R. Hiptmair
L. Scarabosio
C. Urzua Torres

Spring Term 2015

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 13

Problem 13.1 The One-Dimensional Wave Equation (Core problem)

[NPDE, Section 6.2.2] introduced the one-dimensional wave equation with constant coefficients as a simple model for wave propagation. There, in [NPDE, § 6.2.20], we examined the so-called Cauchy problem, for which the spatial domain is the whole real line. In this problem we return to a bounded spatial domain and impose non-homogeneous Dirichlet boundary conditions that depend on time. F

The 1D wave equation with constant coefficients reads

$$\frac{\mathrm{d}^2 u}{\mathrm{d}t^2} - c\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = 0, \quad \text{on } (0,1) \times (0,T)\ . \tag{13.1.1}$$

The partial differential equation is supplemented with Dirichlet boundary conditions and zero initial conditions

$$u(0,t) = 0, \quad u(1,t) = \begin{cases} \sin t & 0 \le t \le \pi, \\ 0 & \text{otherwise}, \end{cases}$$
$$u(x,0) = 0, \quad \frac{\mathrm{d}u}{\mathrm{d}t}(x,0) = 0.$$

This initial-boundary value problem can be tackled numerically using the method of lines, see [NPDE, Section 6.2.3], which, intermittently, leads to the ODE

$$\mathbf{M}\frac{\mathrm{d}^2 u}{\mathrm{d}t^2}\vec{\boldsymbol{\mu}} + \mathbf{A}\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\varphi}}(t), \tag{13.1.2}$$

for the time-dependent coefficient vector $\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\mu}}(t)$ associated with a spatial Galerkin discretization.

In this task we focus on finite element Galerkin discretization with piecewise linear Lagrangian finite elements on equidistant meshes, see [NPDE, Section 1.5.2.2].

The non-homogeneous Dirichlet boundary condition at $x = 1$ can be taken into account through the use of a locally supported *offset function* as explained in [NPDE, Rem. 1.5.81], see also [NPDE, Section 3.6.5].

(13.1a) Compute the stiffness matrix $\mathbf{A}$ and the mass matrix $\mathbf{M}$ using the trapezoidal rule [NPDE, Eq. (1.5.73)] to evaluate the integrals.

HINT: The mass matrix will be diagonal, because the use of this particular quadrature formula effects "mass lumping" [NPDE, Rem. 6.2.45]. The stiffness matrix has already been computed in [NPDE, Section 1.5.2.2].

Solution: The *full* matrices, including the boundary points, are $N+2 \times N+2$:

$$\mathbf{M} = h \begin{pmatrix} \frac{1}{2} & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & \frac{1}{2} \end{pmatrix}, \qquad \mathbf{A} = \frac{c}{h} \begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}.$$

(13.1b) Find a piecewise linear, time-dependent, and locally supported offset function g, which we can used to incorporate the nonhomogenous Dirichlet boundary condition into the semi-discrete problem.

HINT: [NPDE, Rem. 1.5.81] with additional dependence on time.

Solution: If $b^{N+1}(x)$ is the rightmost basis function, we can use

$$g(x,t) = b^{N+1}(x)u(1,t).$$

(13.1c) What is the right-hand side of the ordinary differential equation (13.1.2) arising from the method of lines?

HINT: The right-hand side will involve the offset function found in subproblem (13.1b).

Solution: Write $\boldsymbol{\mu} = \boldsymbol{\mu}_0 + \boldsymbol{\mu}_g$, where $\boldsymbol{\mu}_0$ is the vector of internal values and $\boldsymbol{\mu}_g$ is the vector of boundary values (both of size $N+2$). Then (13.1.2) reduces to

$$\mathbf{M}\ddot{\vec{\boldsymbol{\mu}}}_0 + \mathbf{A}\vec{\boldsymbol{\mu}}_0 = -\mathbf{A}\vec{\boldsymbol{\mu}}_g.$$

The term $-\mathbf{M}\ddot{\vec{\boldsymbol{\mu}}}_g$ can be removed from the right-hand side because this vector will be zero everywhere except for in the last element ($\mathbf{M}$ is diagonal!), and we will only solve (13.1.2) for the interior nodes anyway.

(13.1d) Write a MATLAB function

```
U = wave(N, K, T, c)
```

that solves (13.1.1) with the method just described, with `N` interior nodes in space, `K` timesteps and final time `T`, and returns the results in a $(N+2) \times (K+1)$-matrix `U`. Use leapfrog timestepping from [NPDE, § 6.2.43], in particular [NPDE, Eq. (6.2.44)] and do not forget the special initial step.

Solution: See Listing 13.1.

Listing 13.1: Implementation for `wave.m`

```
function [U, el, ki, po, xvals, tvals] = wave(N, K, T, c)

    xvals = linspace(0,1,N+2);
```

```
    h = xvals(2);
    tvals = linspace(0,T,K+1);
    tau = tvals(2);
    M = h*speye(N+2, N+2); M(1,1) = h/2; M(N+2,N+2) = h/2;
    A = c*spdiags([-ones(N+2,1), 2*ones(N+2,1),
       -ones(N+2,1)], -1:1, N+2, N+2)/h; A(1,1) = c/h;
       A(N+2,N+2) = c/h;

    fd = 2:N+1;

    U = zeros(N+2,K+1);
    U(N+2,:) = sin(tvals) .* (tvals <= pi);
    nu = zeros(N,K);
    el = []; ki = []; po = [];

    for i=1:K

        % Update nu

        L = -A*U(:,i); L = L(fd);
        if i == 1
            nu = (tau/2) * (M(fd,fd) \ L);
        else
            L = L + M(fd,fd)*nu/tau;
            nu = tau * (M(fd,fd) \ L);
        end

        % Update mu

        U(fd,i+1) = U(fd,i) + tau*nu;

        el = [el, U(:,i+1)' * A * U(:,i+1)];
        tv = (tvals(i)+tvals(i+1))/2;
        exnu = [0; nu; cos(tv)*(tv<=pi)];
        ki = [ki, exnu' * M * exnu/2];
          meanu=(U(:,i)+U(:,i+1))/2;
        po = [po, meanu' * A * meanu/2];

        plot(xvals, U(:,i));
        axis([0, 1, -1, 1]);

        pause(0.005);

    end

end
```

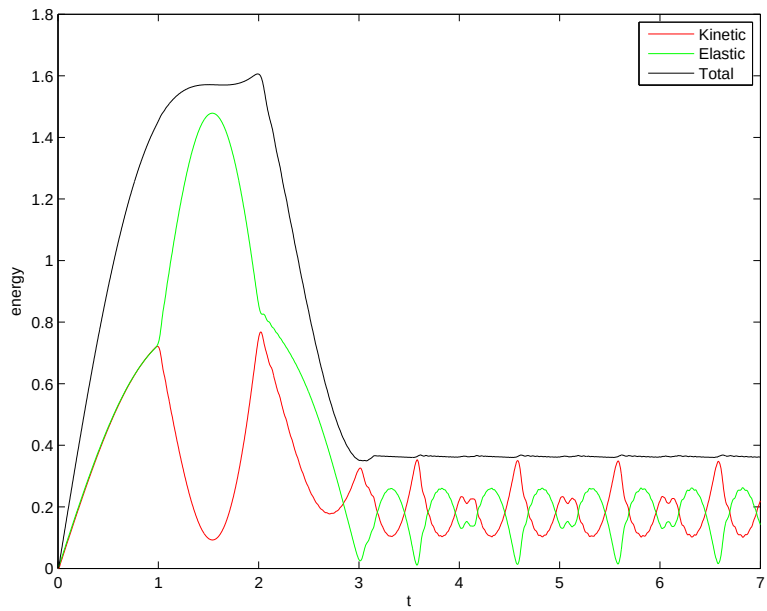


Figure 13.1: Energy plots for subproblem (13.1f).

(13.1e) Run your code with $N = 100$, $K = 2000$, $T = 7$ and $c = 1$. Plot the solution continually (MATLAB command `drawnow`) while solving (use the MATLAB `pause` command to slow down or halt execution so that you can actually see the "movie").

HINT: For debugging purposes: the value of the solution at point $(50, 2000)$ should be 0.0761.

(13.1f) As in [NPDE, Exp. 6.2.46], plot the (discrete) elastic, kinetic and total energies as a function of time.

HINT: Formulas are given in [NPDE, Section 6.2.4]. In particular, [NPDE, Code 6.2.48] may be useful.

Solution: See Figure 13.1.

(13.1g) Describe the behavior of the solution computed in subproblem (13.1e) in qualitative terms related to "wave propagation".

Solution: You should see a wave propagating from right to left as the "string" is raised. When it reaches the right-hand side it should bounce back and forth.

Problem 13.2 Crank-Nicolson Timestepping (Core problem)

In this problem we conduct some analysis of a two-step timestepping scheme for the semi-discrete wave equation.

As an alternative to leapfrog timestepping, for the typical method of lines ODE for wave propagation problems, see [NPDE, Section 6.2.3],

$$\mathbf{M}\ddot{\vec{\mu}} + \mathbf{A}\vec{\mu} = \vec{\varphi}(t),$$

one may use the *Crank-Nicolson method*, in analogy to [NPDE, Eq. (6.2.41)] written as

$$\mathbf{M}\frac{\vec{\boldsymbol{\mu}}^{(j+1)} - 2\vec{\boldsymbol{\mu}}^{(j)} + \vec{\boldsymbol{\mu}}^{(j-1)}}{\tau^2} = \\ -\mathbf{A}\left(\frac{1}{4}\vec{\boldsymbol{\mu}}^{(j+1)} + \frac{1}{2}\vec{\boldsymbol{\mu}}^{(j)} + \frac{1}{4}\vec{\boldsymbol{\mu}}^{(j-1)}\right) + \frac{1}{4}\vec{\boldsymbol{\varphi}}(t_{j+1}) + \frac{1}{2}\vec{\boldsymbol{\varphi}}(t_j) + \frac{1}{4}\vec{\boldsymbol{\varphi}}(t_{j-1}). \quad (13.2.1)$$

As was done for leapfrog in [NPDE, § 6.2.43], it can also be formulated as a single-step method,

$$\frac{\vec{\boldsymbol{\mu}}^{(j+1)} - \vec{\boldsymbol{\mu}}^{(j)}}{\tau} = \frac{1}{2}\left(\vec{\mathbf{v}}^{(j+1)} + \vec{\mathbf{v}}^{(j)}\right), \quad (13.2.2)$$

$$\mathbf{M}\frac{\vec{\mathbf{v}}^{(j+1)} - \vec{\mathbf{v}}^{(j)}}{\tau} = -\mathbf{A}\frac{\vec{\boldsymbol{\mu}}^{(j+1)} + \vec{\boldsymbol{\mu}}^{(j)}}{2} + \frac{\vec{\boldsymbol{\varphi}}(t_{j+1}) + \vec{\boldsymbol{\varphi}}(t_j)}{2}. \quad (13.2.3)$$

(13.2a) Show that (13.2.1) is equivalent to (13.2.2)–(13.2.3), i.e., they both give the same recursion for $\vec{\boldsymbol{\mu}}^{(j)}$.

Solution: We start from the left-hand side of (13.2.1) and use (13.2.2)–(13.2.3) to arrive at the right-hand side of (13.2.1) and thus the conclusion.

$$\begin{aligned}
\mathbf{M}\frac{\vec{\boldsymbol{\mu}}^{(j+1)} - 2\vec{\boldsymbol{\mu}}^{(j)} + \vec{\boldsymbol{\mu}}^{(j-1)}}{\tau^2} &= \frac{\mathbf{M}}{\tau^2}\left[\vec{\boldsymbol{\mu}}^{(j+1)} - 2\vec{\boldsymbol{\mu}}^{(j)} + \vec{\boldsymbol{\mu}}^{(j-1)}\right] \\
&= \frac{\mathbf{M}}{\tau}\left[\frac{1}{\tau}\left(\vec{\boldsymbol{\mu}}^{(j+1)} - \vec{\boldsymbol{\mu}}^{(j)}\right) - \frac{1}{\tau}\left(\vec{\boldsymbol{\mu}}^{(j)} - \vec{\boldsymbol{\mu}}^{(j-1)}\right)\right] \\
&= \frac{\mathbf{M}}{2\tau}\left[\left(\vec{\mathbf{v}}^{(j+1)} + \vec{\mathbf{v}}^{(j)}\right) - \left(\vec{\mathbf{v}}^{(j)} + \vec{\mathbf{v}}^{(j-1)}\right)\right] \\
&= \frac{1}{2}\left[\frac{\mathbf{M}}{\tau}\left(\vec{\mathbf{v}}^{(j+1)} - \vec{\mathbf{v}}^{(j)}\right) + \frac{\mathbf{M}}{\tau}\left(\vec{\mathbf{v}}^{(j)} - \vec{\mathbf{v}}^{(j-1)}\right)\right] \\
&= -\mathbf{A}\frac{\vec{\boldsymbol{\mu}}^{(j+1)} + 2\vec{\boldsymbol{\mu}}^{(j)} + \vec{\boldsymbol{\mu}}^{(j-1)}}{4} + \frac{\vec{\boldsymbol{\varphi}}(t_{j+1}) + 2\vec{\boldsymbol{\varphi}}(t_j) + \vec{\boldsymbol{\varphi}}(t_{j-1})}{4}.
\end{aligned}$$

(13.2b) Show that (13.2.2)–(13.2.3) conserves the discrete energy,

$$E_j = \frac{1}{2}(\vec{\mathbf{v}}^{(j)})^\top \mathbf{M}\vec{\mathbf{v}}^{(j)} + \frac{1}{2}(\vec{\boldsymbol{\mu}}^{(j)})^\top \mathbf{A}\vec{\boldsymbol{\mu}}^{(j)}.$$

HINT: Take the scalar product of (13.2.2) with $\frac{1}{2}\mathbf{A}\left(\vec{\boldsymbol{\mu}}^{(j)} + \vec{\boldsymbol{\mu}}^{(j+1)}\right)$, and of (13.2.3) with $\frac{1}{2}\left(\vec{\mathbf{v}}^{(j)} + \vec{\mathbf{v}}^{(j+1)}\right)$, and then add them together.

Solution: With $\boldsymbol{\varphi} = 0$ we can readily see that the right-hand sides cancel out. Therefore, we get

$$\begin{aligned}
\left\langle \vec{\boldsymbol{\mu}}^{(j+1)} - \vec{\boldsymbol{\mu}}^{(j)}, \mathbf{A}\vec{\boldsymbol{\mu}}^{(j)} + \mathbf{A}\vec{\boldsymbol{\mu}}^{(j+1)} \right\rangle + \left\langle \mathbf{M}\vec{\mathbf{v}}^{(j+1)} - \mathbf{M}\vec{\mathbf{v}}^{(j)}, \vec{\mathbf{v}}^{(j)} + \vec{\mathbf{v}}^{(j+1)} \right\rangle &= 0 \\
\left\langle \vec{\boldsymbol{\mu}}^{(j+1)}, \mathbf{A}\vec{\boldsymbol{\mu}}^{(j+1)} \right\rangle + \left\langle \mathbf{M}\vec{\mathbf{v}}^{(j+1)}, \vec{\mathbf{v}}^{(j+1)} \right\rangle - \left\langle \vec{\boldsymbol{\mu}}^{(j)}, \mathbf{A}\vec{\boldsymbol{\mu}}^{(j)} \right\rangle - \left\langle \mathbf{M}\vec{\mathbf{v}}^{(j)}, \vec{\mathbf{v}}^{(j)} \right\rangle &= 0 \\
E_{j+1} - E_j &= 0.
\end{aligned}$$

Problem 13.3 Wave Equation with Perfectly Matched Layers

As in [NPDE, § 6.2.20] we consider Cauchy problem for the 1D wave equation (on the unbounded domain $\Omega = \mathbb{R}$):

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - \frac{\partial}{\partial x}\left(c^2(x)\frac{\partial u}{\partial x}\right) &= 0 \qquad \text{on } \Omega \times (0,T), \\ u(x,0) = u_0(x), \quad \frac{\partial u}{\partial t}(x,0) &= v_0(x), \quad \text{on } \Omega, \end{aligned} \tag{13.3.1}$$

where

$$c(x) = 1 \quad \text{for } x \geq 1,\, x \leq -1, \quad \text{and} \quad \operatorname{supp}(u_0),\, \operatorname{supp}(v_0) \subset]-1,1[.$$

We are only interested in that part of the solutions of (13.3.1) that lies inside $[-1,1]$.Nevertheless we can not simply restrict problem (13.3.1) to $[-1,1]$, for instance by imposing Dirichlet boundary conditions, since reflected waves would completely supersede the solution of the Cauchy problem after a short time. Such reflections could be seen in subproblem (13.1e).

Instead we have to use *absorbing boundary conditions* that let waves pass undisturbed. One option are so-called *perfectly matched layers* (PML). This technique is based on introducing a artificial material in a zone outside the region of interest that absorbes waves. In our example these zones are $[-1-L,-1]$ and $[1,1+L]$.

The variational formulation of the PML augmented 1D wave equations then reads: seek $u(t) \in H^1(]-1-L,1+L[)$ and $v(t) \in L^2([-1-L,1+L])$ such that for all $w \in H^1(]-1-L,1+L[)$ and $q \in L^2(]-1-L,1+L[)$

$$\begin{aligned} \int_{-1-L}^{1+L} \frac{\partial u}{\partial t}\, w \,\mathrm{d}x + \int_{-1-L}^{1+L} \sigma(x) u\, w \,\mathrm{d}x + \int_{-1-L}^{1+L} v\, \frac{\partial w}{\partial x} \,\mathrm{d}x &= \int_{-1-L}^{1+L} v_0\, w \,\mathrm{d}x, \\ \int_{-1-L}^{1+L} \frac{\partial v}{\partial t}\, q \,\mathrm{d}x + \int_{-1-L}^{1+L} \sigma(x) v\, q \,\mathrm{d}x - \int_{-1-L}^{1+L} c^2(x) \frac{\partial u}{\partial x}\, q \,\mathrm{d}x &= 0, \end{aligned} \tag{13.3.2}$$

with

$$\sigma = \begin{cases} 0 & -1 < x < 1 \\ \sigma_0 & x < -1, x > 1, \quad \sigma_0 > 0 \end{cases}. \tag{13.3.3}$$

(13.3a) State the standard variational formulation of (13.3.1), if homogeneous Neumann boundary conditions are imposed at $x = \pm 1$.

Solution: As usual, we start by multiplying by a test function and integrating over the domain

$$\int_{-1}^{1} \frac{\partial^2 u(x,t)}{\partial t^2} w(x) \,\mathrm{d}x - \int_{-1}^{1} \frac{\partial}{\partial x}\left(c^2(x)\frac{\partial u(x,t)}{\partial x}\right) w(x) \,\mathrm{d}x = 0, \quad \forall w \in H^1(]-1,1[). \tag{13.3.4}$$

Then use integrations by parts in the second term

$$\int_{-1}^{1} \frac{\partial^2 u(x,t)}{\partial t^2} w(x) \,\mathrm{d}x + \int_{-1}^{1} \left(c^2(x)\frac{\partial u(x,t)}{\partial x}\right) \frac{\partial w(x)}{\partial x} \,\mathrm{d}x - \left[c^2(x)\frac{\partial u(x,t)}{\partial x} w(x)\right]_{|_{x=-1}^{1}} = 0, \tag{13.3.5}$$

$\forall v \in H^1(]-1,1[)$. Since the last term vanishes due to homogeneous Neumann boundary conditions, we finally obtain: seek $u \in V(t)$

$$\int_{-1}^{1} \frac{\partial^2 u(x,t)}{\partial t^2} w(x)\,\mathrm{d}x + \int_{-1}^{1} \left(c^2(x) \frac{\partial u(x,t)}{\partial x} \right) \frac{\partial w(x)}{\partial x}\,\mathrm{d}x = 0, \quad \forall w \in H^1(]-1,1[), \tag{13.3.6}$$

with $V(t) := \{v :]0,T[\to H^1(]-1,1[) : \frac{\partial v(x,t)}{\partial x} = 0 \text{ for } x \in \{-1,1\} \times]0,T[\to \mathbb{R}\}$.

(13.3b) State the variational problem (13.3.2) for $L = 0$, which means that the absorbing layer is ignored.

Solution:

$$\int_{-1}^{1} \frac{\partial u}{\partial t}\, w\ \mathrm{d}x + \int_{-1}^{1} \sigma(x) u\, w\ \mathrm{d}x + \int_{-1}^{1} v\, \frac{\partial w}{\partial x}\ \mathrm{d}x = \int_{-1}^{1} v_0\, w\ \mathrm{d}x,$$
$$\int_{-1}^{1} \frac{\partial v}{\partial t}\, q\ \mathrm{d}x + \int_{-1}^{1} \sigma(x) v\, q\ \mathrm{d}x - \int_{-1}^{1} c^2(x) \frac{\partial u}{\partial x}\, q\ \mathrm{d}x = 0,$$

which considering σ inside $[-1,1]$, boils down to

$$\int_{-1}^{1} \frac{\partial u}{\partial t}\, w\ \mathrm{d}x + \int_{-1}^{1} v\, \frac{\partial w}{\partial x}\ \mathrm{d}x = \int_{-1}^{1} v_0\, w\ \mathrm{d}x, \tag{13.3.7}$$
$$\int_{-1}^{1} \frac{\partial v}{\partial t}\, q\ \mathrm{d}x - \int_{-1}^{1} c^2(x) \frac{\partial u}{\partial x}\, q\ \mathrm{d}x = 0, \tag{13.3.8}$$

(13.3c) Show that the two variational formulations obtained in subproblem (13.3a) and subproblem (13.3b) are equivalent, when one makes the substitution

$$v(x,t) = c(x)^2 \int_0^t \frac{\partial u}{\partial x}(x,\tau)\,\mathrm{d}\tau \quad \Leftrightarrow \quad \frac{\partial v}{\partial t} = c(x)^2 \frac{\partial u}{\partial x}\,. \tag{13.3.9}$$

HINT: Test the right equation in (13.3.9) with a function in $L^2(]-1,1[)$.

Solution: Following the hint, we begin with

$$\int_{-1}^{1} \frac{\partial v}{\partial t} q\,\mathrm{d}x - \int_{-1}^{1} c(x)^2 \frac{\partial u}{\partial x} q\,\mathrm{d}x = 0, \quad \forall q \in L^2(]-1,1[) \tag{13.3.10}$$

and get (13.3.8). Consider now the variational formulation (13.3.6) and integrate over time

$$\int_{-1}^{1} \frac{\partial u(x,t)}{\partial t} w(x)\,\mathrm{d}x - \int_{-1}^{1} v_0(x) w(x)\,\mathrm{d}x + \int_{-1}^{1} v(x,t) \frac{\partial w(x)}{\partial x}\,\mathrm{d}x = 0, \quad \forall v \in H^1(]-1,1[), \tag{13.3.11}$$

Obtaining (13.3.7).

Now we tackle the full discretization of (13.3.2) and we pursue the method-of-lines policy. In particular we resort to a Galerkin finite element discretization in space based on a on a mesh $\mathcal{M}$ with $N+1$ equidistant nodes $x_0 := L-1 \leq \cdots \leq x_N := 1+L$. Then, in (13.3.2), we replace the space $H^1(]-1-L,1+L[)$ with the space $\mathcal{S}_1^0(\mathcal{M})$ of piecewise linear continuous functions, see

[NPDE, § 1.5.62]. As trial and test space for $L^2(]-1-L,1+L[)$ we choose the space $\mathcal{S}_0^{-1}(\mathcal{M})$ of piecewise constant discontinuous functions on $\mathcal{M}$. For both spaces we opt for the canonical local supported nodal basis functions: "tent functions" according to [NPDE, Eq. (1.5.63)] for $\mathcal{S}_1^0(\mathcal{M})$, and the characteristic functions of mesh cells for $\mathcal{S}_0^{-1}(\mathcal{M})$.

The resulting ODE system is discretized via a special variant of leapfrog timestepping, see [NPDE, § 6.2.43]. In each timestep it leads to the following discrete variational equation:

$$\begin{aligned}
\int_{-1-L}^{L+1} \frac{u_N^{(k+1)} - u_N^{(k)}}{\Delta t}\, w_N \,\mathrm{d}x + \int_{-1-L}^{L+1} \sigma \frac{u_N^{(k+1)} + u_N^{(k)}}{2}\, w_N \,\mathrm{d}x + \int_{-1-L}^{L+1} v_N^{(k)} \frac{\partial w_N}{\partial x} \,\mathrm{d}x \\
= \int_{-1-L}^{L+1} v_0\, w_N \,\mathrm{d}x, \\
\int_{-1-L}^{L+1} \frac{v_N^{(k+1)} - v_N^{(k)}}{\Delta t}\, q_N \,\mathrm{d}x + \int_{-1-L}^{L+1} \sigma \frac{v_N^{(k+1)} + v_N^{(k)}}{2}\, q_N \,\mathrm{d}x - \int_{-1-L}^{L+1} c^2 \frac{\partial u_N^{(k+1)}}{\partial x}\, q_N \,\mathrm{d}x = 0,
\end{aligned}$$

with $u_N, w_N \in \mathcal{S}_1^0(\mathcal{M})$ and $v_N, q_N \in \mathcal{S}_0^{-1}(\mathcal{M})$. This scheme will underly the implementation requested in this problem.

(13.3d) Derive first the linear system of equations that has to be solved in each timestep. Describe it using suitable Galerkin matrices and give formulas for their entries.

HINT: The formulas are simple: recall [NPDE, Eq. (1.5.65)] and [NPDE, Eq. (1.5.70)] and make use of the simplifications offered by the equidistant mesh.

Solution: Define

$$\begin{aligned}
\mathbf{M}_\mu^1 &:= \text{Mass matrix corresponding to } \int_{-1-L}^{L+1} \mu\, uw \,\mathrm{d}x \text{ approximated by p.w.linear.} \\
\mathbf{M}_\mu^0 &:= \text{Mass matrix corresponding to } \int_{-1-L}^{L+1} \mu\, uw \,\mathrm{d}x \text{ approximated by p.w.constants.} \\
\mathbf{b}^1 &:= \text{RHS vector corresponding to } \int_{-1-L}^{L+1} v_0 w \,\mathrm{d}x \text{ approximated by p.w.linear.} \\
\mathbf{G}^1 &:= \text{Galerkin matrix corresponding to } \int_{-1-L}^{L+1} v \frac{\partial w}{\partial x} \,\mathrm{d}x \text{ approximated by p.w.linear.} \\
\mathbf{G}_\mu^0 &:= \text{Galerkin matrix corresponding to } \int_{-1-L}^{L+1} \mu \frac{\partial u}{\partial x}\, q \,\mathrm{d}x \text{ approximated by p.w.constants.}
\end{aligned}$$

Re-ordering the equations above, we get

$$\begin{aligned}
\left(\frac{\mathbf{M}_1^1}{dt} + \mathbf{M}_{\sigma/2}^1\right)\mathbf{U}^{(k+1)} &= \mathbf{b}^1 + \left(\frac{\mathbf{M}_1^1}{dt} - \mathbf{M}_{\sigma/2}^1\right)\mathbf{U}^{(k)} - \mathbf{G}^1\mathbf{V}^{(k)} \\
\left(\frac{\mathbf{M}_1^0}{dt} + \mathbf{M}_{\sigma/2}^0\right)\mathbf{V}^{(k+1)} &= \left(\frac{\mathbf{M}_1^0}{dt} - \mathbf{M}_{\sigma/2}^0\right)\mathbf{V}^{(k)} + \mathbf{G}_{c^2}^0\mathbf{U}^{(k+1)}
\end{aligned}$$

(13.3e) Implement the scheme in `main_pml.m` for the inital data given there.

Solution: See Listing 13.2.

Listing 13.2: Implementation for Problem 13.3

```
% 1D Wave equation with perfectly matched layer (PML)
%
% Copyright 2006-2009 Patrick Meury, Holger Heumann
% SAM - Seminar for Applied Mathematics
% ETH-Zentrum
% CH-8092 Zurich, Switzerland

% Initialize constants
L = 1;              % Length of the interval
T = 2;            % Final time
C = 1;              % Speed of sound
ALPHA = 0.75;       % Right-travelling wave
BETA = 0.25;        % Left-travelling wave
U0 = @(x,varargin)(ALPHA+BETA)*pulse_1D(x,varargin{:});      % Initial data
V0 = @(x,varargin)(ALPHA-BETA)*C*dpulse_1D(x,varargin{:});   % Initial velocity
MU = @(x,varargin)C^2*ones(size(x,1),1);                     % Coeff of the wave eq
NPTS = 1000;        % Number of points
NSTEPS = 10000;     % Number of time steps

PML = 0.25;          % Length of PML layer
SIGMA_0 = 50;    % Scaling parameter for absorption profile
SIGMA = @sigma_const_1D; % Absorption profile

XLim = [-(L+PML) L+PML];   % X-axes limits
YLim = [-1.05 1.05];       % Y-axes limits

% Initialize mesh
dx = (XLim(2)-XLim(1))/(NPTS);
Coordinates = transpose(XLim(1):dx:XLim(2));

% Precompute matrices
dt = T/NSTEPS;
QuadRule = gauleg(0,1,4);

MC = M_P0(Coordinates);
ML = M_P1(Coordinates);
ML_sigma = M_P1(Coordinates,QuadRule,SIGMA,SIGMA_0,L);
MC_sigma = M_P0(Coordinates,QuadRule,SIGMA,SIGMA_0,L);
G = assemMatLaplace_P1P0(Coordinates,QuadRule,@(x,varargin)1);
G_mu = assemMatLaplace_P1P0(Coordinates,QuadRule,MU);
G_mu = transpose(G_mu);

S1 = ML/dt + ML_sigma/2;
```

```
S2 = ML/dt - ML_sigma/2;
S3 = MC/dt + MC_sigma/2;
S4 = MC/dt - MC_sigma/2;

% Compute initial data
U_old = ML\assemLoad_P1_1D(Coordinates,QuadRule,U0);
%V_old = zeros(NPTS-1,1);
% projection of V0 to p.w.c
V_old = (V0(Coordinates(1:end-1))+V0(Coordinates(2:end)))/2;

b = assemLoad_P1_1D(Coordinates,QuadRule,V0);

% Integrate ODE system (dissipative leapfrog scheme, CFL
   condition)
fig = figure('Name','1D Wave equation with PML');
plot(Coordinates,U_old,'r-', [-L -L],YLim,'k--', ...
    [ L  L],YLim,'k--');
set(gca,  'XLim',XLim, 'YLim',YLim);
drawnow;

% compute energies
E_p = zeros(NSTEPS+1,1);
E_kin = zeros(NSTEPS+1,1);
% auxiliary variables to consider dofs in [-1,1]
Noffset = length([XLim(1):dx:-L]');
Coordinates_inside = transpose(Coordinates(Noffset):dx:L);
% Galerkin matrix required for H1 seminorm
A = A_P1(Coordinates_inside);
Nend = size(A,1)+ Noffset-1;
dtemp = (U_old(Noffset:Nend)-U0(Coordinates_inside));
E_kin(1) = dtemp'*ML(Noffset:Nend,Noffset:Nend)*dtemp/2;
E_p(1) = (U_old(Noffset:Nend)'*A*U_old(Noffset:Nend))/2;

for i = 1:NSTEPS
    % Compute right-hand side load data
    rhs = b + S2*U_old - G*V_old;

    % Compute new value for U
    U_new = S1\rhs;

    % Compute new value for V
    rhs = S4*V_old + G_mu*U_new;
    V_new = S3\rhs;

    % Compute norms
    dtemp = (U_new(Noffset:Nend)-U_old(Noffset:Nend))/dt;
    E_kin(i+1)    =
```

```
        dtemp'*ML(Noffset:Nend,Noffset:Nend)*dtemp/2;
    E_p(i+1) = (U_new(Noffset:Nend)'*A*U_new(Noffset:Nend))/2;

    % Update old values for U and V
    U_old = U_new;
    V_old = V_new;

    plot(Coordinates,U_old,'r-', [-L -L],YLim,'k--', ...
        [ L  L],YLim,'k--');
    set(gca, 'XLim',XLim, 'YLim',YLim);
    drawnow;

end
%clear all;
%close all;

% Plot energies
figure;
hold on
plot(0:dt:T, E_p, 'b')
plot(0:dt:T, E_kin, 'r')
plot(0:dt:T, E_p + E_kin, 'g')
title('Energy')
xlabel('t')
ylabel('energy')
hold off
```

(13.3f) As in subproblem (13.1f) , plot the (discrete) elastic, kinetic and total energies as a function of time.

HINT: Formulas are given in [NPDE, Section 6.2.4]. In particular, [NPDE, Code 6.2.48] may be useful.

Solution: We use the following formula to compute the error:

$$E(t) = \frac{1}{2}\left\|\frac{\partial u}{\partial t}\right\|^2_{L^2(\Omega)} + \frac{1}{2}|u|^2_{H^1(\Omega)}, \quad \Omega = (0,1).$$

See code Listing 13.2 for details.

(13.3g) Describe the behavior of the solution computed in subproblem (13.3f) in qualitative terms related to "wave propagation".

Solution:

You should observe a decay of energy. See Figure 13.2.

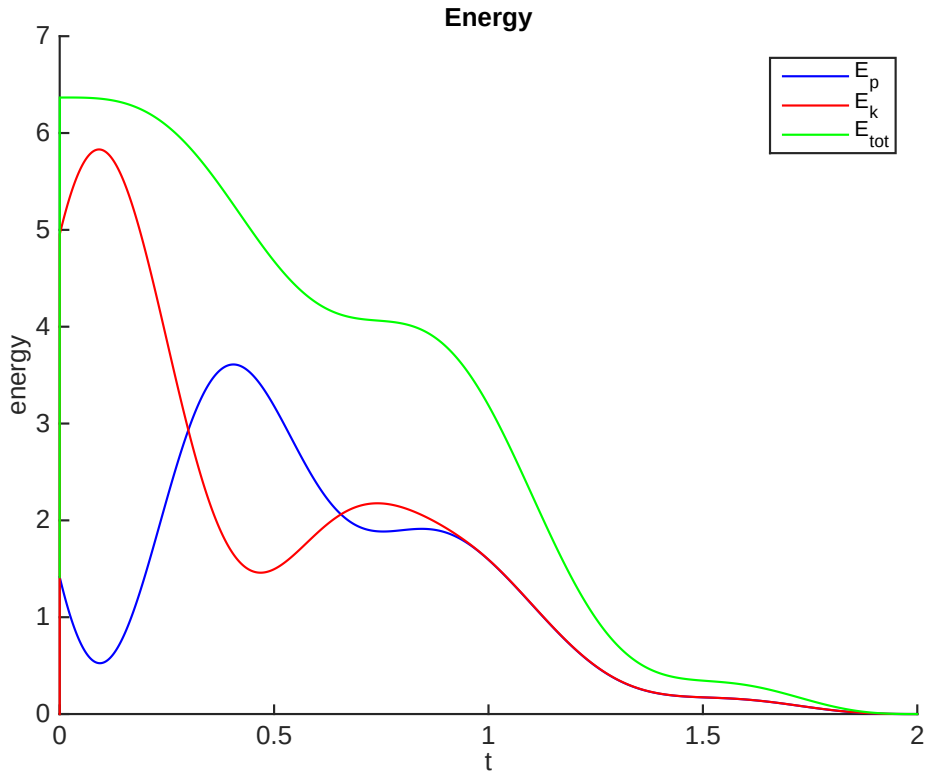


Figure 13.2: Energy for subproblem (13.3g)

Problem 13.4 Helmholtz Equation

So far we have almost always faced initial boundary value problems for the linear wave equation, with the exception of the 1D Cauchy problem from [NPDE, § 6.2.20], where boundary conditions did not occur. In this problem we learn about a situation, where we can drop initial conditions: the *time-periodic* setting.

Now we consider the linear wave equation with homogeneous Dirichlet boundary conditions

$$\begin{aligned} \frac{\partial^2 u(\boldsymbol{x},t)}{\partial t^2} - \Delta u(\boldsymbol{x},t) &= f(\boldsymbol{x},t) \quad \text{in } \Omega \times \mathbb{R}, \\ u(\boldsymbol{x},t) &= 0, \qquad \text{on } \partial\Omega \times \mathbb{R}, \end{aligned} \tag{13.4.1}$$

on a bounded domain $\Omega \subset \mathbb{R}^2$, but for all times $t \in \mathbb{R}$.

We assume a time-periodic excitation

$$f(\boldsymbol{x},t) = \mathrm{Re}\{\hat{f}(\boldsymbol{x})\mathrm{e}^{i\omega t}\}, \tag{13.4.2}$$

with angular frequency $\omega > 0$ (Re designated the real part). The function $\hat{f} : \Omega \to \mathbb{C}$, $\hat{f} \in L^2(\Omega)$, is called the complex amplitude/phasor of f.

(13.4a) Show that

$$u(\boldsymbol{x},t) = \mathrm{Re}\{\hat{u}(\boldsymbol{x})\mathrm{e}^{i\omega t}\}, \tag{13.4.3}$$

solves the variational form of (13.4.1), if $\hat{u} \in H^1(\Omega)$ solves

$$\begin{aligned} -\omega^2 \hat{u}(\boldsymbol{x}) - \Delta \hat{u}((\boldsymbol{x}) &= \hat{f}(\boldsymbol{x}) \quad \text{in } \Omega, \\ \hat{u}(\boldsymbol{x}) &= 0, \qquad \text{on } \partial\Omega. \end{aligned} \tag{13.4.4}$$

Solution: First consider the variational form of (13.4.1)

$$u \in V(t): \quad \int_\Omega \frac{\partial^2 u(\boldsymbol{x},t)}{\partial t^2} v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int_\Omega \mathbf{grad}\, u(\boldsymbol{x},t) \cdot \mathbf{grad}\, v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \int_\Omega f(\boldsymbol{x},t)\, v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \quad \forall v \in H_0^1(\Omega). \tag{13.4.5}$$

Let $u(\boldsymbol{x},t) = \mathrm{Re}\{\hat{u}(\boldsymbol{x})\mathrm{e}^{i\omega t}\}$, replicing this above and by linearity of the derivative, we get

$$\begin{aligned} u \in V(t): \quad & \int_\Omega \mathrm{Re}\{-\omega^2 \hat{u}(\boldsymbol{x})\mathrm{e}^{i\omega t}\}\, v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \\ & + \int_\Omega \mathrm{Re}\{\mathbf{grad}\, \hat{u}(\boldsymbol{x})\mathrm{e}^{i\omega t}\} \cdot \mathbf{grad}\, v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \int_\Omega \mathrm{Re}\{\hat{f}(\boldsymbol{x})\mathrm{e}^{i\omega t}\}\, v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \quad \forall v \in H_0^1(\Omega). \end{aligned}$$

Applying integration by parts in the second term in the left hand side, and using linearity of real part, we obtain

$$u \in V(t): \quad \int_\Omega \mathrm{Re}\{\Big(-\omega^2 \hat{u}(\boldsymbol{x}) - \Delta \hat{u}(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\Big)\mathrm{e}^{i\omega t}\}\, v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0 \quad \forall v \in H_0^1(\Omega).$$

Now, if $\hat{u}(\boldsymbol{x})$ solves (13.4.4), it satisfies

$$\Big(-\omega^2 \hat{u}(\boldsymbol{x}) - \Delta \hat{u}(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\Big) = 0 \quad \text{in } \Omega, \tag{13.4.6}$$

From where we get the desired result.

(13.4b) In class we learned that the hyperbolic evolution governed by (13.4.1) involves an incessant conversion of elastic energy and kinetic energy given by

$$E_{el}(t) = \frac{1}{2} \int_\Omega \|\mathbf{grad}\, u(\boldsymbol{x},t)\|^2 \, \mathrm{d}\boldsymbol{x}, \tag{13.4.7}$$

$$E_{kin}(t) = \frac{1}{2} \int_\Omega \left| \frac{\partial u(\boldsymbol{x},t)}{\partial t} \right|^2 \mathrm{d}\boldsymbol{x} \tag{13.4.8}$$

Show directly, without appealing to [NPDE, Thm. 6.2.29], but using (13.4.7) and (13.4.8), that for $u = u(\boldsymbol{x},t)$ according to (13.4.3) and (13.4.4) the total energy is preserved.

Solution: Using the representation (13.4.3) in (13.4.7) and (13.4.8), we get

$$\begin{aligned} \frac{\partial E(t)}{\partial t} &= \frac{\partial (E_{el}(t) + E_{kin}(t))}{\partial t} \\ &= \frac{1}{2} \int_\Omega \big(2\,\mathrm{Re}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\mathrm{e}^{i\omega t}\} \cdot i\omega\, \mathrm{Re}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\mathrm{e}^{i\omega t}\} - 2i\omega\, \mathrm{Re}\{\hat{u}(\boldsymbol{x})\mathrm{e}^{i\omega t}\}\omega^2\, \mathrm{Re}\{\hat{u}(\boldsymbol{x})\mathrm{e}^{i\omega t}\}\big) \, \mathrm{d}\boldsymbol{x} \end{aligned}$$

Integrate by parts the first term and factorize to get (13.4.4)

$$\begin{aligned} \frac{\partial E(t)}{\partial t} &= \int_\Omega \mathrm{Re}\{\big(-\Delta \hat{u}(\boldsymbol{x}) - \omega^2 \hat{u}(\boldsymbol{x})\big)\mathrm{e}^{i\omega t}\} i\omega\, \mathrm{Re}\{\hat{u}(\boldsymbol{x})\mathrm{e}^{i\omega t}\} \, \mathrm{d}\boldsymbol{x} \\ &= \int_\Omega \mathrm{Re}\{\hat{f}(\boldsymbol{x})\mathrm{e}^{i\omega t}\} i\omega\, \mathrm{Re}\{\hat{u}(\boldsymbol{x})\mathrm{e}^{i\omega t}\} \, \mathrm{d}\boldsymbol{x}. \end{aligned}$$

which is 0 when $\hat{f}(\boldsymbol{x}) = 0$, so it preserves the energy.

(13.4c) Give a formula for the mean elastic and kinetic energy of u given by (13.4.3), that is, express

$$\hat{E}_{el} := \frac{1}{T}\int_0^T \frac{1}{2}\int_\Omega \|\mathbf{grad}\, u(\boldsymbol{x},t)\|^2 \,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t, \tag{13.4.9}$$

$$\hat{E}_{kin} := \frac{1}{T}\int_0^T \frac{1}{2}\int_\Omega \left|\frac{\partial u(\boldsymbol{x},t)}{\partial t}\right|^2 \mathrm{d}\boldsymbol{x}\,\mathrm{d}t, \tag{13.4.10}$$

in terms of suitable expressions for $\hat{u}$, where $T = \frac{2\pi}{\omega}$ is the duration of one period.

Solution: Use (13.4.3) and rewrite $\mathbf{grad}\, u(\boldsymbol{x},t)$ as

$$\mathrm{Re}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\mathrm{e}^{i\omega t}\} = \mathrm{Re}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\}\cos\omega t - \mathrm{Im}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\}\sin\omega t.$$

Plug this expression into (13.4.9), and interchange the order of integration to obtain

$$\begin{aligned}\hat{E}_{el} &= \frac{1}{2T}\int_\Omega\int_0^T \mathrm{Re}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\}^2\cos\omega t^2 - \mathrm{Re}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\}\,\mathrm{Im}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\}\cos\omega t\sin\omega t\,\mathrm{d}t\,\mathrm{d}\boldsymbol{x}\\ &\quad + \frac{1}{2T}\int_\Omega\int_0^T \mathrm{Im}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\}^2\sin\omega t^2\,\mathrm{d}t\,\mathrm{d}\boldsymbol{x}\\ &= \frac{1}{2T}\frac{\pi}{\omega}\int_\Omega \mathrm{Re}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\}^2 + \mathrm{Im}\{\mathbf{grad}(\hat{u}(\boldsymbol{x}))\}^2\,\mathrm{d}\boldsymbol{x} = \frac{1}{4}\int_\Omega |\mathbf{grad}(\hat{u}(\boldsymbol{x}))|^2\,\mathrm{d}\boldsymbol{x}.\end{aligned}$$

Following the same idea, plug

$$\mathrm{Re}\{(\hat{u}(\boldsymbol{x}))\mathrm{e}^{i\omega t}\} = \mathrm{Re}\{(\hat{u}(\boldsymbol{x}))\}\cos\omega t - \mathrm{Im}\{(\hat{u}(\boldsymbol{x}))\}\sin\omega t$$

into $\hat{E}_{kin}$ and get

$$\begin{aligned}\hat{E}_{kin} &= \frac{1}{2T}\int_0^T\int_\Omega \left|\frac{\partial(\mathrm{Re}\{(\hat{u}(\boldsymbol{x}))\}\cos\omega t - \mathrm{Im}\{(\hat{u}(\boldsymbol{x}))\}\sin\omega t)}{\partial t}\right|^2 \mathrm{d}\boldsymbol{x}\,\mathrm{d}t\\ &= \frac{1}{2T}\int_0^T\int_\Omega |-\omega\,\mathrm{Re}\{(\hat{u}(\boldsymbol{x}))\}\sin\omega t - \omega\,\mathrm{Im}\{(\hat{u}(\boldsymbol{x}))\}\cos\omega t|^2\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t.\end{aligned}$$

As above, when interchanging order of integration, we get

$$\hat{E}_{kin} = \frac{1}{2T}\frac{\omega^2\pi}{\omega}\int_\Omega \mathrm{Re}\{(\hat{u}(\boldsymbol{x}))\}^2 + \mathrm{Im}\{(\hat{u}(\boldsymbol{x}))\}^2\,\mathrm{d}\boldsymbol{x} = \frac{\omega^2}{4}\int_\Omega |(\hat{u}(\boldsymbol{x}))|^2\,\mathrm{d}\boldsymbol{x}$$

(13.4d) For $\Omega = \{x \in \mathbb{R}^2 \,:\, \|x\| < 1\}$ and

$$\hat{f}(\boldsymbol{x}) = \begin{cases}\cos^2\left(2\pi\left\|\boldsymbol{x} - \binom{0.5}{0.5}\right\|\right) & \text{, if } \left\|\boldsymbol{x} - \binom{0.5}{0.5}\right\| < \frac{1}{4},\\ 0 & \text{elsewhere.}\end{cases}$$

Implement a `C++` code that solves (13.4.4) for $\omega = 10$ using piecewise linear Lagrangian FE.

HINT: As usual, you are required to use your previous implementations of `DofHandler`, `MatrixAssembler`, `VectorAssembler`, `BoundaryDofs`, and local assemblers, developed in subproblems 7.4, 8.1, and 8.2 (also available in the corresponding solution folders).

Solution: See Listing 13.5 for the code.

Listing 13.3: Implementation of `main.cc` with dirichlet b.c.

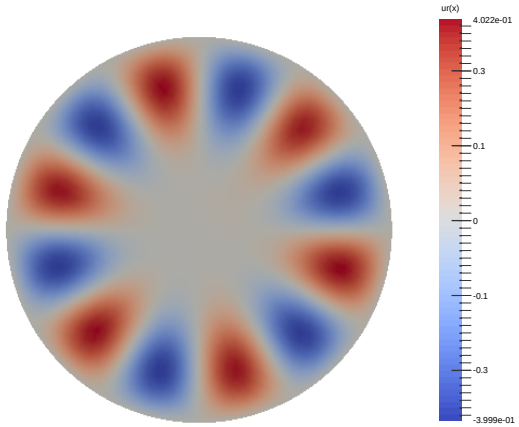
```
const int world_dim = 2;

using namespace std;

using calc_t = double;
using complex_t = complex<calc_t>;

using Matrix = Eigen::SparseMatrix<std::complex<calc_t>, Eigen::RowMajor>;
using Vector = Eigen::VectorXcd;
using IndexVector = vector<bool>;
using GridType = Dune::ALUSimplexGrid<2, 2>;
using GridView = GridType::LeafGridView;
using Coordinate  = Dune::FieldVector<calc_t, world_dim>;
using DofHandler  = NPDE15::LDofHandler<GridView>;

int main(int argc, char *argv[]){
  try{
    // load the grid from file
    std::string fileName = "../_input_meshes/circle_3472.msh";

    // Declare and create mesh using the Gmsh file
    Dune::GridFactory<GridType> gridFactory;
    Dune::GmshReader<GridType>::read(gridFactory, fileName.c_str(), false,
        true);
    std::unique_ptr<GridType> workingGrid(gridFactory.createGrid());
    //workingGrid->globalRefine(3);
    workingGrid->loadBalance();

    // Get the Gridview
    GridView gv = workingGrid->leafGridView();

    // Initialize dof-handler
    DofHandler dofh(gv);

    unsigned N = dofh.size();
    std::cout << "Solving for N =" << N << " unknowns.\n";

    // Load vector
    auto f = [](Coordinate const& x){
      calc_t x1 = x[0]-0.5;
      calc_t x2 = x[1]-0.5;
      calc_t norm = sqrt(x1*x1+x2*x2);
      if (norm<0.25)
     return cos(4*norm)*cos(4*norm);
      return 0.;
    };

    // Get boundary nodes
    IndexVector dirichlet_dofs(N);
    NPDE15::LBoundaryDofs<DofHandler> get_bnd_dofs(dofh);
    get_bnd_dofs(dirichlet_dofs);
    dofh.set_inactive(dirichlet_dofs);

 // assemble rhs and set dirichlet dofs to dirichlet data
    Vector Phi(N); Phi.setZero();
```

```
    NPDE15::VectorAssembler<DofHandler> vecAssembler(dofh);
    vecAssembler(Phi, NPDE15::LLocalFunction(f));
    // Homogeneous dirichlet data
    Vector G(N); G.setZero();
    vecAssembler.set_inactive(Phi, G);

    // Reaction function
    double omega = 10;
    auto c = [&omega](Coordinate const& x){return -omega*omega; };
    auto cb = [&omega](Coordinate const& x){return -complex_t(0,omega); };

    // assemble the system matrix
    std::vector<Eigen::Triplet<std::complex<calc_t>>> triplets;
    NPDE15::MatrixAssembler<DofHandler> matAssembler(dofh);
    matAssembler(triplets, NPDE15::AnalyticalLocalLaplace());
    matAssembler(triplets, NPDE15::LocalMass<complex_t>(c));
    matAssembler.set_inactive(triplets);

    Matrix A(N, N);
    A.setFromTriplets(triplets.begin(), triplets.end());
    A.makeCompressed();

    // solution vector U
    Vector U(N); U.setZero();

    // solve the system
    U = Phi/A; // short-hand, see Pardiso.hpp for more information

    // plot absolute value of the solution:
    vector<double> U_re(N), U_im(N), F_re(N), F_im(N);
    for (unsigned i=0;i<N;++i){
    U_re[i]=real(U[i]);
    U_im[i]=imag(U[i]);
    F_re[i]=real(Phi[i]);
    F_im[i]=imag(Phi[i]);
    }

    cout << "\n\nWriting solution to vtk file... ";
    Dune::VTKWriter<GridView> vtkwriter(gv);
    stringstream name;
    name << "solution_dirichlet";
    vtkwriter.addVertexData(U_re, "ur(x)");
    vtkwriter.addVertexData(U_im, "ui(x)");
    vtkwriter.addVertexData(F_re, "fr(x)");
    vtkwriter.addVertexData(F_im, "fi(x)");
    vtkwriter.write(name.str().c_str());
    cout << "Done.\n";

  }
  // catch exceptions
```

Listing 13.4: Implementation of `LocalMass_()`

```
#ifndef LOCALMASS_HPP_
#define LOCALMASS_HPP_

#include <dune/localfunctions/lagrange/pk.hh>
```

```
#include <dune/geometry/quadraturerules.hh>
#include <dune/common/fmatrix.hh>
#include <dune/istl/matrix.hh>

namespace NPDE15{

  template <class Function, class MatrixCalcType>
  class LocalMassC{
  public:
    using calc_t = double;
    using ElementMatrix = Dune::FieldMatrix<MatrixCalcType,3,3>;
    /*!   \brief Constructor
      \param[in] c Coefficient function used during assembly
    */
    LocalMassC(Function const& c) : c_(c) {};

    /*!   \brief Local Assembler
      \param[in] e Entity to integrate over
      \param[out] local Local contribution matrix
    */
    template <class Element>
    void operator()(Element const& e, ElementMatrix &locMassMat) const{
      const int world_dim = Element::dimension;
      const int elem_dim  = Element::mydimension;
      typedef typename Dune::QuadratureRule<calc_t, elem_dim> QuadRule_t;
      typedef typename Dune::QuadratureRules<calc_t, elem_dim> QuadRules;
      const QuadRule_t & quadRule = QuadRules::rule(e.type(), 3);
      Dune::PkLocalFiniteElement<calc_t,calc_t,elem_dim, 1> localFE;
      assert(localFE.type()==e.type());

      unsigned M=localFE.localBasis().size();
      //locMassMat.setSize(M,M);
      locMassMat=0;
      auto const& egeom = e.geometry();
      for (auto qr : quadRule){
     // get quadrature point in the reference element
     auto const& qp_local_pos = qr.position();
     double  const& w = qr.weight();
     // evaluate shape function values (locally)
     std::vector<Dune::FieldVector<calc_t,1> > shapef_val;
     localFE.localBasis().evaluateFunction(qp_local_pos, shapef_val);
     // evaluate coefficient function (globally!)  at the current
        quadrature position
     auto coeff=c_(egeom.global(qp_local_pos));
     // determinant of transformation from reference element
     double jac_det = egeom.integrationElement(qp_local_pos);
     // add to local contribution matrix
     for (unsigned i=0;i<M;++i){
       for (unsigned j=0;j<M;++j)
         locMassMat[i][j]+= coeff*(shapef_val[i]*shapef_val[j]*w*jac_det);
     }
      } // end for loop quadRule

    }
  private:
    Function const& c_;
  };
```



(a) $\omega = 10$

Figure 13.3: Plot for subproblem (13.4d).

```
  // template deduction helper
  template <class MatrixCalcType, class Function>
  LocalMassC<Function, MatrixCalcType> LocalMass(Function const& q){
    return LocalMassC<Function, MatrixCalcType>(q);
    }

}
#endif
```

Now we reduce the homogeneous boundary condition in (13.4.1) with a special boundary condition, the so-called first order absorbing boundary condition

$$\mathbf{grad}\, u(\boldsymbol{x},t)\cdot \mathbf{n}(\boldsymbol{x}) + \frac{\partial u}{\partial t}(\boldsymbol{x},t) = 0, \quad \text{on } \partial\Omega\times\mathbb{R}. \tag{13.4.11}$$

(13.4e) Find a boundary condition for $\hat{u}$ in (13.4.4) such that, again, $u = u(\boldsymbol{x},t)$ given by (13.4.3) solves the wave equation from (13.4.1) and satisfies (13.4.12).

Solution: By replacing $u(\boldsymbol{x},t) = \mathrm{Re}\{\hat{u}(\boldsymbol{x})\mathrm{e}^{i\omega t}\}$ and following the same arguments as before, we find that if $\hat{u}$ in (13.4.4) also satisties

$$\mathbf{grad}\, \hat{u}(\boldsymbol{x})\cdot \mathbf{n}(\boldsymbol{x}) - i\omega\hat{u}(\boldsymbol{x}) = 0, \quad \text{on } \partial\Omega \tag{13.4.12}$$

then $u(\boldsymbol{x},t)$ solves (13.4.1) and satisfies (13.4.12).

(13.4f) Modify your C++ code so that it can handle the boundary conditions from subproblem (13.4e). Repeat the numerical experiment from subproblem (13.4d).

HINT: Observe here we are using complex numbers. This means you should be careful when declaring the triplets type and slightly modify your implementation of `LocalMass`. You may also use the templated class available in the lecture `svn` repository

`assignments_codes/assignment13/Problem4`

Solution: See Listing 13.5 for the code.

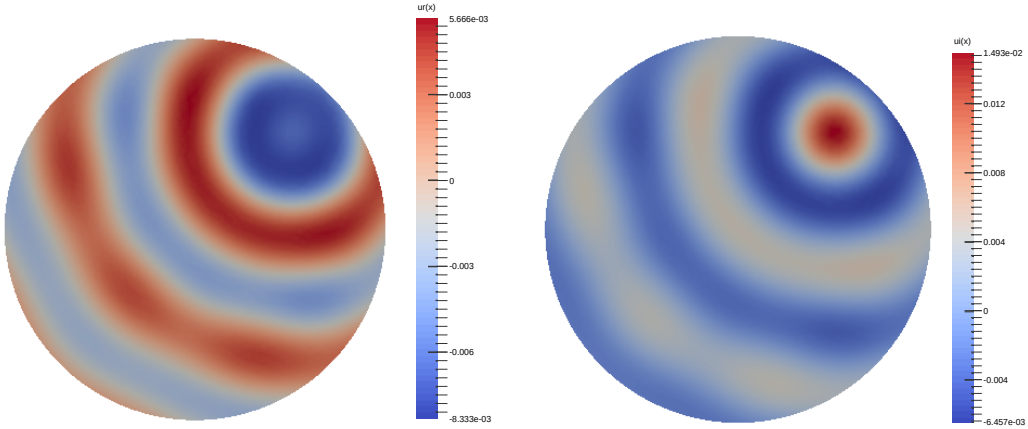
Listing 13.5: Implementation of `main.cc` with robin b.c.

```
const int world_dim = 2;

using namespace std;

using calc_t = double;
using complex_t = complex<calc_t>;

using Matrix = Eigen::SparseMatrix<std::complex<calc_t>, Eigen::RowMajor>;
using Vector = Eigen::VectorXcd;
using IndexVector = vector<bool>;
using GridType = Dune::ALUSimplexGrid<2, 2>;
using GridView = GridType::LeafGridView;
using Coordinate  = Dune::FieldVector<calc_t, world_dim>;
using DofHandler  = NPDE15::LDofHandler<GridView>;

int main(int argc, char *argv[]){
  try{
    // load the grid from file
    std::string fileName = "../_input_meshes/circle_3472.msh";

    // Declare and create mesh using the Gmsh file
    Dune::GridFactory<GridType> gridFactory;
    Dune::GmshReader<GridType>::read(gridFactory, fileName.c_str(), false,
        true);
    std::unique_ptr<GridType> workingGrid(gridFactory.createGrid());
    //workingGrid->globalRefine(3);
    workingGrid->loadBalance();

    // Get the Gridview
    GridView gv = workingGrid->leafGridView();

    // Initialize dof-handler
    DofHandler dofh(gv);

    unsigned N = dofh.size();
    std::cout << "Solving for N =" << N << " unknowns.\n";

    // Load vector
    auto f = [](Coordinate const& x){
      calc_t x1 = x[0]-0.5;
      calc_t x2 = x[1]-0.5;
      calc_t norm = sqrt(x1*x1+x2*x2);
      if (norm<0.25)
     return cos(4*norm)*cos(4*norm);
      return 0.;
    };

    // Get boundary nodes
    IndexVector robin_dofs(N);
    NPDE15::LBoundaryDofs<DofHandler> get_bnd_dofs(dofh);
    get_bnd_dofs(robin_dofs);

  // assemble rhs and set dirichlet dofs to dirichlet data
```

```
    Vector Phi(N); Phi.setZero();
    NPDE15::VectorAssembler<DofHandler> vecAssembler(dofh);
    vecAssembler(Phi, NPDE15::LLocalFunction(f));

    // Reaction function
    double omega = 10;
    auto c = [&omega](Coordinate const& x){return -omega*omega; };
    auto cb = [&omega](Coordinate const& x){return -complex_t(0,omega); };

    // assemble the system matrix
    std::vector<Eigen::Triplet<std::complex<calc_t>>> triplets;
    NPDE15::MatrixAssembler<DofHandler> matAssembler(dofh);
    matAssembler(triplets, NPDE15::AnalyticalLocalLaplace());
    matAssembler(triplets, NPDE15::LocalMass<complex_t>(c));
    matAssembler(triplets, NPDE15::LocalMass<complex_t>(cb), robin_dofs);

    Matrix A(N, N);
    A.setFromTriplets(triplets.begin(), triplets.end());
    A.makeCompressed();

    // solution vector U
    Vector U(N); U.setZero();

    // solve the system
    U = Phi/A; // short-hand, see Pardiso.hpp for more information

    // plot absolute value of the solution:
    vector<double> U_re(N);
    vector<double> U_im(N);
    vector<double> F_re(N);
    vector<double> F_im(N);
    for (unsigned i=0;i<N;++i){
    U_re[i]=real(U[i]);
    U_im[i]=imag(U[i]);
    F_re[i]=real(Phi[i]);
    F_im[i]=imag(Phi[i]);
    }

    cout << "\n\nWriting solution to vtk file... ";
    Dune::VTKWriter<GridView> vtkwriter(gv);
    stringstream name;
    name << "solution";
    vtkwriter.addVertexData(U_re, "ur(x)");
    vtkwriter.addVertexData(U_im, "ui(x)");
    vtkwriter.addVertexData(F_re, "fr(x)");
    vtkwriter.addVertexData(F_im, "fi(x)");
    vtkwriter.write(name.str().c_str());
    cout << "Done.\n";

  }
  // catch exceptions
```

Published on 20.05.2015.

To be submitted on 27.05.2015.



(a) $\operatorname{Re} u(x)$ (b) $\operatorname{Im} u(x)$

Figure 13.4: Plots for subproblem (13.4f).

References

[NPDE] Lecture Slides for the course "Numerical Methods for Partial Differential Equations".SVN revision # 79326.

[NCSE] Lecture Slides for the course "Numerical Methods for CSE".

[LehrFEM] LehrFEM manual.

Last modified on July 15, 2015