Project in TMA4220

Candidatenumber: 10000 & 10028

17. November 2014

0.1 Poisson solver

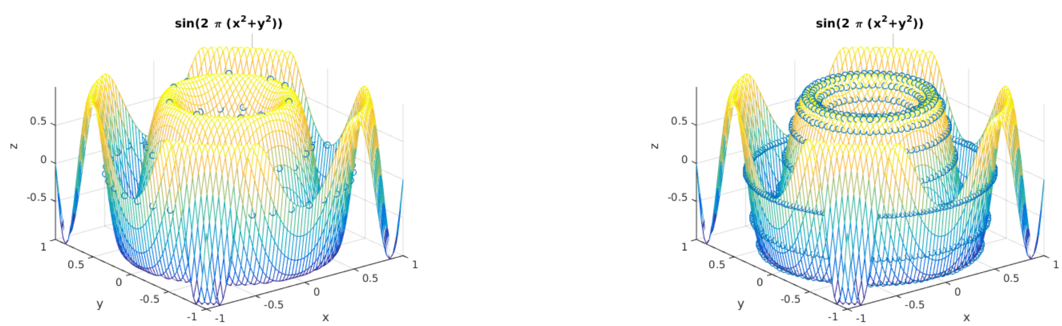
We here present a solver for the Poisson equation,

$$\nabla^2 u = f,$$

in both 2 and 3 dimensions, with both Neumann and Dirichlet boundary conditions, while using the element method. We let $u(r) = \sin(2\pi r^2)$ and $f(r) = \nabla^2 u(r) = -8\pi\cos(2\pi r^2) + 16r^2\pi^2\sin(2\pi r^2)$.

0.1.1 2D solver

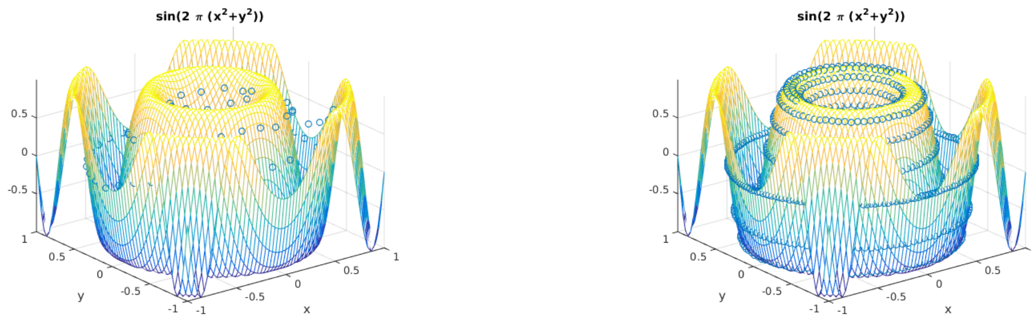
The program `poisson2d.m` solves the Poisson equation in 2 dimensions, with Dirichlet boundary conditions, and created the plots in figure 1.



(a) $N = 125$ (b) $N = 1250$

Figure 1: Scatterplotts of the numerical solution of the Poisson equation in 2 dimensions, with Dirichlet boundary condition and different number of points, N, together with a surfplot of the correct solution.

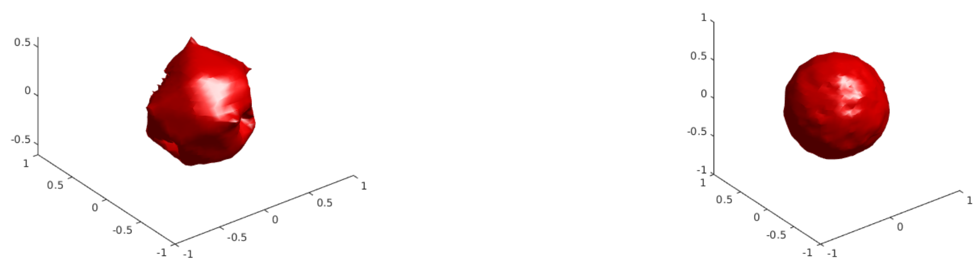
The program `poissonBnd2d.m` solves the Poisson equation in 2 dimensions, with Dirichlet boundary conditions on half the domain, and Neumann on the other half, and created the plots in figure 2.



(a) $N = 125$

(b) $N = 1250$

Figure 2: Scatterplotts of the numerical solution of the Poisson equation in 2 dimensions, with Dirichlet boundary condition on half the boundary, and Neumann on the other half, with and different number of points, N, together with a surfplot of the correct solution.



(a) $N = 125$.

(b) $N = 1250$.

Figure 3: A plot of the numerical solution of the Poisson equation in 3 dimensions, with Dirichlet boundary condition, with different number of points, N. The correct solution is a perfect sphere.

0.1.2 3D solver

The program `poisson3d.m` solves the Poisson equation in 3 dimensions, with Dirichlet boundary conditions, and created the plots in figure 3.

The program `poissonBnd3d.m` solves the Poisson equation in 3 dimensions, with Dirichlet boundary condition on half the boundary, and Neumann on the other half, and created the plots in figure 4

0.1.3 Discussion

As we can see from figure 1 and figure 2, the 2 dimensional case works quite good, and seems to be convergent. Thus `poisson2d.m` and `poissonBnd2d.m` are working solvers for the Poisson equation.
Also the 3 dimensional case with Dirichelt boundary the solver `poisson3d.m` converges. But when including the Neumann boundary condition the A matrix became singular because we needed to include

(a) $N = 125$. (b) $N = 1250$.

Figure 4: A plot of the numerical solution of the Poisson equation in 3 dimensions, with Dirichlet boundary condition on half the boundary, and Neumann on the other half, with different number of points, N. The correct solution is a perfect sphere.

some points on the boundary which we did not need while only having Dirichelt boundary. There also seems to be a problem with while adding the Neumann conditions to b. Thus we conclude that the program `poisson3dBnd.m` does not work correctly. We have no clue why the Neumann conditions works in 2 dimension and not in 3 dimensions, considering that the implementation is almost identical.

0.2 Meat equation

We are here going to present a solution to the heat equation in 3 dimensions using Neumann conditions on the edges, with forward Euler in time, and element-method in space.

0.2.1 Weak formulation

The heat equation

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u \tag{1}$$

Using *Greens formula* we obtain

$$\int_\Omega \frac{\partial u(\mathbf{x})}{\partial t} v(\mathbf{x})\, d\Omega = \int_\Omega \Delta u(\mathbf{x}) v(\mathbf{x})\, d\Omega = -\int_\Omega \nabla u(\mathbf{x}) \nabla v(\mathbf{x})\, d\Omega + \int_\Gamma \frac{\partial u}{\partial n}(\mathbf{x}) v(\mathbf{x})\, d\gamma.$$

This has an unique solution if continuous and weakly coercive. $f \equiv 0$ for our equation.
The scheme is weakly coersive if $a(u,u) + \lambda \|u\|_{L^2(\Omega)} \geq \alpha \|u\|_V$ for $\lambda \geq 0, \alpha > 0$, with $a(u,v) = \int \nabla u(\mathbf{x}) \nabla v(\mathbf{x})\, d\Omega$.
$|a(u,v)| = |\int \nabla u \nabla v\, d\Omega| \leq \sqrt{\int (\nabla u)^2 d\Omega} \sqrt{\int (\nabla v)^2 d\Omega} = \|u\|_V \|v\|_V$ so the the scheme is continuous.
$|a(v,v)| = \int \nabla v \nabla v\, d\Omega = \|v\|_V^2$, so this scheme is weakly coercive for $\alpha \leq 1$. Therefore there exists a unique solution.

Trying to solve our problem using v and u in the space $H^1(\Omega)$, approximated by $X_h^1 = \{v_h \in C^0(\bar{\Omega}) : v_h|_K \in \mathbb{P}_1 \forall K \in \tau_h\}$ in both 2 and 3 dimentions. $V_h = \{v_h \in X_h^1 : v$ satisfying boundary conditions$\}$. Using Lagrangian functions $\phi_j \in V_h$ with the property

$$\phi_j(x_i) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

0.2.2 Galerkin

We can make a Galerkin problem using

$$u_h(\mathbf{x}, t) = \sum_{j=1}^{N_h} u_j(t)\phi(\mathbf{x}) \text{ and } v \in span\{\phi_1, \phi_2, \cdots, \phi_{N_h}\}.$$

This gives

$$\int \sum_{j=1}^{N_h} \frac{\partial u_j}{\partial t} \phi_j \phi_i d\Omega + a(\sum_{j=1}^{N_h} u_j(t)\phi_j, \phi_i) = \int_\Gamma \sum_{j=1}^{N_h} \frac{\partial u_j(t)\phi_j}{\partial n} \phi_i d\gamma$$

$$\sum_{j=1}^{N_h} \frac{\partial u_j}{\partial t} \int \phi_j \phi_i d\Omega + \sum_{j=1}^{N_h} u_j(t) a(\phi_j, \phi_i) = \sum_{j=1}^{N_h} \frac{\partial u_j}{\partial t} m_{ij} + \sum_{j=1}^{N_h} u_j(t) a_{ij} = b_i$$

This is the linear system, $M\dot{\mathbf{u}}(t) + A\mathbf{u}(t) = \mathbf{0}$, the boundary conditions get inserted differently.
The Lagrangian functions ϕ is on the form $\phi_i = \alpha_i x \beta_i y + \gamma_i$. For K_j ϕ_i is 1 in $(x, y) = (x_i, y_i)$ and 0 in the other corners in K. The K elements are triangles in 2 dimentions and tetrahedrons in 3 dimentions, and together they cover our domain Ω.

The Galerkin method let's us finally construct forward Euler method iterations on the form $M\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} + A\mathbf{u}^k = 0$.

0.2.3 Stability of forward euler galerkin

The stability analysis for forward euler in time and FEM in space we find by using w_h^j which is eigenvectors of $a(\cdot, \cdot)$ instead of ϕ, the step-size h may vary in space.

$$\frac{1}{\Delta t} \sum_{j=1}^{N_h} [u_j^{k+1} - u_j^k] \int w_h^j w_h^i + \sum_{j=1}^{N_h} u_j^k a(w_h^j, w_h^i) = 0$$

.

The eigenvalues of the matrix can be calculated

$$a(w_h^j, w_h^i) = \int \nabla w_h^j \nabla w_h^i d\Omega = \lambda_h^j \int w_h^j w_h^i d\Omega = \lambda_h^j (w_h^j, w_h^i) = \lambda_h^j \delta_{ij} = \lambda_h^i$$

giving

$$\frac{u_j^{k+1} - u_j^k}{\Delta t} + u_j^k \lambda_h^i = 0 \Rightarrow u_j^{k+1} = u_j^k (1 - \lambda_h^j \Delta t).$$

For the method to be absolutely stable we need $|1 - \lambda_h^j \Delta t| < 1$.
$-1 < 1 - \lambda_h^j \Delta t < 1 \Rightarrow 0 < \Delta t < \frac{2}{\lambda_h^i}$. The eigenvalues for the stiffness matrix are

$$\max_i \lambda_h^i = \max_i \frac{a(w_h^i, w_h^i)}{\| w_h^i \|_{L^2(\Omega)}^2} \leq \max_i \frac{\| w_h^i \|_V^2}{\| w_h^i \|_{L^2(\Omega)}^2} \simeq (1 + \frac{c}{h^2}), \text{ since } \|\nabla u\|_{L^2(\Omega)} \leq \frac{C\|u\|_{L^2(\Omega)}}{h}.$$

We get that this method only is stable if $\Delta t \leq Ch^2$.

0.2.4 Initial and boundary conditions

We assume that the body initially has a uniform temperature, T_i.
On the boundary we have Neumann-conditions. We divide the boundary in two parts, $\partial\Omega_H$, $\partial\Omega_C$. $\partial\Omega_H$ is the part of the boundary that is in contact with temperature T_H, while $\partial\Omega_C$ is in contact with temperature T_C.

0.2.5 Numerical scheme

The resulting equations that needs to be solved is

$$\begin{aligned} M\tfrac{du}{dt} &= -\alpha A u && \text{on } \Omega \\ \tfrac{\partial u}{d\nu} &= \alpha_H (T_H - u) && \text{on } \partial\Omega_H \\ \tfrac{\partial u}{d\nu} &= \alpha_C (T_C - u) && \text{on } \partial\Omega_C. \end{aligned}$$

This is a simple set of ODEs, and is solved using forward Euler. The numerical scheme can then be written as

$$
\begin{aligned}
u^{i+1} &= u^i - \alpha(M\backslash A)u^i && \text{on } \Omega \\
u^{i+1} &= u^i + \alpha_H(T_H - u^i) && \text{on } \partial\Omega_H \\
u^{i+1} &= u^i + \alpha_C(T_C - u^i) && \text{on } \partial\Omega_C
\end{aligned} \; .
$$

Where α, α_H and α_C are the heat conductivity in the body, heat conductivity between $\partial\Omega_H$ and T_H, the heat conductivity between $\partial\Omega_C$ and T_C, respectivly.

0.2.6 Correctness of Method

To test if A, b and M where correct, we wanted to find

$$
\begin{aligned}
E_A &= ||Au - b_1||_2, \quad & b_1 &= \int_\Omega f v d\Omega \\
E_M &= ||Mu - b_2||_2, \quad & b_2 &= \int_\Omega u v d\Omega
\end{aligned}
$$

with $u(r) = \sin(2\pi r^2)$, $f(r) = \nabla^2 u(r) = -8\pi\cos(2\pi r^2) + 16r^2\pi^2\sin(2\pi r^2)$ on the unit sphere. The program `error3d.m` was made for this purpose, and produced tabel 1, and figure 5. From table 1 it seems that E_A does not converge, but some experimentation with `error3d.m` shows that it decreases extremly slowly with increasing N. For the matrix M this is not an issue and it converges nicely. The 2 dimensional case will not be discussed in this report, but the program `error2d.m` can be used if this is of interest.

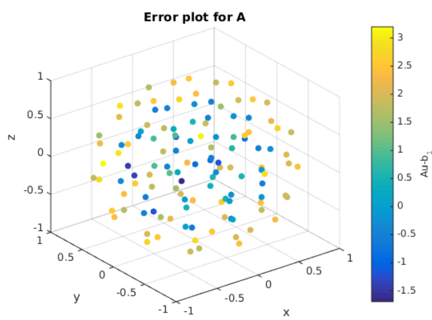
0.2.7 Example of use

As an example of use we wanted to cook a beef. For that we need a lot of different assumptions.

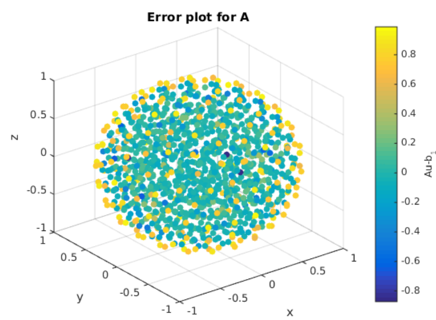
- The beef has a initial uniform temperature $T_i = 10°C$, and is turned when the temperature on the middle poit reaches a temperature $T_f = 60°C$. It is done when the lowest temperature in the beef is $T_f = 60°C$. $T_H = 180°C$, $T_C = 20°C$.
- The beef is discretized with $n_1 = n_2 = n_3 = 5$ with the function `getbeef.m`.
- Thermal conductivities are independant of temperature, and as follows: $\alpha = 8.75\cdot 10^{-8}$, $\alpha_H = 0.7$, $\alpha_C = 3\cdot 10^{-4}$.

#	$N = 125$	$N = 250$	$N = 1250$
E_A	19.50	22.09	12.82
E_M	0.1165	0.0556	0.0121

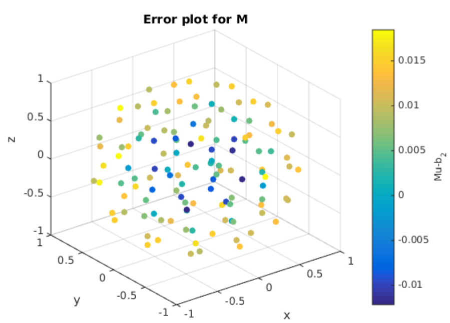
Table 1: Error of A and M, when compared to their corresponding b_1 and b_2.



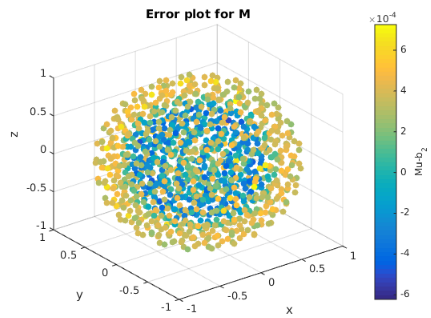
(a) Plot of $Au - b_1$, with $N = 125$.



(b) Plot of $Au - b_1$, with $N = 1250$.



(c) Plot of $Mu - b_2$, with $N = 125$.



(d) Plot of $Mu - b_2$, with $N = 1250$.

Figure 5: A scatterplott showing the pointwise error of A and M, with different number of points, N.

α is given in the task, α_H is assumed to be quite large, while α_C is assumed to be somwhere in between α and α_H. The other constants are chosen freely.

Result

Running the program `beef.m` with these assumptions, gives out the following times:

- Time to turn the beef: 658 seconds
- Total time to the beef is done: 776 seconds

This is close to 13 minutes in total cookingtime, which seems about right for a normal steak. As we can see from figure 6, the edges are all cooked, while the upper middle is some what raw.

Discussion

Although the program `beef.m` gives a time-estimate similar to the reality, it is in no way an accurate description. The numerical scheme has it problems, the A matrix converges extremely slow with N. The program gives different cookingtime with differnet values of n_i.

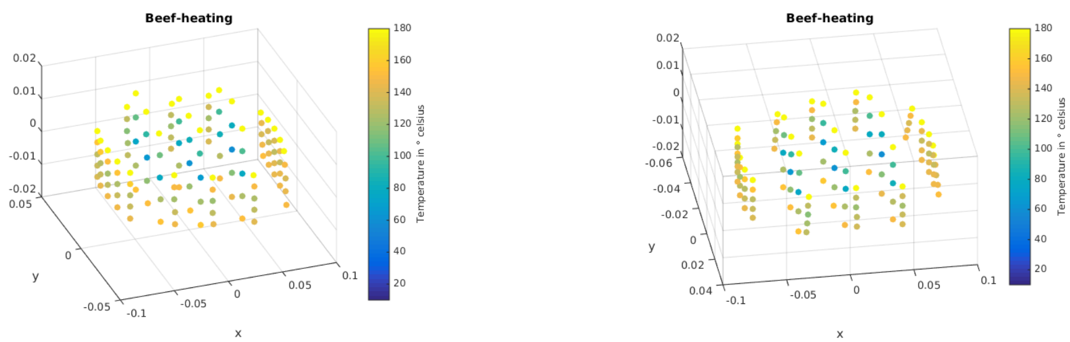


Figure 6: A scatterplott of a cooked beef from two different angles.