[]; }  
    }  
  }  
}
```

```
//network relations between global quantities  
{ Name ElectricalCircuit; Type Network;  
  Case Circuit {  
    { Region E1; Branch {1,2}; }  
    { Region R1; Branch {1,3}; }  
    { Region L1; Branch {3,2}; }  
    { Region C1; Branch {1,2}; }  
  }  
}
```



GetDP solver – Constructing your own model

5. Formulation

- Symbolic equation builder (bilinear and linear forms, term by term) involving local and nonlocal quantities

```
Formulation{
  { Name Maxwell_e; Type FemEquation;
    Quantity {
      { Name e; Type Local; NameOfSpace Hcurl_h; }
    }
    Equation {
      Galerkin { [ 1/mu[] * Dof{Curl e} , {Curl e} ];
                 In Omega; Jacobian Jac1; Integration Int1; }
      Galerkin { DtDt [ epsilon[] * Dof{e} , {e} ];
                 In Omega; Jacobian Jac1; Integration Int1; }
    }
  }
}
```

“Find $\mathbf{e} \in \mathbf{H}_h(\mathbf{curl}; \Omega)$ such that
 $(\mu^{-1} \mathbf{curl} \mathbf{e}, \mathbf{curl} \mathbf{e}') + \partial_t^2(\epsilon \mathbf{e}, \mathbf{e}') = 0, \quad \forall \mathbf{e}' \in \mathbf{H}_h(\mathbf{curl}; \Omega)$ ”

GetDP solver – Constructing your own model

6. Jacobian

- Mapping fom reference to solution space
- Geometrical transformations (axisymmetry, infinite domains, ...)

```
Jacobian{  
  { Name Jac1;  
    Case { //piecewise defined on groups  
      { Region OmegaInf; Jacobian VolSphShell{Rint, Rext}; }  
      { Region OmegaAxi; Jacobian VolAxi; }  
      { Region All; Jacobian Vol; }  
    }  
  }  
}
```

GetDP solver – Constructing your own model

7. Integration

- Numerical and analytic integration methods
- Criterion-based selection

```
Integration {  
  { Name Int1; Criterion Test[];  
    Case {  
      { Type Gauss;  
        Case {  
          { GeoElement Triangle; NumberOfPoints 3; }  
          { GeoElement Tetrahedron; NumberOfPoints 3; }  
        }  
      }  
      { Type Analytic; }  
    }  
  }  
}
```


GetDP solver – Constructing your own model

8. Resolution

- Sequence of solving operations (system generation and solution, time-stepping and nonlinear iterations, multi-formulation coupling, ...)

```
Resolution{
  { Name Parabolic;
    System {
      { Name A; NameOfFormulation Parabolic; }
    }
    Operation{
      InitSolution[A];
      TimeLoopTheta[tmin,tmax,dt,1]{
        Generate[A]; Solve[A]; If[Save[]]{ SaveSolution[A]; }
      }
    }
  }
}
```

GetDP solver – Constructing your own model

9. PostProcessing

- Piecewise definition of quantities of interest
- Local or integral evaluation

```
PostProcessing {
  { Name magfields; NameOfFormulation Dynamic;
    Quantity {
      { Name b;
        Value {
          Local { [ -mu[] * {Grad phi} ]; In OmegaCC; }
          Local { [ mu[] * h ]; In OmegaC; }
        }
      }
    }
  }
}
```

GetDP solver – Constructing your own model

10. PostOperation

- Evaluation and export of post-processing quantities (maps, cuts, ...)

```
PostOperation {  
  { Name Map_b; NameOfPostProcessing magfields;  
    Operation {  
      Print[ b, OnElementsOf Omega, File "b.pos", Format Gmsh ];  
      Print[ b, OnLine {{0,0,0}}{1,0,0}} {100}, File "b.txt" ];  
    }  
  }  
}
```

GetDP solver – Constructing your own model

For our toy problem, the complete “new.pro” file is:

```
Group{
  Omega = Region[1]; Gamma0 = Region[{2,3}];
}

Function{
  alpha[Omega] = 1; f[Omega] = -1;
}

Jacobian {
  { Name Vol; Case { { Region All; Jacobian Vol; } } }
}

Integration {
  { Name I1; Case {
    { Type Gauss; Case { { GeoElement Hexahedron; NumberOfPoints 6; } } }
  }
}

Constraint {
  { Name Dirichlet; Case { { Region Gamma0; Value 0; } } }
}
```

GetDP solver – Constructing your own model

```

FunctionSpace {
  { Name H1_0; Type Form0;
    BasisFunction {
      { Name si; NameOfCoef ui; Function BF_Node;
        Support Omega; Entity NodesOf[ All ]; }
      }
    Constraint {
      { NameOfCoef ui; EntityType NodesOf; NameOfConstraint Dirichlet; }
    }
  }
}

Formulation {
  { Name Laplace; Type FemEquation;
    Quantity {
      { Name u; Type Local; NameOfSpace H1_0; }
    }
    Equation {
      Galerkin { [ - alpha[] * Dof{Grad u} , {Grad u} ];
        In Omega; Jacobian Vol; Integration I1; }
      Galerkin { [ - f[], {u} ];
        In Omega; Jacobian Vol; Integration I1; }
    }
  }
}

```

$$F^0(\Omega) = \text{span}\{s_1(\mathbf{x}), s_2(\mathbf{x}), \dots, s_N(\mathbf{x})\}$$

$$u(\mathbf{x}) \approx u_h(\mathbf{x}) = \sum_{i=1}^N u_i s_i(\mathbf{x})$$

$$- \int_{\Omega} \alpha(\mathbf{x}) \mathbf{grad} u(\mathbf{x}) \cdot \mathbf{grad} u'(\mathbf{x}) d\Omega$$

$$- \int_{\Omega} f(\mathbf{x}) u'(\mathbf{x}) d\Omega$$

$$= 0$$

GetDP solver – Constructing your own model

```
Resolution {  
  { Name MyResolution;  
    System {  
      { Name A; NameOfFormulation Laplace; }  
    }  
    Operation {  
      Generate[A]; Solve[A]; SaveSolution[A];  
    }  
  }  
}
```

$Au = f$

```
PostProcessing {  
  { Name MyPostPro; NameOfFormulation Laplace;  
    PostQuantity {  
      { Name u; Value { Term { [ {u} ]; In Omega; Jacobian Vol; } } }  
    }  
  }  
}
```

```
PostOperation {  
  { Name u; NameOfPostProcessing MyPostPro;  
    Operation {  
      Print[ u, OnElementsOf Omega, File "u.pos" ];  
    }  
  }  
}
```

GetDP solver – Constructing your own model

- Note that Gamma0 in “new.pro” does not actually contain the whole boundary of the magnetometer: the homogeneous Dirichlet boundary condition will only be applied on the electrodes. Question: what happens on the rest of the boundary? (As an exercise, modify “new.geo” to fix this.)
- The complete GetDP documentation is available on <http://getdp.info>
- We will study the physics and the implementation of electromagnetic, thermal and mechanical formulations on Tuesday and Wednesday

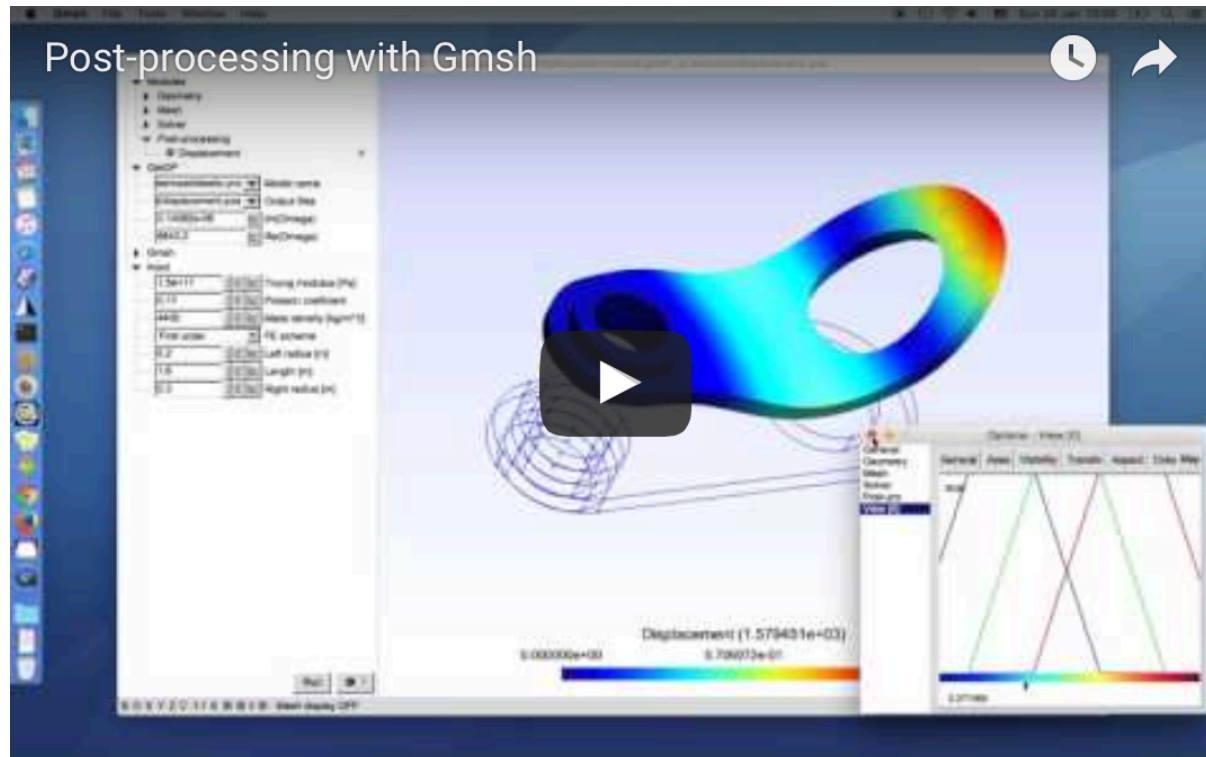
Post-processing

Post-processing with Gmsh

- Once you compute a solution by pressing “Run”, it is merged and displayed automatically in the graphic window; it is called a “view”
- Check/uncheck the views in “Modules->Post-processing” to show or hide them
- The Left/Right keyboard arrows allow to navigate in time
- The Up/Down arrows on the keyboard allow to navigate between views
- Click on the arrow next to a view name to access the view options

Post-processing with Gmsh

- A video tutorial showing how to use basic post-processing features is available here:
<http://youtu.be/oS3T8i07wkl>



Electromagnetism and Heat Transfer

Electromagnetic models

Different types of field computations



Electrostatics — Distribution of electric field due to static charges and/or electric potential difference

$$C = \frac{Q}{V} = \frac{\epsilon S}{d}$$



Electrokinetics — Distribution of static electric current in conductors

$$R = \frac{V}{I} = \frac{l}{\sigma S}$$



Electrodynamics — Distribution of electric field and electric current in materials (insulating & conducting)

$$\delta = \sqrt{\frac{2}{\omega \mu \sigma}}$$



Magnetostatics — Distribution of static magnetic field due to static currents (DC) & permanent magnets

$$L = \frac{\Phi}{m.m.f} = n^2 \frac{\mu_0 S}{l}$$



Magnetodynamics — Distribution of magnetic field and eddy currents due to time-variable sources



Full wave — Propagation of electromagnetic fields

Electrostatics

Phenomena involving time-independent distributions of charges & fields $\partial_t \mathbf{b} = 0$

$\text{curl } \mathbf{e} = 0$	boundary conditions
$\text{div } \mathbf{d} = q$	$\mathbf{n} \times \mathbf{e} _{\Gamma_e} = 0$
$\mathbf{d} = \epsilon \mathbf{e}$	$\mathbf{n} \cdot \mathbf{j} _{\Gamma_j} = 0$

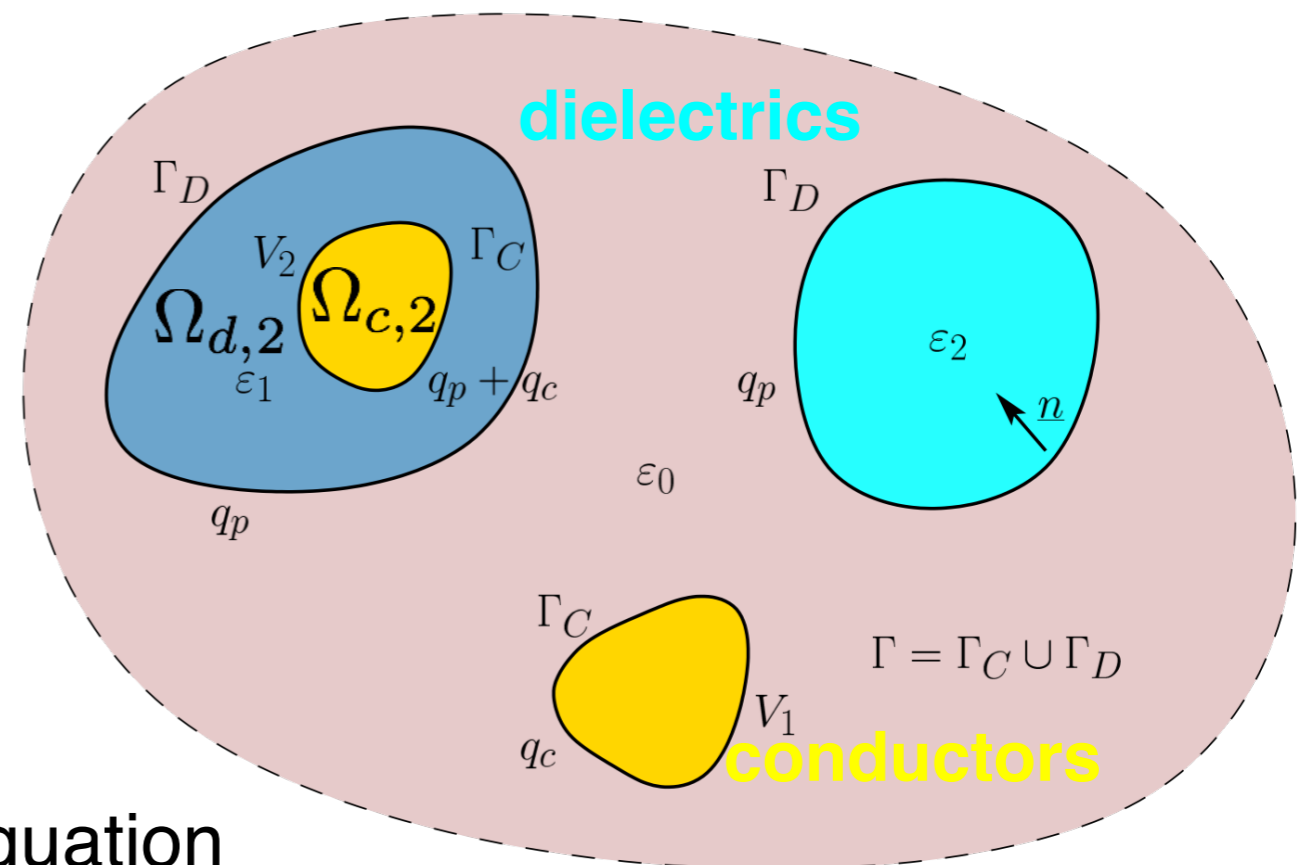
electric scalar potential formulation in

– $\text{div} (\epsilon \text{grad } v) = q, \quad \mathbf{e} = -\text{grad } v$ Ω
 \Downarrow no charges

– $\text{div grad } v = -\Delta v = 0$ Laplace equation

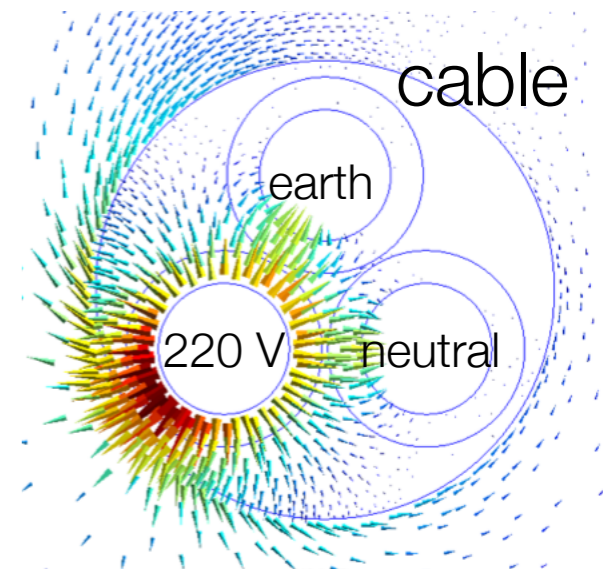
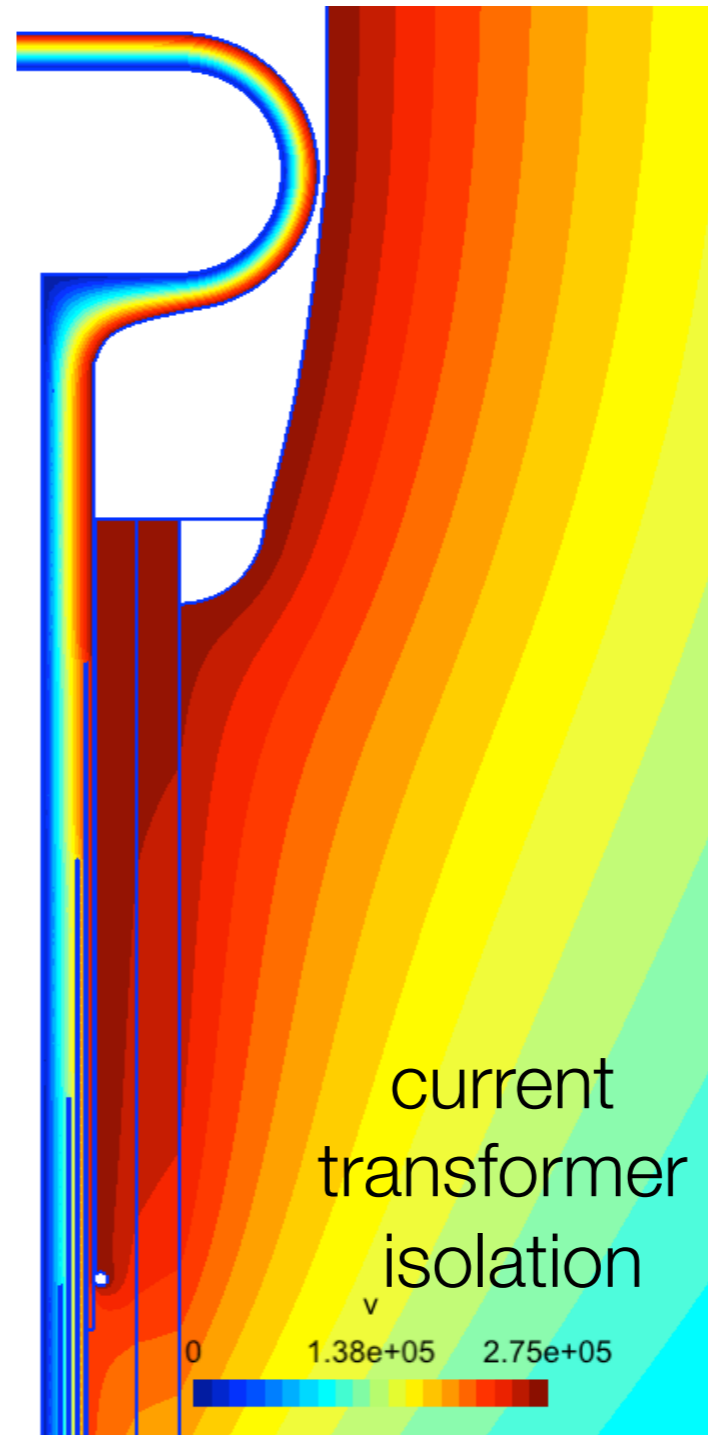
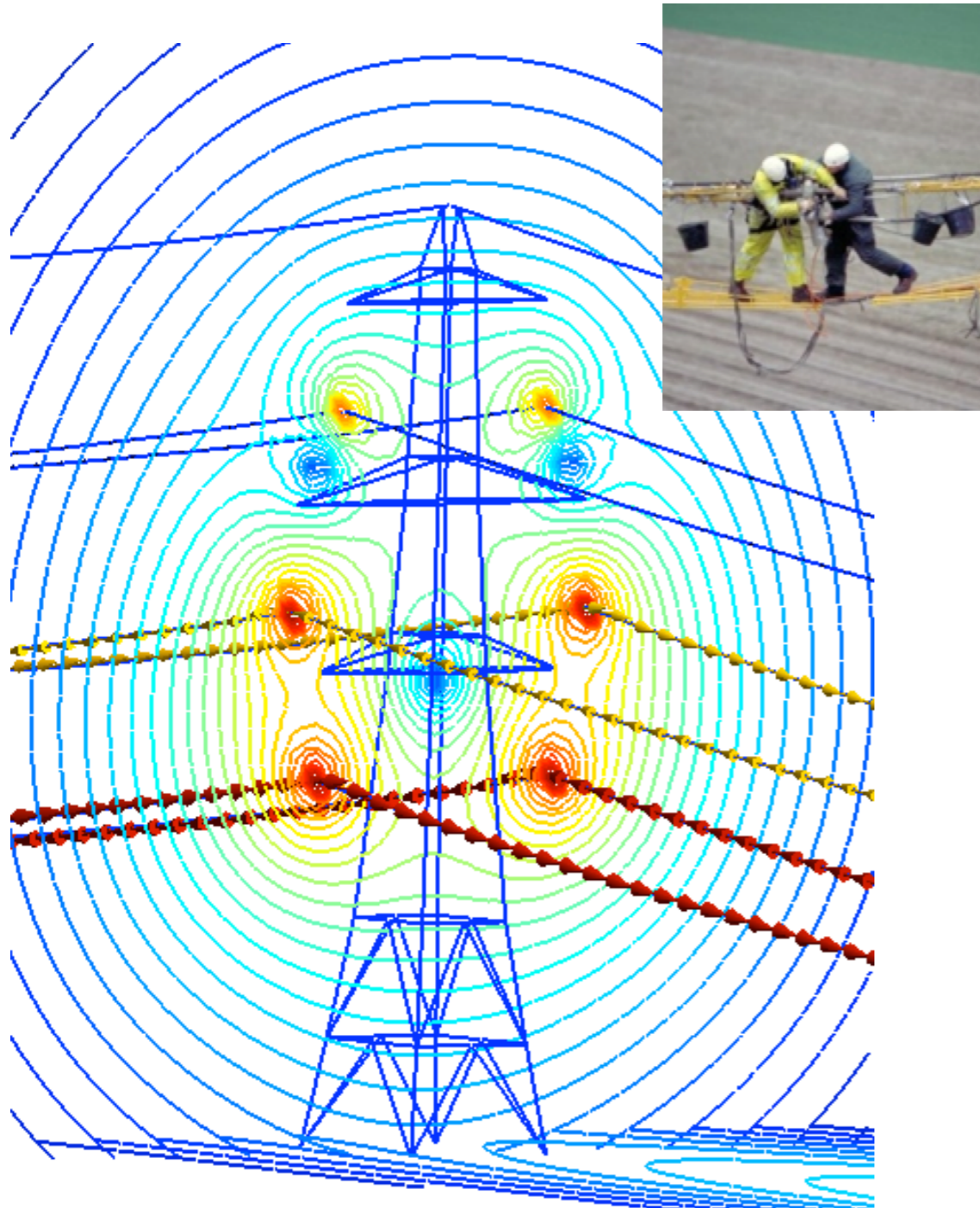
in each conducting region
 imposed potential at boundary

$$\Omega_{c,i}, \quad v = v_i \Rightarrow v|_{\Gamma_{c,i}} = v_i$$



Electrostatics

field next to a 220 kV high voltage line



Electrokinetics

$$\text{curl } \mathbf{e} = 0$$

$$\text{curl } \mathbf{h} = \mathbf{j} \Rightarrow \text{div } \mathbf{j} = 0$$

$$\mathbf{j} = \sigma \mathbf{e}$$

boundary conditions (BCs)

$$\mathbf{n} \times \mathbf{e}|_{\Gamma_e} = 0$$

$$\mathbf{n} \cdot \mathbf{j}|_{\Gamma_j} = 0$$

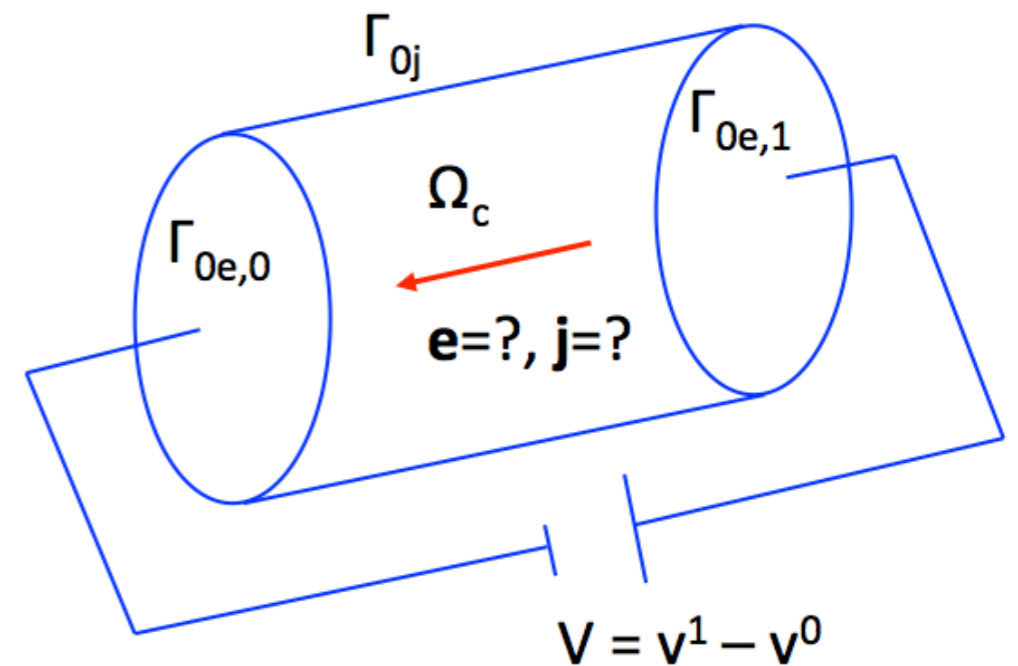
electric scalar potential formulation

$$-\text{div}(\sigma \text{grad } v) = 0, \quad \mathbf{e} = -\text{grad } v$$

formulation for the conducting region Ω_c

in each electrode
imposed potential

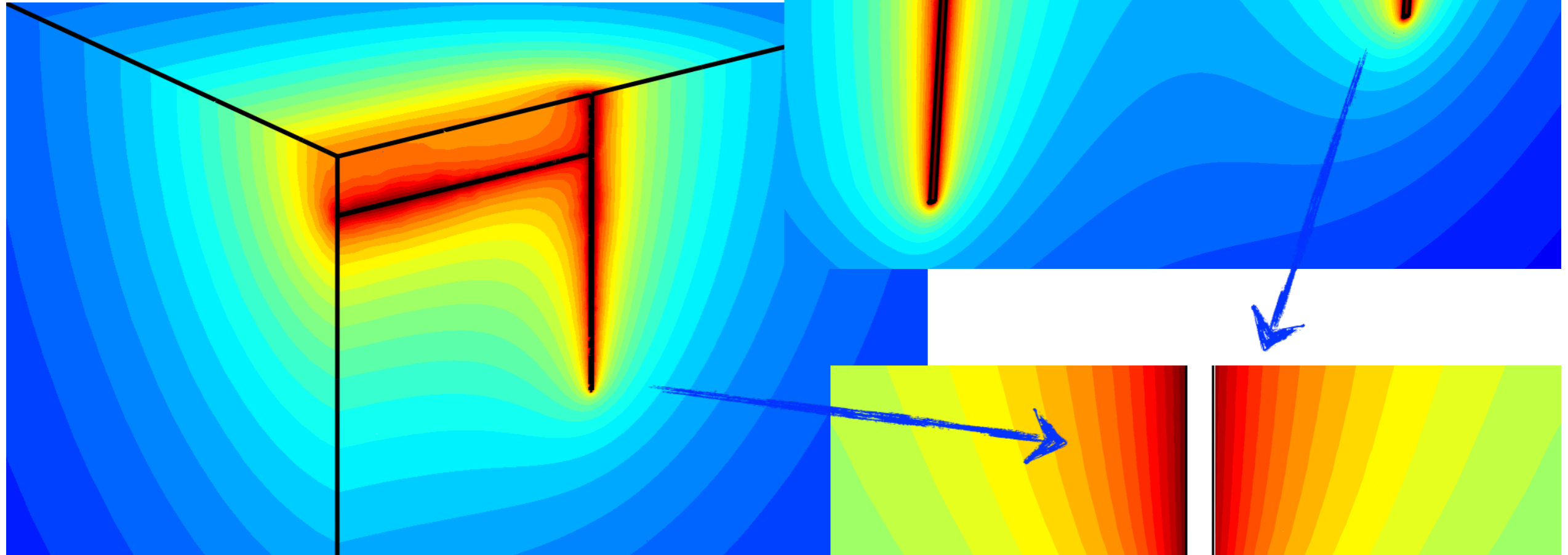
$$\Gamma_{e,i}, \quad v = v_i \Rightarrow v|_{\Gamma_{e,i}} = v_i$$



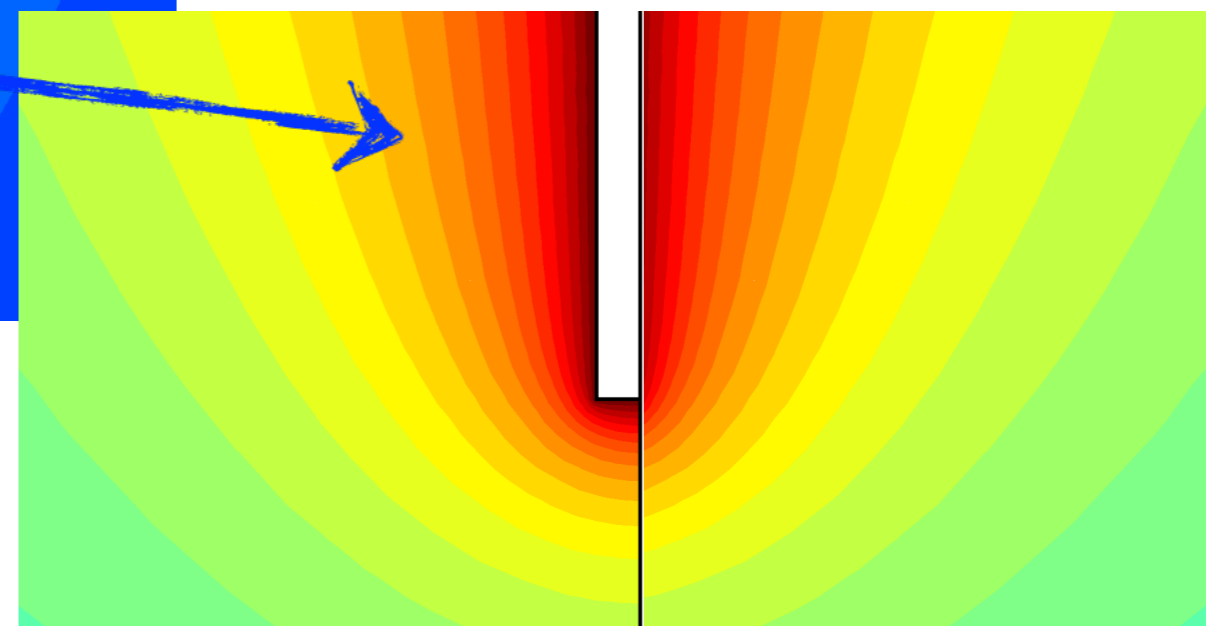
Electrokinetics



grounding systems:
combination of rods & cables



electric potential distribution



Electrodynamics

$$\text{curl } \mathbf{e} = 0$$

$$\text{curl } \mathbf{h} = \mathbf{j} + \partial_t \mathbf{d} \Rightarrow \text{div } (\mathbf{j} + \partial_t \mathbf{d}) = 0$$

$$\mathbf{j} = \sigma \mathbf{e}$$

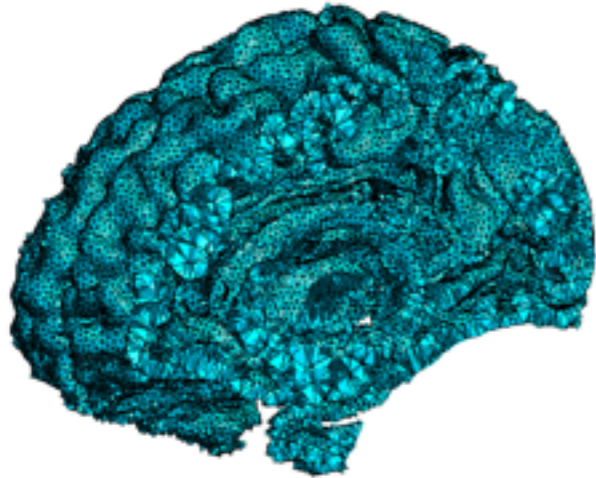
$$\mathbf{d} = \epsilon \mathbf{e}$$

electric scalar potential formulation

$$-\text{div } (\sigma \text{ grad } v + \epsilon \text{ grad } \partial_t v) = 0, \quad \mathbf{e} = -\text{grad } v$$

Electrodynamics

Tissue	Conductivity (S/m)
Muscle	0.2330
White matter	0.0533
Grey matter	0.0753 - 0.5155
Cerebellum	0.0953 - 0.3020
Eyes	1.5000
Spinal chord	0.0274
Lungs	0.0684
Heart	0.0827
Veins	0.7000
Liver	0.0367
-	-



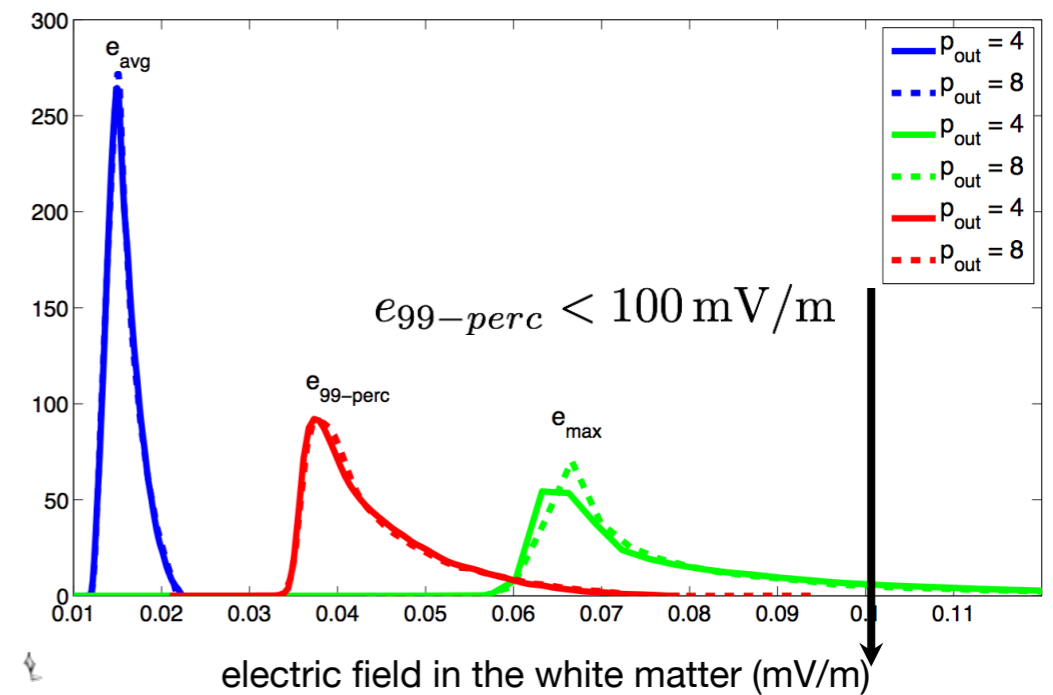
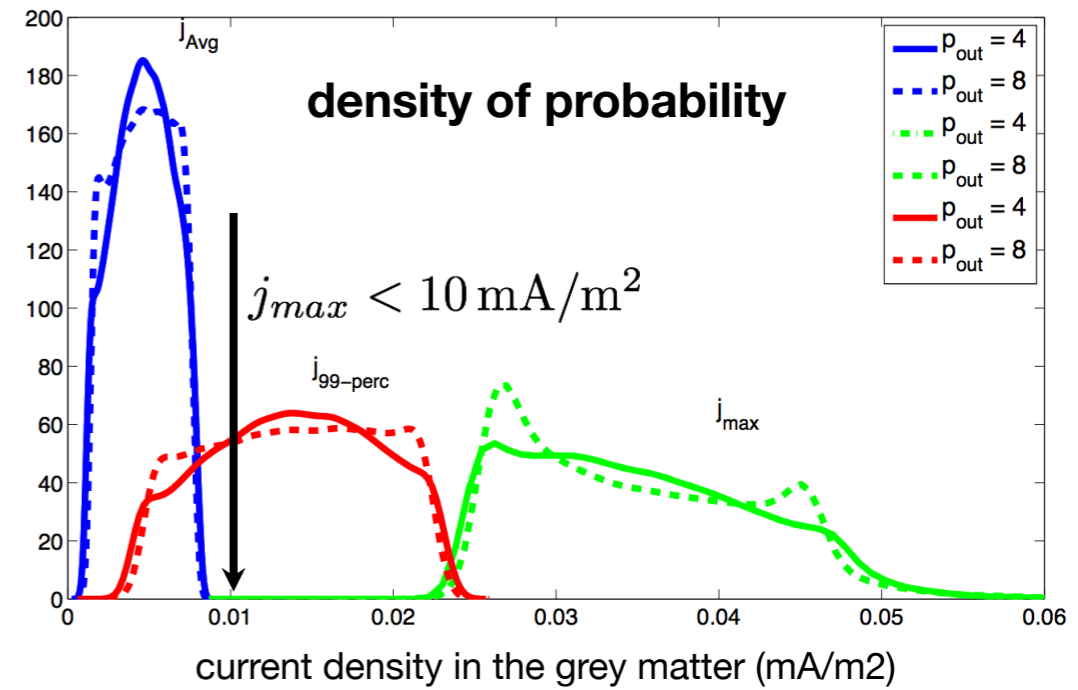
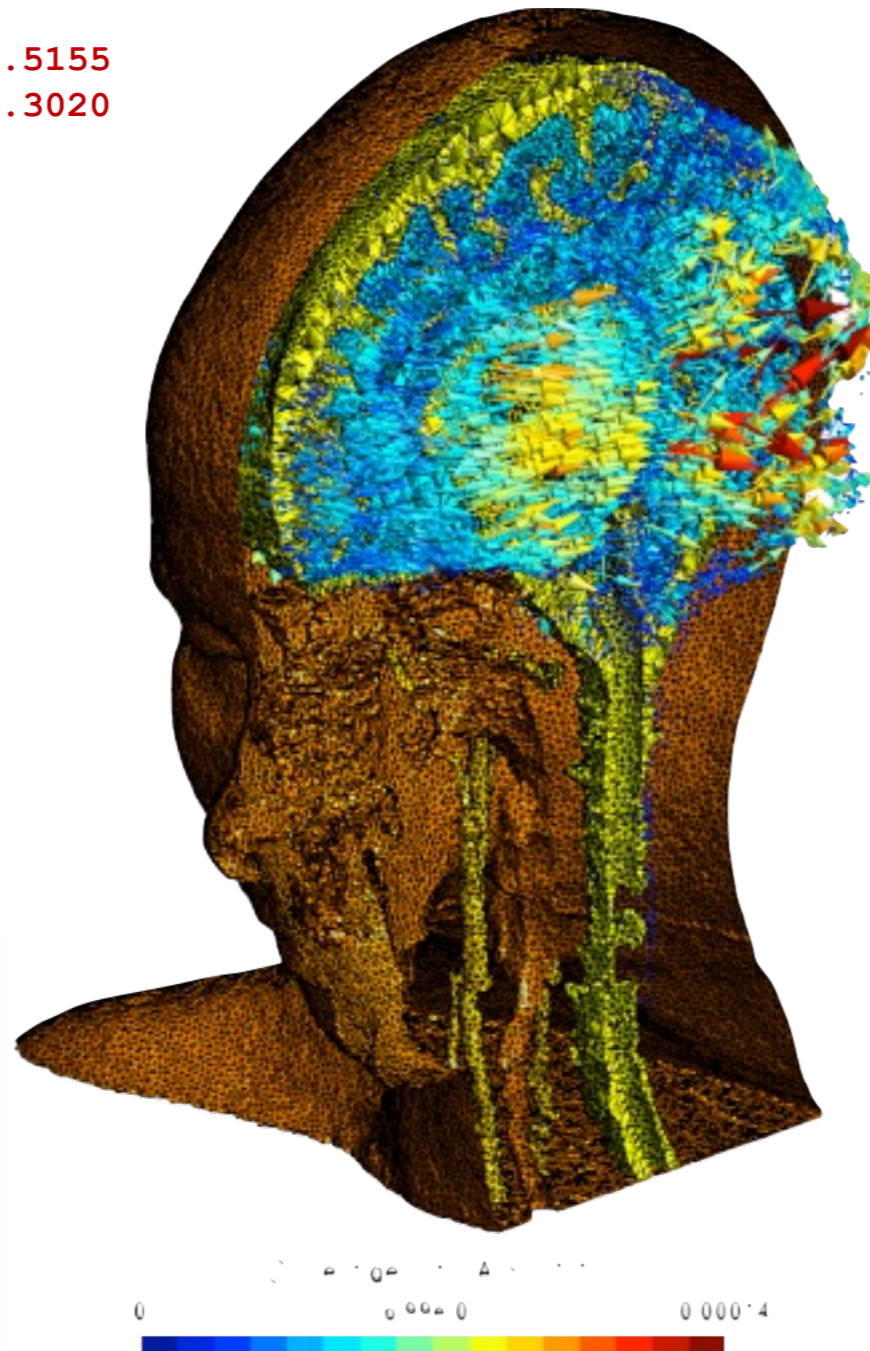
ELLA

Iso2mesh

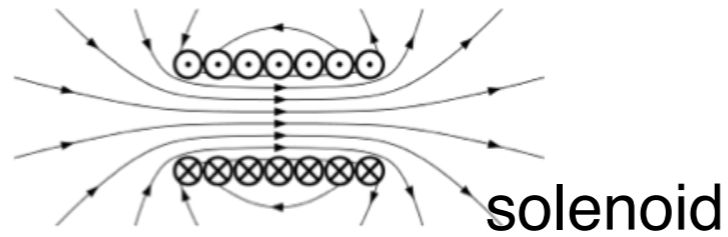
45 different tissues

1e6 nodes

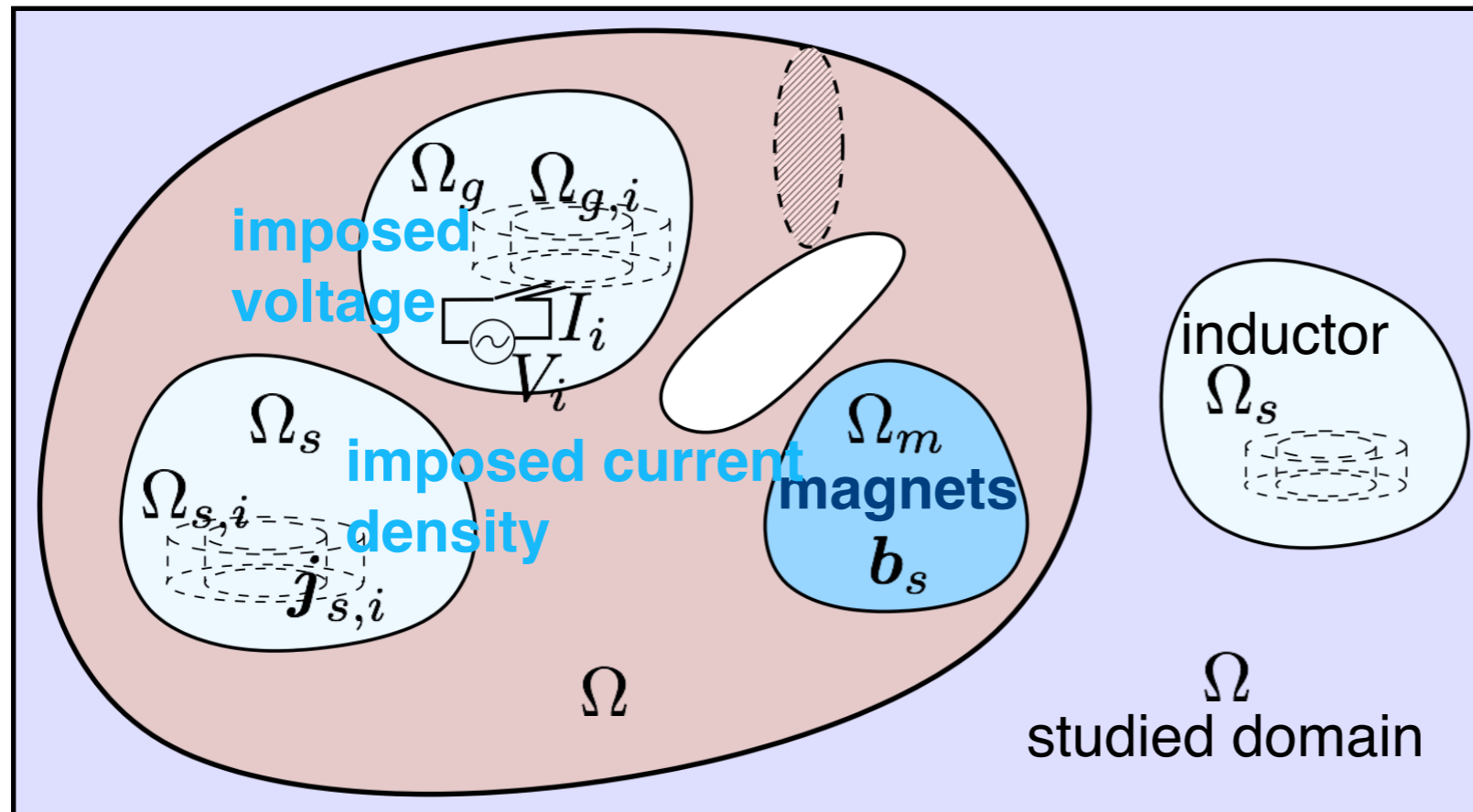
5.8e6 tetrahedra



Magnetostatics



$$L = \frac{\Phi}{m.m.f} = n^2 \frac{\mu_0 S}{l}$$



$$\text{curl } \mathbf{h} = \mathbf{j}_s$$

$$\text{div } \mathbf{b} = 0$$

$$\mathbf{b} = \mu \mathbf{h} (+ \mathbf{b}_s)$$

$$\mathbf{h} = \nu \mathbf{b} (+ \mathbf{h}_s)$$

magnetic vector potential formulation

$$\text{curl } \nu \text{ curl } \mathbf{a} = \mathbf{j}_s, \quad \mathbf{b} = \text{curl } \mathbf{a}$$

Possible sources:

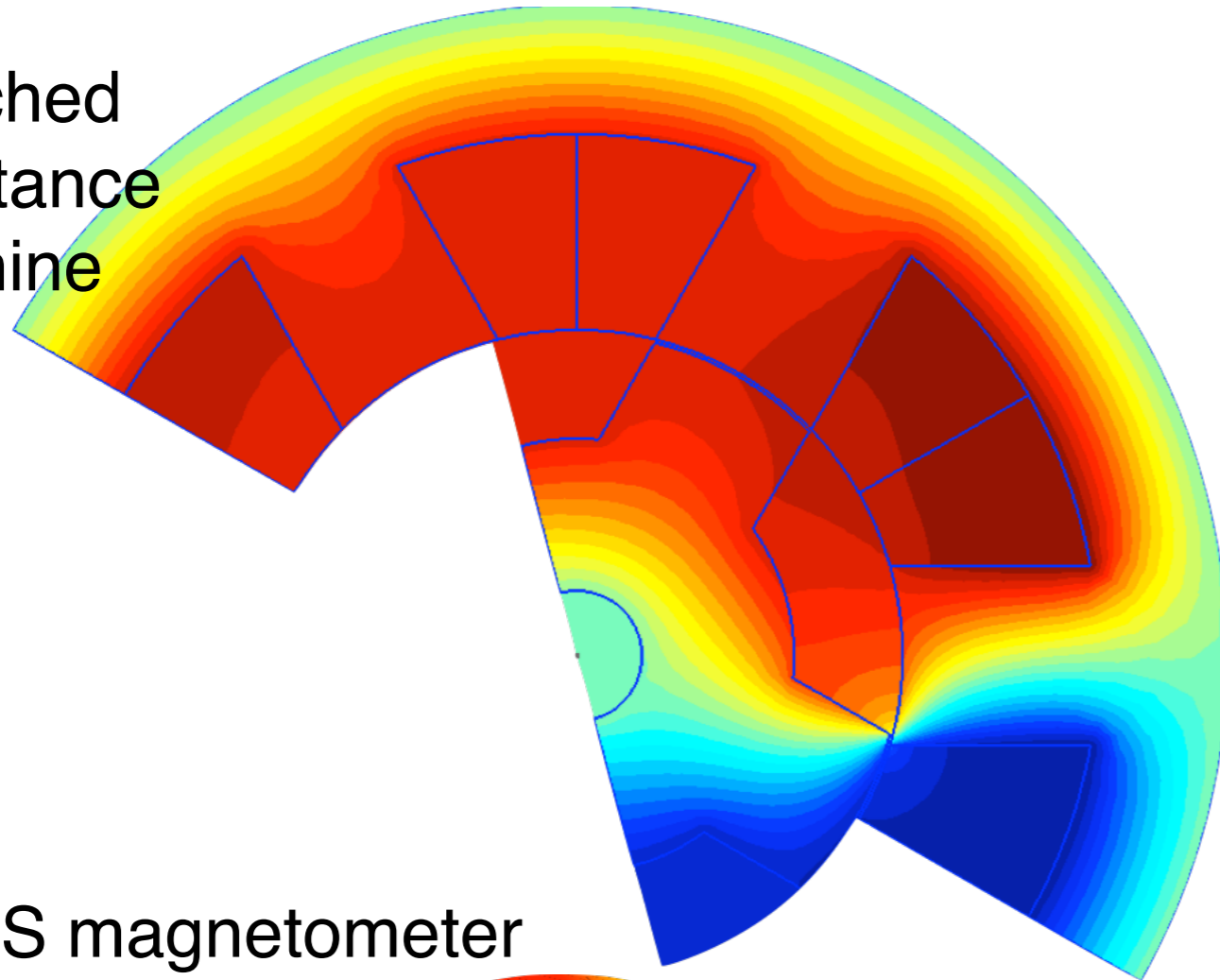
\mathbf{j}_s imposed current density in inductor

\mathbf{b}_s remanent induction if magnets

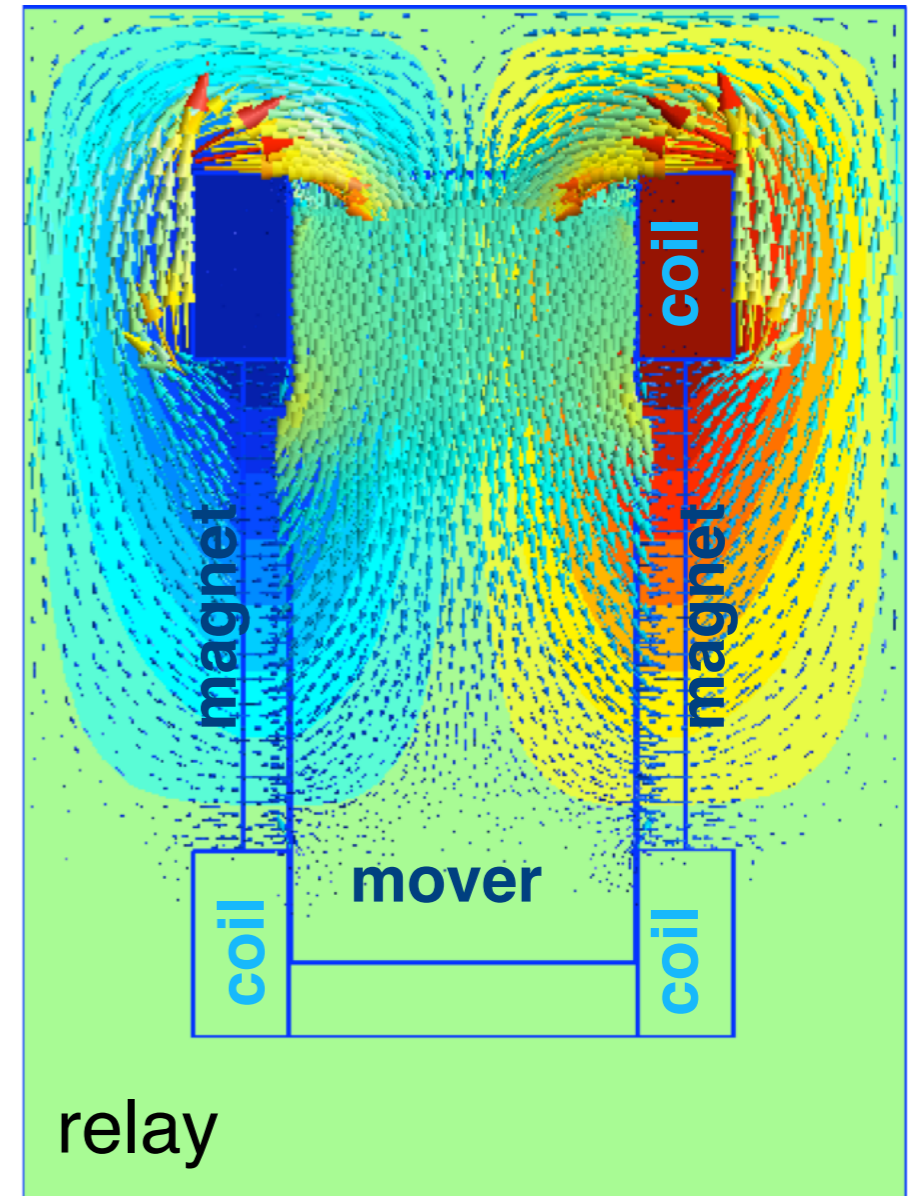
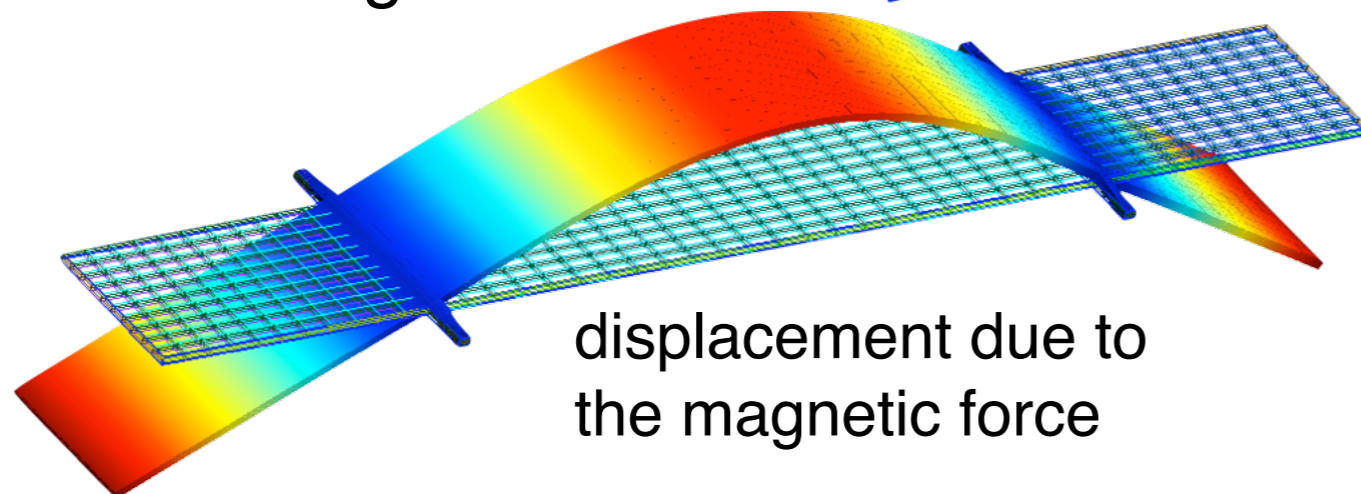
\mathbf{h}_s remanent magnetic field if magnets

Magnetostatics

Switched
reluctance
machine



MEMS magnetometer



Magnetodynamics

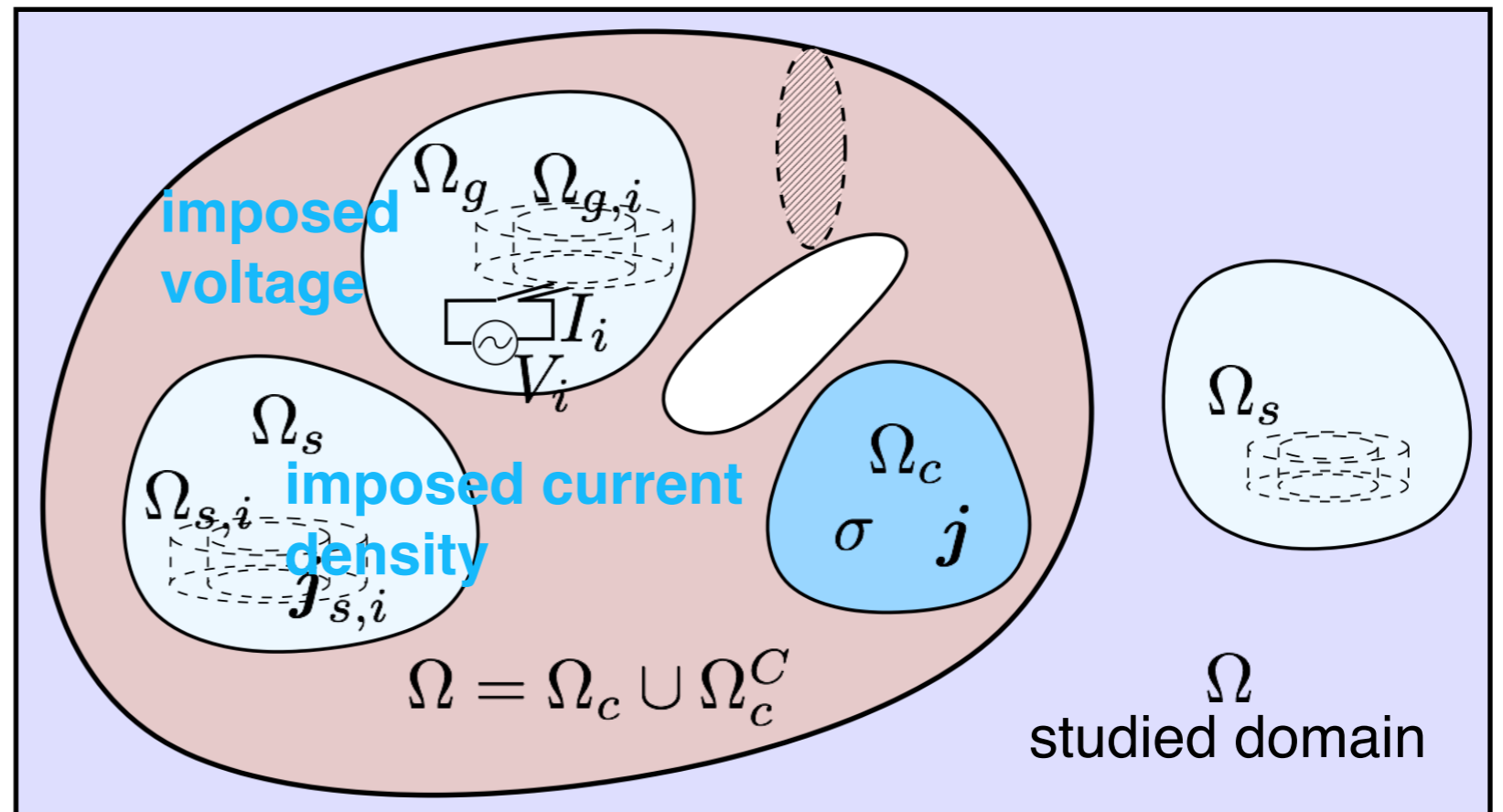
$$\text{curl } \mathbf{h} = \mathbf{j}$$

$$\text{curl } \mathbf{e} = -\partial_t \mathbf{b}$$

$$\text{div } \mathbf{b} = 0$$

$$\mathbf{b} = \mu \mathbf{h} + \mathbf{b}_s$$

$$\mathbf{j} = \sigma \mathbf{e} + \mathbf{j}_s$$



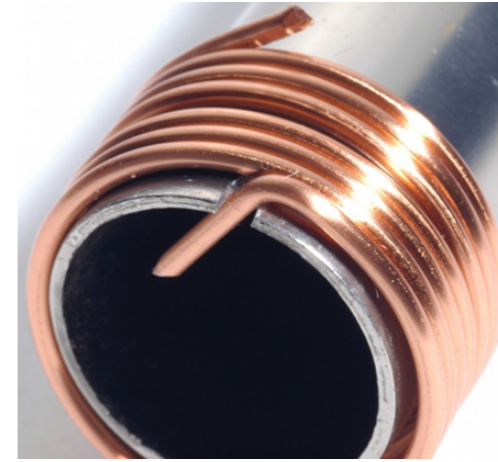
magnetic vector potential formulation

$$\text{curl } \nu \text{ curl } \mathbf{a} + \sigma (\partial_t \mathbf{a} + \text{grad } v) = \mathbf{j}_s, \quad \mathbf{b} = \text{curl } \mathbf{a}$$

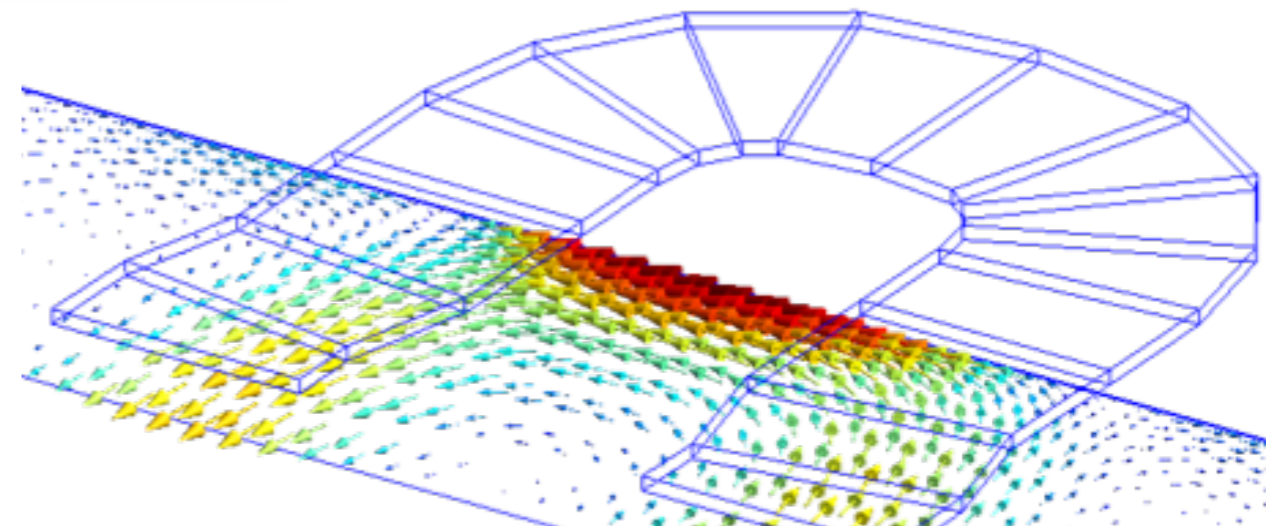
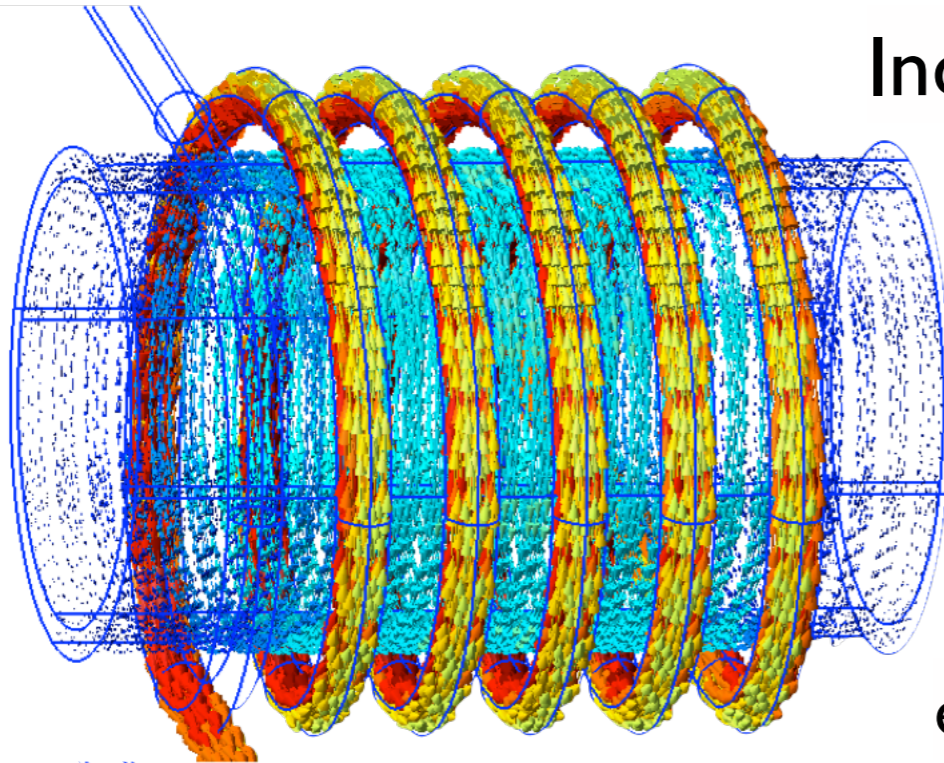
$$\text{reluctivity } \nu = \frac{1}{\mu}$$

$$\mathbf{e} = -\text{grad } v - \partial_t \mathbf{a}$$

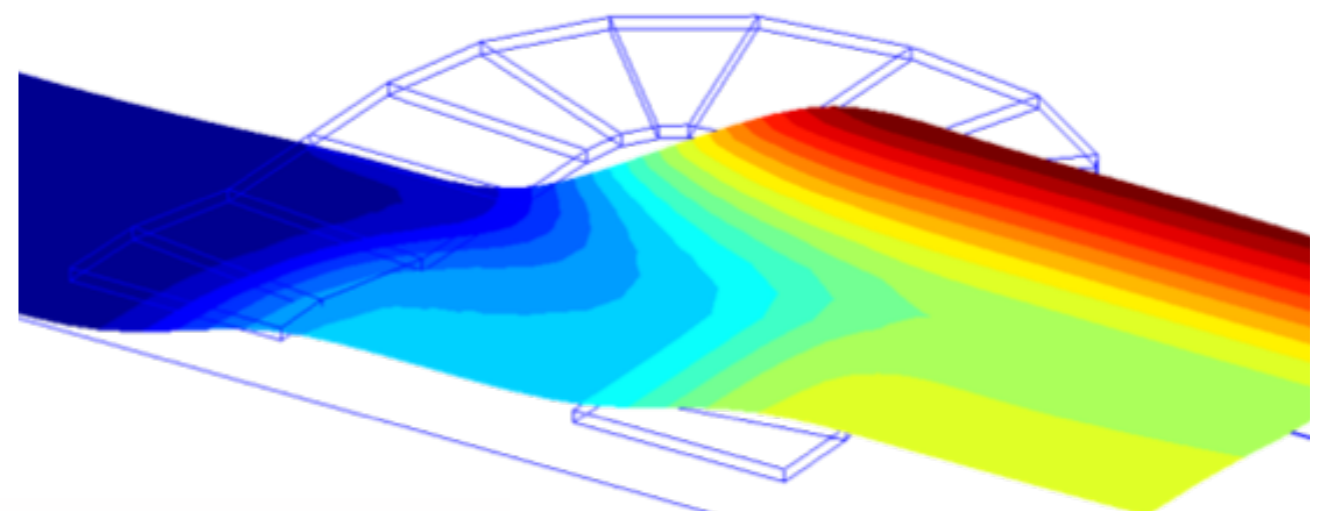
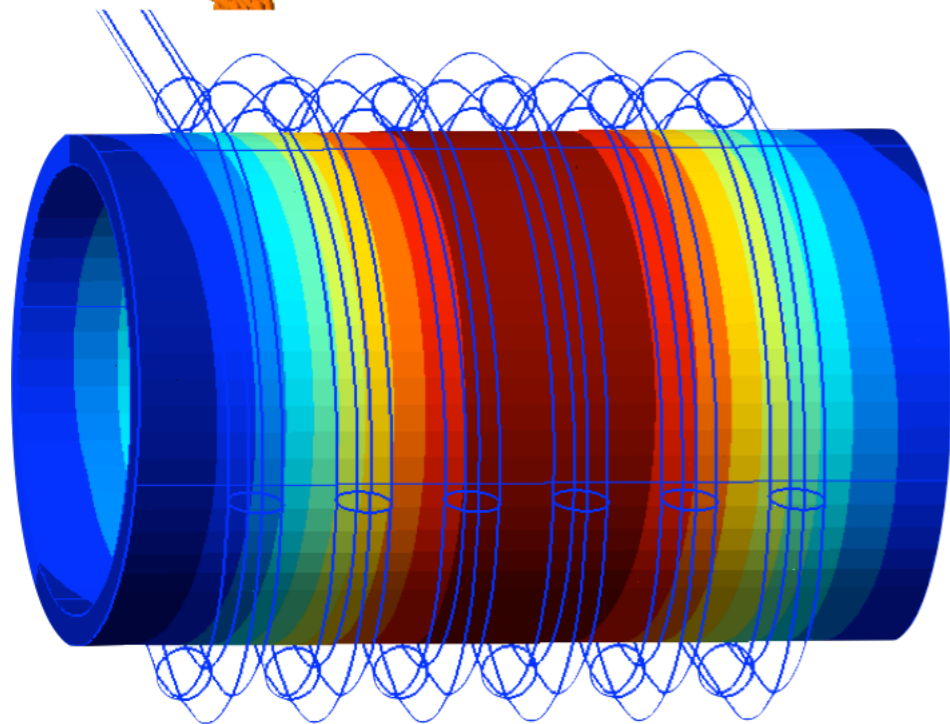
Magnetodynamics



Induction heating



eddy current distribution



temperature distribution

Full wave

$$\text{curl } \mathbf{h} = \mathbf{j} + \partial_t \mathbf{d}$$

$$\text{curl } \mathbf{e} = -\partial_t \mathbf{b}$$

$$\mathbf{b} = \mu \mathbf{h}$$

$$\mathbf{d} = \epsilon \mathbf{e}$$

$$\mathbf{j} = \sigma \mathbf{e}$$

total electric field = scattered field + incident field

$$\mathbf{e} = \mathbf{e}_s + \mathbf{e}_{inc}$$

$$\mathbf{e}_s = (\mathbf{n} \times \mathbf{e}) \times \mathbf{n}$$

+ Silver-Müller radiation condition at infinity
(outgoing waves)

$$\text{curl } \mathbf{e} \times \mathbf{n} - ik((\mathbf{n} \times \mathbf{e}) \times \mathbf{n}) =$$

$$\text{curl } \mathbf{e}_{inc} \times \mathbf{n} - ik((\mathbf{n} \times \mathbf{e}_{inc}) \times \mathbf{n}), \text{ on } \Gamma$$

electric or magnetic field formulation

time domain

$$\text{curl curl } \mathbf{e} + \sigma \mu \partial_t \mathbf{e} + \epsilon \mu \partial_t^2 \mathbf{e} = 0$$

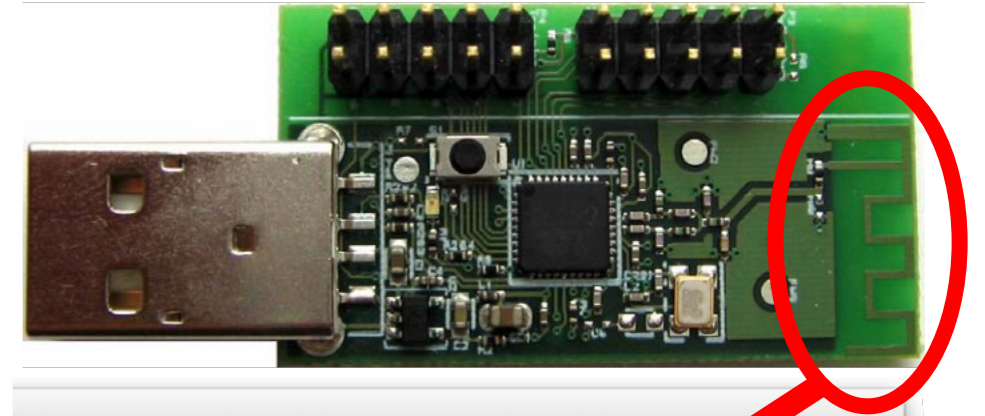
$$\text{curl curl } \mathbf{h} + \sigma \mu \partial_t \mathbf{h} + \epsilon \mu \partial_t^2 \mathbf{h} = 0$$

frequency domain $\omega = 2\pi f$

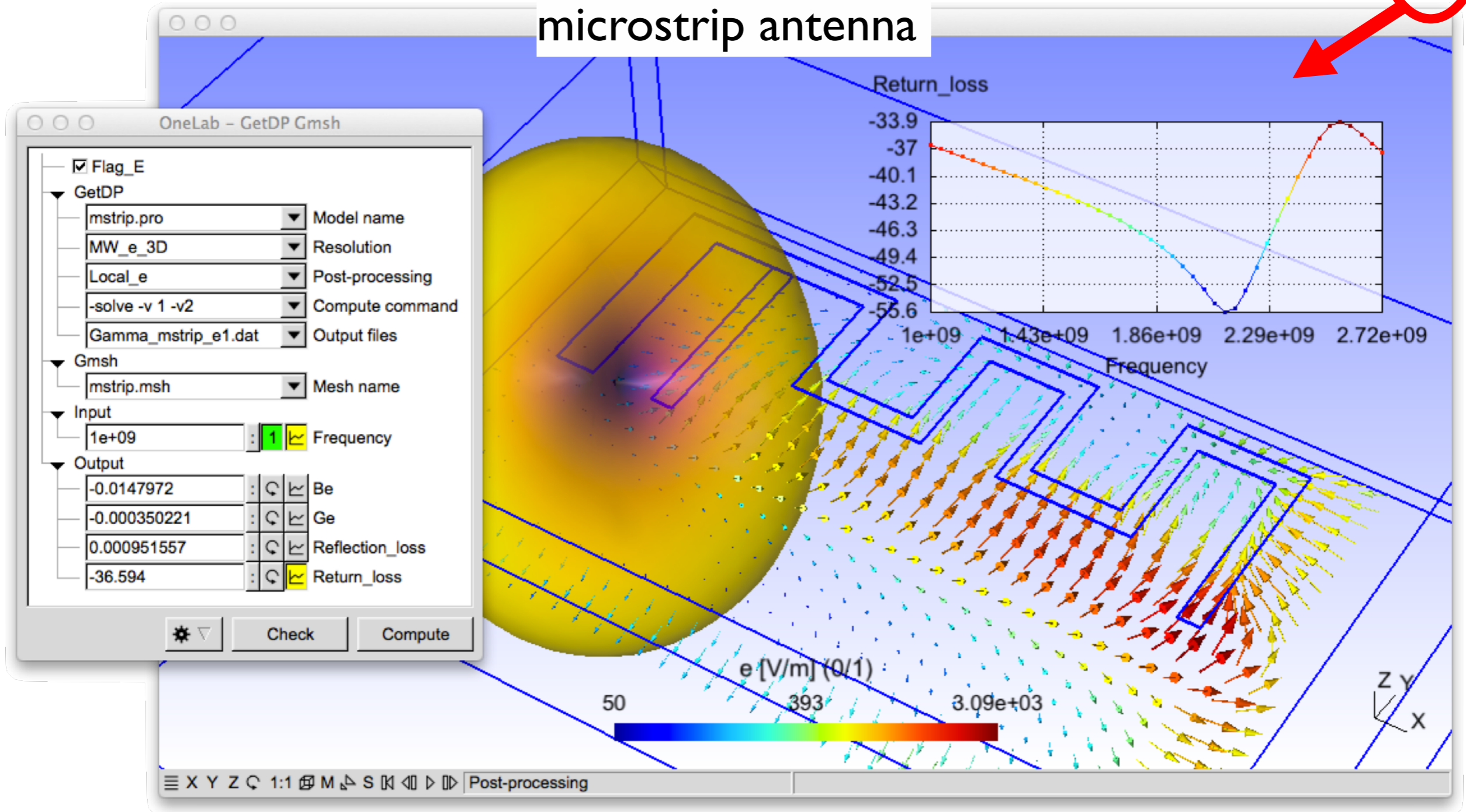
$$\Delta \mathbf{e} - i\omega \sigma \mu \mathbf{e} + \omega^2 \epsilon \mu \mathbf{e} = 0$$

$$\Delta \mathbf{h} - i\omega \sigma \mu \mathbf{h} + \omega^2 \epsilon \mu \mathbf{h} = 0$$

Full wave

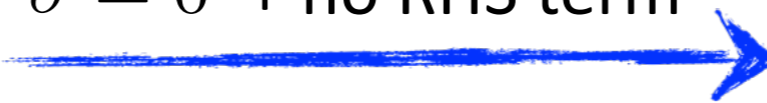


microstrip antenna



Model choice

Maxwell's equations & constitutive relations in frequency domain, without sources:

$$\begin{aligned} \Delta \mathbf{e} - \omega \sigma \mu \mathbf{e} + \omega^2 \epsilon \mu \mathbf{e} &= 0 & \sigma = 0 \text{ + no RHS term} \\ \Delta \mathbf{h} - \omega \sigma \mu \mathbf{h} + \omega^2 \epsilon \mu \mathbf{h} &= 0 & k^2 = \omega^2 \epsilon \mu \end{aligned}$$


Helmholtz equations

$$\Delta \mathbf{e} + k^2 \mathbf{e} = 0$$

$$\Delta \mathbf{h} + k^2 \mathbf{h} = 0$$

Using characteristic lengths

- domain size L
- skin depth $\delta = \sqrt{\frac{2}{\omega \sigma \mu}}$

- wavelength $\lambda = \frac{2\pi}{k}$, wave number $k = \frac{\omega}{c}$, speed of light $c = \frac{1}{\sqrt{\epsilon \mu}}$

allows to write them in non-dimensional form:

$$\left(\frac{3}{L^2} - \frac{2i}{\delta^2} + \frac{4\pi}{\lambda^2} \right) \mathbf{e} = 0$$

$$\left(\frac{3}{L^2} - \frac{2i}{\delta^2} + \frac{4\pi}{\lambda^2} \right) \mathbf{h} = 0$$

Model choice

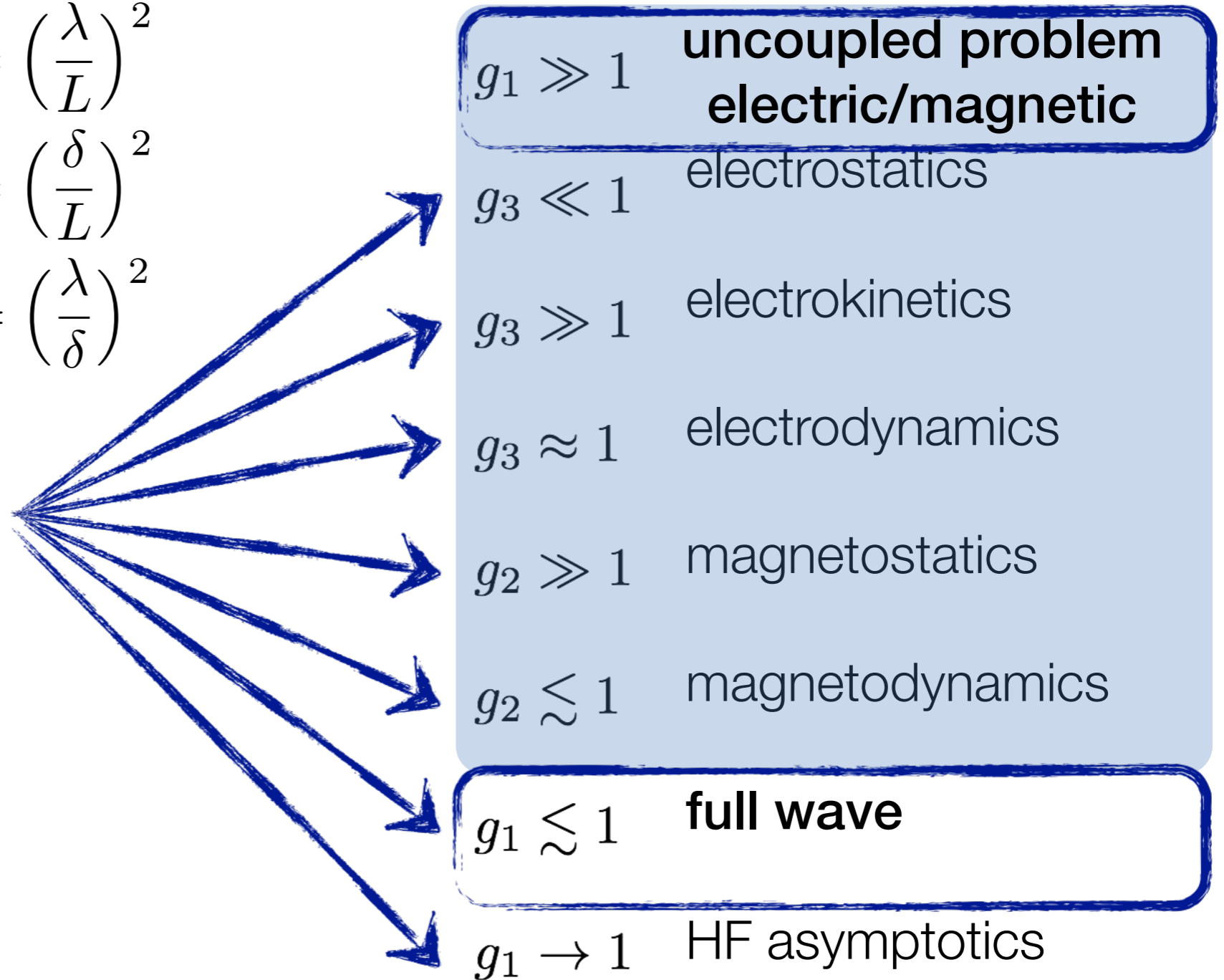
Non-dimensional numbers






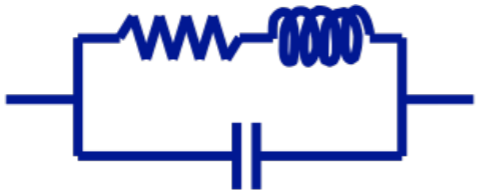
$$\left\{ \begin{array}{l} g_1 = \left(\frac{\lambda}{L}\right)^2 \\ g_2 = \left(\frac{\delta}{L}\right)^2 \\ g_3 = \left(\frac{\lambda}{\delta}\right)^2 \end{array} \right.$$

Maxwell's equations

$$\left(\frac{3}{L^2} - \frac{2i}{\delta^2} + \frac{4\pi}{\lambda^2}\right) \mathbf{e} = 0$$

$$\left(\frac{3}{L^2} - \frac{2i}{\delta^2} + \frac{4\pi}{\lambda^2}\right) \mathbf{h} = 0$$



EM model	governing equations
electrostatics 	$\text{curl } \mathbf{e} = 0, \text{ div } \mathbf{d} = q$ $\mathbf{d} = \epsilon \mathbf{e}$
electrokinetics 	$\text{curl } \mathbf{e} = 0, \text{ div } \mathbf{j} = 0$ $\mathbf{j} = \sigma \mathbf{e}$
electrodynamics 	$\text{curl } \mathbf{e} = 0, \text{ div } (\mathbf{j} + \partial_t \mathbf{d}) = 0$ $\mathbf{j} = \sigma \mathbf{e}, \mathbf{d} = \epsilon \mathbf{e}$
magnetostatics 	$\text{curl } \mathbf{h} = \mathbf{j}, \text{ div } \mathbf{b} = 0$ $\mathbf{b} = \mu \mathbf{h}, \mathbf{h} = \nu \mathbf{b}$
magnetodynamics 	$\text{curl } \mathbf{h} = \mathbf{j}, \text{ curl } \mathbf{e} = -\partial_t \mathbf{b}, \text{ div } \mathbf{b} = 0$ $\mathbf{b} = \mu \mathbf{h}, \mathbf{j} = \sigma \mathbf{e}$
full wave 	$\text{curl } \mathbf{h} = \mathbf{j} + \partial_t \mathbf{d}, \text{ curl } \mathbf{e} = -\partial_t \mathbf{b}, \text{ div } \mathbf{b} = 0$ $\mathbf{b} = \mu \mathbf{h}, \mathbf{j} = \sigma \mathbf{e}, \mathbf{d} = \epsilon \mathbf{e}$

Continuous mathematical structure

Maxwell's equations

$$\text{grad } f_0 \equiv \nabla f_0 = (\partial_x, \partial_y, \partial_z) f_0$$

$$\text{curl } \mathbf{f}_1 \equiv \nabla \times \mathbf{f}_1 \equiv (\partial_x, \partial_y, \partial_z) \times \mathbf{f}_1$$

$$\text{div } \mathbf{f}_2 \equiv \nabla \cdot \mathbf{f}_2 \equiv (\partial_x, \partial_y, \partial_z) \cdot \mathbf{f}_2$$

$$\text{curl } \mathbf{h} - \partial_t \mathbf{d} = \mathbf{j}$$

$$\text{curl } \mathbf{e} + \partial_t \mathbf{b} = 0$$

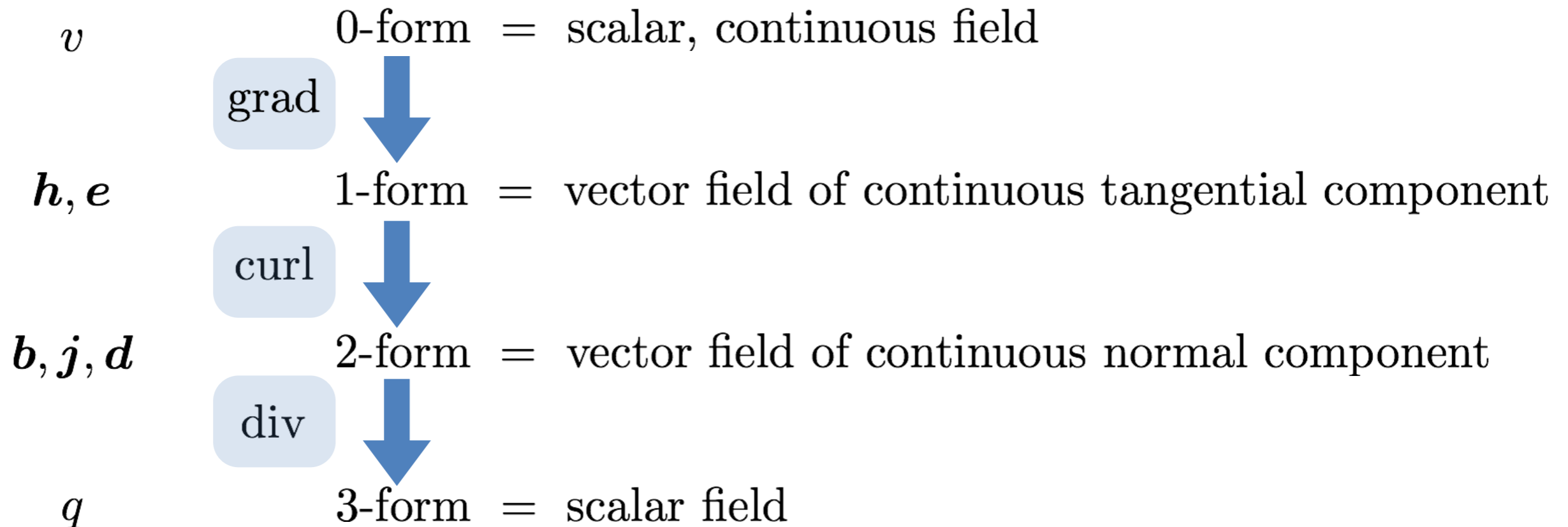
$$\text{div } \mathbf{b} = 0$$

$$\text{div } \mathbf{d} = q$$

$$\mathbf{b} = \mathcal{B}(\mathbf{e}, \mathbf{h}) = \mu \mathbf{h} (+\mathbf{b}_s)$$

$$\mathbf{d} = \mathcal{D}(\mathbf{e}, \mathbf{h}) = \epsilon \mathbf{e} (+\mathbf{d}_s)$$

$$\mathbf{j} = \mathcal{J}(\mathbf{e}, \mathbf{h}) = \sigma \mathbf{e} (+\mathbf{j}_s)$$



Square integrable vector fields

Useful definitions

The solutions of Maxwell's equations belong to spaces of square integrable scalar and vector fields $L^2(\Omega)$ and $\mathbf{L}^2(\Omega)$. They are defined by

$$L^2(\Omega) = \left\{ u : \int_{\Omega} u^2(\mathbf{x}) d\mathbf{x} < \infty \right\}$$

$$\mathbf{L}^2(\Omega) = \left\{ \mathbf{u} : \int_{\Omega} \|\mathbf{u}(\mathbf{x})\|^2 d\mathbf{x} < \infty \right\}$$

The scalar product of $u, v \in L^2(\Omega)$ and $\mathbf{u}, \mathbf{v} \in \mathbf{L}^2(\Omega)$ is defined by

$$(u, v)_{\Omega} = \int_{\Omega} u(\mathbf{x}) v(\mathbf{x}) d\mathbf{x} \quad \text{and} \quad (\mathbf{u}, \mathbf{v})_{\Omega} = \int_{\Omega} \mathbf{u}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x}) d\mathbf{x}$$

The norm of $u \in L^2(\Omega)$ and $\mathbf{u} \in \mathbf{L}^2(\Omega)$ is defined by

$$\|u\|_{L^2(\Omega)} = (u, u)_{\Omega}^{1/2} = \left[\int_{\Omega} u^2(\mathbf{x}) d\mathbf{x} \right]^{1/2}$$

$$\|\mathbf{u}\|_{\mathbf{L}^2(\Omega)} = (\mathbf{u}, \mathbf{u})_{\Omega}^{1/2} = \left[\int_{\Omega} \|\mathbf{u}(\mathbf{x})\|^2 d\mathbf{x} \right]^{1/2}$$

Classification of 2nd order PDEs

EM problems involve linear second order partial differential equations (PDE), for the unknown field u and the source function $f(x,y)$, that can be written (2D):

$$\mathcal{L}(u) = a \frac{\partial^2 u(x, y)}{\partial x^2} + b \frac{\partial^2 u(x, y)}{\partial x \partial y} + c \frac{\partial^2 u(x, y)}{\partial y^2} + d \frac{\partial u(x, y)}{\partial x} + e \frac{\partial u(x, y)}{\partial y} + gu = f$$

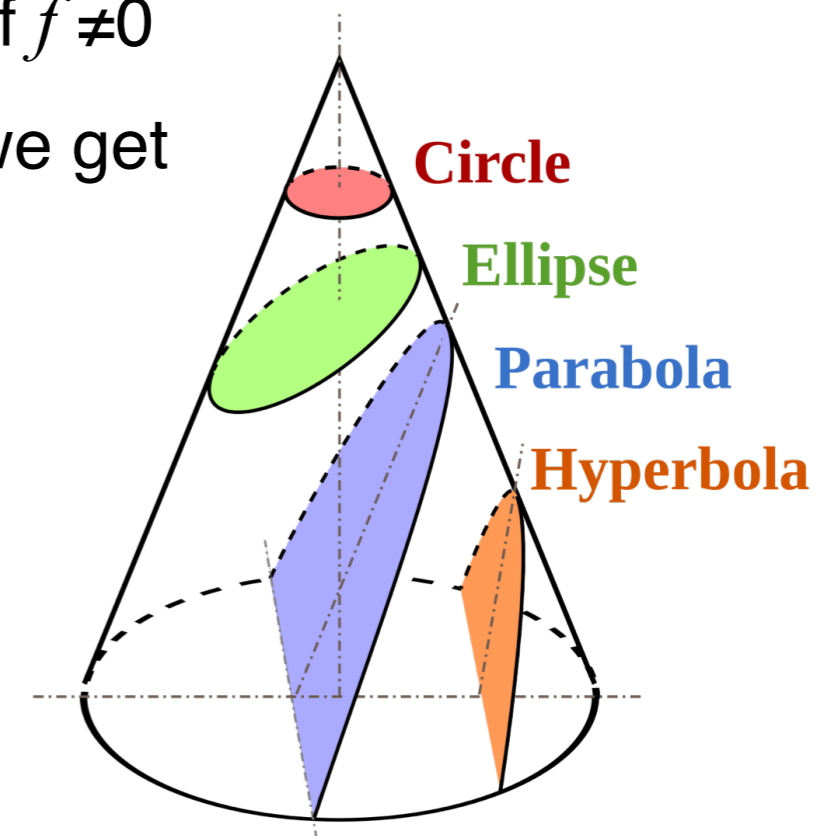
with coefficients a, b, c, d, e, g are functions of x, y , but they can also depend on u (nonlinear PDE), homogeneous if $f=0$, inhomogeneous if $f \neq 0$

From the homogeneous case with constant coefficients, we get

$$ax^2 + bxy + cy^2 + dx + ey + g = 0$$

$$\begin{vmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & g \end{vmatrix} = (ac - b^2/4)g + bed/4 - cd^2/4 - ae^2/4$$

$$\begin{cases} b^2 - 4ac < 0 & \text{elliptic} & \Rightarrow & \text{steady-state problems} \\ b^2 - 4ac = 0 & \text{parabolic} & \Rightarrow & \text{diffusion problems} \\ b^2 - 4ac > 0 & \text{hyperbolic} & \Rightarrow & \text{propagation problems} \end{cases}$$



Classification of 2nd order PDEs (cont'd)

elliptic e.g. electrostatics $\mathbf{e} = -\text{grad } v, \mathbf{d} = \epsilon \mathbf{e}$

$$-\Delta v = -\frac{\partial^2 v(x, y, z)}{\partial x^2} - \frac{\partial^2 v(x, y, z)}{\partial y^2} - \frac{\partial^2 v(x, y, z)}{\partial z^2} = \frac{q}{\epsilon}$$

$$\mathbf{n} \times \mathbf{e}|_{\Gamma_e} = 0$$

$$\mathbf{n} \cdot \mathbf{d}|_{\Gamma_d} = 0$$

parabolic e.g. magnetodynamics

$$\text{curl } \nu \text{ curl } \mathbf{a} + \sigma (\partial_t \mathbf{a} + \text{grad } v) = \mathbf{j}_s, \quad \mathbf{b} = \text{curl } \mathbf{a}$$

$$\mathbf{n} \times \mathbf{h}|_{\Gamma_h} = 0$$

$$\mathbf{n} \cdot \mathbf{b}|_{\Gamma_b} = 0 \quad \mathbf{a}(t = t_0) = \mathbf{a}_0 \quad \mathbf{e} = -\text{grad } v - \partial_t \mathbf{a}$$

hyperbolic e.g. full wave in time domain

$$\text{curl curl } \mathbf{e} + \sigma \mu \partial_t \mathbf{e} + \epsilon \mu \partial_t^2 \mathbf{e} = 0$$

$$\mathbf{n} \times (\mathbf{e}^{tot} - \mathbf{e}^{inc}) + \sqrt{\frac{\mu}{\epsilon}} \mathbf{n} \times (\mathbf{n} \times (\mathbf{h}^{tot} - \mathbf{h}^{inc})) = 0$$

$$\mathbf{e}(t = t_n) = \mathbf{e}_n$$

$$\mathbf{e}(t = t_{n-1}) = \mathbf{e}_{n-1}$$

Strong and weak formulations

notation

$$(u, v)_\Omega = \int_\Omega u \cdot v \, d\Omega$$

$$\langle u, v \rangle_\Gamma = \int_\Gamma u \cdot v \, d\Gamma$$

Strong formulation

$$\mathcal{L}u = f \text{ in } \Omega$$

$$\mathcal{B}u = g \text{ on } \Gamma$$

- ✓ \mathcal{L} differential operator of order n
- ✓ \mathcal{L}^* adjoint of \mathcal{L}
- ✓ \mathcal{B} differential operator imposing BC
- ✓ f function in Ω , g function on $\Gamma = \partial\Omega$
- ✓ u unknown function
- ✓ Q_g linear function of v

Weak formulation

find u so that

$$(u, \mathcal{L}^*v)_\Omega - (f, v)_\Omega + \int_\Gamma Q_g(v) \, ds = 0, \quad \forall v \in V(\Omega)$$

Continuous system $\Rightarrow \infty \times \infty$
 Discrete system $\Rightarrow N \times N$
 \Rightarrow **numerical solution**

Green formulae

$$(\mathcal{L}u, v)_\Omega - (u, \mathcal{L}^*v)_\Omega = \int_\Gamma Q(u, v) \, ds$$

Q bilinear function of u and v

$$\left\{ \begin{array}{l} \text{grad-div type} \\ (\mathbf{v}, \text{grad } u)_\Omega + (\text{div } \mathbf{v}, u)_\Omega = \langle u, \hat{n} \cdot \mathbf{v} \rangle_\Gamma \\ \text{curl-curl type} \\ (\mathbf{v}, \text{curl } \mathbf{w})_\Omega - (\text{curl } \mathbf{v}, \mathbf{w})_\Omega = \langle \mathbf{v} \times \hat{n}, \mathbf{w} \rangle_\Gamma \end{array} \right.$$

Strong and weak formulations (cont'd)

e.g. strong electrostatic formulation $\mathbf{e} = -\text{grad } v, \mathbf{d} = \epsilon \mathbf{e}$

$$-\Delta v = -\frac{\partial^2 v(x, y, z)}{\partial x^2} - \frac{\partial^2 v(x, y, z)}{\partial y^2} - \frac{\partial^2 v(x, y, z)}{\partial z^2} = \frac{q}{\epsilon} \Rightarrow \begin{cases} \mathcal{L} = -\Delta \\ f = \frac{q}{\epsilon} \\ u = v \end{cases}$$

$$\mathbf{n} \times \mathbf{e}|_{\Gamma_e} = 0$$

$$\mathbf{n} \cdot \mathbf{d}|_{\Gamma_d} = 0$$

governing equations and BCs

e.g. weak electrostatic formulation

$$\int_{\Omega} -\Delta v \cdot w \, d\Omega = \int_{\Omega} \left(-\text{div}(\text{grad } v) \right) \cdot w \, d\Omega = \int_{\Omega} \frac{q}{\epsilon} w \, d\Omega$$



$$\mathbf{v} \cdot \text{grad } u + u \text{div } \mathbf{v} = \text{div}(u\mathbf{v})$$

**Green formula:
integrating by parts**

$$\int_{\Omega} \left(-\text{div}(w \text{grad } v) + \text{grad } v \cdot \text{grad } w \right) d\Omega = \int_{\Omega} \frac{q}{\epsilon} w \, d\Omega$$

Constraints

$$\mathcal{L}u = f \text{ in } \Omega$$

$$\mathcal{B}u = g \text{ on } \Gamma$$

- Local constraints
 - boundary conditions (BCs) on local fields at the boundary of the domain
 - their choice influences the final solution
 - they can be exploited to reduce the computational domain
 - interface conditions (ICs): coupling of fields between subdomains
- Global constraints
 - Flux or circulations of fields to be fixed (current, voltage, e.m.f., m.m.f, charge)
 - Flux or circulations of fields to be connected (circuit coupling)

Boundary conditions (BCs)

$$\mathcal{L}u = f \text{ in } \Omega$$

$$\mathcal{B}u = g \text{ on } \Gamma$$

- Dirichlet BCs: fix the unknown at the boundary to a given value

$$u|_{\Gamma} = u_0 \begin{cases} = 0, & \text{homogeneous BC} \\ \neq 0, & \text{inhomogeneous BC} \end{cases}$$

- Neumann BCs: fix the normal derivative of the unknown at the boundary to a given value

$$\frac{\partial u(\mathbf{x})}{\partial n} = w(\mathbf{x}), \quad \mathbf{x} \text{ on } \Gamma$$

- Mixed BCs: a combination of Dirichlet and Neumann BCs

$$\frac{\partial u(\mathbf{x})}{\partial n} + f_1(\mathbf{x})u = f_2(\mathbf{x}), \quad \mathbf{x} \text{ on } \Gamma$$

$f_1(\mathbf{x})u$ and $f_2(\mathbf{x})$ explicitly known

Boundary conditions (BCs) (cont'd)

$$\mathcal{L}u = f \text{ in } \Omega$$

$$\mathcal{B}u = g \text{ on } \Gamma$$

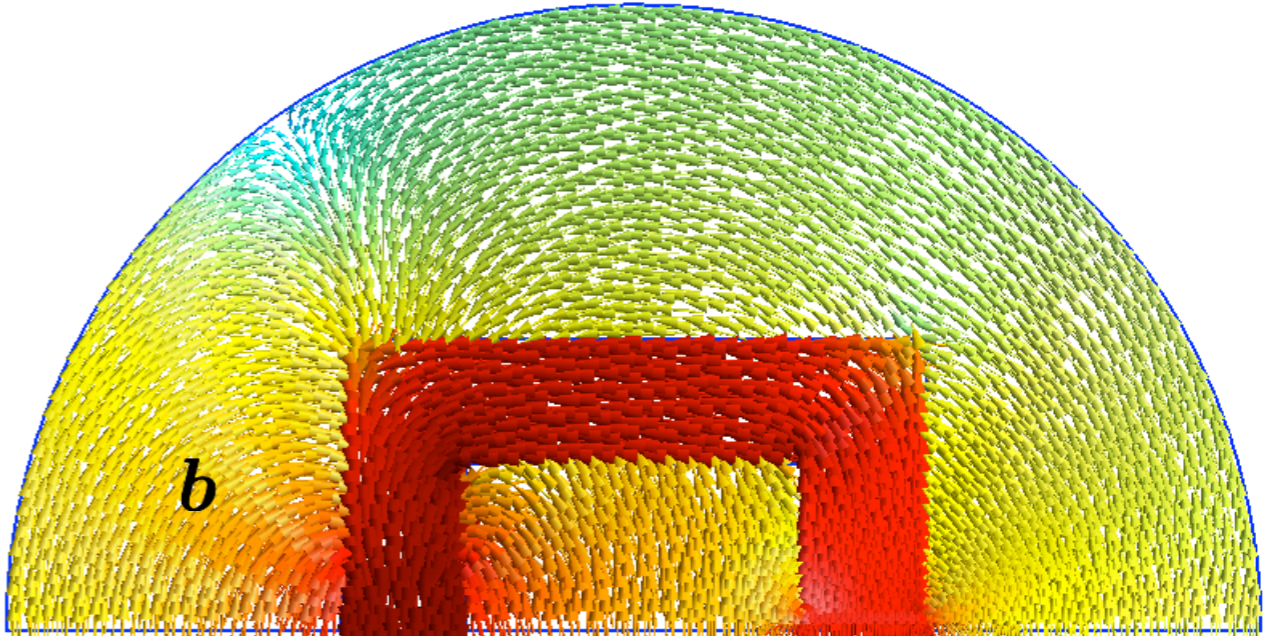
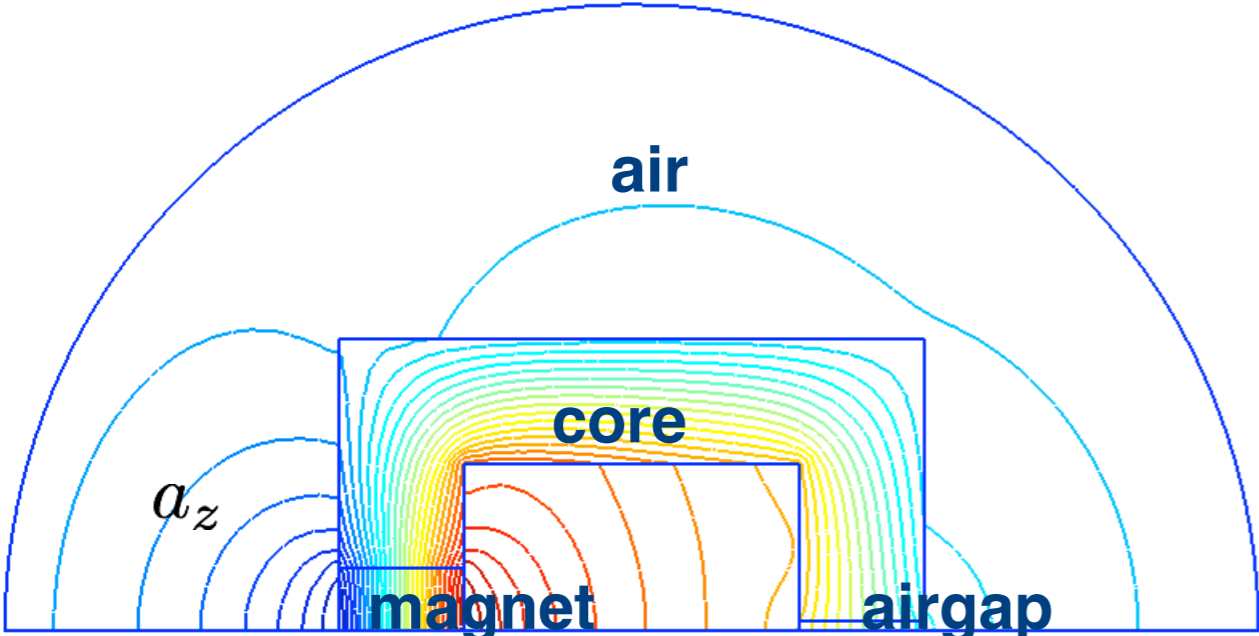
- Periodic BCs: fixing the unknown at the boundary to a given value

$$u(\mathbf{x}_0) + C_1 u(\mathbf{x}_1) = C_2, \quad \mathbf{x}_0 \text{ on } \Gamma_0, \quad \mathbf{x}_1 \text{ on } \Gamma_1$$

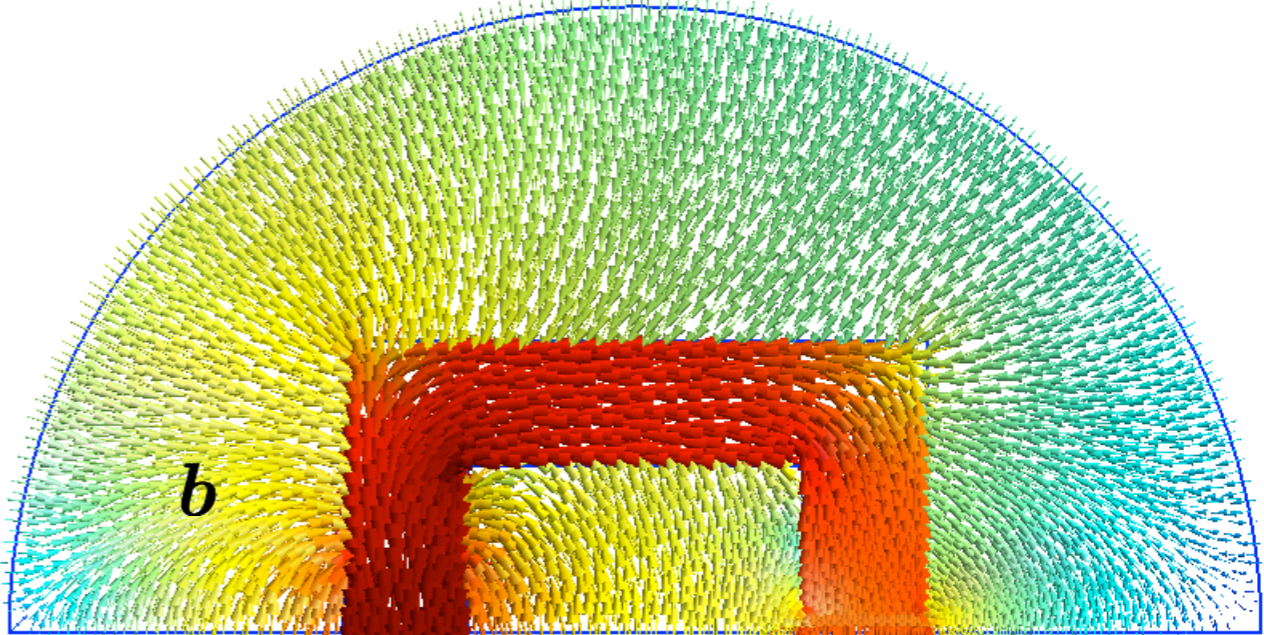
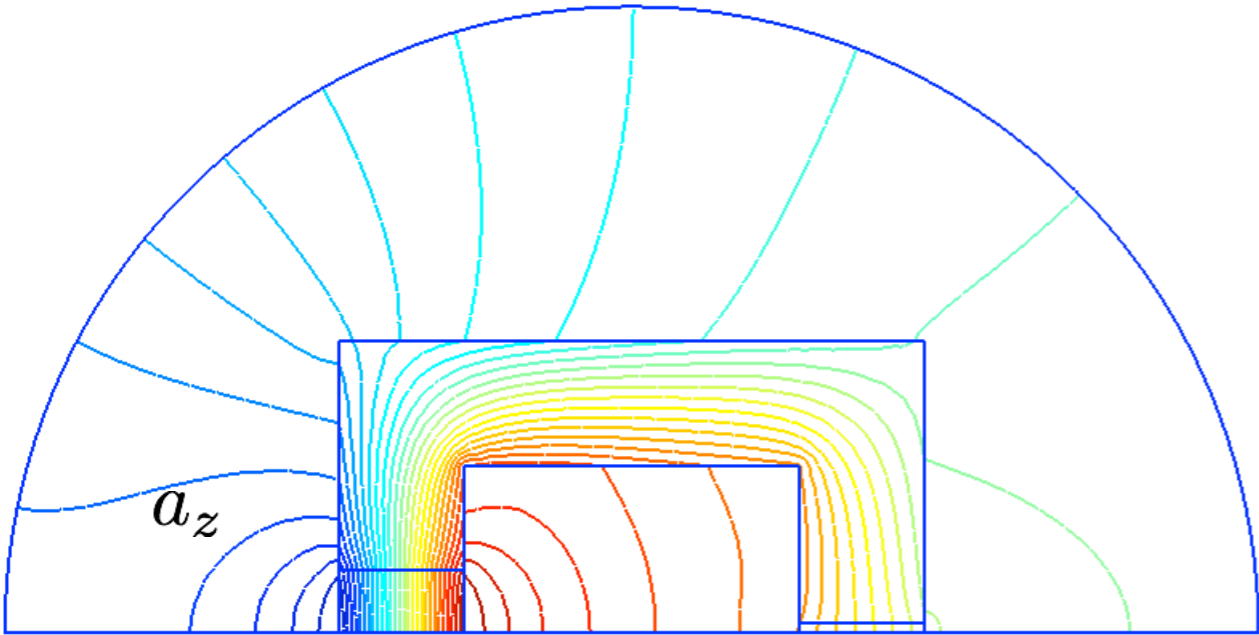
$$\text{or } \frac{\partial u(\mathbf{x}_0)}{\partial n} + C'_1 \frac{\partial u(\mathbf{x}_1)}{\partial n} = C'_2, \quad \mathbf{x}_0 \text{ on } \Gamma_0, \quad \mathbf{x}_1 \text{ on } \Gamma_1$$

- Floating BCs: unknown fixed to a value that still has to be determined, often related to global boundary conditions

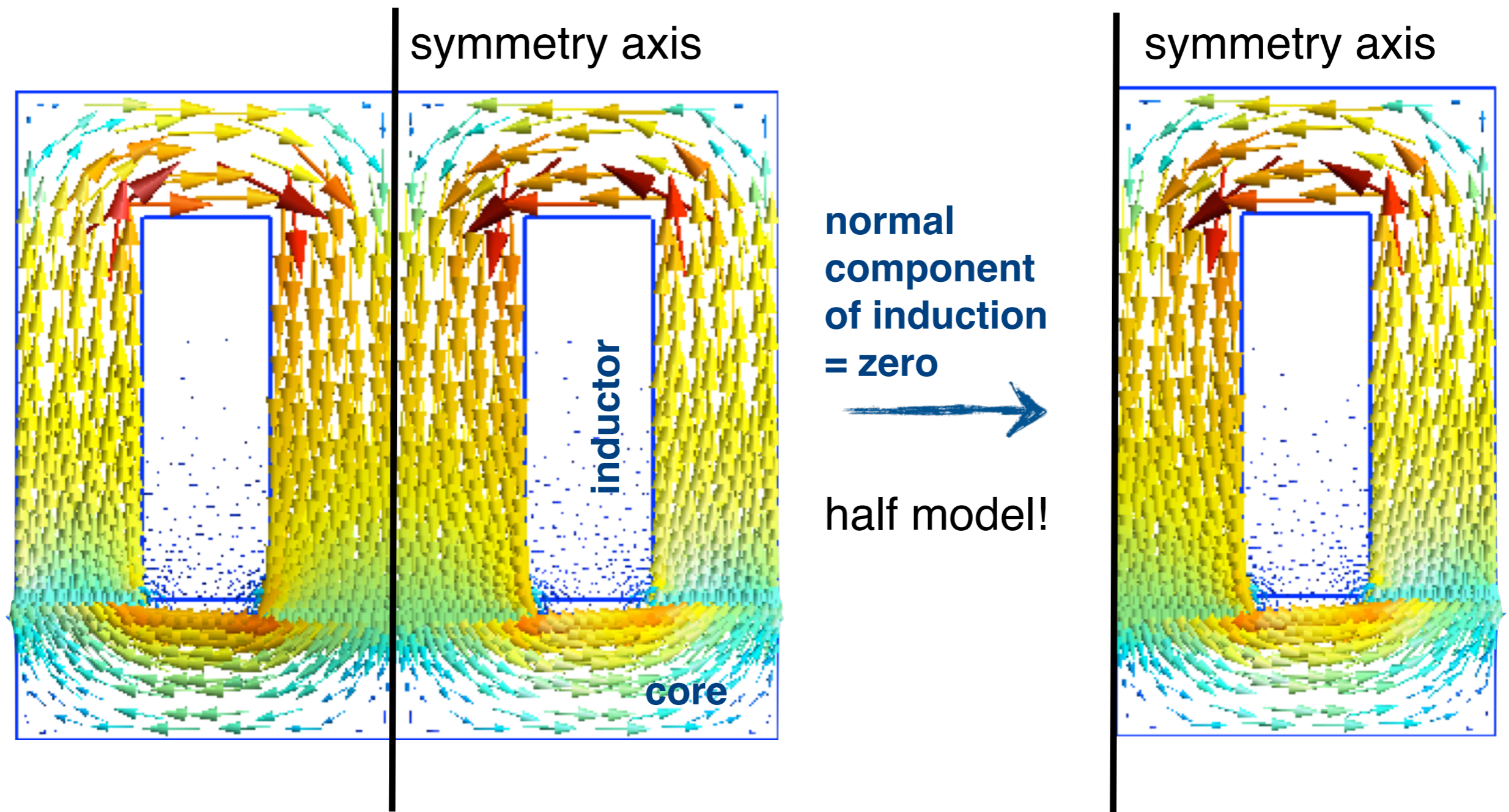
Dirichlet BC



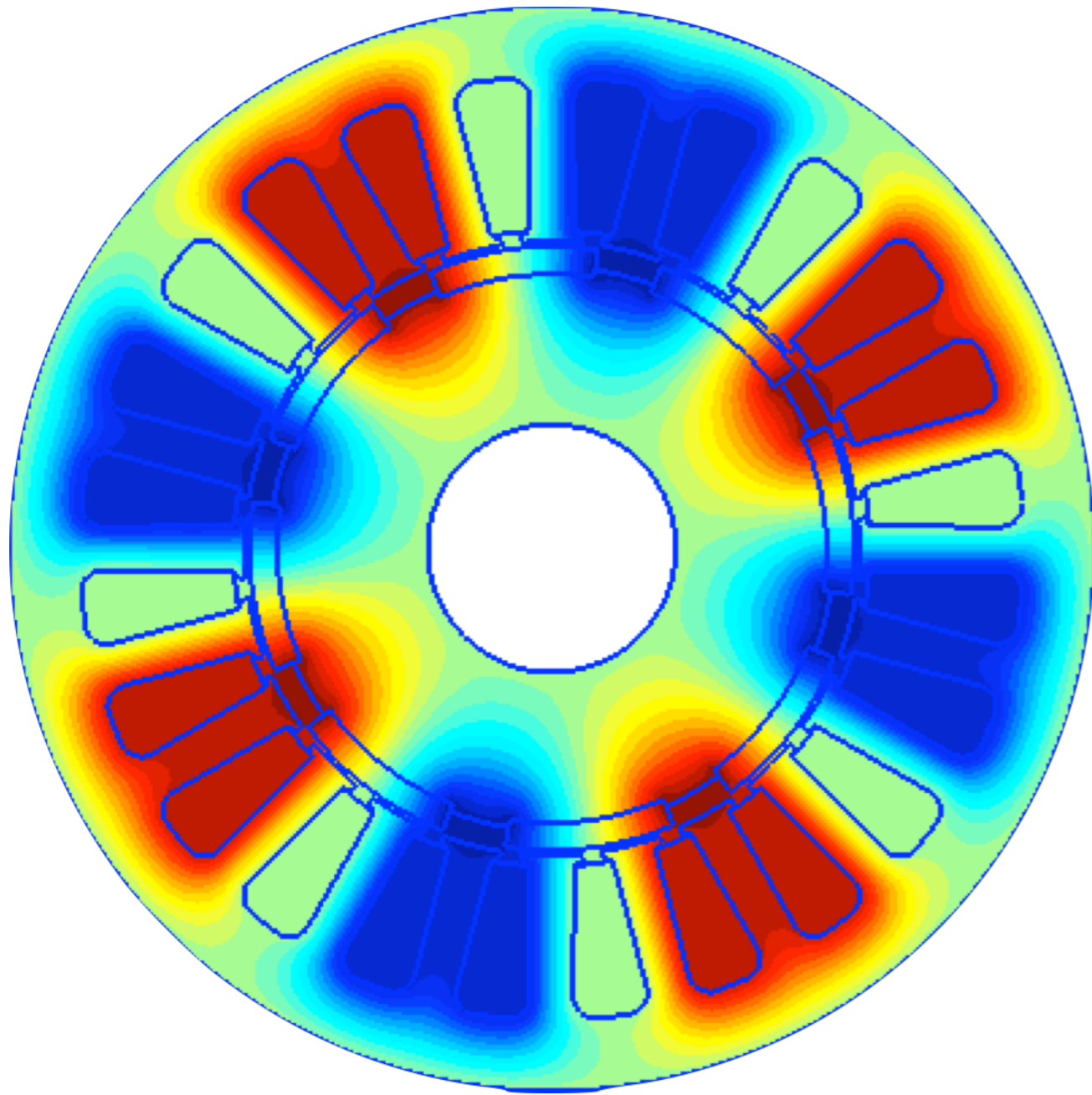
Neumann BC



BCs — symmetry

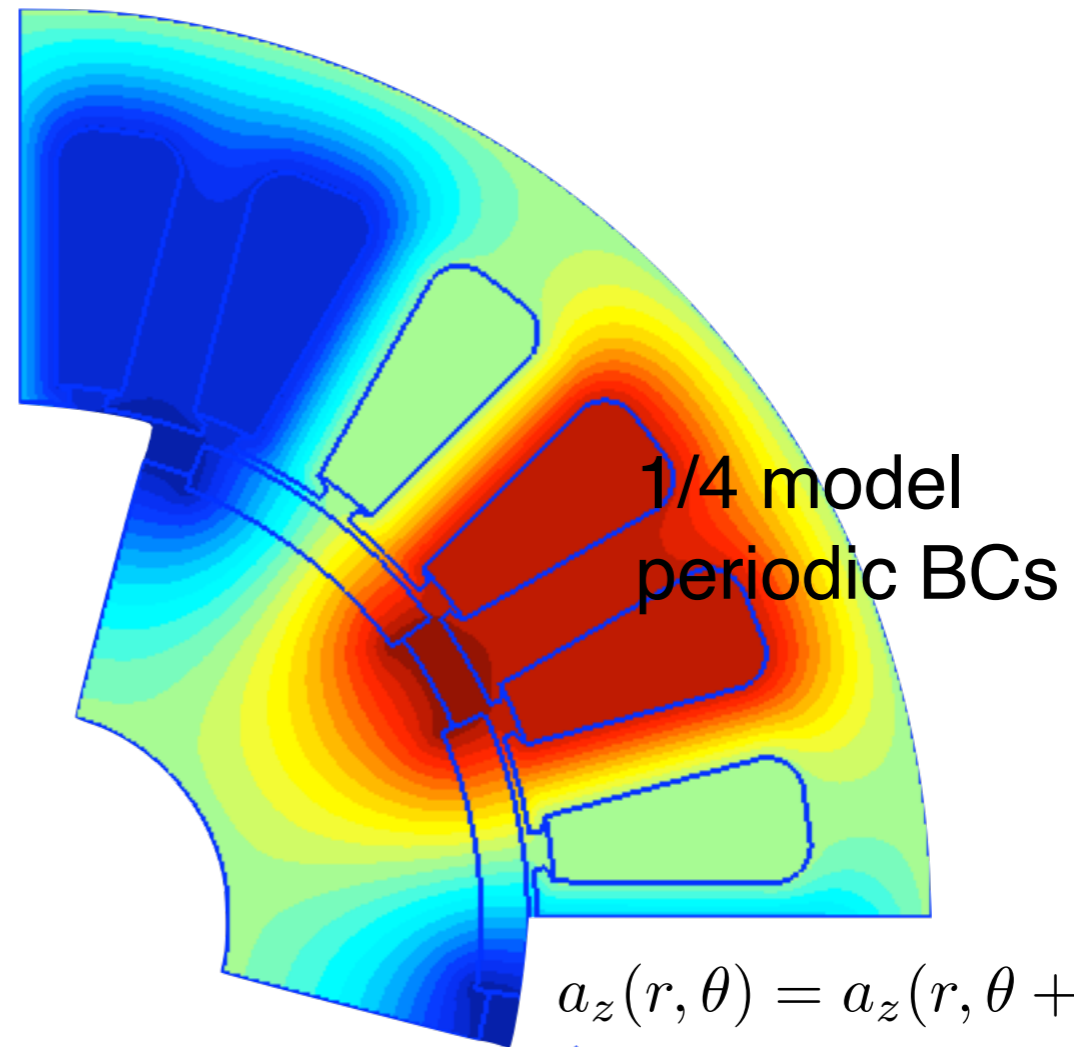


Periodic BCs



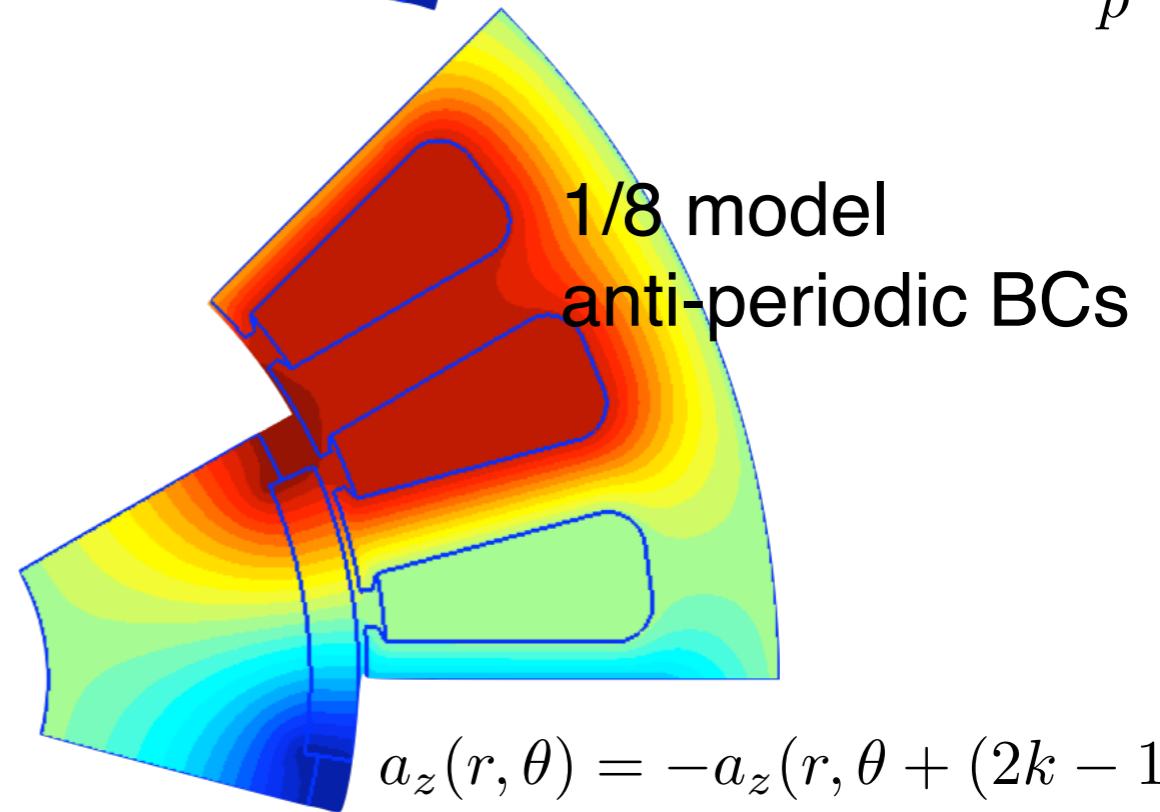
p = number of poles

$k = 1, 2, 3, \dots$



1/4 model
periodic BCs

$$a_z(r, \theta) = a_z(r, \theta + 2k \frac{\pi}{p})$$



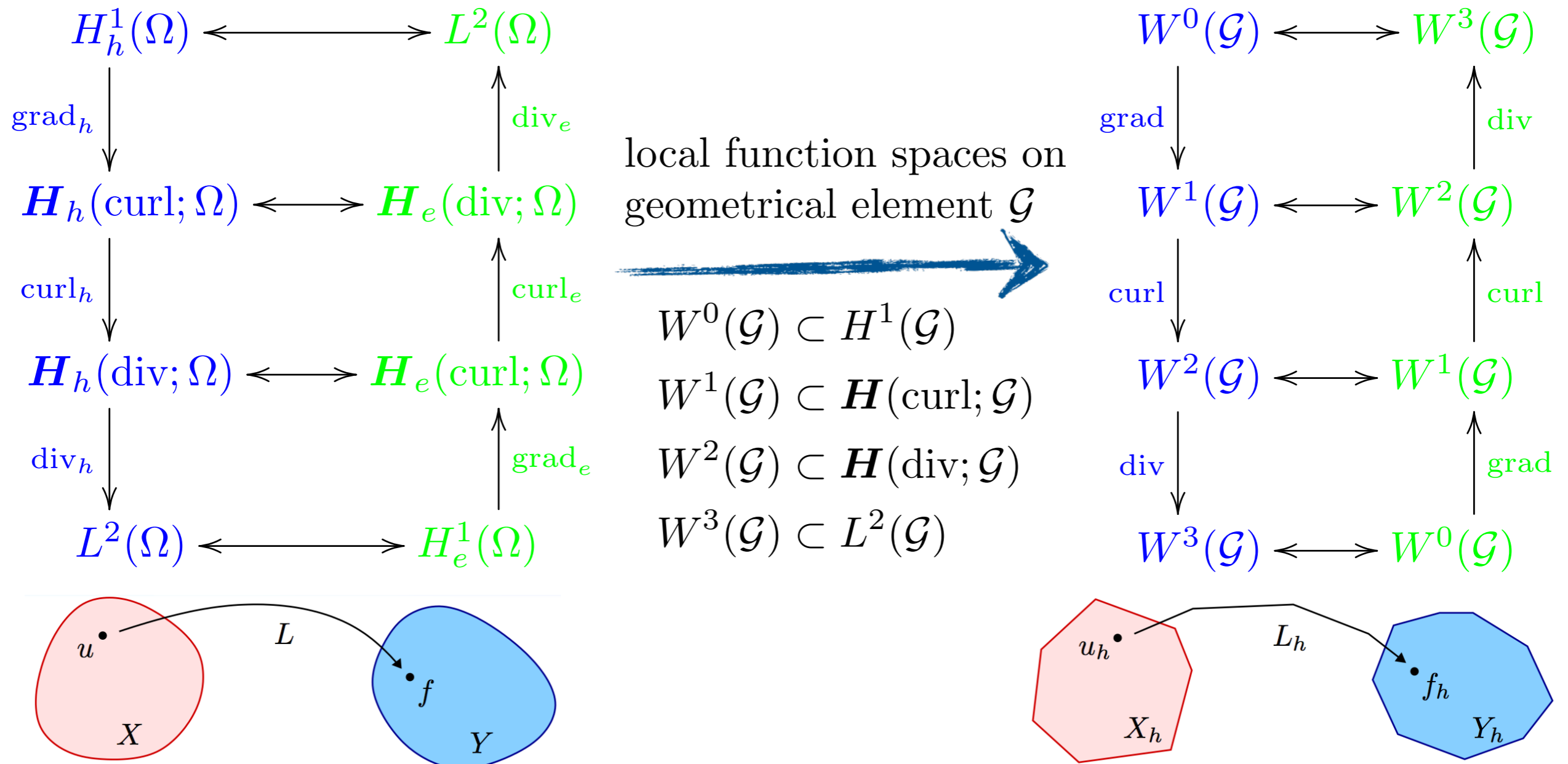
1/8 model
anti-periodic BCs

$$a_z(r, \theta) = -a_z(r, \theta + (2k - 1) \frac{\pi}{p})$$

Discrete mathematical structure
Whitney elements

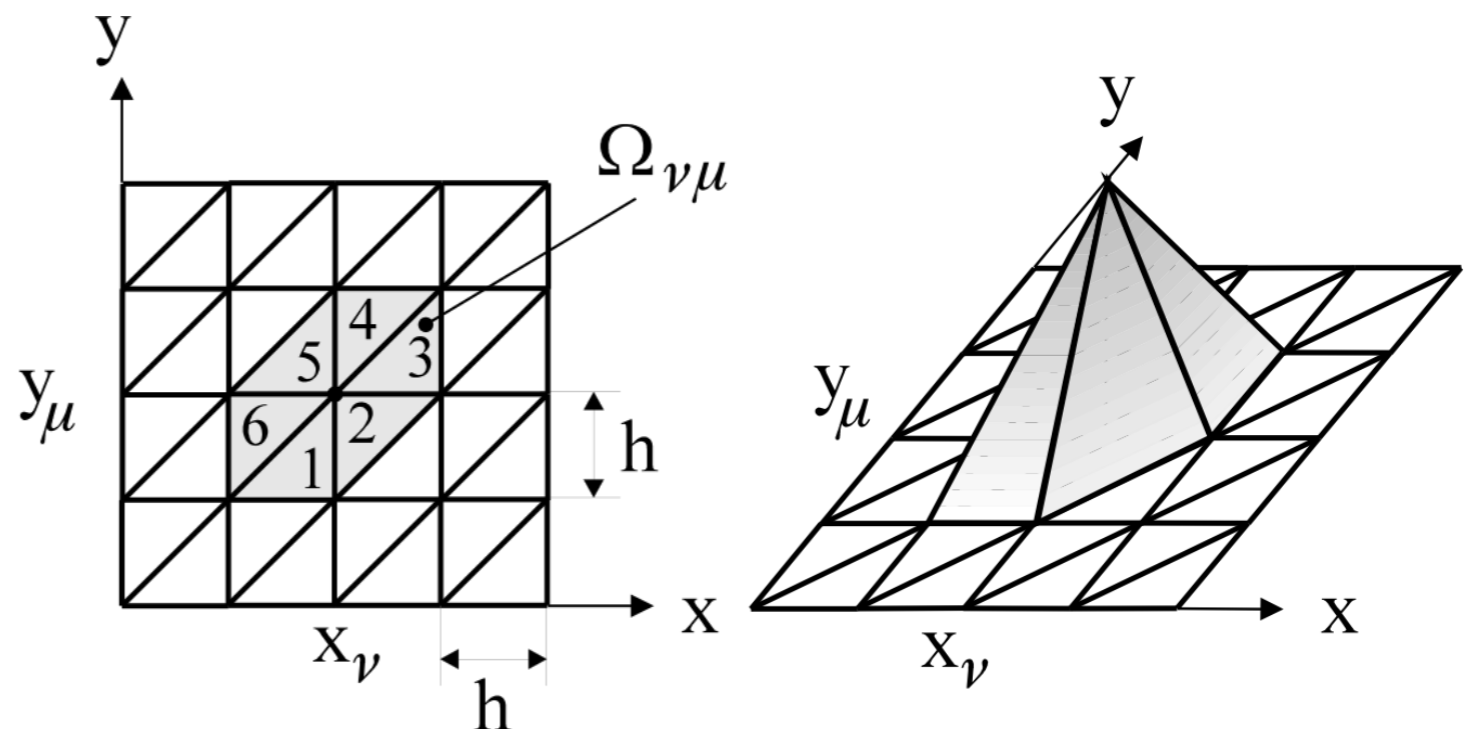
Discrete mathematical structure

Replace the continuous spaces (infinite dimension) by discrete spaces (finite dimension)

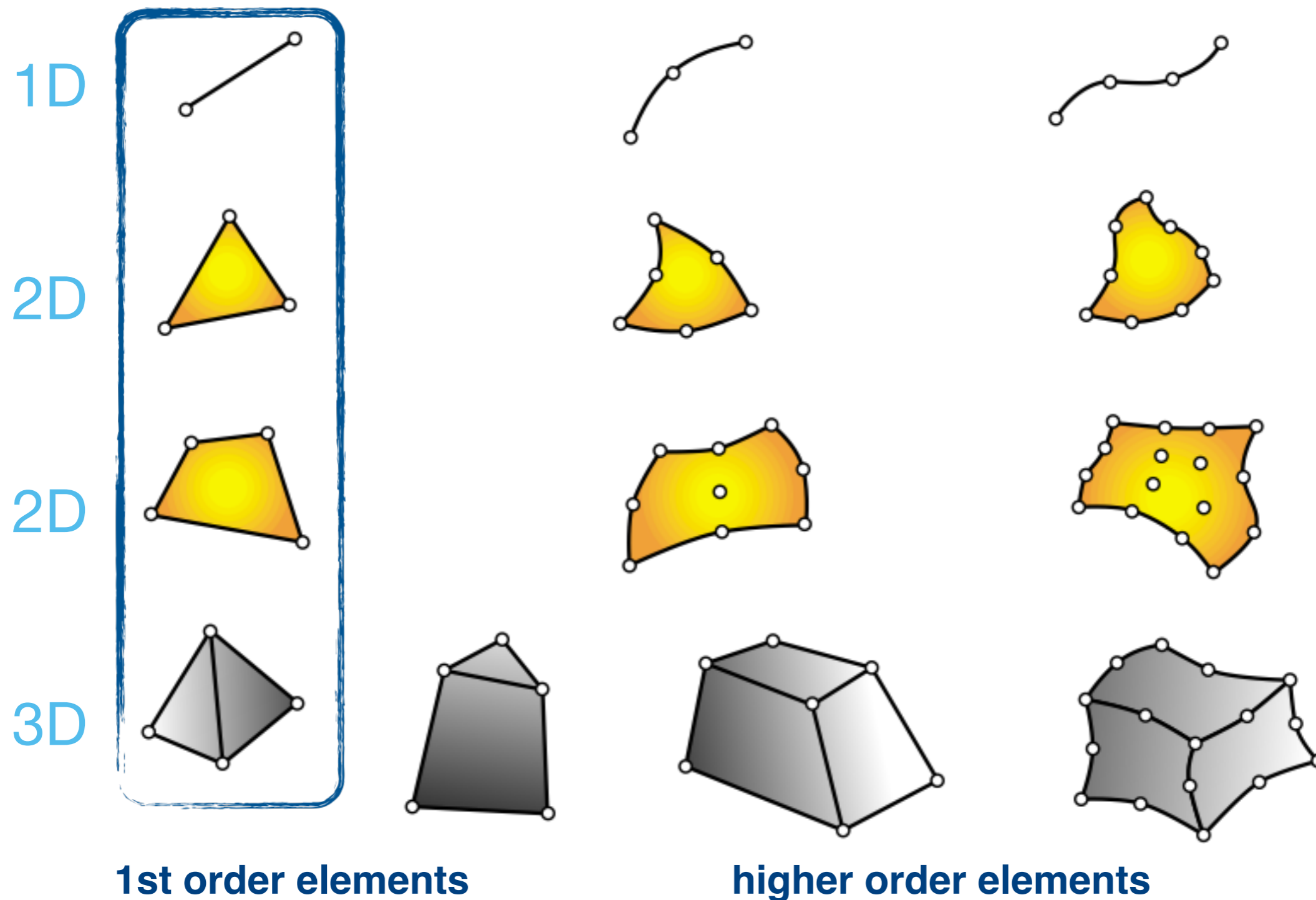


Finite elements

- set of linearly independent test and trial functions (also called basis/shape functions and weighting functions)
- commonly piecewise polynomial
- defined at a structured grid or an unstructured mesh
- compact support
- scalar or vectorial functions



Finite elements geometry defined by nodes

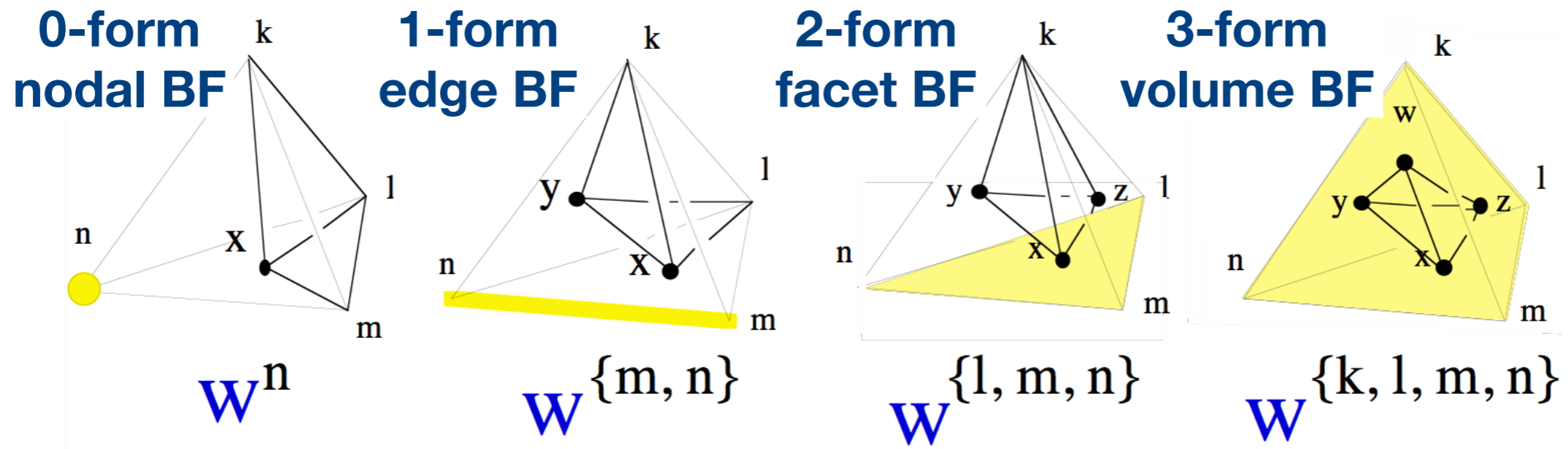


The Whitney elements

Finite element (\mathcal{G}, Σ, S) :

- ✓ geometrical element \mathcal{G}
- ✓ $\Sigma =$ set of N Dofs
- ✓ S function of finite dimension N

Let us consider a mesh of Ω formed by geometrical elements \mathcal{G} with nodes \mathcal{N} , edges \mathcal{E} , faces \mathcal{F} , volumes \mathcal{V} ,



The Whitney elements of order p are expressed as

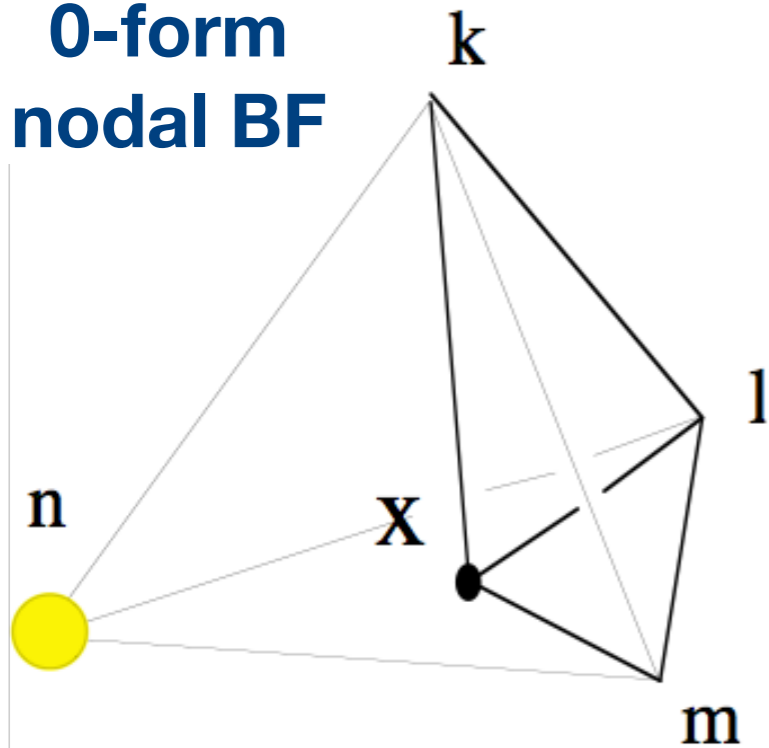
$$\mathbf{w}_{n_0, \dots, n_p} = p! \sum_{j=0}^p (-1)^m \zeta_{n_m} \text{grad } \zeta_{n_0} \times \dots \times \text{grad } \zeta_{n_{m-1}} \times \text{grad } \zeta_{n_{m+1}} \times \dots \times \text{grad } \zeta_{n_p}$$

with $\zeta_n(x)$ barycentric weight of x with respect to node n in \mathcal{G}

The Whitney elements of order 0

Nodal elements

**0-form
nodal BF**



$$w_n = \zeta_n$$

with $n \in \mathcal{N}$ (node set)

span space $W^0(\mathcal{G})$

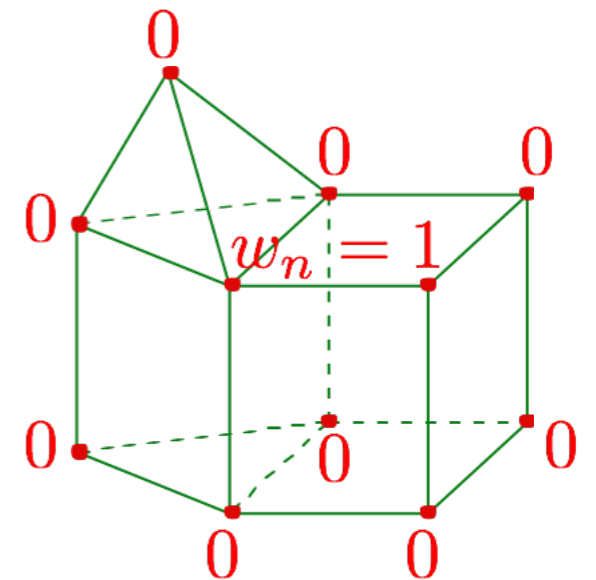
The interpolation of a function u is given by

$$I(u) = \sum_{x_i \in \mathcal{N}} u_i w_i$$

$$\text{with } u_i = \alpha_i(u) = u(x_i)$$

w_n

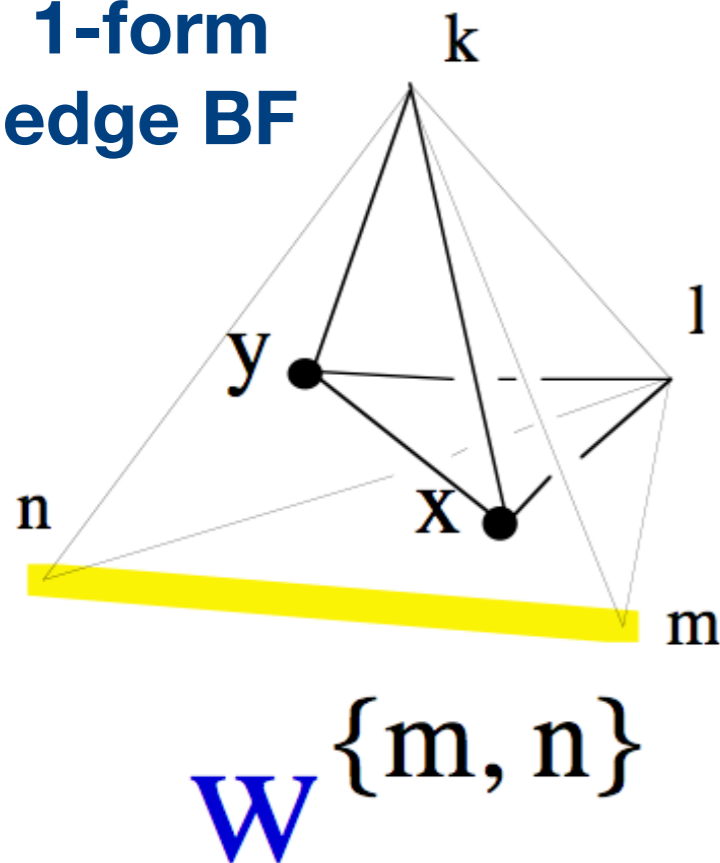
- ✓ piecewise linear continuous:
first order scalar Lagrange finite elements
- ✓ discretisation of scalar fields
- ✓ $w_n = 1$ at node n , 0 at other nodes
- ✓ $w_n = 1$ is continuous across faces



The Whitney elements of order 1

Edge elements

**1-form
edge BF**



$$\boldsymbol{w}_e = \boldsymbol{w}_{\{m, n\}} = \zeta_m \text{grad } \zeta_n - \zeta_n \text{grad } \zeta_m,$$

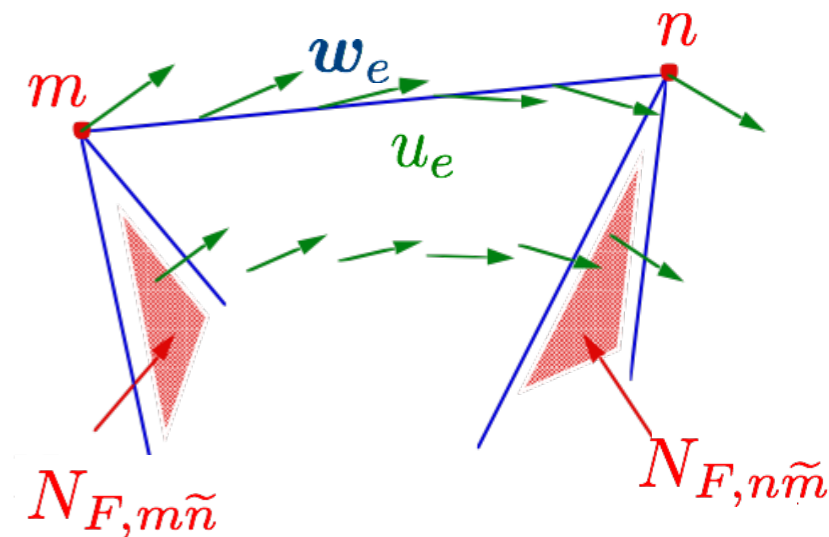
with $e \in \mathcal{E}$ (edge set)
span space $W^1(\mathcal{G})$

The interpolation of a function \boldsymbol{u} is given by

$$I(\boldsymbol{u}) = \sum_{e \in \mathcal{E}} u_e \boldsymbol{w}_e$$

$$\text{with } u_e = \alpha_e(\boldsymbol{u}) = \int_e \boldsymbol{u} \cdot d\boldsymbol{l}, \quad \forall e \in \mathcal{E}$$

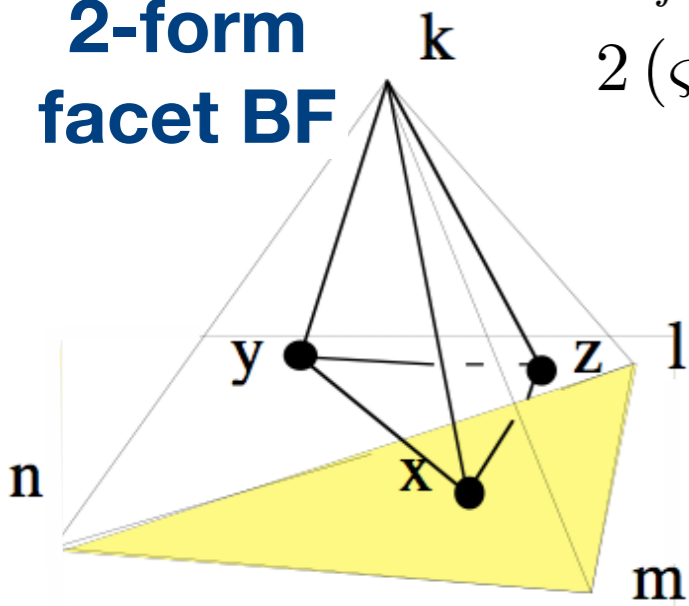
- ✓ Dof = circulations of field along edges of mesh
- ✓ discretisation of 1-forms, e.g. \boldsymbol{h} , \boldsymbol{e}
- ✓ tangential component continuous across faces
- ✓ circulation of $\boldsymbol{w}_e = 1$ along edge e , 0 across other edges



Whitney elements of order 2

Face elements

**2-form
facet BF**



W {l, m, n}

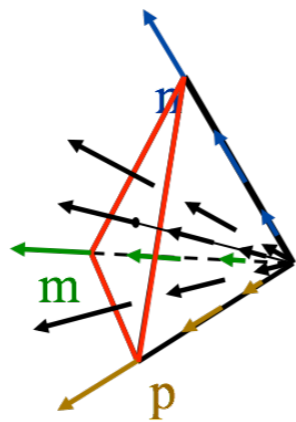
$$\boldsymbol{w}_f = \boldsymbol{w}_{\{l,m,n\}} =$$

$$2 (\varsigma_l \text{grad } \varsigma_m \times \text{grad } \varsigma_n - \varsigma_m \text{grad } \varsigma_l \times \text{grad } \varsigma_n + \varsigma_n \text{grad } \varsigma_l \times \text{grad } \varsigma_m)$$

with $f = \{l, m, n\} \in \mathcal{F}$ (face set)

span space $W^2(\mathcal{G})$

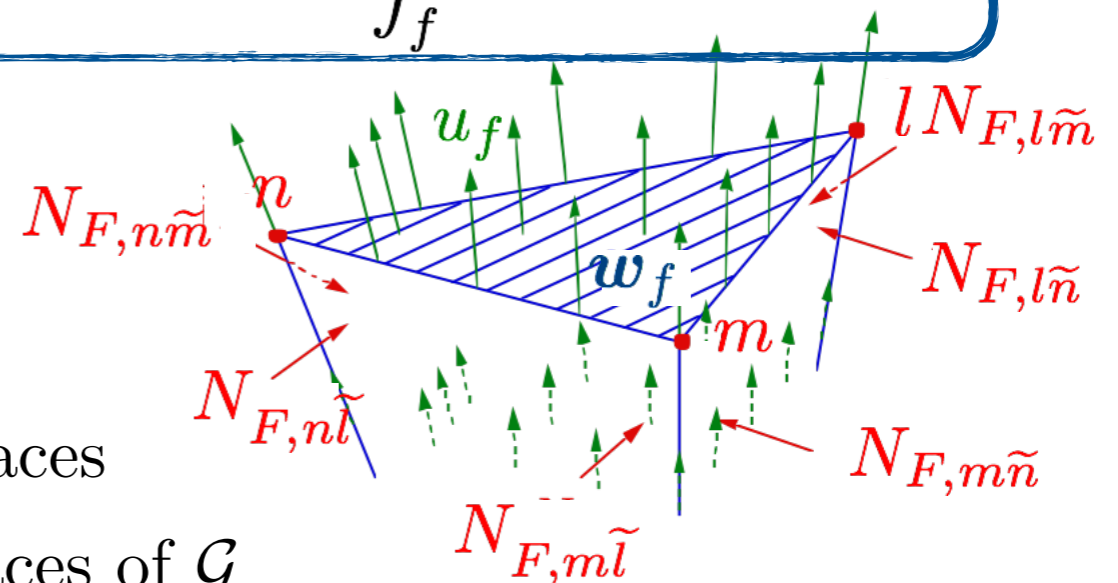
The interpolation of a function \boldsymbol{u} is given by



$$I(\boldsymbol{u}) = \sum_{f \in \mathcal{F}} u_f \boldsymbol{w}_f$$

$$\text{with } u_f = \alpha_f(\boldsymbol{u}) = \int_f \boldsymbol{u} \cdot \boldsymbol{n} ds, \forall f \in \mathcal{F}$$

- ✓ Dof = flux through faces of mesh
- ✓ discretisation of 2-forms, e.g. \boldsymbol{b} , \boldsymbol{j}
- ✓ normal component continuous across interfaces
- ✓ flux of $\boldsymbol{w}_f = 1$ across face, 0 across other faces of \mathcal{G}



Whitney elements of order 3

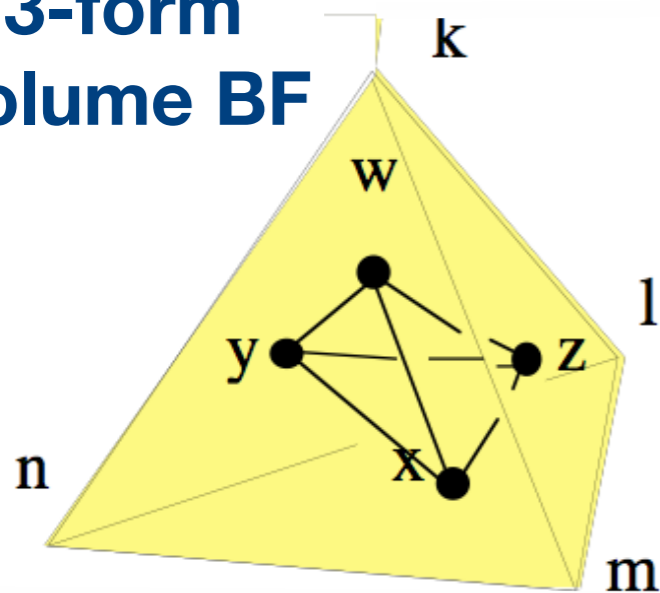
Volume elements

$$w_v = w_{\{k,l,m,n\}} = 6 \left(\varsigma_k \text{grad } \varsigma_l \times \text{grad } \varsigma_m \times \text{grad } \varsigma_n - \varsigma_l \text{grad } \varsigma_k \times \text{grad } \varsigma_m \times \text{grad } \varsigma_n + \right. \\ \left. \varsigma_n \text{grad } \varsigma_k \times \text{grad } \varsigma_l \times \text{grad } \varsigma_m - \varsigma_n \text{grad } \varsigma_k \times \text{grad } \varsigma_l \times \text{grad } \varsigma_m \right)$$

with $v = \{k, l, m, n\} \in \mathcal{V}$ (volume set)

span space $W^3(\mathcal{G})$

**3-form
volume BF**



W $\{k, l, m, n\}$

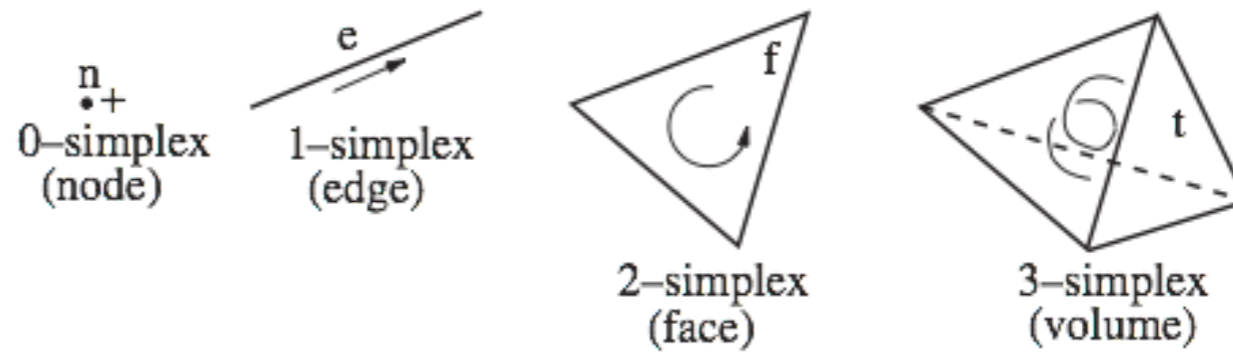
The interpolation of a function u is given by

$$I(u) = \sum_{v \in \mathcal{V}} u_v w_v$$

$$\text{with } u_v = \alpha_v(u) = \int_v u \, dv$$

- ✓ piecewise constant functions
- ✓ Dof = integration over its volume
- ✓ discretisation of densities
- ✓ $\sum w_v = 1$ over the volume of \mathcal{G} , 0 over other volumes

Conformity

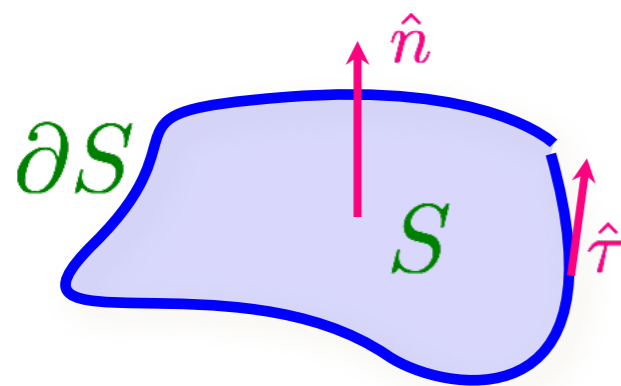


- ✓ **Nodal elements:** Conforming finite elements (in $\mathcal{H}^1(\Omega)$) interpolate scalar fields that are **continuous across any interface**.

Discretisation of scalar quantities: potentials φ , v , temperature...

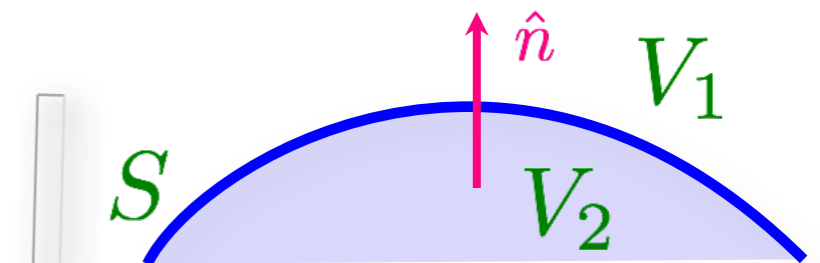
- ✓ **Edge elements:** Curl-conforming finite elements (in $\mathbf{H}(\text{curl}; \Omega)$) ensure the **continuity of the tangential component** of the field.

Discretisation of the magnetic field \mathbf{h} , the magnetic vector potential \mathbf{a} or the electric field \mathbf{e} .



$$\int_{\partial S} \mathbf{h} \cdot \hat{\tau} \, dl = \int_S (\mathbf{j} + \partial_t \mathbf{d}) \cdot \hat{n} \, ds$$

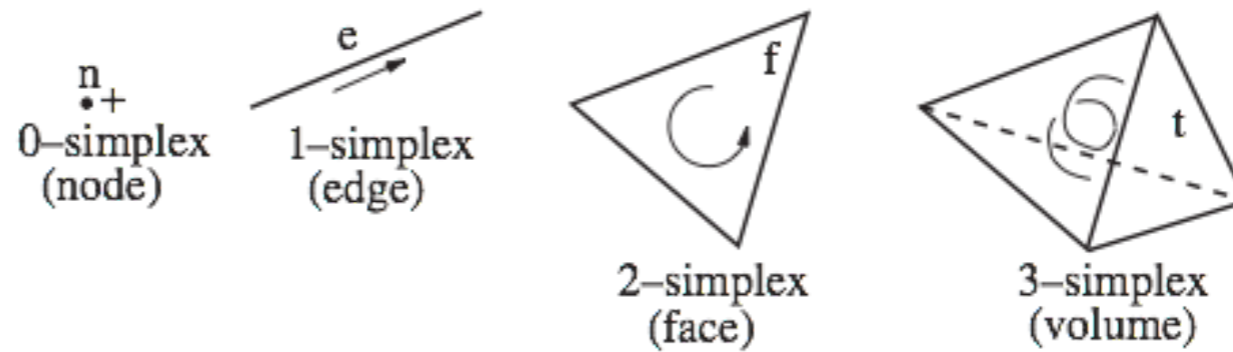
$$\int_{\partial S} \mathbf{e} \cdot \hat{\tau} \, dl = - \int_S \partial_t \mathbf{b} \cdot \hat{n} \, ds$$



$$\hat{n} \times (\mathbf{h}_2 - \mathbf{h}_1)|_S = \mathbf{j}_s$$

$$\hat{n} \times (\mathbf{e}_2 - \mathbf{e}_1)|_S = 0$$

Conformity



- ✓ **Face elements:** Div-conforming finite elements (in $\mathbf{H}(\text{div}; \Omega)$) ensure the **continuity of the normal component** of the interpolated field. Discretisation of the magnetic flux density \mathbf{b} , the current density \mathbf{j} or the electric flux density \mathbf{d} .

The diagram shows a face element S (a blue shaded region) with boundary ∂S (a blue line). A normal vector $\hat{\mathbf{n}}$ is shown pointing outwards from the face. The boundary is also labeled with $\hat{\boldsymbol{\tau}}$.

$$\int_S \mathbf{b} \cdot \hat{\mathbf{n}} \, ds = 0$$

$$\int_S \mathbf{d} \cdot \hat{\mathbf{n}} \, ds = \int_V \rho \, dv$$

$$\int_S \mathbf{j} \cdot \hat{\mathbf{n}} \, ds = 0$$

The diagram shows a volume element V (a blue shaded region) with boundary S (a blue line). A normal vector $\hat{\mathbf{n}}$ is shown pointing outwards from the boundary. The volume is divided into two sub-volumes V_1 and V_2 .

$$\hat{\mathbf{n}} \cdot (\mathbf{b}_2 - \mathbf{b}_1)|_S = 0$$

$$\hat{\mathbf{n}} \cdot (\mathbf{d}_2 - \mathbf{d}_1)|_S = \rho_s$$

- ✓ **Volume elements:** Finite elements in $L^2(\Omega)$ do not impose any continuity (**discontinuous**) between elements on the interpolated field. Discretisation of quantities that may vary from one element to the other e.g. the electric charge density ρ .

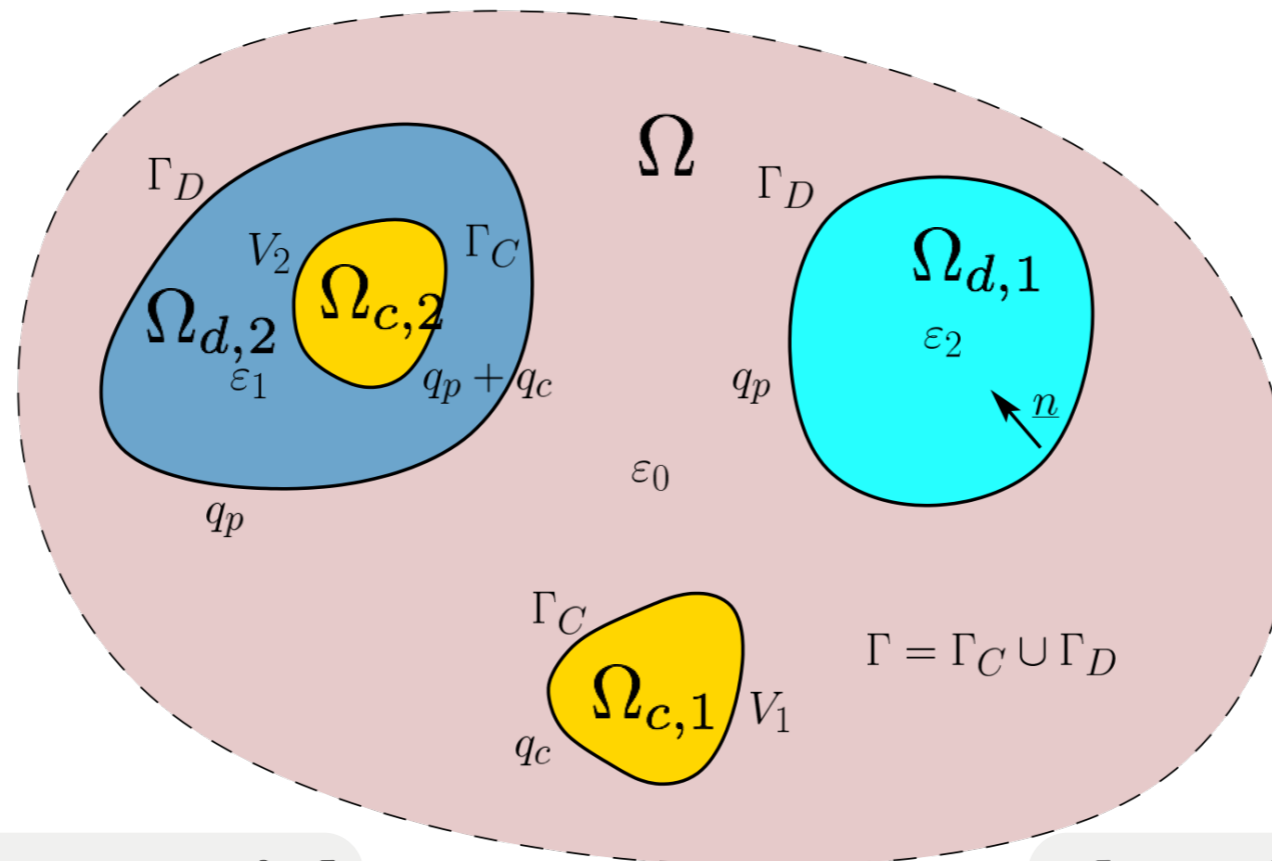
$$\int_S \mathbf{d} \cdot \hat{\mathbf{n}} \, ds = \int_V \rho \, dv$$

Electrokinetic formulation and discretization

Electrostatics



Phenomena involving time-independent distributions of charges & fields



“*d* side”

“*e* side”

electric vector potential
(*u*-) formulation

electric scalar potential
(*v*-) formulation

boundary conditions

$$\text{curl } \mathbf{e} = 0$$

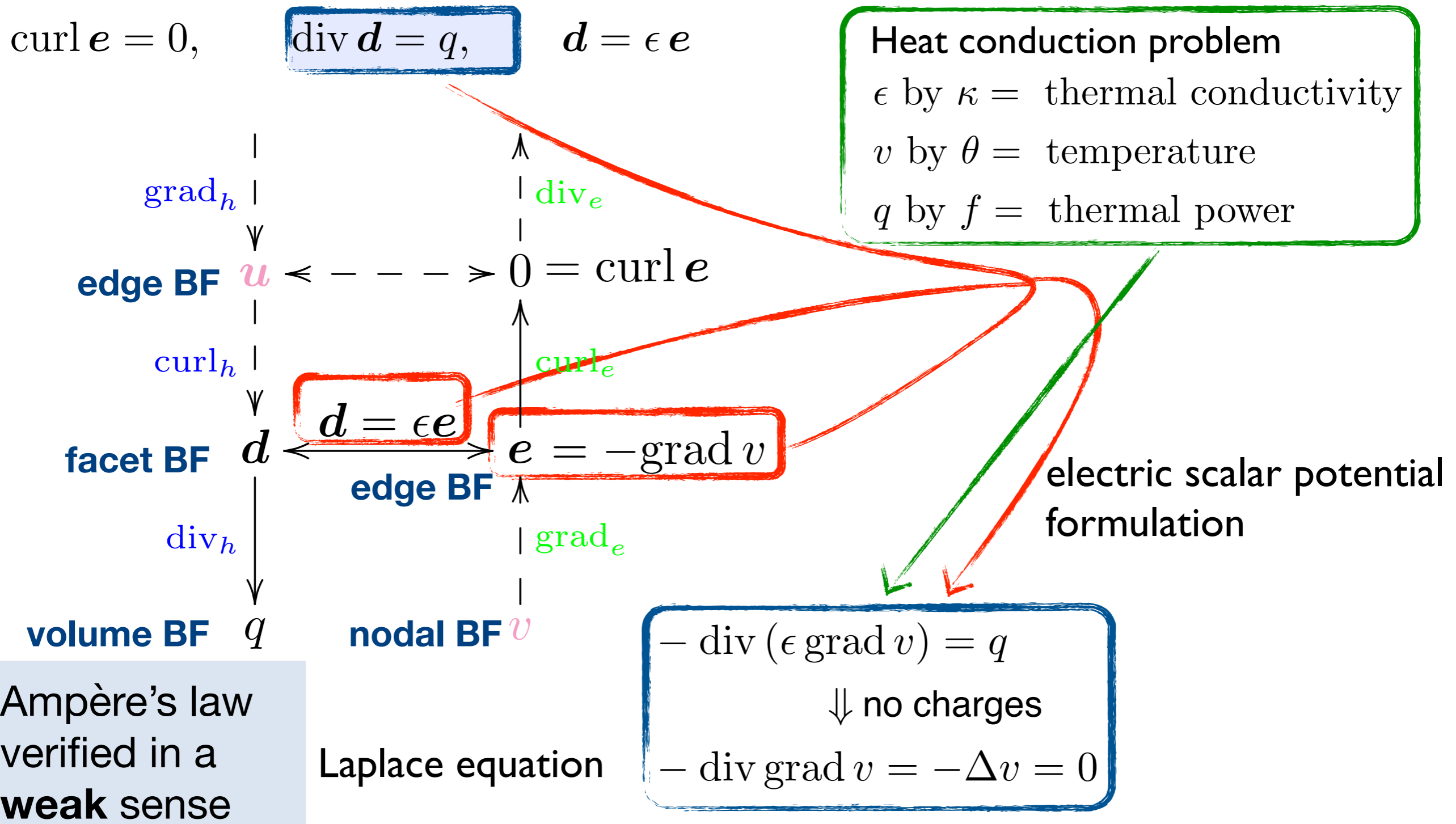
$$\text{div } \mathbf{d} = q$$

$$\mathbf{d} = \epsilon \mathbf{e}$$

$$\mathbf{n} \times \mathbf{e}|_{\Gamma_e} = 0$$

$$\mathbf{n} \cdot \mathbf{d}|_{\Gamma_d} = 0$$

Electrostatics



Spatial discretization — electrostatics

Weighted residual approach

$$-\operatorname{div}(\epsilon \operatorname{grad} v) = q \quad \text{in } \Omega$$

We integrate the equation weighted by test functions $w_i(\boldsymbol{x})$ over the whole domain:

$$\int_{\Omega} \left(-\operatorname{div}(\epsilon \operatorname{grad} v) \right) w_i \, d\Omega = \int_{\Omega} q w_i \, d\Omega, \quad \forall w_i(\boldsymbol{x})$$

$w_i(\boldsymbol{x})$ weighting or test functions

Spatial discretization — electrostatics (II)

search wiki vector calculus

Weak formulation

$$\forall w_i(\mathbf{x})$$

$$\int_{\Omega} \left(-\operatorname{div}(\epsilon \operatorname{grad} v) \right) w_i \, d\Omega = \int_{\Omega} q w_i \, d\Omega$$



$$\mathbf{v} \cdot \operatorname{grad} u + u \operatorname{div} \mathbf{v} = \operatorname{div}(u\mathbf{v})$$

integration by parts
Green formula

$$\int_{\Omega} \left(-\operatorname{div}(w_i \epsilon \operatorname{grad} v) + \epsilon \operatorname{grad} v \cdot \operatorname{grad} w_i \right) d\Omega = \int_{\Omega} q w_i \, d\Omega$$



$$\int_{\Omega} \operatorname{div} \mathbf{a} \, d\Omega = \oint_{\partial\Omega} \mathbf{a} \, d\Gamma$$

divergence theorem

$$\int_{\partial\Omega} w_i (-\epsilon \operatorname{grad} v) \, d\Gamma + \int_{\Omega} \epsilon \operatorname{grad} v \cdot \operatorname{grad} w_i \, d\Omega = \int_{\Omega} q w_i \, d\Omega$$

only the first derivative of the electric potential is now required

Spatial discretization — electrostatics (III)

Boundary integral

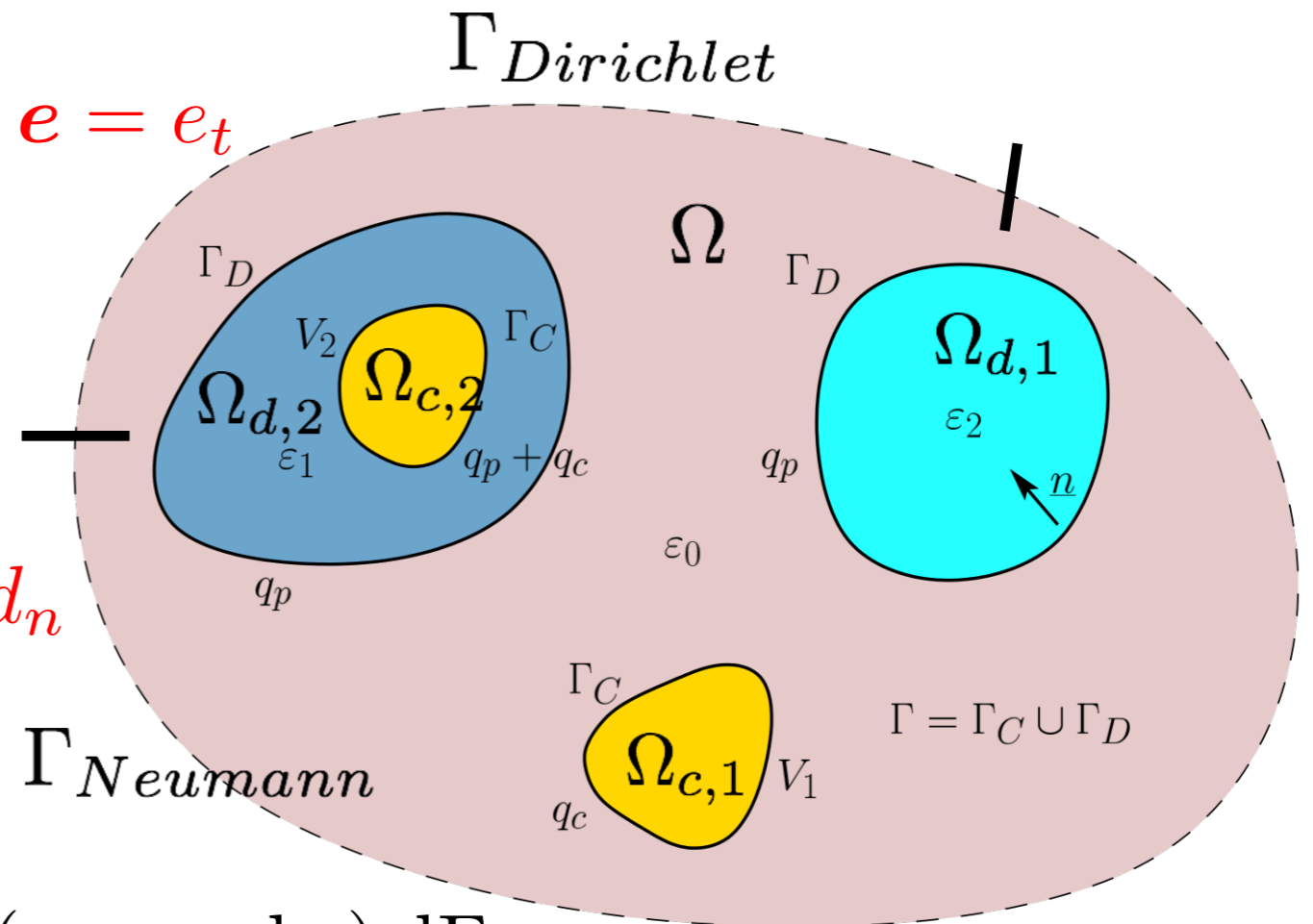
$$\int_{\partial\Omega} w_i \underbrace{(-\epsilon \operatorname{grad} v)}_{\mathbf{n} \cdot \mathbf{d} = d_n} d\Gamma$$

$$\mathbf{n} \cdot \mathbf{d} = d_n$$

$$\forall w_i(\mathbf{x})$$

$$\mathbf{n} \times \mathbf{e} = \mathbf{e}_t$$

$$\mathbf{n} \cdot \mathbf{d} = d_n$$



$$\underbrace{\int_{\Gamma_{Dirichlet}} w_i(-\epsilon \operatorname{grad} v) d\Gamma}_{=0} + \underbrace{\int_{\Gamma_{Neumann}} w_i(-\epsilon \operatorname{grad} v) d\Gamma}_{=0}$$

$$= 0 \quad \forall w_i(\mathbf{x})$$

$$= 0$$

if $w_i = 0$ on $\Gamma_{Dirichlet}$

essential BC

natural BC

Spatial discretization — electrostatics (IV)

$$v = \sum_j u_j s_j \quad \text{with } s_j(\mathbf{x}) = 0 \text{ at } \Gamma_{Dirichlet}$$

$$\begin{cases} s_j(\mathbf{x}) & \text{shape functions} \\ u_j & \text{unknowns, degrees of freedom} \end{cases}$$

Ritz-Galerkin method $s_j(\mathbf{x}) = w_j(\mathbf{x})$

Petrov-Galerkin method $s_j(\mathbf{x}) \neq w_j(\mathbf{x})$

$$\int_{\Omega} \epsilon \operatorname{grad} v \cdot \operatorname{grad} w_i \, d\Omega = \int_{\Omega} q w_i \, d\Omega$$



$$\sum_j u_j \underbrace{\int_{\Omega} \epsilon \operatorname{grad} w_j \cdot \operatorname{grad} w_i \, d\Omega}_{= k_{ij}} = \underbrace{\int_{\Omega} q w_i \, d\Omega}_{= f_i}$$



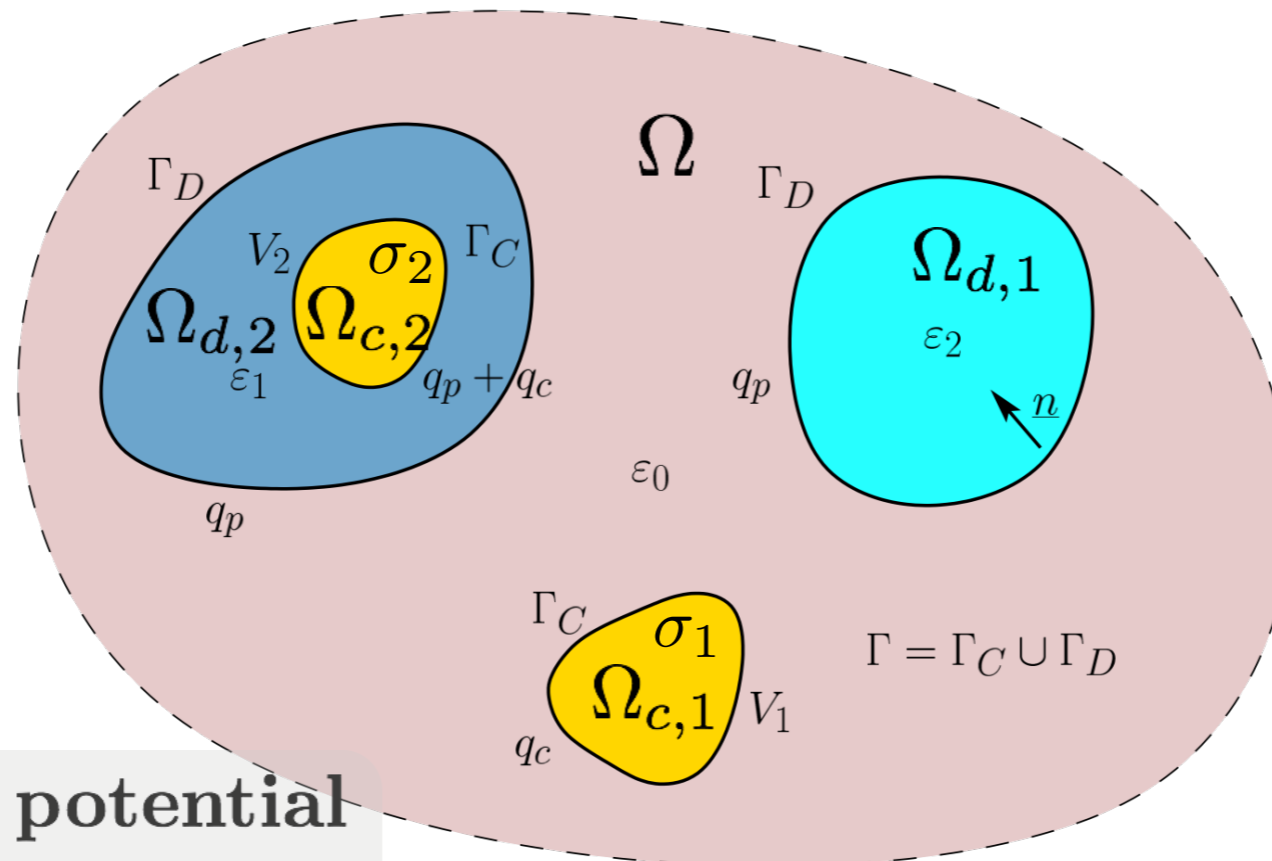
matrix system

$$[k_{ij}][u_j] = [f_i]$$



Electrokinetics

Phenomena involving time-independent (static) currents in conductors



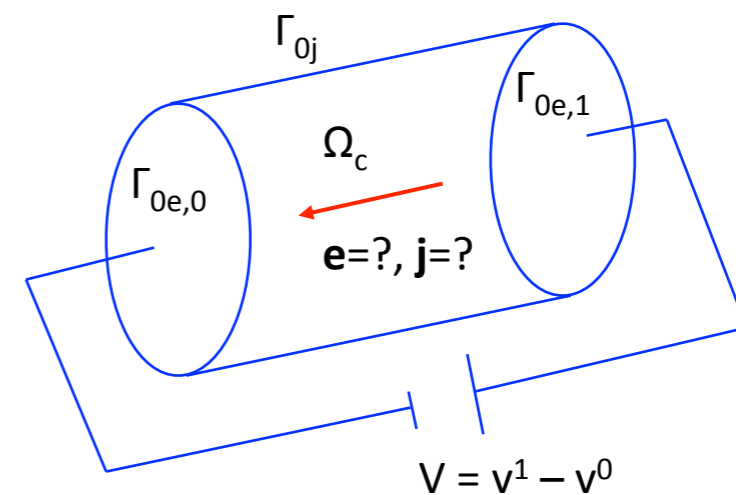
electric scalar potential
(v -) formulation

$$\begin{aligned} \text{curl } \mathbf{e} &= 0 \\ \text{div } \mathbf{j} &= 0 \\ \mathbf{j} &= \sigma \mathbf{e} \end{aligned}$$

boundary conditions

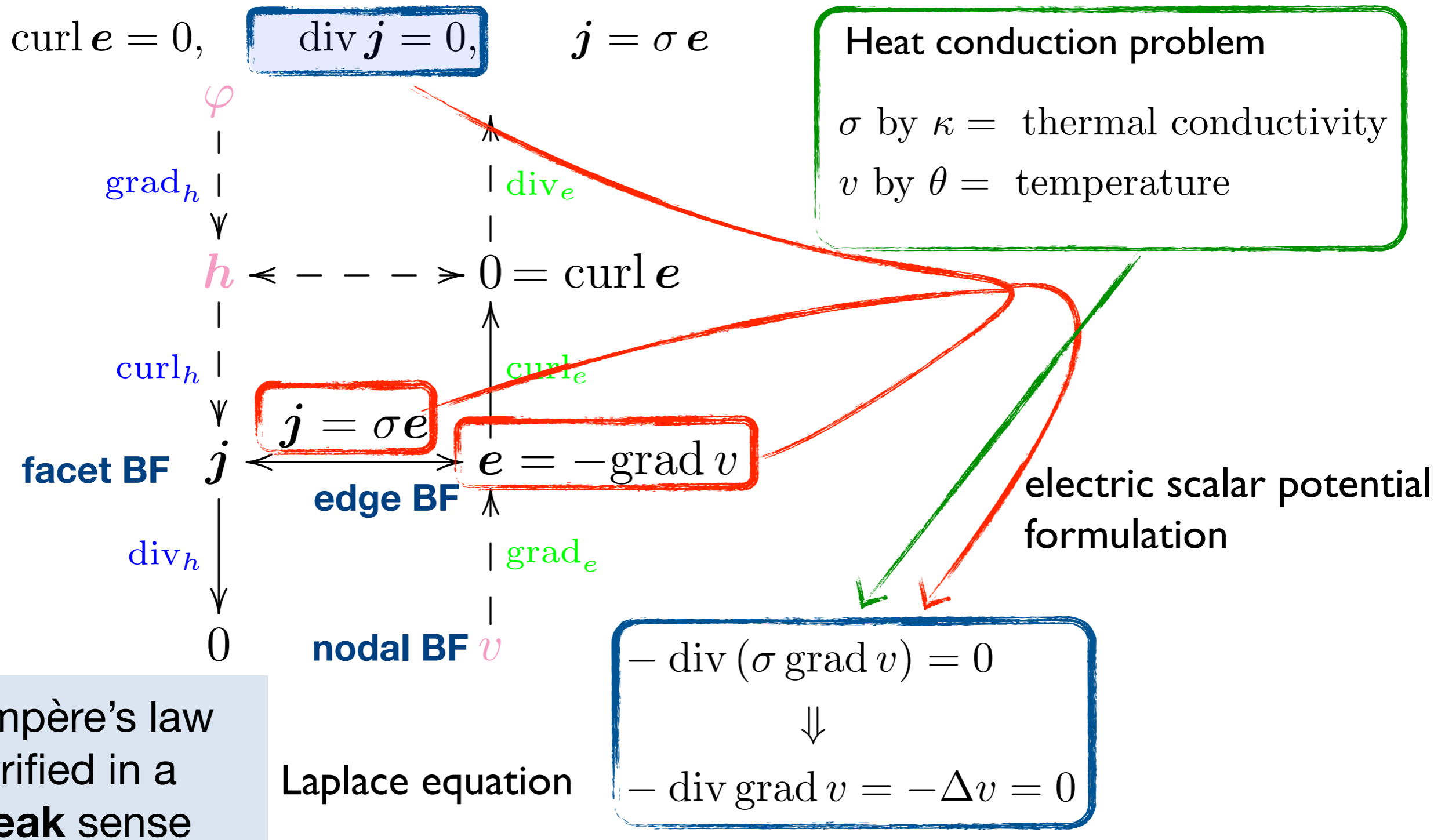
$$\begin{aligned} \mathbf{n} \times \mathbf{e}|_{\Gamma_e} &= 0 \\ \mathbf{n} \cdot \mathbf{j}|_{\Gamma_j} &= 0 \end{aligned}$$

“e side”





Electrokinetics



Spatial discretization — electrokinetics

Weighted residual approach

$$-\operatorname{div}(\sigma \operatorname{grad} v) = 0 \quad \text{in } \Omega_c$$

We integrate the equation weighted by test functions $w_i(\boldsymbol{x})$ over the whole domain:

$$\int_{\Omega} \left(-\operatorname{div}(\sigma \operatorname{grad} v) \right) w_i d\Omega = 0, \quad \forall w_i(\boldsymbol{x})$$

$w_i(\boldsymbol{x})$ weighting or test functions

Spatial discretization — electrokinetics (II)

Weak formulation $\forall w_i(\mathbf{x})$

$$\int_{\Omega} \left(-\operatorname{div}(\sigma \operatorname{grad} v) \right) w_i \, d\Omega = 0$$



$$\mathbf{v} \cdot \operatorname{grad} u + u \operatorname{div} \mathbf{v} = \operatorname{div}(u\mathbf{v})$$

integration by parts
Green formula

$$\int_{\Omega} \left(-\operatorname{div}(w_i \sigma \operatorname{grad} v) + \sigma \operatorname{grad} v \cdot \operatorname{grad} w_i \right) d\Omega = 0$$



$$\int_{\Omega} \operatorname{div} \mathbf{a} \, d\Omega = \oint_{\partial\Omega} \mathbf{a} \, d\Gamma$$

divergence theorem

$$\int_{\partial\Omega} w_i (-\sigma \operatorname{grad} v) \, d\Gamma + \int_{\Omega} \sigma \operatorname{grad} v \cdot \operatorname{grad} w_i \, d\Omega = 0$$

only the first derivative of the electric potential is now required

Spatial discretization — electrokinetics (III)

Boundary integral

$$\int_{\partial\Omega} w_i \underbrace{(-\sigma \text{grad } v)}_{\text{natural BC}} d\Gamma$$

$$\mathbf{n} \cdot \mathbf{j} = j_n = 0$$

$$\forall w_i(\mathbf{x})$$

$$\underbrace{\int_{\Gamma_{Dirichlet}} w_i(-\sigma \text{grad } v) d\Gamma}_{\text{essential BC}} + \underbrace{\int_{\Gamma_{Neumann}} w_i(-\sigma \text{grad } v) d\Gamma}_{\text{natural BC}}$$

$$= 0 \quad \forall w_i(\mathbf{x})$$

if $w_i = 0$ on $\Gamma_{Dirichlet}$

essential BC

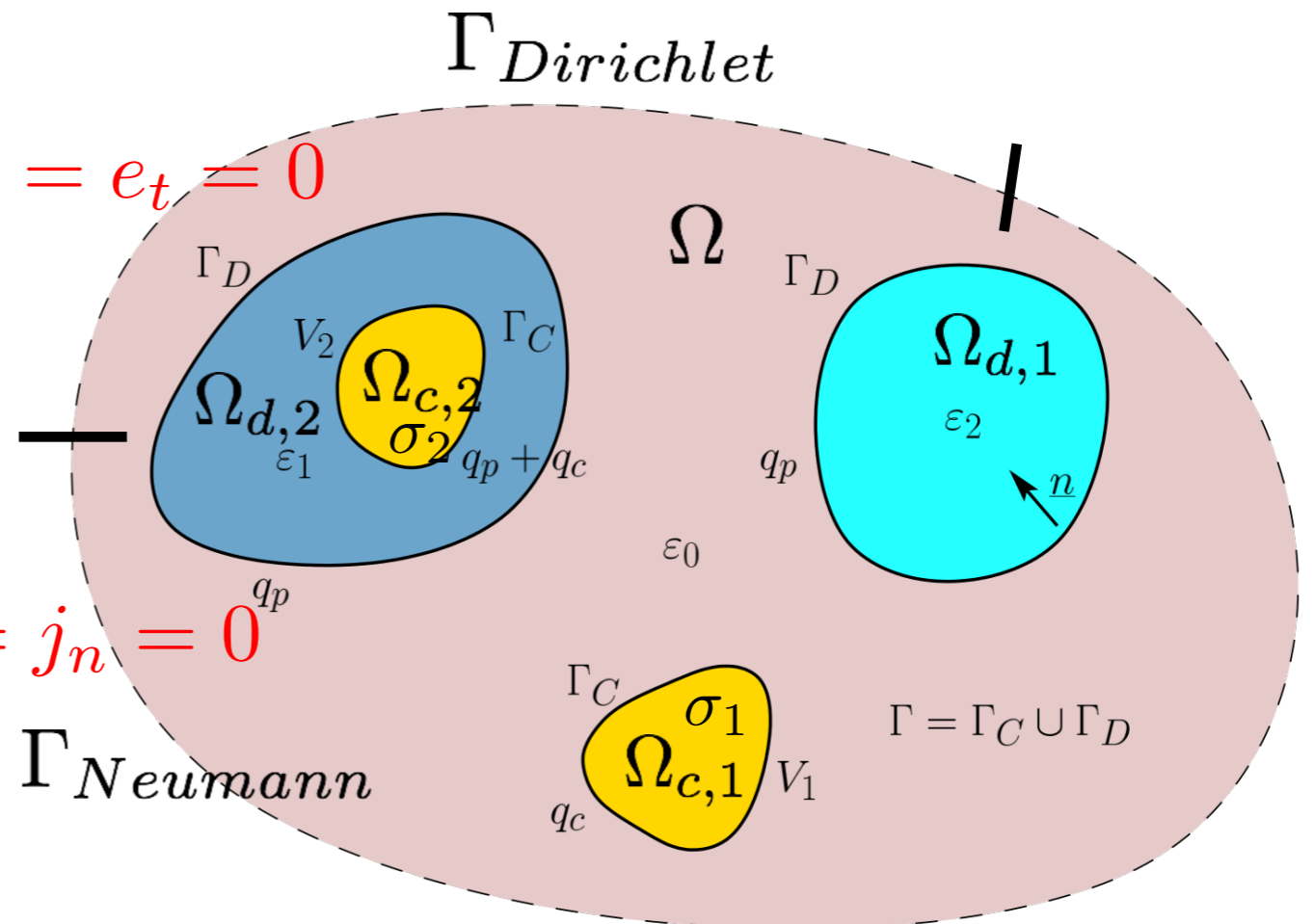
natural BC

$$\mathbf{n} \times \mathbf{e}|_{\Gamma_e} = 0$$

$$\mathbf{n} \cdot \mathbf{j}|_{\Gamma_j} = 0$$

$$\mathbf{n} \times \mathbf{e} = e_t = 0$$

$$\mathbf{n} \cdot \mathbf{j} = j_n = 0$$



Spatial discretization — electrokinetics (IV)

$$v = \sum_j u_j s_j \quad \text{with } s_j(\mathbf{x}) = 0 \text{ at } \Gamma_{Dirichlet}$$

$$\begin{cases} s_j(\mathbf{x}) & \text{shape functions} \\ u_j & \text{unknowns, degrees of freedom} \end{cases}$$

Ritz-Galerkin method $s_j(\mathbf{x}) = w_j(\mathbf{x})$

Petrov-Galerkin method $s_j(\mathbf{x}) \neq w_j(\mathbf{x})$

$$\int_{\Omega} \sigma \operatorname{grad} v \cdot \operatorname{grad} w_i \, d\Omega = 0$$

$$\sum_j u_j \underbrace{\int_{\Omega} \sigma \operatorname{grad} w_j \cdot \operatorname{grad} w_i \, d\Omega}_{= k_{ij}} = 0$$

matrix system

$$[k_{ij}][u_j] = [0]$$

Source via constraints
in U vector (potential at electrodes)

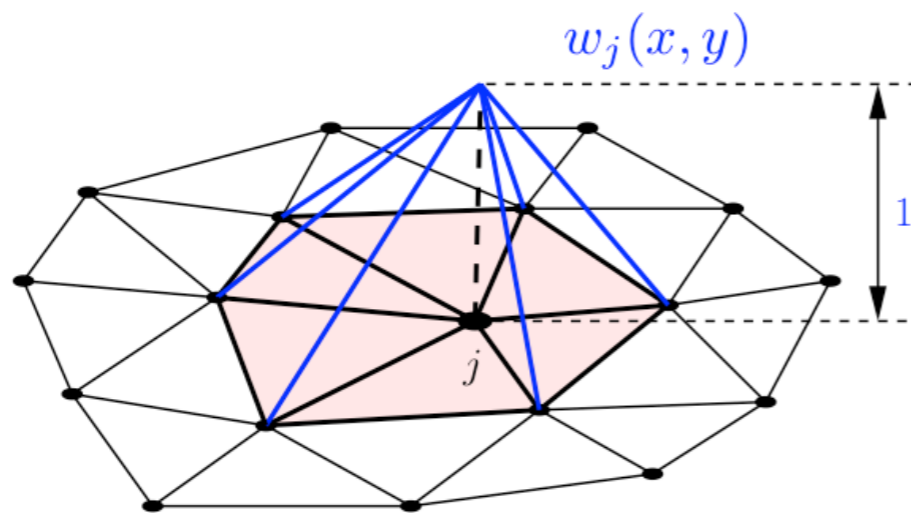
Function space – electric scalar potential

$$v = \sum_{n \in \mathcal{N}} v_n s_n$$

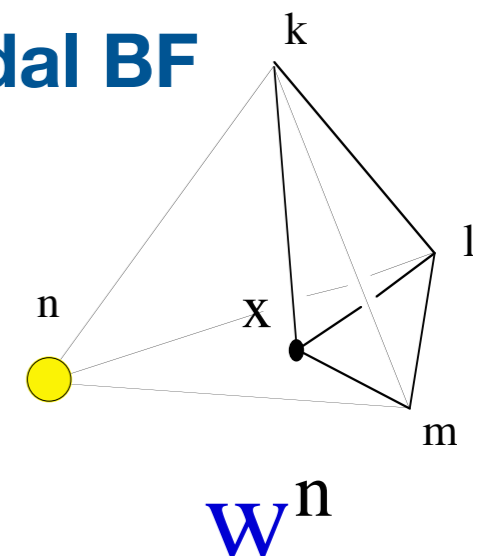
```

FunctionSpace{
  { Name Hgrad_v_EleKin; Type Form0; //discrete function space for H1_h
    BasisFunction {
      { Name sn; NameOfCoef vn; Function BF_Node; //“P1 FEs”
        Support DomainC_Ele; Entity NodesOf[All]; }
    }
    Constraint {
      { NameOfCoef vn; EntityType NodesOf; NameOfConstraint ElectricScalarPotential; }
    }
  }
}

```



**0-form
nodal BF**



Electrokinetic formulation: build equation!

```

Formulation {
  { Name Electrokinetics_v ; Type FemEquation ;
    Quantity {
      { Name v ; Type Local ; NameOfSpace Hgrad_v_EleKin ; }
    }
    Equation {
      Galerkin { [ sigma[] * Dof{d v} , {d v} ] ;
        In DomainC_Ele ; Jacobian Vol ; Integration GradGrad ; }
    }
  }
}

```

basis function

weighing function

$$\int_{\partial\Omega} w_i(-\sigma \text{grad } v) d\Gamma + \int_{\Omega} \sigma \text{grad } v \cdot \text{grad } w_i d\Omega = 0$$

Dirichlet constraint

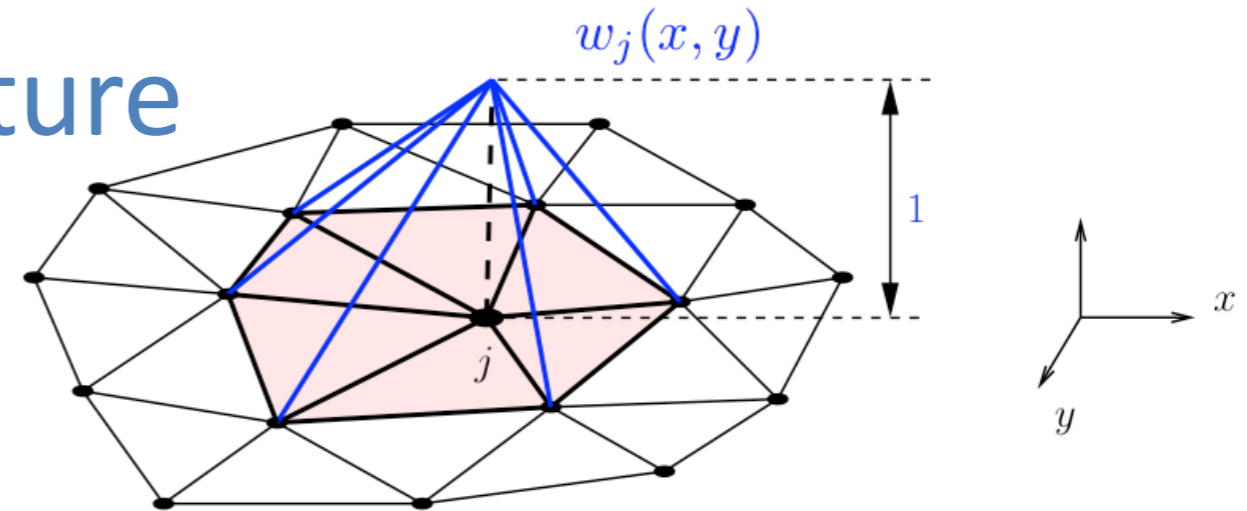
```

Constraint {
  { Name ElectricScalarPotential ;
    Case {
      { Region Dirichlet0 ; Value 0. ; }
      { Region Dirichlet1 ; Value V_imposed ; }
    }
  }
}

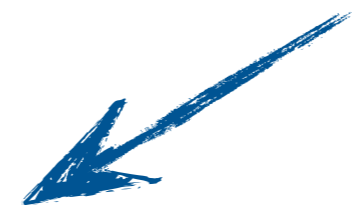
```

Function space – temperature

$$T = \sum_{n \in \mathcal{N}} T_n s_n$$



```
FunctionSpace{
  { Name Hgrad_T; Type Form0; //discrete function space for H1_h
    BasisFunction {
      { Name sn; NameOfCoef Tn; Function BF_Node; //‘‘P1 FEs’’
        Support Domain_The; Entity NodesOf[All]; }
    }
  Constraint {
    { NameOfCoef Tn; EntityType NodesOf; NameOfConstraint Temperature; }
  }
}
```



```
Constraint {
  { Name Temperature ;
    Case {
      { Region Dirichlet0 ; Value 20. ; }
      { Region Dirichlet1 ; Value 20. ; }
    }
  }
}
```

Thermal formulation: build equation!

```

Formulation {
  { Name Thermal_T ; Type FemEquation ;
    Quantity {
      { Name T ; Type Local ; NameOfSpace Hgrad_T ; }
      { Name v ; Type Local ; NameOfSpace Hgrad_v_EleKin ; }
    }
    Equation {

      Galerkin { [ kappa[] * Dof{d T} , {d T} ] ;
                 In Domain_The; Integration GradGrad ; Jacobian Vol ; }

      Galerkin { DtDof[ rhoc[] * Dof{T} , {T} ] ;
                 In Domain_The; Integration GradGrad ; Jacobian Vol ; }

      Galerkin { [ -1/sigma[] * SquNorm[sigma[]*{d v}] , {T} ] ;
                 In Domain_The ; Integration GradGrad ; Jacobian Vol ; }

    }
  }
}

```

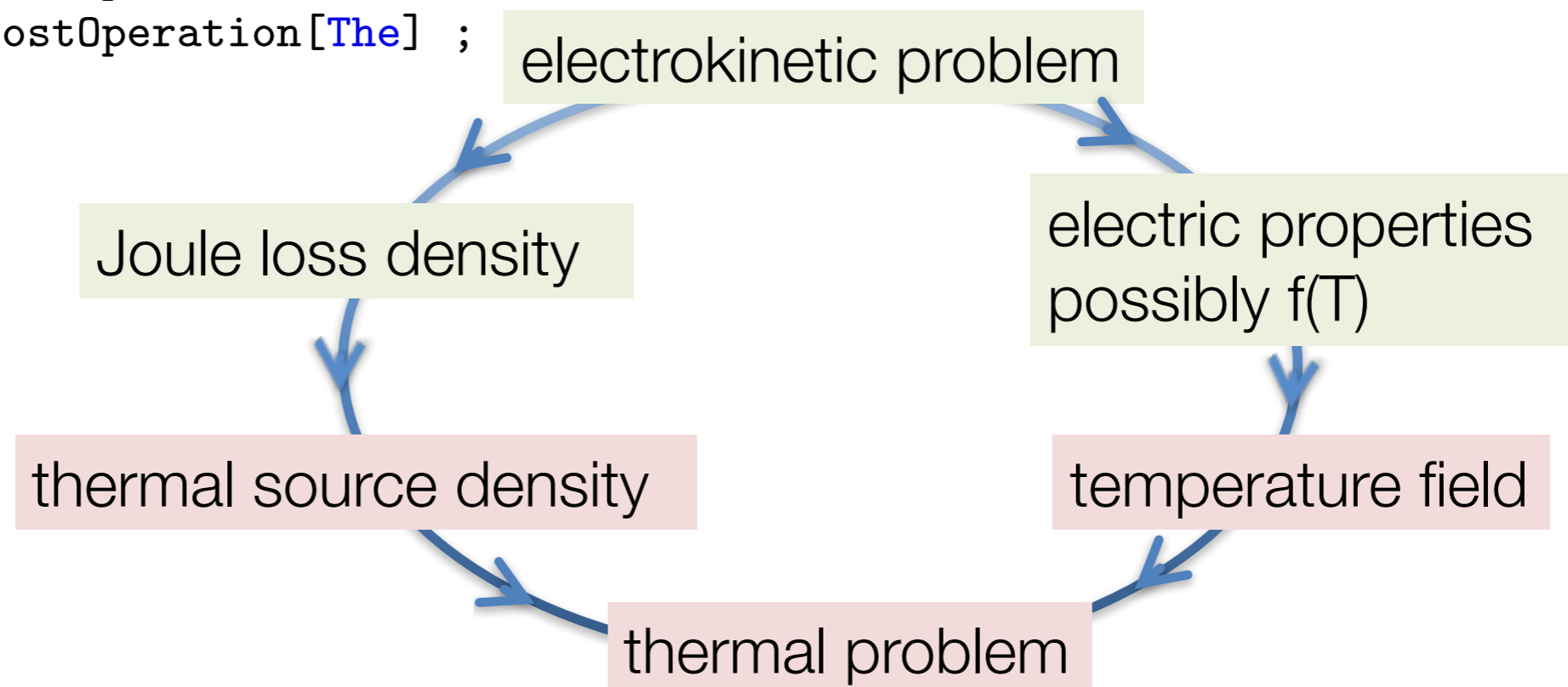
$$\int_{\Omega_T} \kappa \operatorname{grad} T \cdot \operatorname{grad} w_i \, d\Omega_T + \int_{\Omega_T} \rho_T c_p \partial_t T \cdot w_i \, d\Omega_T + = \int_{\Omega_T} Q_v \cdot w_i \, d\Omega_T$$

$$\rho c_p \partial_t T = \operatorname{div} (\kappa \operatorname{grad} T) + Q_v$$

Heat conduction equation

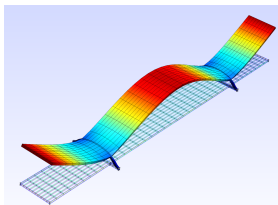
Coupling: Electrokinetic + thermal resolution

```
Resolution {  
  { Name Analysis ;  
    System {  
      { Name Sys_EleKin ; NameOfFormulation Electrokinetics_v ; }  
      { Name Sys_The ; NameOfFormulation Thermal_T ; }  
    }  
    Operation {  
      InitSolution[Sys_The] ;  
      InitSolution[Sys_EleKin] ;  
      Generate[Sys_EleKin] ; Solve[Sys_EleKin] ; SaveSolution[Sys_EleKin] ;  
      Generate[Sys_The] ; Solve[Sys_The] ; SaveSolution[Sys_The] ;  
      PostOperation[EleKin] ;  
      PostOperation[The] ;  
    }  
  }  
}
```



Mechanics and Elasticity

Magnetometer = Electromagnetism + Elasticity



classical Electromagnetism

Vector fields: $\vec{B}, \vec{H}, \vec{J}, \dots$

Vector analysis: $\text{div}, \text{curl}, \text{grad}$

Assumes Euclidean space



**Euclidean
metric**

classical Elasticity

Displacement field \vec{u}

Symmetric tensors ε, σ

Assumes Euclidean space

Magnetometer = Electromagnetism + Elasticity

classical Electromagnetism

Vector fields: $\vec{B}, \vec{H}, \vec{J}, \dots$

Vector analysis: $\text{div}, \text{curl}, \text{grad}$

Assumes Euclidean space



classical Elasticity

Displacement field \vec{u}

Symmetric tensors ε, σ

Assumes Euclidean space

Euclidean metric

- ◇ Classical Elasticity and classical Electromagnetism assume a Euclidean metric
- ◇ This assumption
 - engineers background
 - hides the central role of metric
 - electromagnetic world and elasticity world cannot communicate (no common concepts, the door is locked)
 - forbids clear electromechanical concepts (Maxwell stress tensor, electromagnetic forces)
 - forbids clear sensitivity analysis (cf tomorrow)

Magnetometer = Electromagnetism + Elasticity

classical Electromagnetism

Vector fields: $\vec{B}, \vec{H}, \vec{J}, \dots$

Vector analysis: $\text{div}, \text{curl}, \text{grad}$

Assumes Euclidean space



classical Elasticity

Displacement field \vec{u}

Symmetric tensors ε, σ

Assumes Euclidean space

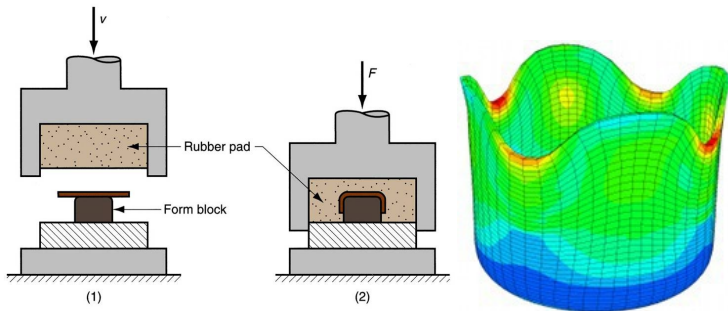
Euclidean metric

- ◇ One could have done a classical introduction, boring,
- ◇ We are going to open the door and see how the two worlds communicate
- ◇ Closer to research
- ◇ Define the metric tensor and demonstrate its fundamental role (giant step to understand curved spaces and general relativity)
- ◇ Show that all physical quantities we need are tensors of various kinds
- ◇ Finally, review the elastic part of the magnetometer model.

Elasticity

2 formulations for Elasticity

- ◇ full general geometric approach (large strains, finite strain, nonlinear elasticity)
- ◇ linear elasticity, infinitesimal strain: linearization of the latter (complicated → simple)



Pathway to tensors

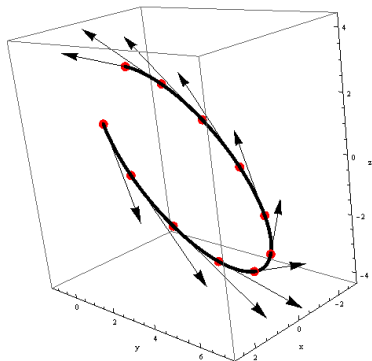
This all relies on a *chain* of basic mathematical concepts

- ◇ continuous medium (manifold)
- ◇ curve
- ◇ vector
- ◇ covector
- ◇ tensors

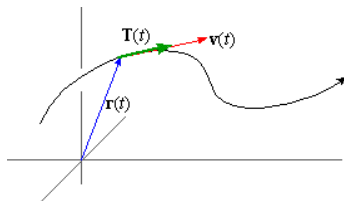
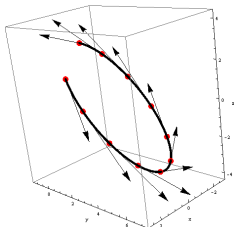
Manifold \rightarrow curve

Manifold:

- ◇ Analogy: sponge
- ◇ continuous set of points
- ◇ no defined distance between points $\langle \rangle$ rigid body
- ◇ no tearing, no creation of voids, ...
- ◇ Infinitely many curves pass through each point P of Ω



Curve \rightarrow tangent vector



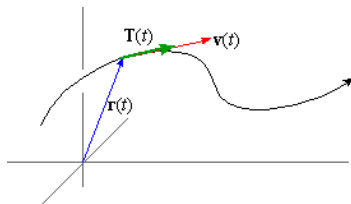
$$\begin{aligned}C(\lambda) &= \{x^i(\lambda)\}, \lambda \in [A, B] \\ &= \{x(\lambda), y(\lambda), z(\lambda)\}\end{aligned}$$

$$\begin{aligned}\vec{V}(\lambda) &= \left\{ \frac{dx^i}{d\lambda}(\lambda) \right\} \\ &= \left\{ \frac{dx}{d\lambda}, \frac{dy}{d\lambda}, \frac{dz}{d\lambda} \right\} \\ &= \{V^x, V^y, V^z\}\end{aligned}$$

$$\begin{aligned}\vec{r}(t) &= \{x^i(t)\}, t \in [A, B] \\ &= \{x(t), y(t), z(t)\}\end{aligned}$$

$$\begin{aligned}\vec{v}(t) &= \left\{ \frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt} \right\} \\ &= \{v^x, v^y, v^z\}\end{aligned}$$

Curve \rightarrow tangent vector



- ◇ Infinitely many curves pass through each point P of Ω
- ◇ Each curve has its *tangent vector* $\vec{V} = \{V^x, V^y, V^z\}$ in P
- ◇ the set of all tangent vectors at P is the tangent space $T_P\Omega$, a linear space
- ◇ if the curve is a trajectory, and the curve parameter λ is the time t , the tangent vector is the velocity
- ◇ Note the upper (contravariant) indices for vector components

Vector \rightarrow covector

- ◇ A covector field is a quantity that makes sense when integrated over a curve (circulation)
- ◇ Examples: force field \vec{f} , gravity field $m\vec{g}$, electric field \vec{E} , magnetic field \vec{H}

Mechanical work

the curve $C(P_A, P_B) = \{x^k(t)\}$ is a trajectory from P_A to P_B

$$\begin{aligned} W &= \int_{C(P_A, P_B)} \vec{f} \cdot d\vec{C} && \text{conventional vector representation} \\ &= \int_{C(P_A, P_B)} f_i dC^i && \text{note the lower (covariant) indices for } \vec{f} \\ &= \int_A^B f_i \frac{dx^i}{dt} dt && \text{using the parametrisation of the curve, implicit sum} \end{aligned}$$

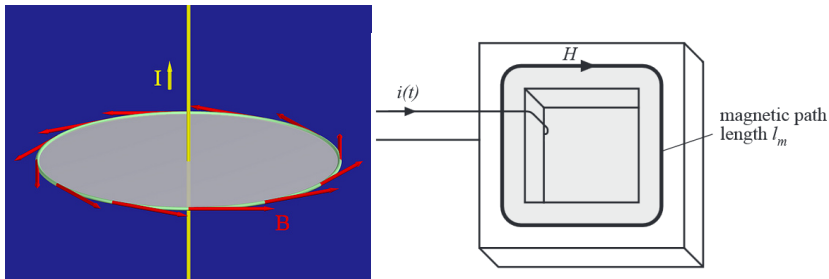
Vector \rightarrow covector

Potential gravitational energy

$$\begin{aligned} V &= \int_{C(P_A, P_B)} m\vec{g} \cdot d\vec{C} \\ &= \int_{C(P_A, P_B)} mg \operatorname{grad} z \cdot d\vec{C} && \operatorname{grad} z = \vec{e}_z \\ &= \int_A^B mg \frac{dz}{dx^i} \frac{dx^i}{d\lambda} d\lambda && \text{the gradient is a covector, cov. indices} \\ &= mg(z_B - z_A) \end{aligned}$$

Vector \rightarrow covector

Ampere law



$$I = \oint_C \vec{H} \cdot d\vec{C}$$

$$= \oint_C H_i \frac{dx^i}{d\lambda} d\lambda = \int_C \left(H_x \frac{dx}{d\lambda} + H_y \frac{dy}{d\lambda} + H_z \frac{dz}{d\lambda} \right) d\lambda$$

$$\vec{B} = \mu_0 \vec{H} = \oint_C H_i dC^i$$

<http://agni.phys.iit.edu/vpa/images/mag1.jpg>,
https://c1.staticflickr.com/3/2481/3789856303_7d9f858c0a.jpg

Vector and covectors \rightarrow tensors

- ◇ Vectors: velocities or tangent vectors of curves
- ◇ Covectors: physical quantities that make sense when integrated over a curve or a trajectory
- ◇ distinguished by covariant or contravariant indices
A vector: $\{V^x, V^y, V^z\}$
A covector: $\{\alpha_x, \alpha_y, \alpha_z\}$
- ◇ covector = first example of a tensor, the simplest one, a tensor with one vector argument and zero covector argument:

$$\alpha(\vec{V}) = \alpha_i V^i \in \mathbb{R}$$

- ◇ contraction of repeated indices, one covariant, one contravariant

Tensors: a general definition

- ◇ Generalisation of the principle of contraction for m vector arguments and n covector arguments:

$$T(\underbrace{\vec{V}, \dots, \vec{W}}_{m \text{ args}}, \underbrace{\alpha, \dots, \beta}_{n \text{ args}}) = T_{i \dots j}^{k \dots l} V^i \dots W^j \alpha_k \dots \beta_l$$

- ◇ Linearity for all arguments, e.g. for the first one

$$T(a\vec{V} + b\vec{W}, \dots) = aT(\vec{V}, \dots) + bT(\vec{W}, \dots)$$

- ◇ To the classroom: “Cite tensors...”
- ◇ In tensor analysis, tensors are NOT always 2 dimensional arrays of numbers!
- ◇ Scalars, vectors and covectors are the simplest tensor types

Tensors: transformation matrices

- ◇ Tensors with one vector argument ($m = 1$) and one covector argument ($n = 1$) are transformation matrices

$$T(\vec{V}, \alpha) = T_i^k V^i \alpha_k$$

- ◇ They have 9 independent components T_i^k (9 terms at the right-hand side of the above identity, due to implicit sums)
- ◇ Contracting the transformation matrix with its vector argument, $T(\vec{V}, \cdot)$, gives a tensor with one covector argument, i.e. a new vector
- ◇ Contracting the transformation matrix with its covector argument, $T(\cdot, \alpha)$, gives a tensor with one vector argument, i.e. a new covector
- ◇ Hence the name *transformation matrix*. Summary so far

m	n	
0	0	scalar field
0	1	vector field
1	0	covector field
1	1	transformation matrix
...		

Tensors with several arguments of the same nature

- ◇ Tensors with 2 vector arguments have also 9 independent components T_{ij} (like transformations matrices)

$$T(\vec{V}, \vec{W}) = T_{ij} V^i W^j.$$

- ◇ When there are at least two arguments of the same nature, the order of the arguments matters in general

$$T(\vec{V}, \vec{W}) = T_{ij} V^i W^j \neq T(\vec{W}, \vec{V}) = T_{ij} W^i V^j.$$

- ◇ The behaviour under swapping arguments of the same nature allows to define very important tensor subclasses: symmetric and antisymmetric tensors

Symmetric tensors

- ◇ A **symmetric tensor** verifies

$$T(\vec{V}, \vec{W}) = T(\vec{W}, \vec{V}) \quad \forall \vec{V}, \vec{W}$$

- ◇ The above condition holds in particular for the basis vectors
 $\vec{e}_x = \{\delta_x^i\} = \{1, 0, 0\}$, $\vec{e}_y = \{\delta_y^i\} = \{0, 1, 0\}$, $\vec{e}_z = \{\delta_z^i\} = \{0, 0, 1\}$.
- ◇ Hence

$$T(\vec{e}_x, \vec{e}_y) = T(\vec{e}_y, \vec{e}_x) \Rightarrow T_{xy} = T_{yx}$$

- ◇ This yields in total 3 non trivial independent relationships

$$T_{xy} = T_{yx} \quad , \quad T_{yz} = T_{zy} \quad , \quad T_{zx} = T_{xz}$$

- ◇ Symmetric tensors with 2 vector arguments have thus only $6=9-3$ independent components.
- ◇ Examples: metric tensor g , deformation tensor ε

Antisymmetric tensors

- ◇ Another important class of tensors are the **antisymmetric tensor**:

$$T(\vec{V}, \vec{W}) = -T(\vec{W}, \vec{V}) \quad \forall \vec{V}, \vec{W}$$

- ◇ Components with repeated indices of antisymmetric tensors always vanish

$$T(\vec{e}_x, \vec{e}_x) = -T(\vec{e}_x, \vec{e}_x) \Rightarrow T_{xx} = -T_{xx} \Rightarrow T_{xx} = 0$$

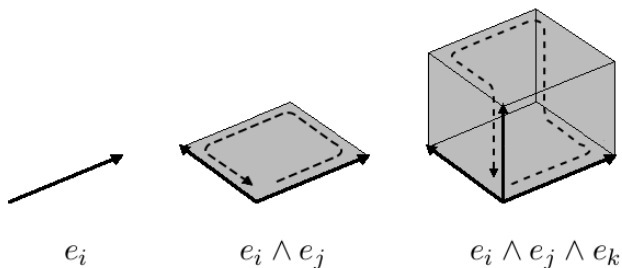
- ◇ Antisymmetry yields in total 6 non trivial independent relationships

$$T_{xx} = 0 \quad , \quad T_{yy} = 0 \quad , \quad T_{zz} = 0$$

$$T_{xy} = -T_{yx} \quad , \quad T_{yz} = -T_{zy} \quad , \quad T_{zx} = -T_{xz}$$

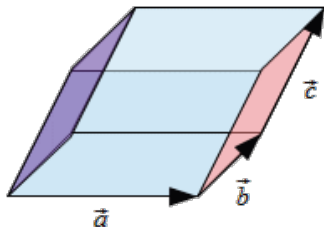
- ◇ Antisymmetric tensors with 2 vector arguments have thus $3=9-6$ independent components, like vectors and covectors.
- ◇ Examples: current density \vec{J} , magnetic flux density \vec{B}

Flux density



- ◇ A flux density associates a number to a facet made of 2 nonparallel vectors
- ◇ The flux is zero if the two vectors are parallel
- ◇ The flux changes sign if the vectors are swapped
- ◇ \Rightarrow a flux density is an antisymmetric tensor with 2 vector arguments ($m = 2, n = 0$)
- ◇ It has 3 components like a vector or a covector...

Volume density



- ◇ A volume density associates a number to a parallelepiped made of 3 nonparallel vectors
- ◇ The number is zero if the any two vectors out of the three are parallel (flat parallelepiped)
- ◇ The number changes sign if any two vectors are swapped (negative volume according to the right-hand rule)
- ◇ \Rightarrow a volume density is an antisymmetric tensor with 3 vector arguments ($m = 3, n = 0$)
- ◇ How many independent components has such a tensor?

Volume density

- ◇ A tensor with 3 vector arguments ($m = 3, n = 0$) has 27 independent components (27 terms in the sum):

$$T(\vec{U}, \vec{V}, \vec{W}) = T_{ijk} U^i V^j W^k$$

- ◇ A **volume density** ρ is a fully antisymmetric tensor with 3 vector arguments

$$\rho(\vec{U}, \vec{V}, \vec{W}) = -\rho(\vec{V}, \vec{U}, \vec{W}) = -\rho(\vec{U}, \vec{W}, \vec{V}) \dots \quad \forall \vec{U}, \vec{V}, \vec{W}$$

and so on for any argument permutation

- ◇ All 21 components with a repeated index vanish: e.g. $\rho_{xxy} = 0$
- ◇ Only 6 components exist with no repeated indices, for which one has 5 relationships

$$\rho_{xyz} = \rho_{yzx} = \rho_{zxy} = -\rho_{xzy} = -\rho_{zyx} = -\rho_{yxz}$$

- ◇ Antisymmetric tensors with 3 vector arguments have thus only 1 independent components, like a scalar field

Differential forms

- ◇ Differential forms of order m are completely antisymmetric tensors with m vector arguments and zero covector arguments
- ◇ Originate from the theory of integration over geometrical regions
- ◇ **Stokes theorem**: duality with the boundary operator ∂

$$\int_{\Omega} d\alpha = \int_{\partial\Omega} \alpha$$

$$\int_{C(P_A, P_B)} \text{grad} f \cdot d\vec{C} = f|_{\partial C(P_A, P_B)} = f(P_B) - f(P_A)$$

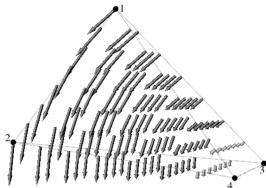
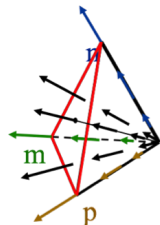
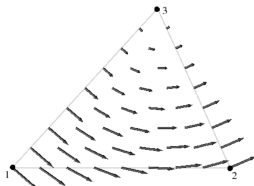
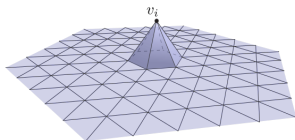
$$\int_S \text{curl} \vec{f} \cdot d\vec{S} = \int_{\partial S} \vec{f} \cdot d\partial\vec{S}$$

$$\int_V \text{div} \vec{f} dV = \int_{\partial V} \vec{f} \cdot d\partial\vec{V} = \int_{\partial V} \vec{f} \cdot \vec{n} d\partial V$$

- ◇ Accurate mathematical representation of electromagnetic fields

Differential forms: shape functions

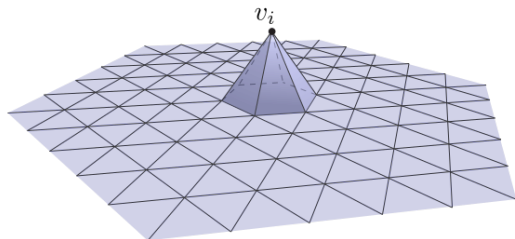
m		
0	0-form	scalar function
1	1-form	covector field, circulation density
2	2-form	flux density
3	3-form	volume density



0-forms: shape functions

m		
0	0-form	scalar function
1	1-form	covector field, circulation density
2	2-form	flux density
3	3-form	volume density

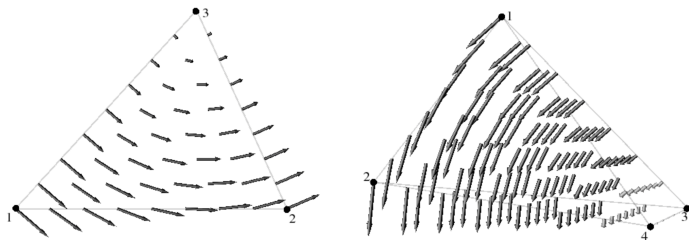
The value of the shape function v_i of node i of the mesh is 1 at node i , and zero at all other nodes of the mesh.



1-forms: shape functions

m		
0	0-form	scalar function
1	1-form	covector field, circulation density
2	2-form	flux density
3	3-form	volume density

The circulation of the shape function k is 1 over edge k of the mesh, and zero over all other edges of the mesh.

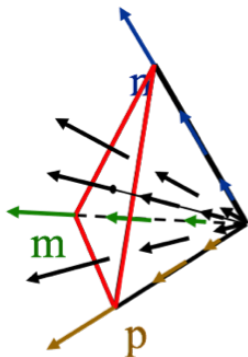


<http://www.iue.tuwien.ac.at/phd/nentchev/node40.html>,
<http://www.iue.tuwien.ac.at/phd/nentchev/node43.html>

2-forms: shape functions

m		
0	0-form	scalar function
1	1-form	covector field, circulation density
2	2-form	flux density
3	3-form	volume density

The flux of the shape function k is 1 across facet k of the mesh, and zero across all other facets of the mesh.



3-forms: shape functions

m		
0	0-form	scalar function
1	1-form	covector field, circulation density
2	2-form	flux density
3	3-form	volume density

The integral of the shape function k is 1 over element k of the mesh, and zero over all other elements of the mesh.

Draw it . . .

Tensors: overview

m	n	S/A	
0	0		scalar field
0	1		vector field
1	0		covector field
1	1		transformation matrix
2	0	S	metric tensor g , deformation tensor ε
2	0	A	flux density \vec{B} , displacement current \vec{D}
0	2	S	(stress tensor σ)
3	0	A	volume density ρ
...			

Tensors: partial antisymmetrization

- ◇ Partially antisymmetrized tensors are also possible
- ◇ The force density $\rho \vec{F}$ is a covector valued volume density

$$\vec{F} = \int_V \rho \vec{F} dV$$

- ◇ It is a tensor with 4 vector arguments, antisymmetrised with respect to the first 3 only.
- ◇ The stress tensor σ is a symmetric tensor with two covector arguments in

$$W = \sigma : \varepsilon = \sigma^{ij} \varepsilon_{ij}$$

- ◇ But the stress tensor σ is a tensor with three vector arguments, antisymmetrized over the first 2, when regarded as a surface density of force, or a flux of momentum.

$$\vec{F} = - \int_S \sigma \cdot d\vec{S} = - \int_S \sigma \cdot \vec{n} dS$$

Tensors: overview

- ◇ Physical quantities are tensor fields with well-defined arguments and symmetry properties
- ◇ Partially antisymmetrized tensors are also possible
- ◇ Many confusions in Euclidean space:
vector, covectors, flux densities, covector densities have all 3 components, and are all regarded as “vectors” in vector analysis, despite their very different tensorial natures

Elasticity: equilibrium equation

$$\vec{F} = - \int_S \sigma \cdot d\vec{S} = - \int_S \sigma \cdot \vec{n} dS \quad (*)$$

$$\vec{F} = \int_V \rho \vec{F} dV \quad \text{force density}$$

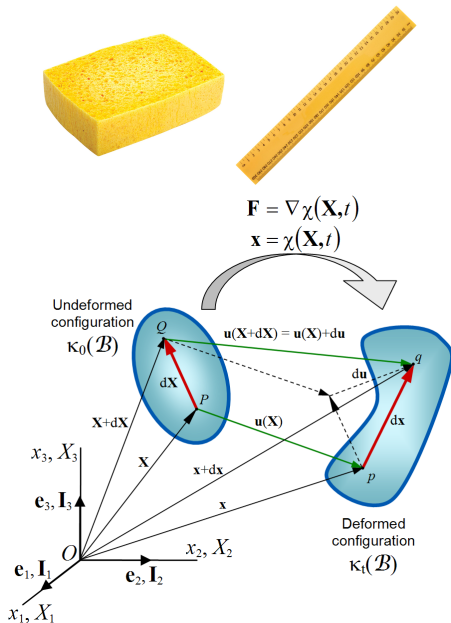
$$= - \int_{\partial V} \sigma \cdot \vec{n} d\partial V \quad \text{from (*)}$$

$$= - \int_V \text{div } \sigma dV \quad \text{Stokes theorem}$$

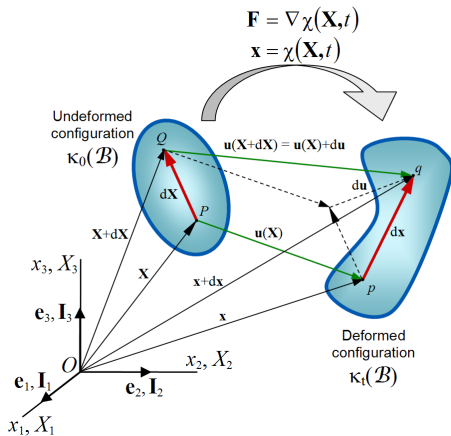
Hence, for free, the **equilibrium equation** which is a purely geometric statement, directly derived from the geometrical nature of σ and of force densities:

$$\text{div } \sigma + \rho \vec{F} = 0$$

Finite strain elasticity: placement map



Finite strain elasticity: placement map



- ◇ Material manifold: sponge
- ◇ Euclidean space: ruler, notion of distance between points
- ◇ placement map χ : description of motion

Metric tensor

- ◇ The metric tensor g is a symmetric tensor with two vector arguments
- ◇ The coefficients $g_{ij}(x, y, z)$ may depend on position (curved space)
- ◇ Defines the distance on a manifold
- ◇ Used in mesh generation also

Length of a curve

General case

- ◇ Consider a curve $C(P_A, P_B)$ from P_A to P_B with tangent vector \vec{V}
- ◇ The idea is to integrate a unit tangent vector along the curve to measure its length
- ◇ Normalized tangent vector, using the metric tensor g

$$\vec{t} = \frac{\vec{V}}{\sqrt{g(\vec{V}, \vec{V})}}$$

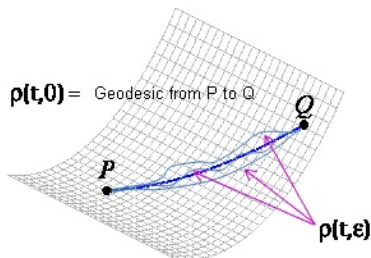
- ◇ Density of curve length is the covector $g(\vec{t})$, using again g
- ◇ Length of the curve

$$L_C = \int_{C(P_A, P_B)} g(\vec{t})$$

Distance between two points P_A and P_B

- ◇ Among all curves between P_A and P_B , the geodesic is the one with the minimal length
- ◇ The distance between P_A and P_B is the length of the geodesic

$$d(P_A, P_B) = \min_C L_C = \min_C \int_{C(P_A, P_B)} g(\vec{t})$$



Distance between two points P_A and P_B

Euclidean case

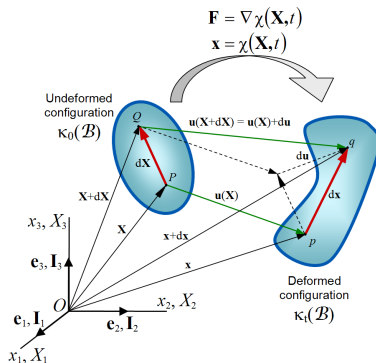
In the Euclidean space, everything is more simple

- ◇ $g_{ij} = \delta_{ij}$ using Cartesian coordinates
- ◇ Geodesics are straight lines
- ◇ The space is flat, distance depends only on end points

$$d(P_A, P_B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

- ◇ $\vec{t} = \frac{\vec{V}}{|\vec{V}|}$
- ◇ and the metric appears explicitly nowhere...

Placement map χ



- ◇ Material manifold
- ◇ Euclidean space
- ◇ Placement map χ : description of motion
- ◇ the placement map is a smooth invertible mapping
- ◇ It is sufficient to map all tensor quantities as well

Finite strain, another byproduct of g

Cauchy-Green tensor: \rightarrow distance in the reference configuration

$$\begin{aligned} C &= \chi g && \text{a non trivial metric tensor} \\ &= F^T F && F \text{ is the jacobian matrix of } \chi \end{aligned}$$

Green tensor

$$\begin{aligned} E &= \frac{1}{2}(\chi g - g) && \text{general expression} \\ &= \frac{1}{2}(C - \mathbb{I}) && \text{in Euclidean space, } g = \mathbb{I}, g_{ij} = \delta_{ij} \\ E_{ij} &= \frac{1}{2} \left(\frac{du^k}{dx^j} \delta_{ki} + \frac{du^k}{dx^i} \delta_{kj} + \frac{du^k}{dx^i} \frac{du^k}{dx^j} \right) && \delta_{ki} \text{ factors needed to balance indices} \end{aligned}$$

Infinitesimal strain: linearisation of E for small deformations

$$\varepsilon_{ij} = \frac{1}{2} \left(\frac{du^k}{dx^j} \delta_{ki} + \frac{du^k}{dx^i} \delta_{kj} \right) \stackrel{!}{=} \frac{1}{2} \left(\frac{du^i}{dx^j} + \frac{du^j}{dx^i} \right) = \frac{1}{2} ((\nabla u) + (\nabla u)^T)$$

Hooke law

- ◇ Last piece of the elastic model: the constitutive relationship $\sigma(\nabla u)$ or $\sigma(\varepsilon)$
- ◇ We now proceed in Euclidean space, and stop balancing covariant and contravariant indices (classical elasticity). Implicit summation remains on repeated indices.
- ◇ Classically, Hooke tensor with 81 components, then simplifying assumptions

$$\sigma_{ij} = H_{ijkl} \varepsilon_{kl}$$

- ◇ A more intuitive approach is to decompose the 9 dimensional space of 3x3 tensors into relevant geometric subspaces
- ◇ Let A be a 3x3 tensor, \mathbb{I} be the identity tensor. One has

$$\underset{\text{dim}=9}{A} = \underset{\text{dim}=5}{\text{dev}(A)} + \underset{\text{dim}=1}{\mathbb{I} \text{vol}(A)} + \underset{\text{dim}=3}{\text{antisym}(A)}$$

Hooke law

$$\underset{\text{dim}=9}{\mathbf{A}} = \underset{\text{dim}=5}{\text{dev}(\mathbf{A})} + \mathbb{I} \underset{\text{dim}=1}{\text{vol}(\mathbf{A})} + \underset{\text{dim}=3}{\text{antisym}(\mathbf{A})}$$

with

$$\text{sym}(\mathbf{A}) = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$$

$$\text{antisym}(\mathbf{A}) = \frac{1}{2}(\mathbf{A} - \mathbf{A}^T)$$

rotations

$$\text{vol}(\mathbf{A}) = \frac{1}{3} \sum_k A_{kk} = \frac{1}{3} \text{trace}(\mathbf{A})$$

compression/dilatation

$$\text{dev}(\mathbf{A}) = \text{sym}(\mathbf{A}) - \mathbb{I} \text{vol}(\mathbf{A})$$

shear

Hooke law (linear isotropic material) is then just a linear relation between the different subspaces:

$$\text{vol}(\boldsymbol{\sigma}) = 3K \text{vol}(\nabla u)$$

$3K =$ bulk modulus

$$\text{dev}(\boldsymbol{\sigma}) = 2G \text{dev}(\nabla u)$$

$2G =$ shear modulus

$$\text{antisym}(\boldsymbol{\sigma}) = 0$$

rotations yield no stress

Hooke law

$$\begin{aligned}\sigma &= \text{dev}(\sigma) + \mathbb{I} \text{vol}(\sigma) + \text{antisym}(\sigma) \\ &= 2G \text{dev}(\nabla u) + 3K \mathbb{I} \text{vol}(\nabla u) + 0 \\ &= (3K - 2G) \mathbb{I} \text{vol}(\nabla u) + 2G \text{sym}(\nabla u) \\ &= 3\lambda \mathbb{I} \text{vol}(\nabla u) + 2\mu \text{sym}(\nabla u) \\ &= \lambda \mathbb{I} \text{trace}(\nabla u) + 2\mu \text{sym}(\nabla u)\end{aligned}$$

Lamé coefficients

$$3\lambda = 3K - 2G = \frac{3E\nu}{(1+\nu)(1-2\nu)}$$

$$2\mu = 2G = \frac{E}{1+\nu}$$

$$3\lambda + 2\mu = 3K = \frac{E}{1-2\nu}$$

$$\lambda + 2\mu = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)}$$

Inverse Hooke law

$$\text{vol}(\nabla u) = \frac{1}{3K} \text{vol}(\sigma)$$

$3K =$ bulk modulus

$$\text{dev}(\nabla u) = \frac{1}{2G} \text{dev}(\sigma)$$

$2G =$ shear modulus

$$\text{antisym}(\sigma) = 0$$

rotations yield no stress

$$\begin{aligned} \nabla u &= \text{dev}(\nabla u) + \mathbb{I} \text{vol}(\nabla u) + \text{antisym}(\nabla u) \\ &= \frac{1}{2G} \text{dev}(\sigma) + \frac{1}{3K} \mathbb{I} \text{vol}(\sigma) + \text{antisym}(\nabla u) \\ &= \left(\frac{1}{3K} - \frac{1}{2G} \right) \mathbb{I} \text{vol}(\sigma) + \frac{1}{2G} \sigma + \text{antisym}(\nabla u) \end{aligned}$$

Inverse Hooke law

$$\begin{aligned}\varepsilon &= \nabla u - \text{antisym}(\nabla u) = \left(\frac{1}{3K} - \frac{1}{2G}\right)\mathbb{I} \text{vol}(\sigma) + \frac{1}{2G} \sigma \\ &= -\frac{\nu}{E} \text{trace}(\sigma)\mathbb{I} + \frac{1+\nu}{E} \sigma\end{aligned}$$

- ◇ ∇u cannot be expressed uniquely from σ
- ◇ $\text{antisym}(\nabla u)$ must be obtained another way
- ◇ Lagrange multiplier in stress formulation

Magnetometer: Exercise 1

- ◇ Use course notes to write down Hooke law under the matrix form

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{xy} \\ \sigma_{xz} \\ \sigma_{yx} \\ \sigma_{yy} \\ \sigma_{yz} \\ \sigma_{zx} \\ \sigma_{zy} \\ \sigma_{zz} \end{pmatrix} = (?) \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial u}{\partial z} \\ \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial v}{\partial z} \\ \frac{\partial w}{\partial x} \\ \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial z} \end{pmatrix}$$

- ◇ Where is this to be found in the Magnetometer model files (filename, line number)?
- ◇ Check and compare.

Magnetometer: Exercise 2

- ◇ Where is the coupling magnetic force defined in the Magnetometer model files (filename, line number)?
- ◇ Write down its algebraic expression.
- ◇ What is the name of this force?

Magnetometer: Exercise 3

- ◇ The magnetometer bar can be regarded as a beam with rectangular cross section and flexural rigidity $EI = Eab^3/12$, where a is the width and b the thickness of the beam.
- ◇ Find different ways to double the rigidity of the beam.
- ◇ Make a static Onelab analysis and compare the obtained deflections with the reference situation.

Magnetometer: Exercise 4

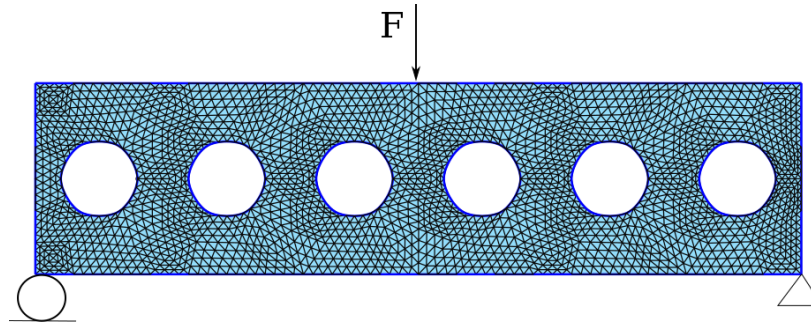
- ◇ Still in the static case, how can you double the deflection by modifying the applied voltage?
- ◇ How can you double the deflection by modifying electric conductivity of the magnetometer bar?

Optimization

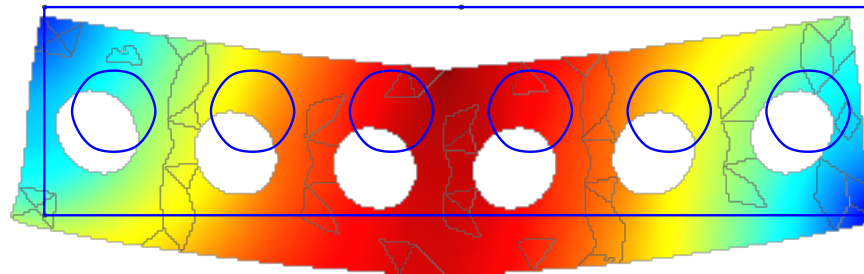
Design problem

Statement of the design problem

- Starting point: existing CAD model of a bridge



- Use FEM to account for the real geometry

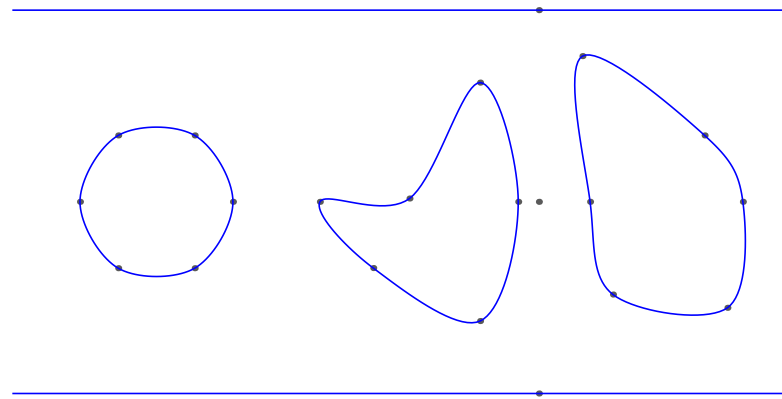


- “Optimal design” should find automatically the structure that minimizes the bridge deflection (for a given load F and support conditions)!

Selection of design variables

Shape optimization

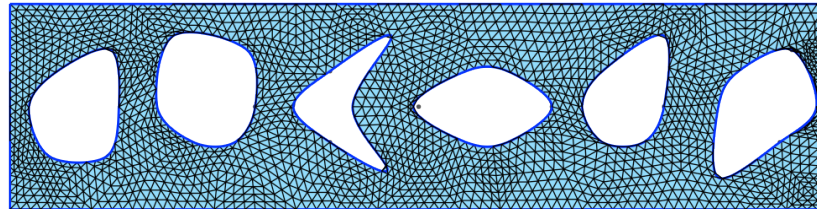
- Start with an existing CAD (from a previous design)



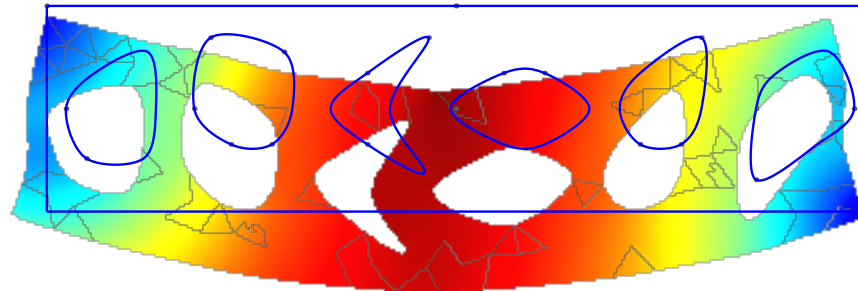
- Design variables are the parameters of existing CAD
 - radius of a circle
 - major/minor axis of ellipse
 - position of nurbs control point

Shape optimization

- Each value of design variables provides a novel geometry that must be re-meshed

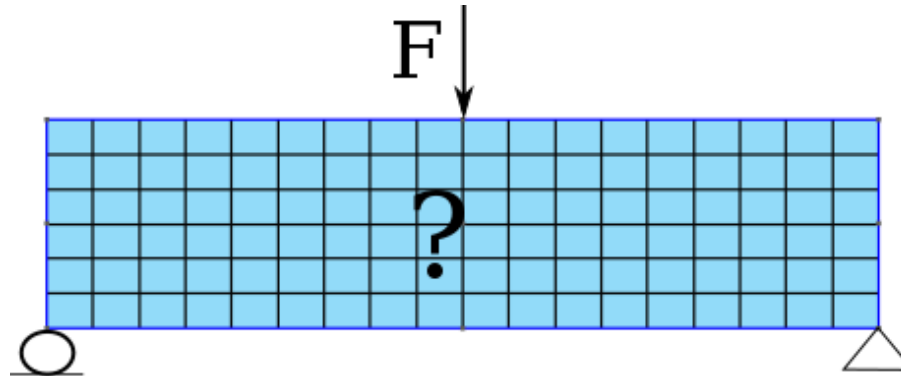


- A FEM analysis can then be applied to obtain the physical solution for the new geometry



Topology optimization

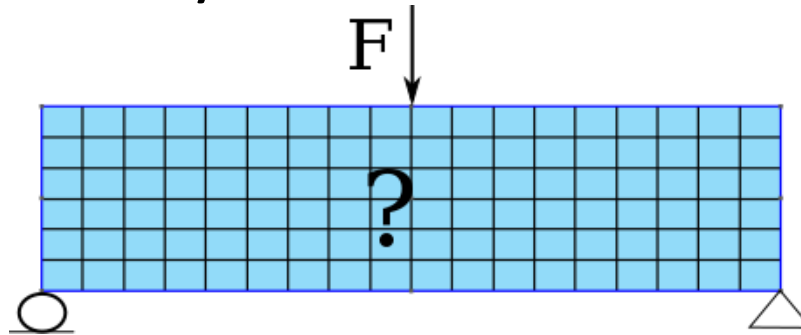
- Start from scratch with a block of material (steel)
 - with known loading and support conditions
 - without any initial guess of the initial structure



- What would be the “best” structure that
 - respects the loading and support conditions?
 - minimises the deflection?
 - uses only a given fraction of the initial volume?

Topology optimization

- Typically define 1 design variable/ FE as the material (e.g. steel) density that can be either 0 (void) or 1 (solid)



- In practice there are millions of FE for finer mesh
 - Impossible to check each combination since a FEM analysis is required for each of them
 - Relaxation of the problem: design variables vary continuously from 0 to 1 -> much more efficient!

FEM-analysis setting

Analysis setting - FEM model

- The physical behavior of the system is provided by PDEs written in a strong form (linear elastic model)

$$\operatorname{div} \sigma(\epsilon) + \mathbf{g} = 0 \quad \text{in } \Omega$$

$$\sigma(\epsilon) = C_{ijkl} \epsilon_{kl} \mathbf{e}_i \mathbf{e}_j^T \quad \text{in } \Omega$$

$$\mathbf{z} = 0 \quad \text{on } \partial\Omega$$

- The problem is equivalently written in a weak form with a residual

$$r(\boldsymbol{\tau}, \mathbf{z}^*, \bar{\mathbf{z}}) \equiv \int_{\Omega} \left(\sigma(\epsilon^*) : \nabla \bar{\mathbf{z}} - \mathbf{g} \cdot \bar{\mathbf{z}} \right) \mathbf{d}\Omega = 0, \quad \forall \bar{\mathbf{z}} \in Z_z^0$$

- The solution \mathbf{z}^* of the system depends implicitly and nonlinearly on design variables, i.e. on geometry through the FE mesh.

Analysis setting - FEM model

- Application of FEM in a mesh with N nodes, leads to the solution of linear system of algebraic equations

$$\bar{\mathbf{z}}^T \left(\mathbf{K}(\tau) \mathbf{z}^* - \mathbf{g}(\tau) \right) = 0, \forall \bar{\mathbf{z}} \in Z_z^0$$

- $\mathbf{z}(N \times 1)$ is the unknown field vector defined in each node of the FE mesh
- $\mathbf{K}(N \times N)$ is the stiffness matrix of the physical problem
- $\mathbf{g}(N \times 1)$ is a loading term (takes into account sources)

Analysis setting - FEM model

- Application of FEM in a mesh with N nodes, leads to the solution of (non)linear system of algebraic equations

$$\bar{\mathbf{z}}^T \left(\mathbf{K}(\tau) \mathbf{z}^*(\tau) - \mathbf{g}(\tau) \right) = 0, \forall \bar{\mathbf{z}} \in Z_z^0$$

- $\mathbf{z}(N \times 1)$ is the unknown field vector defined in each node of the FE mesh
- $\mathbf{K}(N \times N)$ is the stiffness matrix of the physical problem
- $\mathbf{g}(N \times 1)$ is a loading term (takes into account sources)

Analysis setting - FEM model

- FEM leads to the solution of (non)linear system of algebraic equations (in a mesh with N nodes)

$$\bar{\mathbf{z}}^T \left(\mathbf{K}(\tau) \mathbf{z}^*(\tau) - \mathbf{g}(\tau) \right) = 0, \forall \bar{\mathbf{z}} \in Z_z^0$$

- where $\mathbf{z}(N \times 1)$ is the unknown field vector
- $\mathbf{K}(N \times N)$ is the stiffness matrix of the physical problem
- $\mathbf{g}(N \times 1)$ is a loading term (takes into account sources)
- The system is usually solved using the Newton-Raphson method

Mathematical formulation of the optimization problem

Optimization Problem Formulation

- Find the n design variables τ that
 - minimize the internal energy of the structure
 - while using only a fraction α of the initial volume V_0

$$\begin{aligned} \min_{\tau} \quad & \frac{1}{2} \int_{\Omega} \sigma(\epsilon^*) : \epsilon^* \, d\Omega \\ \text{s.t.} \quad & V(\tau) \leq \alpha V_0, \\ & \tau_i^{\min} \leq \tau_i \leq \tau_i^{\max}, \quad i = 1, \dots, n \\ & r(\tau, \mathbf{z}^*, \bar{\mathbf{z}}) = 0, \quad \forall \bar{\mathbf{z}} \in Z_z^0. \end{aligned}$$

- Consideration of real CAD with the FEM constraint!
- Bound constraints on design variables.

Optimization Problem Formulation

- For each design variable (or configuration) τ
 - Mesh the structure (for shape optimization)
 - Run FEM analysis (time consuming) : z^*
 - evaluate objective and constraints (postprocessing)

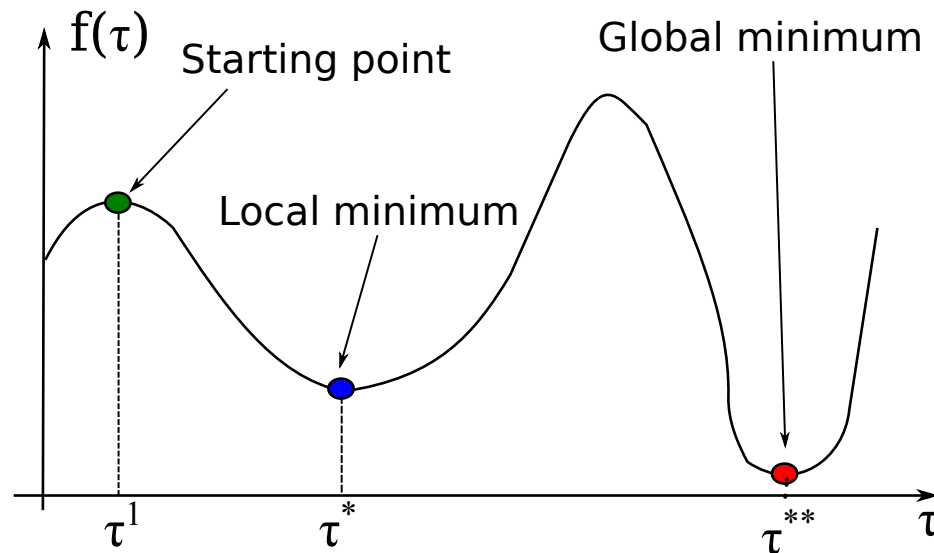
$$\begin{aligned} \min_{\tau} \quad & \frac{1}{2} \int_{\Omega} \sigma(\epsilon^*) : \epsilon^* \, d\Omega \\ \text{s.t.} \quad & V(\tau) \leq \alpha V_0, \\ & \tau_i^{min} \leq \tau_i \leq \tau_i^{max}, \quad i = 1, \dots, n \\ & r(\tau, \mathbf{z}^*, \bar{\mathbf{z}}) = 0, \quad \forall \bar{\mathbf{z}} \in Z_z^0. \end{aligned}$$

- Implicit and nonlinear dependence of z^* on τ
 - each function depends implicitly and nonlinearly on τ
 - time consuming evaluation since FEM is required
- Practically, there can be mil. of τ and inequalities!

Optimizer – resolution of the
optimization problem

Optimizer

- The functions in the optimization problem depend implicitly and nonlinearly on design variables (possibly very large number).



- How can we find the minimum of that function (assuming that we have only 1 design variable)?

Gradient-free optimizer

- Naturally converges to the global optimum!
- #function evaluations for a naïve method is $\frac{n!}{n!(n!-m!)}$
- For a small optimization problem,

$$n = 10, m = 5 : 252$$

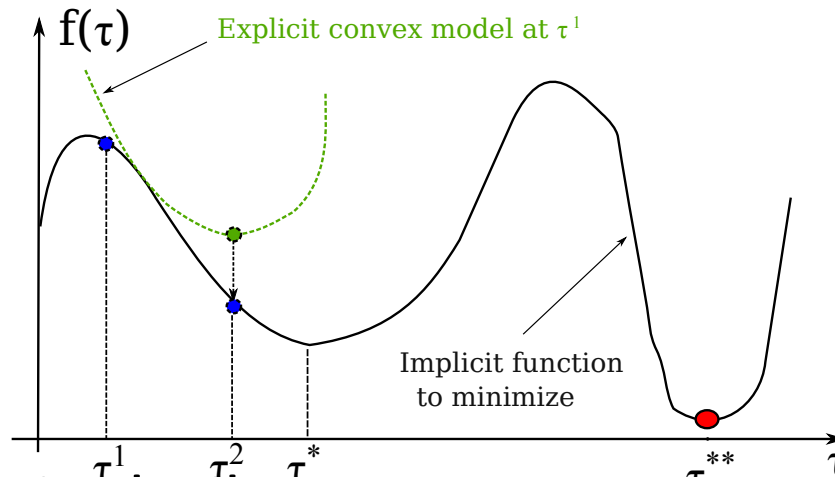
$$n = 20, m = 10 : 185000$$

$$n = 100, m = 50 : 10^{29}$$

- Impossible to explore the whole space with a naive method!
- Clever algorithms that can explore the design space are required!

Gradient-based optimizer

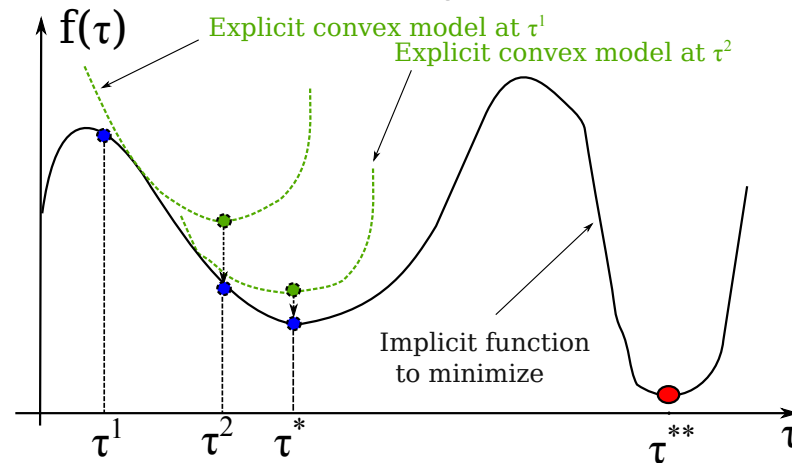
- Idea: At a given point τ^i run a FEM analysis to evaluate the function and its derivative τ^i .
- Then compute an explicit model (sort of Taylor) of the function (in which function evaluations do not take time compared to FEM!).



- The model is built as convex: τ^1 easy to find its minimum (conjugate gradient, or interior point).

Gradient-based optimizer

- Once the optimum of the explicit model at a given point is found, a new explicit model is built at that point and the procedure is repeated until convergence to the real optimum.

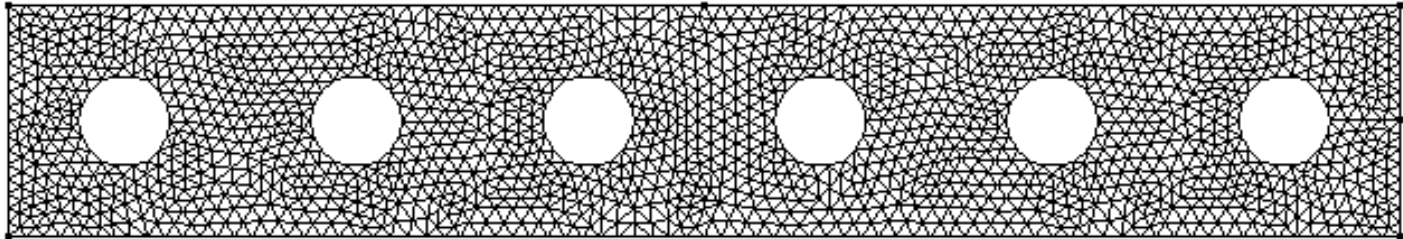


- Stay always feasible during iterations and higher convergence rate than gradient-free methods!

Application: optimal design of a bridge

Optimal design of a bridge

- Shape optimization approach



- Topology optimization approach (0-blue, 1-red)



- Fortunately the classical bridge is recovered!

Sensitivity analysis

Introduction

- “Sensitivity” is the derivative of a performance function f (appearing in the optimization problem) w.r.t. a design variable

$$\frac{d}{d\tau} f(\tau, \mathbf{z}^*(\tau)) = ?$$

- The performance function depends on the solution \mathbf{z}^* of the physical problem (at equilibrium)

$$\bar{\mathbf{z}}^T \left(\mathbf{K}(\tau) \mathbf{z}^*(\tau) - \mathbf{g}(\tau) \right) = 0, \forall \bar{\mathbf{z}} \in Z_z^0$$

- \mathbf{K}^{-1} is a NxN matrix with N number of nodes in mesh
-> obtained during the resolution of the problem!

Finite difference method

- First order Taylor approximation of $f(\tau, \mathbf{z}^*(\tau))$

$$\frac{df}{d\tau}(\tau, \mathbf{z}^*) \approx \frac{f(\tau + \delta\tau, \mathbf{z}_{\tau+\delta\tau}^*) - f(\tau, \mathbf{z}^*)}{\delta\tau}$$

- What is $\mathbf{z}_{\tau+\delta\tau}^*$? Solve anew the linear problem with the perturbed geometry (with perturbed mesh)

$$\bar{\mathbf{z}}^T \left(\mathbf{K}(\tau + \delta\tau) \mathbf{z}_{\tau+\delta\tau}^* - \mathbf{g}(\tau + \delta\tau) \right) = 0, \forall \bar{\mathbf{z}} \in Z_z^0$$

- Pros/Cons
 - Simplest method to compute the sensitivity
 - Very slow when the $\#\tau$ is large since $\#\tau$ additional FEM are required!

Analytic approach – direct method

- Sensitivity of the performance function

$$\frac{df}{d\tau}(\tau, \mathbf{z}^*) = \frac{\partial f}{\partial \tau}(\tau, \mathbf{z}^*) + \frac{\partial f}{\partial \mathbf{z}^*} \frac{d\mathbf{z}^*}{d\tau} ?$$

- Requires determining $\frac{d\mathbf{z}^*}{d\tau}$

$$\bar{\mathbf{z}}^T \frac{d}{d\tau} \left(\mathbf{K}(\tau) \mathbf{z}^*(\tau) - \mathbf{g}(\tau) \right) = 0, \forall \bar{\mathbf{z}} \in Z_z^0$$

- We end up with a linear system for $\frac{d\mathbf{z}^*}{d\tau}$

$$\bar{\mathbf{z}}^T \left(\mathbf{K}(\tau) \frac{d\mathbf{z}^*}{d\tau} + \left(\frac{\partial \mathbf{K}}{\partial \tau} \mathbf{z}^* - \frac{\partial \mathbf{g}}{\partial \tau} \right) \right) = 0, \forall \bar{\mathbf{z}} \in Z_z^0$$

- \mathbf{K}^{-1} known as equilibrium of physical system enforced
- with a loading depending only on design variable

Analytic approach – adjoint method

- Sensitivity of the performance function

$$\frac{df}{d\tau}(\tau, \mathbf{z}^*) = \frac{\partial f}{\partial \tau}(\tau, \mathbf{z}^*) + \frac{\partial f}{\partial \mathbf{z}^*} \frac{d\mathbf{z}^*}{d\tau} ?$$

- $\frac{d\mathbf{z}^*}{d\tau}$ is obtained from the direct method

$$\frac{d\mathbf{z}^*}{d\tau} = \mathbf{K}^{-1} \left(\frac{\partial \mathbf{K}}{\partial \tau} \mathbf{z}^*(\tau) - \frac{\partial \mathbf{g}}{\partial \tau} \right)$$

- The latter is injected into the sensitivity equation

$$\begin{aligned} \frac{df}{d\tau}(\tau, \mathbf{z}^*) &= \frac{\partial f}{\partial \tau}(\tau, \mathbf{z}^*) \\ &+ \frac{\partial f}{\partial \mathbf{z}^*} \cdot \mathbf{K}^{-1} \left(\frac{\partial \mathbf{K}}{\partial \tau} \mathbf{z}^*(\tau) - \frac{\partial \mathbf{g}}{\partial \tau} \right) \end{aligned}$$

Analytic approach – adjoint method

- Derivative of the function

$$\frac{df}{d\tau}(\tau, \mathbf{z}^*) = \frac{\partial f}{\partial \tau}(\tau, \mathbf{z}^*) + \lambda^T \left(\frac{\partial \mathbf{g}}{\partial \tau} - \frac{\partial \mathbf{K}}{\partial \tau} \mathbf{z}^* \right)$$

- Where λ is the solution of the linear adjoint system

$$\bar{\lambda}^T \left(\mathbf{K} \lambda - \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}^*) \right) = 0, \forall \bar{\lambda} \in Z_{\lambda}^0$$

- \mathbf{K}^{-1} known as equilibrium of physical system enforced
- with a loading depending only on the function

Discussion (real problem: $\#\tau \gg \#,f \gg$)

- Direct method

$$\frac{df}{d\tau}(\tau, \mathbf{z}^*) = \frac{\partial f}{\partial \tau}(\tau, \mathbf{z}^*) + \frac{\partial f}{\partial \mathbf{z}^*} \frac{d\mathbf{z}^*}{d\tau}$$

- Back-substitution for each design variable

$$\bar{\mathbf{z}}^T \left(\mathbf{K}(\tau) \frac{d\mathbf{z}^*}{d\tau} + \left(\frac{\partial \mathbf{K}}{\partial \tau} \mathbf{z}^* - \frac{\partial \mathbf{g}}{\partial \tau} \right) \right) = 0, \forall \bar{\mathbf{z}} \in Z_z^0$$

- Use only when $\#\tau \ll \#,f$!

- Adjoint method

$$\frac{df}{d\tau}(\tau, \mathbf{z}^*) = \frac{\partial f}{\partial \tau}(\tau, \mathbf{z}^*) + \lambda^T \left(\frac{\partial \mathbf{g}}{\partial \tau} - \frac{\partial \mathbf{K}}{\partial \tau} \mathbf{z}^* \right)$$

- Back-substitution for each performance function

$$\bar{\lambda}^T \left(\mathbf{K} \lambda - \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}^*) \right) = 0, \forall \bar{\lambda} \in Z_\lambda^0$$

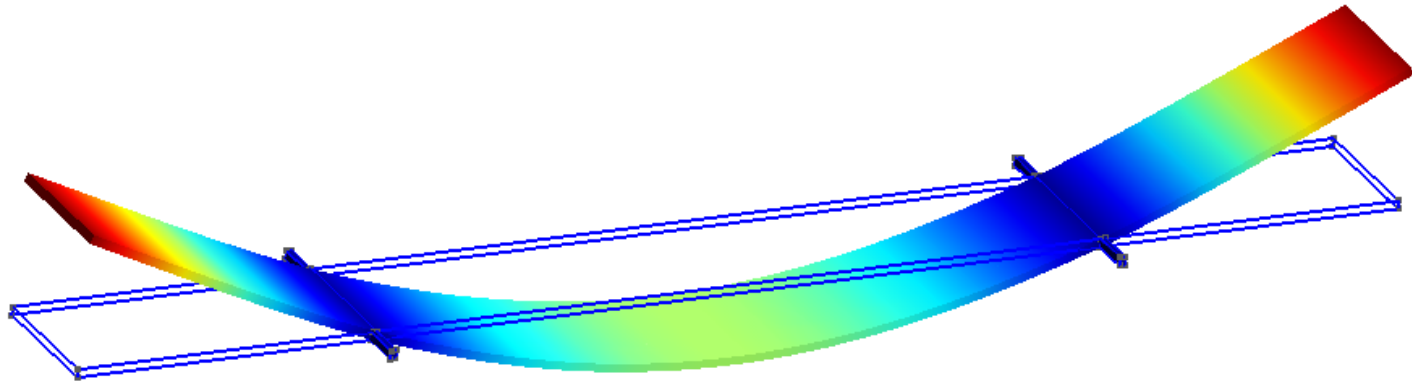
- Use only when $\#\tau \gg \#,f$!

- \mathbf{K}^{-1} known as equilibrium of physical system enforced!

Tutorial – Shape optimization of the magnetometer fundamental frequency

Physical problem

- Eigenmode at a given f_0 frequency



- Design problem
 - change the existing geometry
 - in order to set f_0 to a desired frequency f_0^d
 - How can we do this?

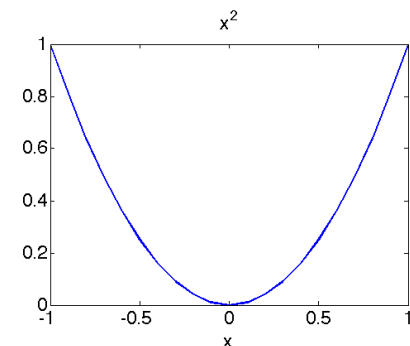
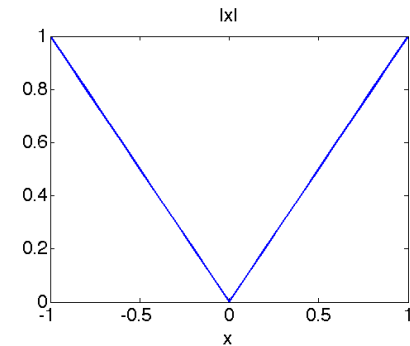
Shape optimization

- Choice of design variable(s)
 - Support length, or width, or position?
 - Relevant variables should give high sensitivity!
- Formulation of the optimization problem
 - First try

$$\min_{\tau_{min} \leq \tau \leq \tau_{max}} |f_0(\tau) - f_0^d|$$

- Second try

$$\min_{\tau_{min} \leq \tau \leq \tau_{max}} (f_0(\tau) - f_0^d)^2$$



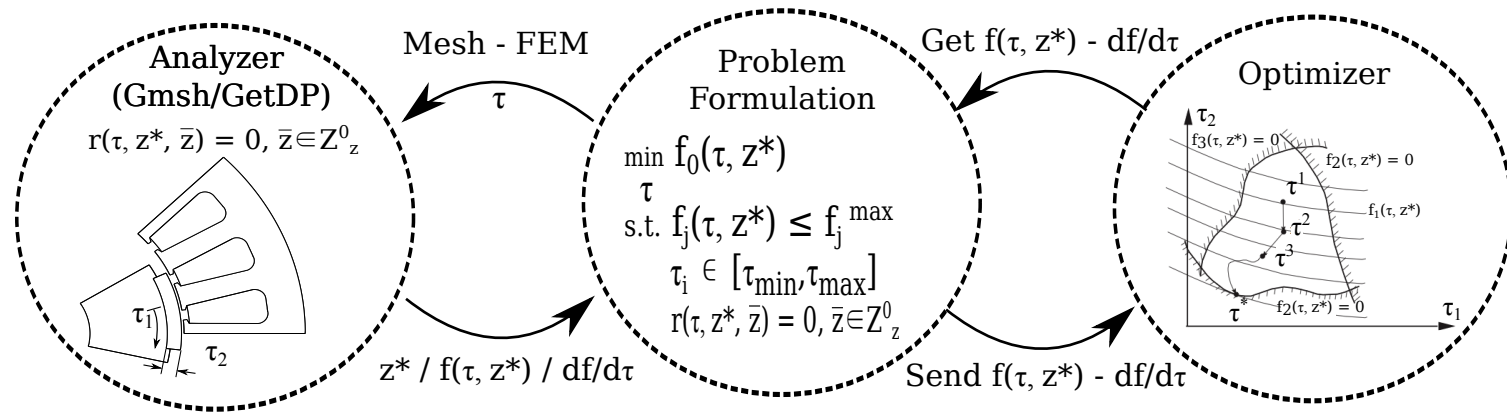
Shape optimization

- Solve the unconstrained optimization problem for τ set as support position

$$\min_{\tau_{min} \leq \tau \leq \tau_{max}} (f_0(\tau) - f_0^d)^2$$

- The problem is solved with the onelab-based optimization code written in Python (optdriver.py).
- Hint: check graphically if the optimizer finds the correct optimum by plotting the objective.

Onelab-based optimization



Optimization algorithm: (handle both shape and topology optimization)

Initialize design variables (user input)

Loop (until convergence to an optimum) ...

- Set the CAD (design variables) and mesh the CAD
- Analysis (FEM): evaluate performance functions
- Sensitivity analysis (FEM): evaluate the gradient of performance functions
- Call optimizer: update the design variables

Onelab-based optimization (Gmsh/GetDP)

- Define the objective as a function (.pro) and add the desired fundamental frequency in onelab database

```
Function {  
  // desired fundamental frequency (onelab parameter)  
  DefineConstant[  
    f0d = {1e5, Name StrCat("Desired fundamental frequency [Hz]")}  
  ];  
  
  // fundamental frequency  
  f0[] = $EigenvalueReal/(2*Pi);  
  
  // objective function  
  objective[] = SquNorm[ f0[] - f0d];  
}
```

Onelab-based optimization (Gmsh/GetDP)

- Call the objective function “objective[]” in the post-processing associated with the elastic model.

```
// Define objective function in post-processing
PostProcessing {
  { Name Elasticity3D ; NameOfFormulation Elasticity3D_u_coupled_transient;
    NameOfSystem Sys_Mec;
    PostQuantity {
      { Name objective ;
        Value {
          Term { Type Global; [ objective[] ]; In Domain; }
        }
      }
    }
  }
}
```

- Call the post-operation

```
// Send the objective function in Onelab database
PostOperation {
  { Name Mec ; NameOfPostProcessing Elasticity3D;
    Operation {
      Print[ objective, OnRegion Domain, TimeStep 0,
        SendToServer "Output/diffEigenFreqNorm" ];
    }
  }
}
```

Onelab-based optimization (Python)

- Create a onelab client and specify the problem
 - Objective and design variables as defined in onelab
 - Onelab parameters defining the model (e.g. f_0^d)
 - Resolution used in .pro

```
# Create a onelab client
clientOnelab = onelab.client(__file__)
if clientOnelab.action == 'check': exit(0)

# Set options for both optimization problem and CAD/FEM model
options = setOptions(clientOnelab,{
    'CADFEMOptions':{
        'Input/0Type of analysis':0,
        'Optimization/Desired natural frequency [Hz]':2.5e05},
    'file':clientOnelab.getPath('magnetometer'),
    'resolution':'Analysis',
    'objective':'Output/diffEigenFreqNorm',
    'constraints':[],
    'designVariables':['Input/Geometry/5Support position [m]'],
    'iterMax':50
})
```

Onelab-based optimization (Python)

- Specify the starting value, lower (τ_{min}) and upper bounds (τ_{max})

```
# Starting value of design variables
desVar = [1.5e-04]

# Lower and upper bounds of design variables
lowerBoundDesVar = [6.0e-05]
upperBoundDesVar = [2.0e-04]

# Upper bound of constraints
upperBoundConstr = []
```

- Create an optimization solver (mma)

```
# Create an optimizer client
clientOpt = mma.client({
    'm':options['m'],
    'xmin':lowerBoundDesVar, 'xmax':upperBoundDesVar,
    'asyinit':0.2, 'asyincr':0.8, 'asydecr':0.3
})
```

- Solve the optimization problem

```
# Call the optimization routine
xopt,objOpt,constrOpt = optimize(clientOnelab, clientOpt,
    desVar, lowerBoundDesVar, upperBoundDesVar, upperBoundConstr, options)
```

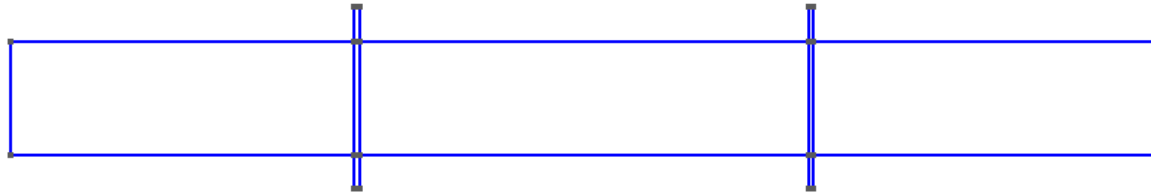
Run shape optimization ...

```
$ gmesh optdriver.py - -v 1
```

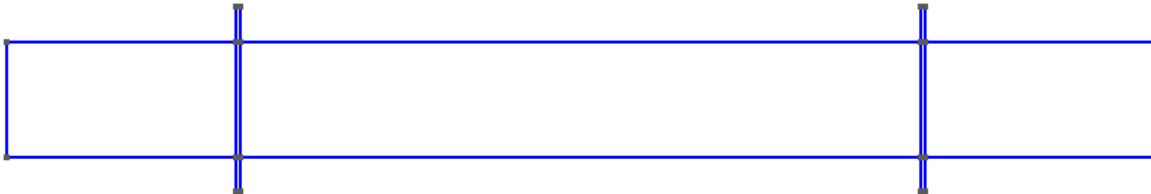
```
-----  
===                               Optimization Problem                               ===  
-----  
Model setting  
  * "Optimization/Desired fundamental frequency [Hz]": 250000.0  
  * "Input/Geometry/00Mesh size factor": 1  
  * "Input/0Type of analysis": 0  
Design variables (n:1)  
  * "Input/Geometry/5Support position [m]": 6.00e-05<=1.50e-04<=2.00e-04  
Performance functions (m:0)  
  * Objective: "Output/diffEigenFreqNorm"  
Stop criteria  
  Design variables tol. : 1e-06  
  Maximum number of iterations : 50  
-----  
It.   1,x0: 1.500e-04,f0: 7.609e+09,change: 1.495e-05  
It.   2,x0: 1.351e-04,f0: 3.596e+09,change: 2.265e-05  
It.   3,x0: 1.124e-04,f0: 3.048e+08,change: 1.780e-05  
It.   4,x0: 9.465e-05,f0: 8.853e+06,change: 6.870e-06  
It.   5,x0: 1.015e-04,f0: 5.935e+06,change: 1.235e-06  
It.   6,x0: 1.003e-04,f0: 1.476e+06,change: 2.674e-07  
It.   7,x0: 1.000e-04,f0: 1.476e+06,change: 2.674e-07  
-----
```

Shape optimization results

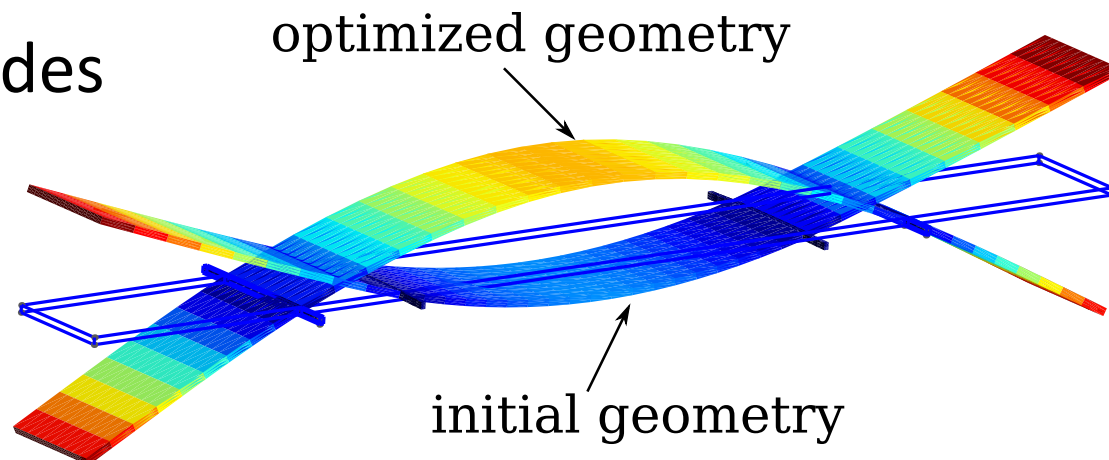
- Initial geometry



- Optimized geometry



- Eigenmodes



Shape optimization results

