Technische Universität Chemnitz
Dr. M. Winkler

Chemnitz, 12th June 2019
hand in on 26th June 2019

Numerical Methods for Partial Differential Equations

Sheet 9

Exercise 23: Implementation of finite elements

Download the files `area_integrator.m`, `affine_transformation.m`, `FE_Lagrange_1.m`, `myinitmesh.m`, `myrefinemesh.m`, `quadrature_unit_triangle_area.m`, `plot_function.m`, `setupmesh.m` and `SolvePoisson.m` from the website of the lecture.

(a) Create a mesh (grid) of the square $[-1,1]^2$ by using

```
mesh = myinitmesh('squareg', 'hmax', 0.5);
```

The mesh can be plotted with the command "`pdemesh(mesh.p, mesh.be, mesh.t)`" and its components can be accessed via "`mesh.p`", "`mesh.e`" and "`mesh.t`" Get familiar with the data structure of the mesh and read "`help myinitmesh`" and "`help initmesh`".

(b) The file `FE_Lagrange_1.m` contains the skeleton for the implementation of a first order Lagrange element. Complete the implementation by appropriately changing all lines/blocks marked with "FIXME". Afterwards, you are able to plot functions from the finite element space by using "`plot_function(mesh, fe, u)`", where `fe` is the finite element and `u` is a vector of the correct size.

(c) The function `area_integrator.m` assembles for each finite element the stiffness matrix A, the mass matrix M and the force vector f. Correct the lines marked with "FIXME" here as well. Section 13.1 from the lecture might be helpful.

(d) Write a Matlab script `SolvePoisson.m` which uses the routines from (b) and (c) to solve the Poisson equation

$$-\operatorname{div}(a\nabla u) = f \text{ in } \Omega, \qquad u = 0 \text{ on } \Gamma,$$

with linear finite elements. The coefficients a and f have to be implemented as functions. The function handles are provided to `area_integrator` as function parameters (type "`help area_integrator`" to get an explanation).

For testing use $a \equiv 1$ and $f(x,y) = 4 - 2(x^2+y^2)$. The exact solution reads $u(x,y) = (1-x^2)(1-y^2)$.

After the assembly of A and f one has to incorporate homogeneous Dirichlet boundary conditions. This can be realized by removing rows and columns from A

and f which belong to Dirichlet DOFs. The indices of the boundary nodes can be determined from `mesh.be` (boundary edges). The Matlab commands `unique` and `setdiff` might be helpful. Don't forget to incorporate the Dirichlet conditions again after having solved the equation system.

(e) Implement some other finite elements, e.g., the Lagrange elements $\mathbb{P}_2$, $\mathbb{P}_3$ or the cubic Hermite element.

Exercise 24: Implementation of finite elements for time-dependent problems

(a) We consider the **heat equation**

$$\begin{aligned} \partial_t u - \Delta u &= f \quad \text{in } \Omega \times (0,T], \\ u &= 0 \quad \text{on } \partial\Omega \times (0,T], \\ u(\cdot,0) &= u_0 \quad \text{in } \Omega, \end{aligned}$$

where $u(x,t)$ describes the temperature of Ω in the point $x \in \Omega$ at time $t \in [0,T]$. Moreover, $f\colon \Omega \times [0,T] \to \mathbb{R}$ is a heat source and $u_0\colon \Omega \to \mathbb{R}$ is the initial temperature.

(i) Fix a time point $t \in (0,T]$ and derive the weak formulation:

$$\int_\Omega \partial_t u(\cdot,t)\, v \,\mathrm{d}x + \int_\Omega \nabla u(\cdot,t) \cdot \nabla v \,\mathrm{d}x = \int_\Omega f(\cdot,t)\, v \,\mathrm{d}x \quad \forall v \in H^1(\Omega).$$

(ii) Apply a finite-element discretization with respect to the space variable only. This means, the coefficients (DOF values) denoted by $u_i = \sigma_i(u_h)$ are still time-dependent! Hence, the semi-discrete solution can be represented by

$$u_h(x,t) = \sum_{i=1}^{S} u_i(t)\varphi_i(x),$$

where φ_i, $i = 1,\dots,S$, are the global shape functions. In a matrix-vector notation this approach leads to a first-order ODE system of the form

$$M\,\dot{u}(t) + A\,u(t) = f(t).$$

Identify the matrices M and A as well as the vector f.

(iii) Use a one-step method (e. g. implicit Euler or Crank-Nicolson scheme) to get a time-discretization as well. To this end, introduce an equidistant time grid $t_k := k\,\tau$, $\tau = T/N$ with N the number of timesteps, and compute the solution

$$u_h^0 := \mathcal{I}_{\mathcal{T}}(u_0), \quad u_h^1 = u_h(\cdot,t_1), \quad u_h^2 = u_h(\cdot,t_2), \quad \dots$$

from one time step to the next one. To do so, replace the time derivative $\dot{u}_h$ by $\frac{1}{\tau}(u_h^{k+1} - u_h^k)$ and evaluate the the remaining terms $A\,u$ and f in the time

point $t = \theta t_{k+1} + (1-\theta)t_k$ for some $\theta \in [0,1]$. In order to evaluate u_h in this time point, use a linear interpolation in time, i. e., $u_h(\cdot,t) = \theta\, u_h^{k+1} + (1-\theta)\, u_h^k$. This approach leads to

- the implicit Euler scheme for $\theta = 1$,
- the Crank-Nicolson method for $\theta = 1/2$.

Implement your approach in Matlab. A template `SolveHeat.m` is available on the website. Reuse the routines implemented in Exercise 23. As a test case consider $u_0 \equiv 0$ and $f(x) = 1$ is $x \in B_{0.2}(0.5, 0.2)$ and $f(x) = 0$ otherwise.

Hint: The Crank-Nicolson-scheme is not unconditionally stable. The time step size τ must be sufficiently small compared to the space grid size h.

The approach discussed here is also referred to as "method of (vertical) lines" (german: "(vertikale) Linienmethode"). A more detailed explanation can be found in the literature or internet.

(b) Consider also the **wave equation**

$$\begin{aligned} \partial_{tt} u - \Delta u &= 0 && \text{in } \Omega \times (0,T], \\ u &= 0 && \text{on } \partial\Omega \times (0,T], \\ u(\cdot,0) &= u_0 && \text{in } \Omega, \\ \partial_t u(\cdot,0) &= v_0 && \text{in } \Omega, \end{aligned}$$

describing the amplitude $u(x,t)$ of a wave in $x \in \Omega$ at $t \in [0,T]$. Here, u_0 denotes the initial amplitude and v_0 the initial impulse. A finite element discretization in space, as in (a), will yield a second-order ODE system of the form

$$M\,\ddot{u}(t) + A\,u(t) = 0.$$

Transform this ODE to a first-order system by introducing an additional variable $v = \dot{u}$. Then, apply the Crank-Nicolson method or the implicit Euler method to solve this problem on an equidistant time grid.

Test your problem with $u_0 \equiv 0$ and

$$v_0(x) = 10 \cdot \left[\exp(-100\,|x - (0,0.8)|^2) + \exp(-100\,|x + (0,0.8)|^2)\right].$$

This models the water level of a lake after throwing two stones into it.

The template `SolveWave.m` might serve as a starting point for your implementation.

Homework 20: Proving convergence rates experimentally

Download the files `lshapeg.m`, `polyg.m`, `test_FEM.m` from the web site of the lecture.

(a) Solve the PDE

$$\begin{aligned} -a_{11}\, D_1^2 u - a_{22}\, D_2^2 u + c\, u &= f \quad \text{in } \Omega = (-1,1)^2 \\ \frac{\partial u}{\partial n} &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

for various parameters $a_{11}, a_{22}, c \geq 0$.

(b) Solve the above PDE on a sequence of meshes. In order to compute the error, one can prolongate coefficients of finite element functions to refined meshes:

```
% Assume that 'u' is the coefficient vector associated
% with the finite element 'fe' on the mesh 'mesh'.

% Refine the mesh
[fine_mesh, P] = myrefinemesh( mesh, fe );

% Prolongate the solutions to the new mesh
v = P * u;

% Now, 'v' is the coefficient vector of the same
% function on the finer mesh 'fine_mesh'
```

Now, the $H^1(\Omega)$-error of the solution on a coarse mesh can be approximated by computing the distance to the solution on a very fine mesh.

Hint: To compute the error use the formula from Sheet 8, Exercise 22.

(c) Compute the experimental convergence rates (see Sheet 3, Homework 3) and draw the error curve connecting the data points $(h_k, \|u - u_{h_k}\|_{H^1(\Omega)})$ into a plot with logarithmic axes. Do the results coincide with the theoretically predicted behavior from Theorem 15.3?