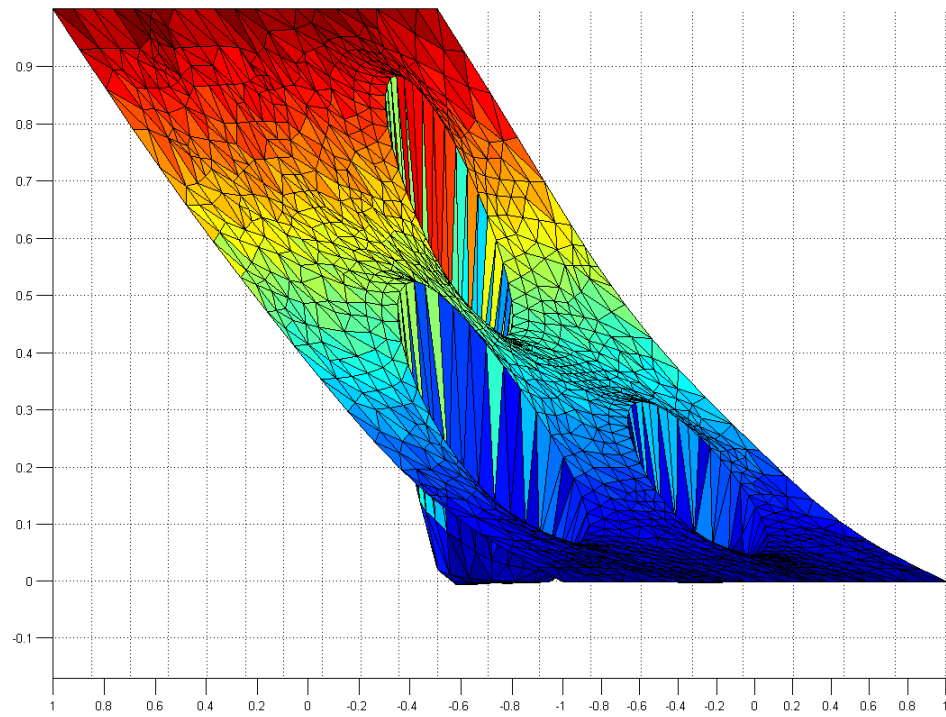


# FEM CODE FOR THERMAL INSTATIONARY ANALISYS OF AN ETEROGENEOUS MATERIAL

Stefano Boccelli

July 2014



# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Mesh Generator</b>	<b>6</b>
2.1	Introduction . . . . .	6
2.2	Automated Procedure . . . . .	6
2.3	Removing undesired triangles . . . . .	6
2.4	Saving the mesh . . . . .	7
2.5	Mesh for the analized problem . . . . .	7
<b>3</b>	<b>Preprocessor</b>	<b>9</b>
<b>4</b>	<b>Solver</b>	<b>11</b>
<b>5</b>	<b>Numerical Analisis</b>	<b>13</b>

# 1 Introduction

With this document I wish to give you a brief overview of a finite element code I've implemented under Octave / MATLAB, able to solve parabolic problems.

I've written this code for learning purposes, while studying for a course of Numerical Methods for Differential Problems in Politecnico di Milano. It's quite simple, still not implementing non-homogeneous Neumann conditions, but I upload this anyway since it's cute and it might be useful to someone studying the basics of FEM methods! Oh, also the mesh generator is pretty cool, although rough, and playing around with the parameters you should be able to obtain an acceptable mesh.

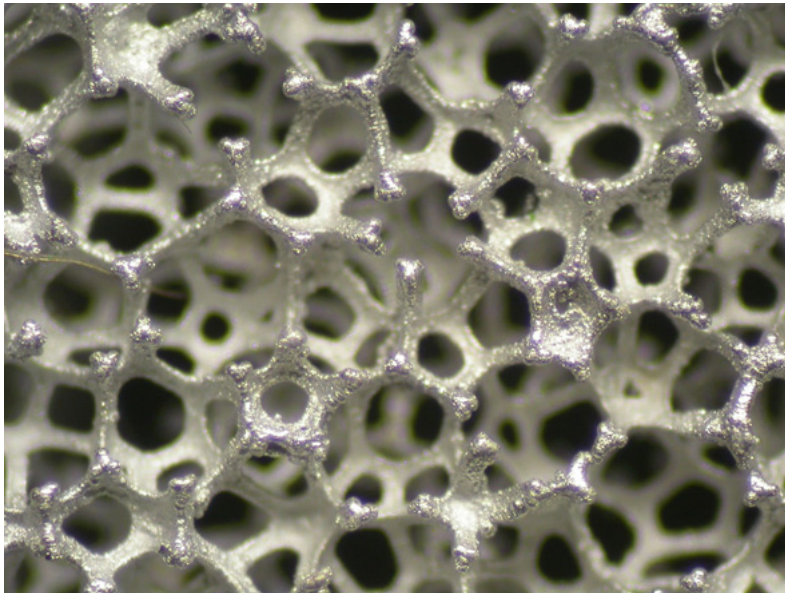
The code is born for a specific purpose: simulating the temperature transient into a Emmental piece of cheese. Then I made a mistake, imposing a lower thermal diffusivity for the air in the bubbles than the cheese itself, so the result was something different.. like a piece of wood into emmental for example! You can fix that by modifying a file..

Ok, the point is that if want to study the temperature transient inside a non homogeneous material, like a porous metal, a sponge, carbon particles into a metal microstructure or maybe some cheese, you clearly have to set a different value for the thermal diffusivity  $\mu$  in every region. Consequences are:

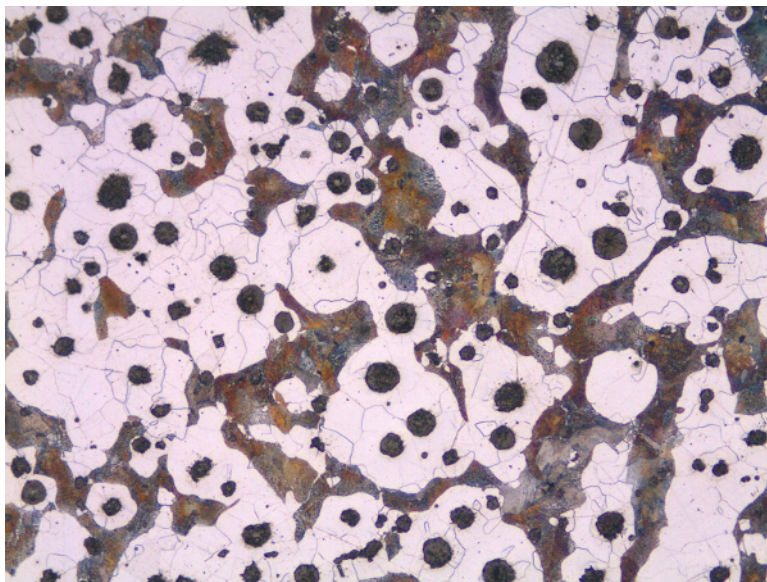
1. Need to create a as-uniform-as-possible mesh in the region of the interface
2. Ability of the preprocessor to assign different values of  $\mu$  to certain triangles (pretty easy after all)
3. We obtain solutions much more interesting than the usual  $\mu = \text{const}$  case!!

In the following paragraphs I'm showing the main features of the components: *mesh generator*, *preprocessor* and in the end the *solver*. Then I'll show the results of a simulation.

Here are some examples of heterogeneous materials:



*Porous Aluminum*



*Carbon into a ferrous alloy*



*Everyday materials*

## 2 Mesh Generator

### 2.1 Introduction

The implemented mesh generator creates a Delaunay triangulation exploiting a “built-in” Octave / MATLAB algorithm.

The first step is the definition of a point cloud, defining the desired border.

The numerical domain is defined through the file *mesh\_prelude\_rect2.m*.

The heart of the mesh generator is the file *MAIN\_grid.m*, whom calling activates an iterative procedure showing to the user the mesh regularity details and allows to:

- automatically modify the mesh, searching for too much distorted triangles and dividing them into smaller ones
- adding points by hand
- saving the mesh
- removing triangles

### 2.2 Automated Procedure

The mesh generator implements an automated procedure (as I briefly pointed out in the previous paragraph) to make the mesh more regular. The procedure focuses on the following steps:

1. identification of the triangles having a sphericity or area bigger than a certain limit specified in the file *triangulation\_parameters.m*
2. generation of some more points on the longer triangle sides
3. “equivalence” procedure: elimination of points more clustered than a certain radius, called *cuttingR* and defined into the *triangulation\_parameters.m* file

### 2.3 Removing undesired triangles

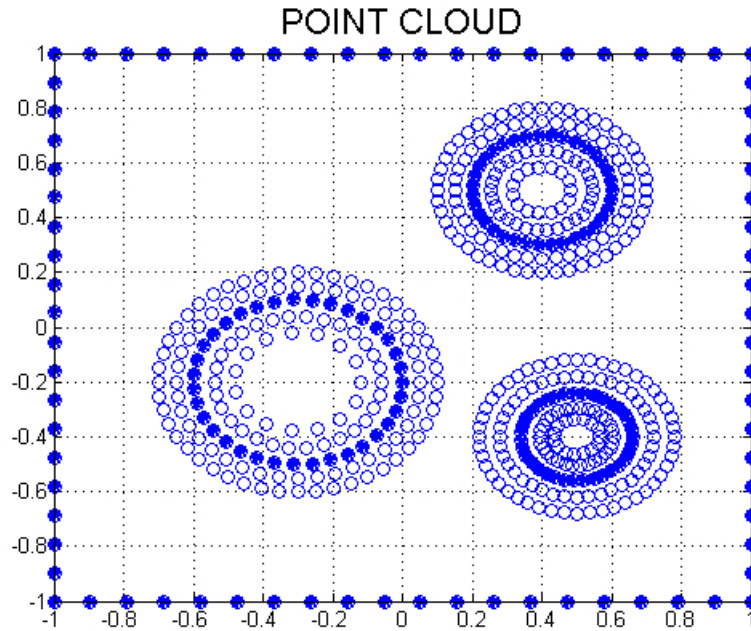
In the case there were internal borders into the domain, the triangulation procedure will triangulate anyway the region they define. It’s necessary to eliminate those triangles before saving the mesh.

## 2.4 Saving the mesh

The saving procedure creates some files called *msh\_\*.dat* containing the mesh details. Those files are read by the preprocessor for the imposition of boundary conditions.

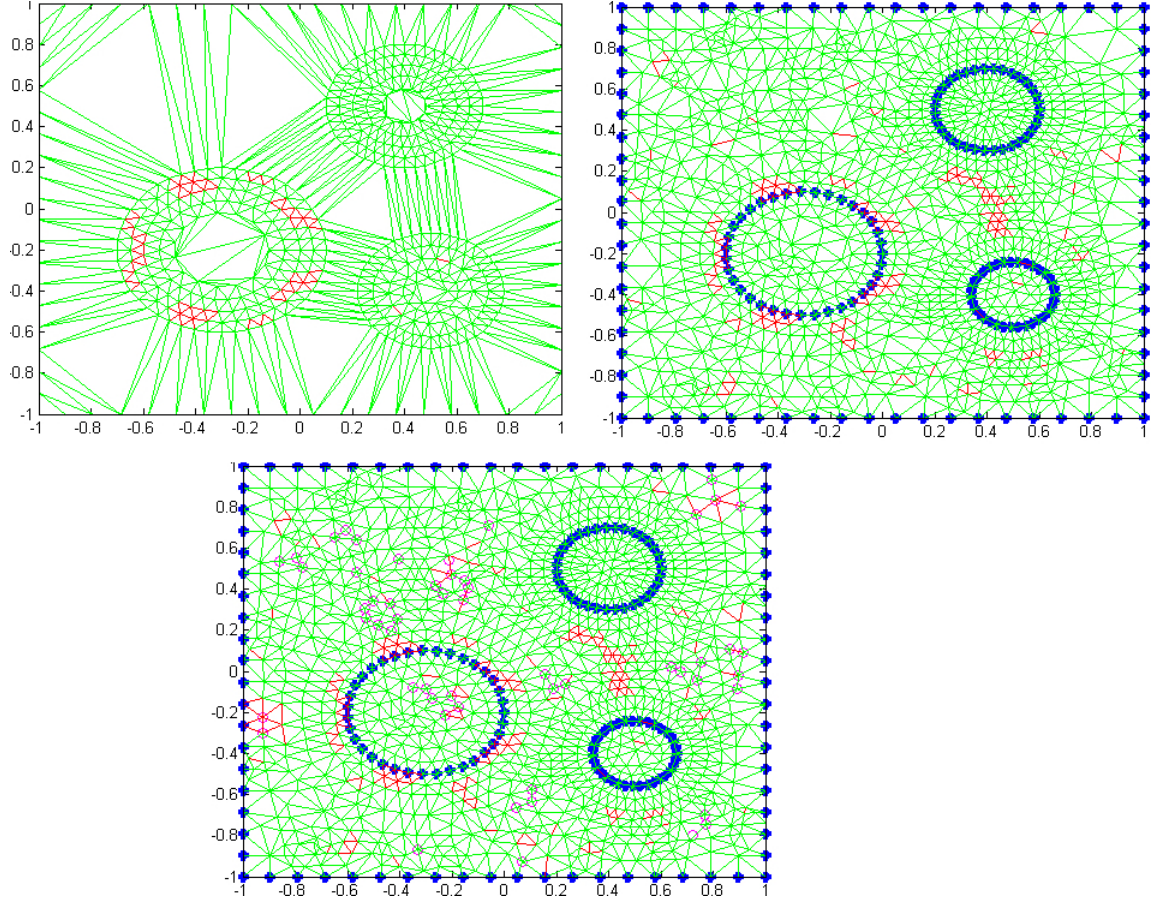
## 2.5 Mesh for the analized problem

Here is shown the initial point cloud for the thermal transient problem I examined. Points belonging to domain borders (and internal discontinuous  $\mu$  borders) are marked deeper. The rest are auxiliary points, added to obtain an omogeneous and structured mesh in the critical interface regions.



Starting nodes





First trial mesh and final one

In the image you can see some red and some green triangles: the mesh generator draws in red the triangles whose sphericity is smaller than a certain threshold. I mistakenly chose a maximum sphericity of 3, so almost no triangles were drawn in red. The points circled in purple have been added with the mesher option “add points by hand”, to achieve a little more regularity in the final mesh.



### 3 Preprocessor

The implemented preprocessor asks to choose nodes where to apply boundary conditions and triangles on which you want to impose a value of the  $\mu$  constant different than the default.

First of all the preprocessor imports the files that have been generated by the mesh generator: it's necessary for those files to be located in the working directory.

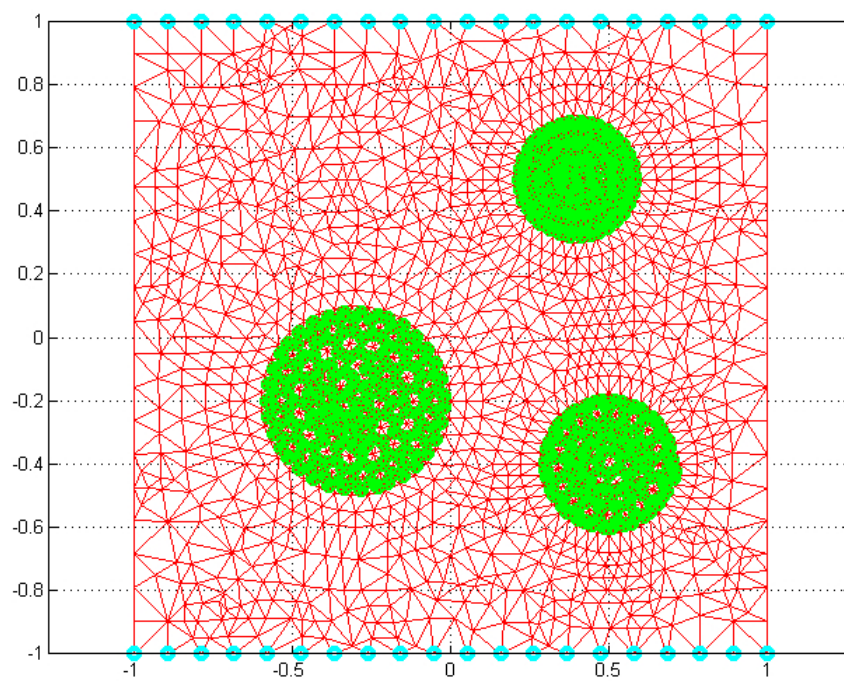
To impose the BCs or the value of  $\mu$  on the triangle you do as follows: let's talk about Neumann conditions for example.. In the file *bcs\_value.m* you have to define as many functions  $N_i$  as many different Neumann values you want to impose. The  $N_i$ s are then assembled into the cell array `Neu_funct{}`, *being careful to the order!!*. During the preprocessor running, a *Label* is asked: the preprocessor is referring to the position (of the function you want to impose) into the cell array. Here's an example:

```
N_one = @(x,y) 0; % omogeneous Neumann
N_two = @(x,y) 2*x;
Neu_funct = {N_one, N_two};
```

So, label = 1 is `Neu_funct(1)`, which is `N_one`!

The result of the preprocessing is memorized. It's necessary to open the Solver without clearing the memorized variables!!

In the following picture, the preprocessed mesh: eterogeneous material regions are selected and the opportune  $\mu$  have been defined. The preprocessor shows in light blue the points over which Dirichlet conditions have been imposed, in purple Neumann points and in green the triangles with modified thermal diffusivity.



## 4 Solver

The implemented solver works with the variables saved by the preprocessor into the structure `msh`.\*

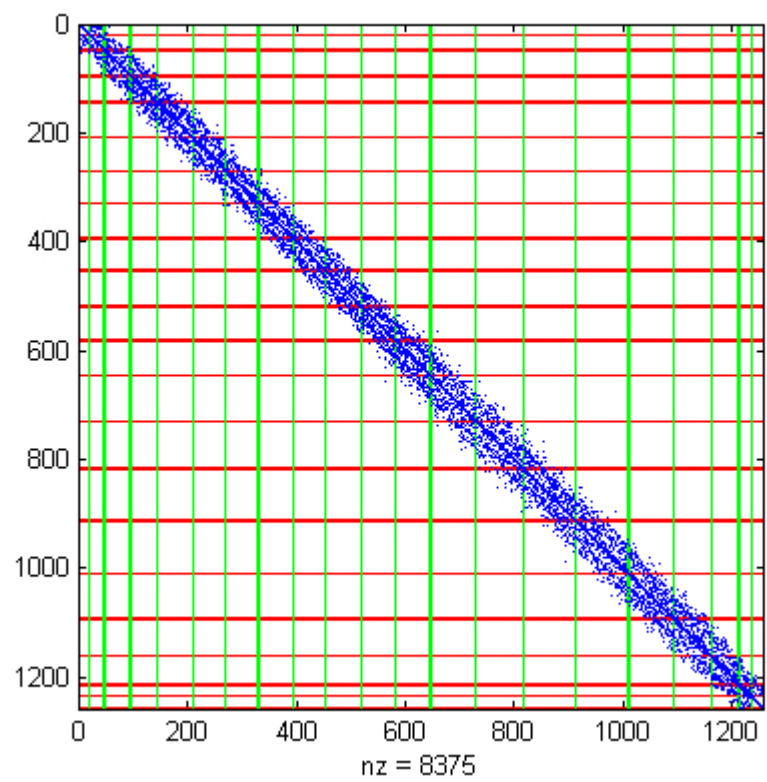
You can modify the file `domain_parameters_parabolic.m` to choose the proper  $\Theta$  - method constant and the timestep and time interval.

First of all the solver assembles mass and stiffness matrix, integrating P1 shape functions over the grid triangles. BCs are imposed by *direct imposition over rows*, and then the matrix is “simmetrized”. The quadrature formula used to integrate shape functions is  $2^{nd}$  order and it evaluates the functions in the mid points of the triangle over which we are integrating.

Since we are solving a parabolic problem, the known term is adjourned every step: it's then necessary to correct it at every step to impose BCs, so completing the direct imposition over rows started out of the cycle by processing A.

**CUIDADO: the solver is actually only able to impose Neumann = 0 conditions!**

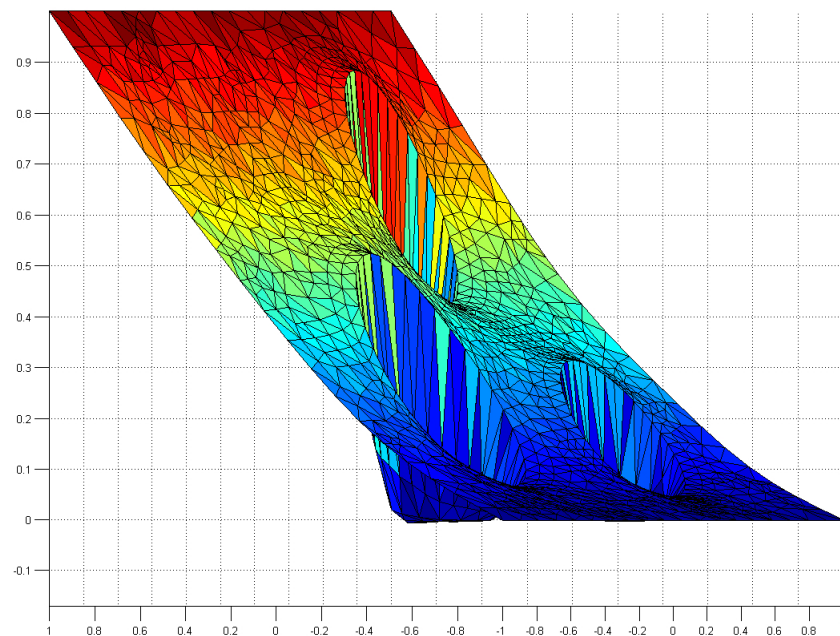
In the following picture the sparsity pattern of the matrix A. Rows and columns modified are drawn with red and green lines.

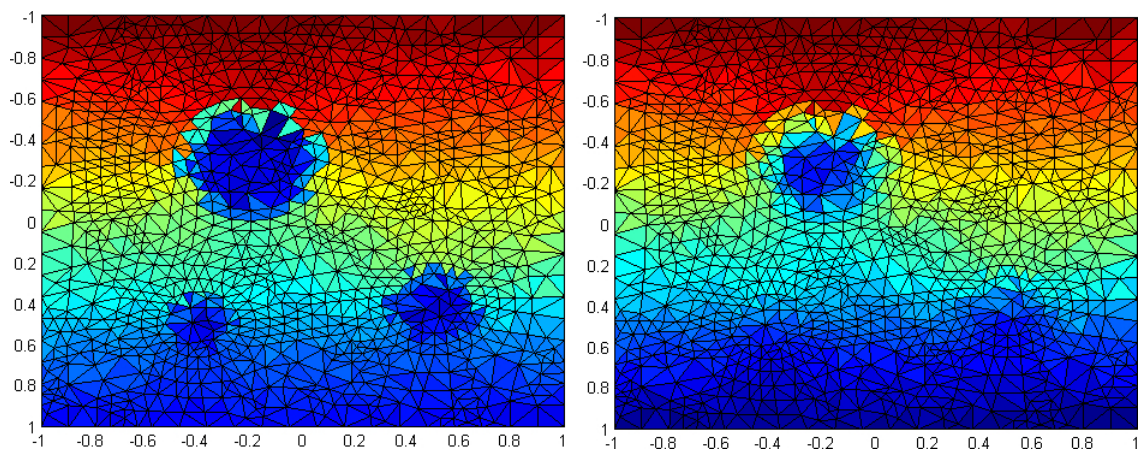


## 5 Numerical Analysis

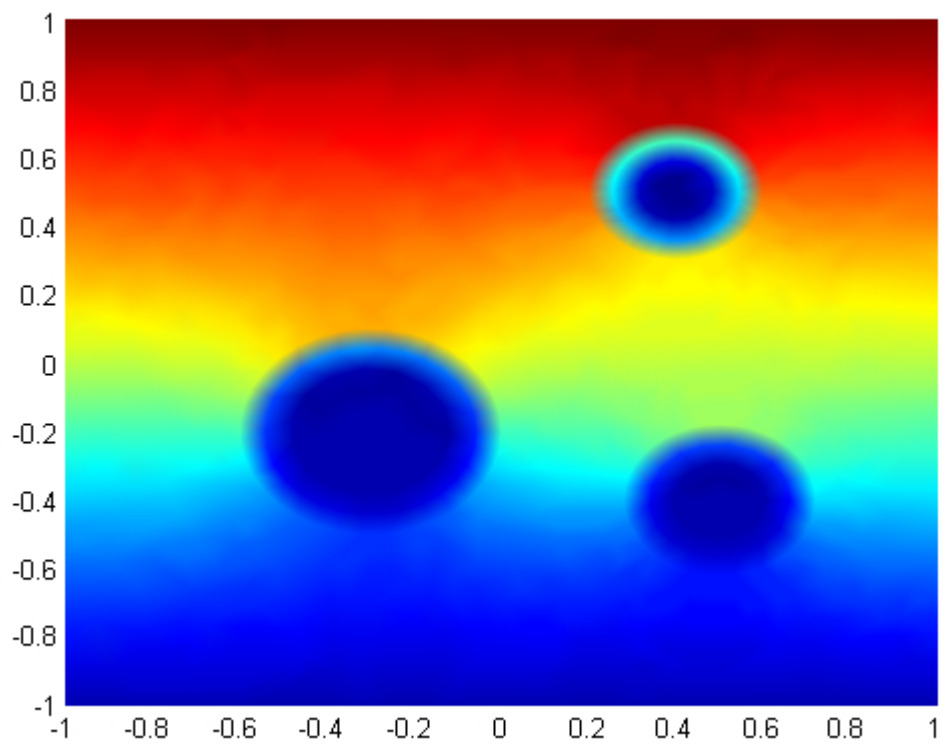
For the following numerical analysis, Dirichlet boundary conditions have been set on the lower border and homogeneous on the upper. Initial temperature was homogeneous too. On the two sides, I chose  $\text{Neumann} = 0$  in order to simulate the symmetry: it's not a tiny slice of material, it's a long and thin slice and we only want to simulate a part of it.  $\text{Neumann} = 0$ : no heat flux.

In the following picture, a few screenshots of the solution at different timesteps.





Solutions at various timesteps



Solution represented by interpolating the values on the nodes and then smoothing

## **6 Final Remarks**

Grazie di aver volato con noi!