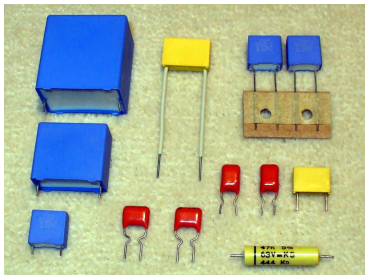
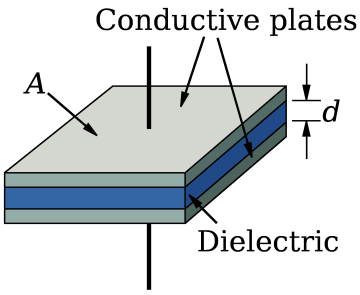
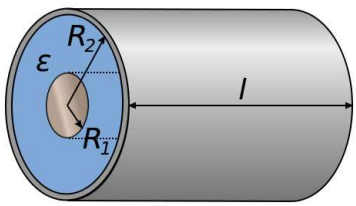
Example 2: electric potential and electric field in a condenser

A condenser is an electric component used to store energy elettrostatically in an electric field.



Condensers may have different shapes, but always contain at least two conductors separated by a dielectric.

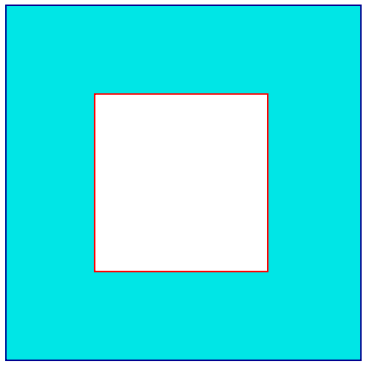




Once set a potential difference across the conductors, positive charge collects on one of them and negative on the other one. Moreover an electric field develops through the dieletric.

The model problem

We consider a cave square section condenser and determine the electric potential inside its dieletric, supposing that on the external boundary/conductor Γ_0 and on the internal boundary/conductor Γ_1 different electric potential are fixed.



Γ_0 is in blue, Γ_1 is in red and the dieletric is in light blue.

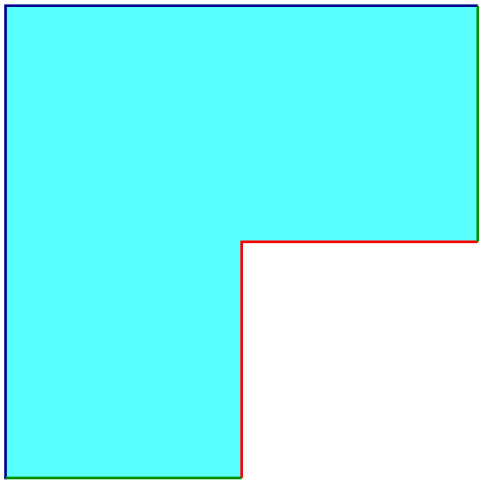
$$\begin{cases} -\Delta u = 0 & \text{in } \Omega \\ u = g & \text{on } \Gamma_0 \cup \Gamma_1 \end{cases}$$

where $\Omega = (-2,2)^2 \setminus (-1,1)^2 \subset \mathbb{R}^2$

u is the electric potential

$$g(x) = \begin{cases} 0 & \text{on } \Gamma_0 \\ 1 & \text{on } \Gamma_1 \end{cases}$$

For simmetry reasons we restrict our study to the upper left quadrant.



The light blue domain Ω is the dieletric. The blue line is Γ_0, the red one is Γ_1. In green the artificial boundary $\Gamma 2$.

As a side effect we have to set conditions on the new generated artificial boundary Γ_2.
A natural choice would be to set homogeneous Neumann boundary conditions.

$$\frac{\partial u}{\partial n} = 0 \qquad \text{on } \Gamma_2$$

where n is the outer normal to Γ_2

Why homogeneous Neumann boundary conditions ?

The simmetric property of the problem and the uniqueness of its solution yield

$$u(x_1, x_2) = u(-x_1, x_2)$$

Consequently the derivative should coincide

$$\frac{\partial}{\partial_{x_1}} u(x_1, x_2) = -\frac{\partial}{\partial_{x_1}} u(-x_1, x_2)$$

Therefore on the axis $x_1 = 0$ we can write

$$\frac{\partial}{\partial_{x_1}} u(0, x_2) = -\frac{\partial}{\partial_{x_1}} u(0, x_2)$$

and thus

$$\frac{\partial}{\partial_{x_1}} u(0, x_2) = \frac{\partial}{\partial n} u = 0$$

Arguing in a similar way for x_2 and on
$\{x \in \Omega \mid x_2 = 0\} \subset \Gamma_2$
we conclude that homogeneous Neumann boundary condition have to be set on Γ_2.

With the help of Alberta, trought numerical simulation we try to answer these questions

- We are interested in the shape of the graph of u. How do the level lines (equal potentials) look like? Are they smooth?

- What happens if the potential on Γ_1 doubles?
  (Linear dependence of the solution from the boundary values)

- More interesting in practice is the electric field which correspond to ∇u. Where is the electric filed bigger?
  (Reentrant corner in $\partial\Omega$ causes a singularity)

ALBERTA: finite element toolbox

ALBERTA is a toolbox for fast and flexible implementation of finite element software for real life applications. ALBERTA is a library with data structures and functions for adaptive finite element simulations in one, two and three space dimension, written in the programming language ANSI-C. It has been developed in Germany by A. Schmidt, K. Siebert and others.

The web site:

www.alberta-fem.de

Documentation/manual:

- A.Schmidt, K Siebert "Design of Adaptive Finite Element Software: The finite element Toolbox ALBERTA" Lecture Notes in Computational Science and Engineering, 2005 , Springer LNCSE series

where you can find it on our server:

/opt/alberta-1.2

Here alberta-1.2-demo contains e few example codes.
There are three versions depending on the dimension of the space: 1D, 2D, 3D.
The directory COMMON contains the part of the code dimension-independent.

In 1D, 2D, 3D we can find:

- the Makefile

- the executable files

- in the subdirectory INIT: files of parameters (for example:`ellipt.dat`)

- in the subdirectory Macro: data files for the initial meshes

NOTE: It is not necessary to keep the same directories subdivision for new codes!

Parameters

Problem data may depend on parameters, moreover many procedures need parameters. Since it is often very helpful to change these values without recompiling the program, they are usually initialized from parameters files. In order to avoid a fixed list of parameters ALBERTA uses the following concept. Every parameter consits of two strings: a key string by which the parameter is identified and a second one containing the parameter values.

The definition of a parameter has the following syntax:

```
key:  parameter values % optional comment
```

Each key definition must be placed in one line and is terminated by a ':'
Each parameter definition must have at least one parameter value otherwise it is ignored. Parameter values can also be specified in so called continuation lines. A line is a continuation line if the last two character in the preceeding line are '\' and the newline character.

See for example file elliptic.dat

```
macro file name:       Macro/macro.amc
global refinements:    1
polynomial degree:     3

% graphic windows:
%  solution, estimate, and mesh if size > 0
graphic windows:       500 500 0
......
```

Parameters are stored as a sequence of words in one string.

To initialize parameters from file use Alberta's functions:

```
void init_parameters(int print,
                     const char * filename)
void add_parameters(int print, const char * key,
                    const char * value)
ADD_PARAMETERS(int, const char*, const char *)
```

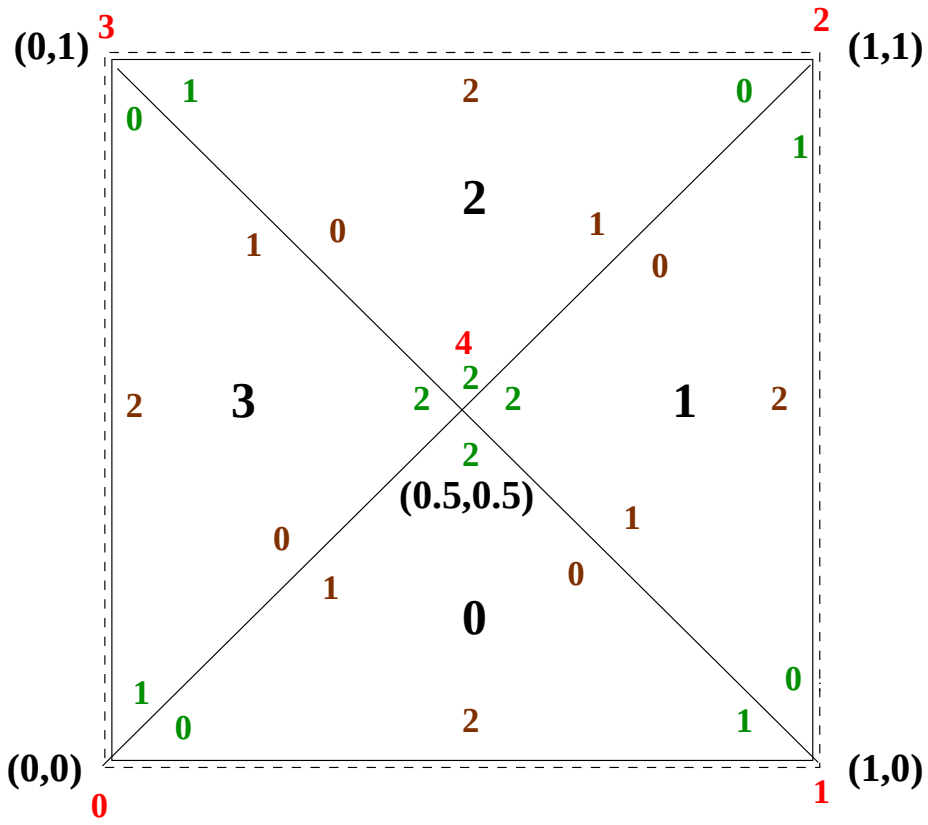
to access the value of a parameter use

```
int get_parameters(int info,
                   const char * key,
                   const char * format, ...)
int GET_PARAMETERS(int, const char *,
                   const char *,...)
```

to display all parameters

```
void print_parameters(void)
```

Example of macro mesh



	0	interior
boundaries :	1	Dirichlet
	−1	Neumann
element neighbours:	−1	no neighbours

File macro.amc

```
DIM: 2
DIM_OF_WORLD: 2
number of elements: 4
number of vertices: 5
element vertices:
0 1 4
1 2 4
2 3 4
3 0 4
element boundaries:
0 0 1
0 0 1
0 0 1
0 0 1
vertex coordinates:
0.0 0.0
1.0 0.0
1.0 1.0
0.0 1.0
0.5 0.5
element neighbours:
1 3 -1
2 0 -1
3 1 -1
0 2 -1
```