Figura 1: The bar we are considering in this example. It exchanges heat through one end, kept at constant temperature, and with the surrounding medium. The other end is insulated.

1 Heat exchange in a one-dimensional bar

We consider a bar of length L and constant thermal conductivity k (see figure). One end of the bar is kept at constant temperature T_0, while the other end is under adiabatic conditions (zero thermal flux). The bar exchanges heat with the surrounding air at temperature T_a. Using a onedimensional model, the steady state solution satiefies

$$-k\frac{d^2}{dx^2}T + h_p(T - T_a) = 0 \quad 0 < x < L, \tag{1}$$

con condizioni al bordo date da

$$T(0) = T_0 \qquad \frac{d}{dx}T(L) = 0. \tag{2}$$

The coefficient of convective heat exchange per unit length h_p [W/m^{2}K] is assumed constant. It is linked to the coefficient per unit area h [W/mK] by the relation

$$h_p = \frac{hp}{S},$$

p being the perimeter and S the section of the bar. Since the bar has a rectangular section, with sides of length a_1 and a_2 we can write

$$h_p = \frac{2h(a_1 + a_2)}{a_1 a_2}.$$

Equations (**??**) and (**??**) may be rewritten in terms of the temperature difference $\theta = T - T_a$ and normalized by setting

$$x \to x/L.$$

Thus, in the following x indicates the normalized (a-dimensional) abscissa and the domain becomes the interval $(0, 1)$. In the normalized variables the problem is

$$-\frac{d^2}{dx^2}\theta + a\theta = 0 \quad 0 < x < 1, \tag{3}$$

with boundary conditions

$$\theta(0) = \theta_0 = T_0 - T_a \qquad \frac{d}{dx}\theta(1) = 0, \tag{4}$$

where

$$a = \frac{L^2 h_p}{k} = \frac{2L^2 h(a_1 + a_2)}{k a_1 a_2}.$$

We consider a uniform grid of M elements in the interval $[0, 1]$ and we discretize (**??**) with linear finite elements. We indicate with $u_i = u_h(x_i)$, $i = 0, \ldots, M$ the approximation of θ at the nodes $x_i = hi$, being $h = 1/M$.

The problem unknowns are given by u_i, $i = 1, \ldots, M$, since, thanks to the boundary condition, $u_0 = \theta_0$. We operate in the usual way to obtain a linear system

$$A\mathbf{u} = \mathbf{b}, \tag{5}$$

with

$$\mathbf{u} = [u_1, \ldots, u_n]^T, \quad \mathbf{b} = [\theta_0, 0, \ldots, 0]^T$$

and $A \in \mathbb{R}^{M\times M}$ the matrix given by

$$A = \begin{bmatrix} 2+h^2a & -1 & 0 & \dots & \dots & 0 \\ -1 & 2+h^2a & -1 & \dots & \dots & 0 \\ & & \dots & \dots & & \\ 0 & \dots & \dots & -1 & 2+h^2a & -1 \\ 0 & \dots & \dots & \dots & -1 & 1 \end{bmatrix}.$$

Matrix A is symmetric positive definite, thus we may use the Gauss-Siedel iterative scheme for the solution of the linear system.

One may verify that the single iteration of Gauss-Siedel can be written as

$$u_i^{(k+1)} = \frac{u_{i-1}^{(k)} + u_{i+1}^{(k)}}{2+h^2a}, \quad i = 1, \dots, M-1$$

and

$$u_M^{(k+1)} = u_{M-1}^{(k)},$$

being k the iteration index. We terminate the iterations when $||\mathbf{u}^{(k+1)} - \mathbf{u}^{(k)}|| \leq \tau$, for a given tolerance $\tau > 0$, or when $k \geq k_{max}$ (no convergence within a maximal number of iterations k_{max}).

1.1 The exact solution

The exact solution of problem (**??**)-(**??**) is

$$\theta(x) = \theta_0 \frac{\cosh[\sqrt{a}(1-x)]}{\cosh(\sqrt{a})}.$$

1.2 The program heat_exchange.cpp

In the directory `Heat_Exchange` you have a prototype program simply called `main.cpp` that solves the problem with the proposed numerical scheme.

Is a simple program and it does use little use of advanced C++ programming. It is not general, and difficult to extend to other finite elements or other numerical schemes for the solution of the linear system.

It is just a first example on which the students may elaborate further. You have a `Makefile` that allow to compile the code by simply typing `make main` or just `make` in the directory where the program is kept.

You may generate the executable directly, for instance with

```
g++ -std=c++11 -o main *.cpp
```

The file `parameters.hpp` defines a struct with the default values of the parameters, namely

Variabile	Nome nel pr.	Valore	Variabile	Nome nel pr.	Valore
L	`L`	40	a_1	`a1`	4
a_2	`a2`	50	T_0	`To`	46
T_a	`Te`	20	k	`k`	0.164
h	`hc`	200×10^{-6}			

. Those values may be changed by using a `GetPot` file, the default name being `Parameters.pot`.

The program accepts arguments: it synopsis is

```
main [-h] [-v] -p parameterFile (default: parameters.pot)
-h this help
-v verbose output
```

and produces a file,`result.dat` containing the approximate solution in the format

$$x_i \quad u_i \quad \theta(x_i),$$

a line for each node, including the node at $x = 0$.

1.2.1 Visualization

To visualize the results you may use `xmgrace` or `gnuplot` (or even MATLAB or Octave).

The *gnuplot* commands to visualize the results are

```
gnuplot
gnuplot> plot "result.dat" u 1:2 w lp title "uh", "result.dat" u 1:3 w l title "uex"
```

2 Possible extensions

Here some possible extensions in order of difficulty

- Allow the user to change the name of the file with the result, for instance indicating the name in the getpot file, or in the command line.
- Change the stopping criterion to use the L^2 or the H^1 norm, instead of the discrete one.
- Build the matrix explicitly and use different linear solvers, for instance the Eigen library, or other available libraries. Allow the user to specify the solver.
- Generalize the code for transient problems, using suitable time integration schemes;
- Generalize the code to allow variable (in space) parameters and non uniform grid. You need to use numerical quadrature;
- Generalize the code to allow higher order finite elements.
- Generalize the code to allow parameter that depends on the solution itself (non-linear problem).

2.1 Uso di vector<double>

We have used the template class `vector<double>` defined in the **Standard Library** (std), instead of native C style vectors. This simplifies a lot the handling, particularly the memory handling. A version using native arrays would replace

```
vector<double> theta(M+1);
```

with

```
double * theta = new double[M+1];
```

The command `new double[M+1]` builds a pointer to an array of doubles that can be addressed by `theta[i]`.

Remember that in this case the handling of the memory is responsibility of the programmer, and you have to delete the array when not needed anymore, using

```
delete[] theta;
```

The `[]` is here required since `theta` is an array. Writing just `delete theta` is an **error** since the program will free only the first element of the array and you will have a part of memory that will not be released (a so-called **memory leak**).

The use of `vector<float>` eliminates this problem, since memory is handled by the class destructor.