MAP 4371 Spring 2019 - Final Project

(Tentative)

1 Organization

Download the files

```
grid.py
problem.py
tests.py
```

from WebCourses and place them in the project directory. The first two files are described in the text below and the last file contains tests for grading. Running the latter file will load your code, run some examples and compare the results to some correct values. If you execute the file after downloading it will produce many error messages because your code is not yet implemented. After you have implemented everything correctly, it should produce an output like

```
Fingerprint: cb42b3a1e1d12190d49c4d4f5b509089

test_assemble_heatsink_matrix (__main__.ProblemTests) ... ok
test_assemble_heatsink_rhs (__main__.ProblemTests) ... ok
test_assemble_square_matrix (__main__.ProblemTests) ... ok
test_assemble_square_rhs (__main__.ProblemTests) ... ok
test_heat_flux_south (__main__.ProblemTests) ... ok
test_sparse (__main__.ProblemTests) ... ok
test_trapezoid (__main__.ProblemTests) ... ok

----------------------------------------------------------------------
Ran 7 tests in 0.011s

OK
```

To turn in the project, we meet in a classroom (TBA), on date (TBA), where I will ask you to run the test, see your plots and selectively ask you about some parts of your code. You get credit for each running test, but no partial credit for a failed test. The oral questions are not for credit but merely to ensure that you wrote the code yourself.

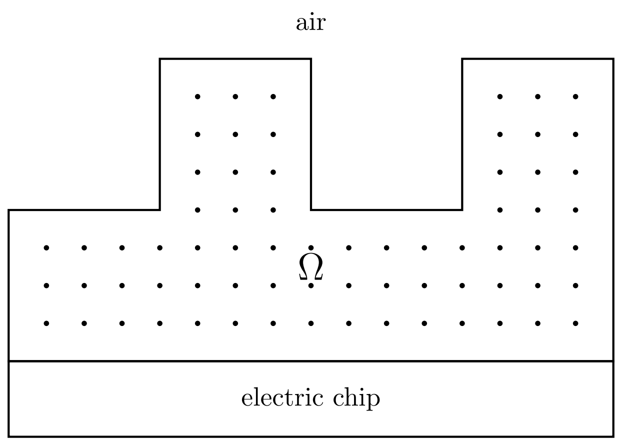

Figure 1:

2 Overview

In this project, we consider a heatsink, which is for example a “riffled” piece of metal one often finds on top of electric chips for cooling. The riffled structure increases the surface area and therefore the amount of heat it can dissipate into the environment. In our case, we consider only a cross section in two dimensions, so that we obtain a domain Ω as shown in Figure 1 for two “fins”. The temperature u inside of the domain Ω is given by the heat equation

$$u_t - \Delta u = 0. \tag{1}$$

To model the heat flow from the chip to the surrounding air, we need appropriate boundary conditions. For the lower boundary Γ_D, connecting to the chip, we assume Dirichlet boundary conditions $u|_{\Gamma_d} = u_D$, i.e. we enforce that the temperature at the bottom is u_D, the temperature of the chip. On the remaining part of the boundary $\Gamma_N = \partial\Omega \setminus \Gamma_D$, the heat dissipates into the surrounding air. Although this is physically not correct, for simplicity we model this by some Neumann boundary conditions, i.e. we prescribe the heat flux through the boundary, so that $\frac{\partial u}{\partial n} = u_N$, where n is the outward normal unit vector and u_N a fixed given number. For the initial temperature in the heatsink, we set $u(x,0) = u_0$ so that in summary we obtain

$$u(x,0) = 1, \qquad u|_{\Gamma_D} = 1, \qquad \left.\frac{\partial u}{\partial n}\right|_{\Gamma_N} = -0.5$$

where we have already fixed some concrete values.

3 Solution of the time dependent problem

We solve this problem in two steps: First, we use the standard four point finite difference stencil from the Poisson equation to discretize the Laplacian $-\Delta u$ together with the boundary values. This yields an equation of the form

$$U_t + AU = F$$

for an appropriate matrix A and vector F. The solution $U(t) = [U_1(t), \dots, U_n(t)]$ still depends on time so that $U_i(t) \approx u(x_i, y_i, t)$ for the ith grid point (x_i, y_i). Note that this is a large system of ODEs for which we can apply any ODE solver. However, recalling the eigenvalues of A form the lecture, this system is stiff, so that we use an implicit solver, in our case the trapezoid method.

4 Heat flux

After a while, the heatsink will settle into a stationary temperature distribution that does no longer change with time. It satisfies $u_t = 0$ and thus is given by

$$-\Delta u = 0, \qquad u|_{\Gamma_D} = 1, \qquad \left.\frac{\partial u}{\partial n}\right|_{\Gamma_N} = -0.5. \tag{2}$$

In order to gain some information on how well the heat sink is working, we compute the heat flux through the interface of the heatsink and the chip, i.e. through Γ_D. It is given by

$$H := \int_{\Gamma_D} \frac{\partial u}{\partial n}. \tag{3}$$

Since we don't know the correct solution u, we approximate this quantity based on the result U of our numerical computations. This function is only defined on the grid and therefore we approximate the directional derivative $\frac{\partial u}{\partial n}$ by a one sided second order finite difference $D_n U$. Then, we can approximate the integral by

$$H \approx h \sum_{i:\, x_i \in \Gamma_D} D_n U_i$$

where h is the resolution of the grid and the sum only contains indices i for which the corresponding grid point x_i are in the Dirichlet boundary Γ_D.

5 Implementation

For the spacial discretization we need a grid. Because the domain Ω is more complicated than the examples we have seen so far, it is provided by the function `make_grid_heatsink` in the file `grid.py`. You can find some documentation on how this function works in the file itself. For comparison, the file also contains the function `make_grid_square`, which works in the same way, but produces a simple rectangular grid.

1. Implement all functions and classes in the file `problem.py` according to the descriptions in the file. This file contains the code that is graded and tested by `test.py`.

2. Solve the heat equation (1) with the functions provided in `problem.py` with 100 time steps in the interval $t \in [0, 1]$. For the grid choose `level=4` and `n_fins=4`. Make solution plots for four intermediate time steps.

   For plotting, use the Matplotlib function `plot_trisurf`. Unfortunately, in our case this does not produce good plots unless you provide the extra option `triangles` (see documentation). You can generate these triangles directly with the command `triangulate(grid)` provided in `grid.py`, where `grid` is the grid object for our domain.

3. For `level=4` and `n_fins` $\in \{2, 4, 8\}$ compute the heat flux (3) for the stationary problem (2) and plot the solutions.