R. Hiptmair
L. Scarabosio
C. Urzua Torres

Spring Term 2015

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 12

Problem 12.1 Radau-3 Timestepping (Core problem)

This short problem studies the full discretization of a linear second-order parabolic initial-boundary value problem as discussed in [NPDE, Section 6.1.1]. The focus will be on the implementation of higher-order implicit Runge-Kutta timestepping.

We consider the parabolic IBVP with homogeneous Dirichlet boundary conditions and zero initial conditions

$$\begin{aligned} \dot{u} - \Delta u &= f(\boldsymbol{x},t) && \text{in } \Omega \times (0,1), \\ u &= 0 && \text{on } \partial\Omega \times (0,1), \\ u &= 0 && \text{on } \Omega \times \{0\}, \end{aligned} \tag{12.1.1}$$

where the spatial domain Ω is the unit disk

$$\Omega := \{\boldsymbol{x} \in \mathbb{R}^2,\ \|\boldsymbol{x}\| < 1\}.$$

The time-dependent source function is given by

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

We pursue the method of lines approach, see [NPDE, Section 6.1.4]. Spatial discretization relies on linear Lagrangian finite elements on a triangular mesh $\mathcal{M}$, that is, $V_{0,N} = \mathcal{S}^0_{1,0}(\mathcal{M})$, using a polygonal approximation of $\partial\Omega$.

When higher order timestepping schemes are desired for the initial value problem arising from the method of lines approach, L(π)-stable implicit Runge-Kutta methods ($\rightarrow$ [NPDE, Def. 6.1.39]) are the methods of choice. One example is the Radau-3 scheme, a 2-stage implicit Runge-Kutta method, whose description via a Butcher scheme is given in [NPDE, Eq. (6.1.86)].

(12.1a) Devise and implement (in C++ or MATLAB) a numerical experiment for determining the order of (algebraic) convergence of the Radau-3 method, when applied to the initial value problem for a scalar ODE $\dot{y} = -y$, $y(0) = 1$, on $[0,2]$. Which order do you find?

HINT: [NPDE, § 1.6.26] discussed "measurements" for rates of convergence. Measure the error $\max_k |y(t_k) - y^{(k)}|$ for various timestep sizes $\tau > 0$ (equidistant timesteps).

Solution: Here $\dot{y} = -y$. Applying the Radau scheme, we get

$$\kappa_1 = -y - \tau\frac{5}{12}\kappa_1 + \tau\frac{1}{12}\kappa_2 \tag{12.1.2}$$

$$\kappa_2 = -y - \tau\frac{3}{4}\kappa_1 - \tau\frac{1}{4}\kappa_2 \tag{12.1.3}$$

solving this system, we get

$$\kappa_1 = \frac{-(1+\frac{1}{3}\tau)y}{(1+\tau\frac{5}{12})(1+\tau\frac{1}{4})+\frac{\tau^2}{16}}, \qquad \kappa_2 = \frac{-(1-\frac{1}{3}\tau)y}{(1+\tau\frac{5}{12})(1+\tau\frac{1}{4})+\frac{\tau^2}{16}} \tag{12.1.4}$$

Finally, updating the solution, we get

$$Y^{j+1} = Y^j + \tau\frac{3}{4}\kappa_1 + \tau\frac{1}{4}\kappa_2 \tag{12.1.5}$$

$$= Y^j\left(1 - \frac{\tau(1+\frac{\tau}{6})}{(1+\tau\frac{5}{12})(1+\tau\frac{1}{4})+\frac{\tau^2}{16}}\right) \tag{12.1.6}$$

As expected, 3rd order convergence is achieved. See Listing 12.1 for details.

Listing 12.1: Implementation of convergence

```
16   double tau = 1./N;
17   Vector Y(N+1), Yex(N+1);
18   Y[0] = 1; Yex[0] = 1;
19
20   for(int i = 1; i <= N; i++){
21     Y[i] = (1 - (tau*((tau/6. + 1)/(((1 + 5*tau/12.)*(1+tau/4.)) +
           tau*tau/16. ))))*Y[i-1];
22     Yex[i] = Yex[0]*exp(-tau*i);
23   }
24   err  = (Y -Yex).lpNorm<Eigen::Infinity>();
25 }
26
27 int main(int argc, char *argv[]){
28   Vector err(7), err2(7);
29   Vector ConvRate(7);
30   for(int k = 0; k < 7; k++){
31     int N = 20 * pow(2,k);
32     Radau_Scalar(N, err[k], err2[k]);
33     if(k>0){
34       ConvRate[k-1] = log(err[k]/err[k-1])/log(2.0);
35     }
36   }
37     std::cout << "Done.\n";
38     std::cout << ConvRate ;
39
40   return 0;
41 }
```

(12.1b) The application of s-stage Runge-Kutta single step methods according to [NPDE, Def. 6.1.39] entails the computation of s increments in each timestep. For the linear ordinary differential equation [NPDE, Eq. (6.1.29)] arising from the method of lines approach to a linear parabolic evolution problem the increments can be obtained by solving a linear system of equations, see [NPDE, § 6.1.42] and [NPDE, Eq. (6.1.43)].

Describe this linear system of equations (in block form) for the RADAU-3 method for (12.1.1) using the notations $\mathbf{A}$ and $\mathbf{M}$ for the $N \times N$ finite element Galerkin matrices associated with the bilinear form for $-\Delta$ and the $L^2(\Omega)$-inner product, respectively. Write $\vec{\varphi}(t)$ for the time-dependent right hand side vector obtained by Galerkin discretization of the source term.

Solution: From [NPDE, § 6.1.42] we know

$$\mathbf{M}\vec{\kappa}_1 + \tau\frac{5}{12}\mathbf{A}\vec{\kappa}_1 + \tau\frac{-1}{12}\mathbf{A}\vec{\kappa}_2 = \vec{\varphi}(t_j + \frac{1}{3}\tau) - \mathbf{A}\vec{\mu}^{(j)} \tag{12.1.7}$$

$$\mathbf{M}\vec{\kappa}_2 + \tau\frac{3}{4}\mathbf{A}\vec{\kappa}_1 + \tau\frac{1}{4}\mathbf{A}\vec{\kappa}_2 = \vec{\varphi}(t_j + \tau) - \mathbf{A}\vec{\mu}^{(j)} \tag{12.1.8}$$

which gives rise to

$$\begin{bmatrix} \mathbf{M} + \tau\frac{5}{12}\mathbf{A} & \tau\frac{-1}{12}\mathbf{A} \\ \tau\frac{3}{4}\mathbf{A} & \mathbf{M} + \tau\frac{1}{4}\mathbf{A} \end{bmatrix} \begin{pmatrix} \vec{\kappa}_1 \\ \vec{\kappa}_2 \end{pmatrix} = \begin{pmatrix} \vec{\varphi}(t_j + \frac{1}{3}\tau) - \mathbf{A}\vec{\mu}^{(j)} \\ \vec{\varphi}(t_j + \tau) - \mathbf{A}\vec{\mu}^{(j)} \end{pmatrix}$$

(12.1c) Complete the class Radau3 provided in the `svn` repository, by implementing the method

```
void computeSystemMatrix_(Matrix & MK, Matrix & A, double dt)
```

which takes as argument the timestep `dt`, computes the required triplets, and finally fills the coefficient matrix `MK`$\in \mathbb{R}^{2N,2N}$ for the linear system supplying the increments, see subproblem (12.1b). The argument `A` returns the stiffness Matrix $\mathbf{A} \in \mathbb{R}^{N,N}$, $N := \dim \mathcal{S}_1^0(\mathcal{M})$, for $-\Delta$.

HINT: First get the triplets for the mass matrix and `S`, then build the system block matrix `MK` on the triplets level by using an offset. The use of index-value triplets for the initialization of sparse matrices in the C++ template library Eigen is explained in [NPDE, Rem. 3.6.45]. [NPDE, Code 3.6.49] provides an example for the use of triplets.

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). You will find the new required files for this problem in

```
assignments_codes/assignment12/Problem1
```

Solution: See Listing 12.2 for the code.

Listing 12.2: Implementation of `computeSystemMatrix_()`

```
111 template <class DofHandler>
112 void Radau3<DofHandler>::computeSystemMatrix_(Matrix & MK, Matrix & A,
        double dt) {
113   // Assemble 2n*2n time stepping matrix
114   NPDE15::MatrixAssembler<DofHandler> matAssembler(dofh);
115   // compute triplets
116   Triplets A_trp, M_trp, triplets;
117   matAssembler(A_trp, NPDE15::AnalyticalLocalLaplace());
118   matAssembler.set_inactive(A_trp);
119   matAssembler(M_trp, NPDE15::AnalyticalLocalMass());
120   matAssembler.set_inactive(M_trp);
121
122   // add stiffness entries to time stepping matrix (blockwise using
        offset = N )
123   for (auto &a : A_trp){
124     double val = dt*a.value();
125     triplets.push_back(Triplet(a.row()   , a.col()   , 5./12.*val) );
          //top-left
          submatrix
```

```
126         triplets.push_back(Triplet(a.row()   , a.col()+N, -1./12.*val));
              //top-right
              submatrix
127         triplets.push_back(Triplet(a.row()+N, a.col()   , 0.75*val)   );
              //bottom-left
              submatrix
128         triplets.push_back(Triplet(a.row()+N, a.col()+N, 0.25*val)   );
              //bottom-right
              submatrix
129       }
130       // and add mass entries ...
131       for (auto &m : M_trp){
132         triplets.push_back(Triplet(m.row(),  m.col(),  m.value())); //
              top-left submatrix
133         triplets.push_back(Triplet(m.row()+N,m.col()+N,m.value())); //
              bottom-right submatrix
134       }
135       // make system matrix from triplets
136       MK.setFromTriplets(triplets.begin(), triplets.end());
137       MK.makeCompressed();
138       // make auxiliary stiffness matrix
139       A.setFromTriplets(A_trp.begin(),A_trp.end());
140       A.makeCompressed();
141   }
```

(12.1d) Now, inside the class Radau3, implement the method

```
template <class Function>
void solve(Function const& f, double dt);
```

for solving (12.1.1) approximately based on the spatial and temporal discretizations as described above. The argument f gives the load function in procedural form (where $f = f(x,t)$), and `dt` specifies the (fixed) timestep.

HINT: Remember `LocalFunction` takes as an argument a load function depending only on `x`. You may use a `lambda` function to convert f to a function depending only on x (for a fixed t).

Solution: See Listing 12.3 for the code.

Listing 12.3: Implementation of `solve()`

```
53  template <class DofHandler>
54  template <class Function>
55  void Radau3<DofHandler>::solve(Function const& f, double dt){
56
57   // Get boundary nodes
58       IndexVector dirichlet_dofs(N);
59       NPDE15::LBoundaryDofs<DofHandler> get_bnd_nodes(dofh);
60       get_bnd_nodes(dirichlet_dofs);
61       dofh.set_inactive(dirichlet_dofs);
62
63       // Get timestepping and auxiliary matrix
64       Matrix MK(2*N, 2*N);
65       Matrix A(N,N);
66       computeSystemMatrix_(MK, A, dt);
67
68       // solution vector u, zero initial conditions
69       Vector U(N); U.setZero();
70       // increment (calculated at each time step)
```

```
71      Vector k(2*N); k.setZero();
72
73      unsigned timeiter = 1;
74
75      calc_t t = 0.0;
76      auto fx = [&t, &f](Coordinate const& x) { return f(x,t); };
77
78      NPDE15::VectorAssembler<DofHandler> vecAssembler(dofh);
79      for (calc_t time = 0; time <= 1; time += dt){
80        Vector L1(N),L2(N); L1.setZero();L2.setZero();
81        Vector G(N); G.setZero();
82        t = time + dt/3.;
83        vecAssembler(L1, NPDE15::LLocalFunction(fx));
84        vecAssembler.set_inactive(L1, G);
85        t = time + dt;
86        vecAssembler(L2, NPDE15::LLocalFunction(fx));
87        vecAssembler.set_inactive(L2, G);
88        // A*u vector
89        Vector Au = A*U;
90        // concatenate vectors (2N-sized r.h.s.  vector)
91        Vector L(2*N);
92        L << L1 - Au, L2 - Au;
93
94        // solve for increment
95        k = L/MK;
96        // apply update
97        U += dt*(0.75*k.head(N)+0.25*k.tail(N));
98
99        if(timeiter==floor(1./dt)){
100     Dune::VTKWriter<GridView> vtkwriter(gv);
101     std::stringstream outputName;
102     outputName << "solution";
103     vtkwriter.addVertexData(U, "u_app(x)");
104     vtkwriter.write(outputName.str().c_str());
105      }
106      std::cout << "Time Iteration #" << timeiter++ << "\n";
107     }
```

(12.1e) Complete `main.cc` To solve and plot the approximate solution of (12.1.1) at final time $T = 1$ obtained by the method from subproblem (12.1c) using the mesh `circle_320.msh` and the timestep $\tau = 0.02$. Plot the obtained solution using `Paraview`.

HINT: For validation purposes you may compare your plot with that provided on the lecture's `svn` repository.

Solution: Include lines 99 to 105 in Listing 12.3 and write main to call `Radau3::solve()`, see Listing 12.4 for details.

Listing 12.4: Implementation of `main.cpp`

```
28  int main(int argc, char *argv[]){
29    try{
30      // load the grid from file
31      std::string FileName = "circle_320.msh";
32
33      // Declare and create mesh using the Gmsh file
```

```
34      Dune::GridFactory<GridType> gridFactory;
35      Dune::GmshReader<GridType>::read(gridFactory, FileName.c_str(), false,
            true);
36      std::unique_ptr<GridType> workingGrid(gridFactory.createGrid());
37      workingGrid->loadBalance();
38
39      // Get the Gridview
40      GridView gv = workingGrid->leafGridView();
41
42      // Initialize dof-handler
43      DofHandler dofh(gv);
44
45      unsigned N = dofh.size();
46      std::cout << "Solving for N =" << N << " unknowns.\n";
47
48      // Load function
49      auto f = [](Coordinate const& x, calc_t t){
50        calc_t x1 = x[0]-0.5*cos(M_PI*t);
51        calc_t x2 = x[1]-0.5*sin(M_PI*t);
52        calc_t norm = sqrt(x1*x1+x2*x2);
53        if (norm<0.5)
54       return 1.;
55        return 0.;
56      };
57
58      Radau3<DofHandler> radau_evolution(dofh);
59      radau_evolution.solve(f, 0.02);
60      std::cout << "Done.\n";
61    }
62    // catch exceptions
63    catch (Dune::Exception &e){
64      std::cerr << "Dune reported error: " << e << std::endl;
65    }
66    catch (...)
67      std::cerr << "Unknown exception thrown!" << std::endl;
68    }
69    return 0;
70  }
```

The output should look like the figure below.

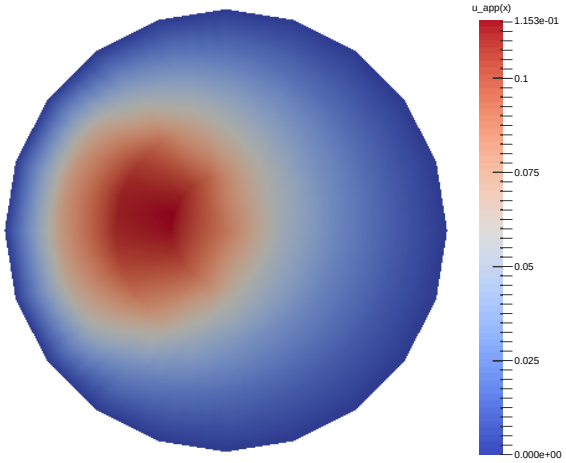


Figure 12.1: Solution for subproblem (12.1d)

Problem 12.2 Decaying Solution by Implicit Euler Timestepping (Core problem)

In this problem we practice the diagonalization technique that was a key tool in the stability analysis of single step methods for semi-discrete parabolic evolution problems, see the presentation in [NPDE, Section 6.1.5.2] and [NPDE, Eq. (6.1.61)].

In class we learned that solutions of the abstract variational linear parabolic evolution problem

$$\begin{aligned} \mathsf{m}(\dot{u},v) + \mathsf{a}(u,v) = 0 \quad \forall v \in V_0, \quad 0 < t < T, \\ u(0) = u_0 \in V_0, \end{aligned} \tag{12.2.1}$$

where both m and a are symmetric positive definite bilinear forms on V_0 that satisfy, see [NPDE, Eq. (6.1.20)],

$$\exists \gamma > 0 : \quad \mathsf{a}(v,v) \geq \gamma \mathsf{m}(v,v) \quad \forall v \in V_0, \tag{12.2.2}$$

display an exponential decay of their m-norm and energy norm, see [NPDE, Lemma 6.1.22].

A simple L(π)-stable [NPDE, Def. 6.1.84] implicit timestepping scheme for the semi-discrete linear parabolic evolution problem

$$\begin{aligned} \mathbf{M}\left\{\frac{\mathrm{d}}{\mathrm{d}t}\vec{\boldsymbol{\mu}}(t)\right\} + \mathbf{A}\vec{\boldsymbol{\mu}} = 0, \quad 0 < t < T, \\ \vec{\boldsymbol{\mu}}(0) = \vec{\boldsymbol{\mu}}_0, \end{aligned} \tag{12.2.3}$$

arising from the *method of lines* ($\rightarrow$ [NPDE, Section 6.1.4]) is the implicit Euler method, see [NPDE, Eq. (6.1.36)]. Here, $\mathbf{M}$ and $\mathbf{A}$ denote the Galerkin matrices ($\rightarrow$ [NPDE, Section 3.2]) associated with m and a w.r.t. an ordered basis of a finite-dimensional trial and test space $V_{0,N} \subset V_0$.

The question is, to what extent the sequence $\vec{\boldsymbol{\mu}}^{(j)}$ generated by the implicit Euler method inherits the exponential decay of the norms stated in [NPDE, Lemma 6.1.22].

(12.2a) Write down the formula for a step of the implicit Euler method producing $\vec{\mu}^{(j)}$ from $\vec{\mu}^{(j-1)}$. Write $\tau > 0$ for the size of the timestep.

Solution:

$$\vec{\mu}^{(j)} = (\tau\mathbf{A} + \mathbf{M})^{-1}(\mathbf{M}\vec{\mu}^{(j-1)}) \quad \text{or equivalently} \quad \mathbf{M}(\vec{\mu}^{(j)} - \vec{\mu}^{(j-1)}) = -\tau\mathbf{A}\vec{\mu}^{(j)}$$

(12.2b) From the generalized eigenvalue problem $\mathbf{A}\vec{\mu} = \lambda\mathbf{M}\vec{\mu}$ we deduced the existence of a regular matrix $\mathbf{T} \in \mathbb{R}^{N\times N}$ such that (*cf.* the diagonalization technique discussed in [NPDE, Section 6.1.5.2], and, in particular [NPDE, Eq. (6.1.58)], [NPDE, Suppl. 6.1.59])

$$\mathbf{AT} = \mathbf{MTD}, \quad \mathbf{D} := \operatorname{diag}(\lambda_1, \ldots, \lambda_N)\,, \quad \mathbf{T}^\top\mathbf{MT} = \mathbf{I}\,.$$

Rewrite the implicit Euler step from subproblem (12.2a) in terms of the transformed coefficient vectors $\vec{\eta}^{(k)} := \mathbf{T}^\top\mathbf{M}\vec{\mu}^{(k)}$, $k = j-1, j$.

Solution: We multiply both sides of the equation above by $\mathbf{T}^\top$. This results in:

$$\mathbf{T}^\top\mathbf{M}(\vec{\mu}^{(j)} - \vec{\mu}^{(j-1)}) = -\mathbf{T}^\top\tau\mathbf{A}\vec{\mu}^{(j)},$$

we can rewrite it as

$$\mathbf{T}^\top\mathbf{M}(\vec{\mu}^{(j)} - \vec{\mu}^{(j-1)}) = -\tau\mathbf{T}^\top\mathbf{A}\mathbf{M}^{-1}\mathbf{T}^{-\top}\mathbf{T}^\top\mathbf{M}\vec{\mu}^{(j)}$$

Therefore, denoting $\vec{\eta}^{(j)} = \mathbf{T}^\top\mathbf{M}\vec{\mu}^{(j)}$ this becomes

$$\vec{\eta}^{(j)} - \vec{\eta}^{(j-1)} = -\tau\mathbf{T}^\top\mathbf{A}\mathbf{M}^{-1}\mathbf{T}^{-\top}\vec{\eta}^{(j)}$$

Finally, observe $\mathbf{T}^{-1}\mathbf{M}^{-1}\mathbf{T}^{-\top} = \mathbf{I}$ and obtain

$$\vec{\eta}^{(j)} - \vec{\eta}^{(j-1)} = -\tau\mathbf{T}^\top\mathbf{A}\mathbf{T}\vec{\eta}^{(j)} = -\tau\mathbf{D}\vec{\eta}^{(j)}$$

(12.2c) What will the norms $\left\|\vec{\xi}\right\|_{\mathbf{M}} := (\vec{\xi}^\top\mathbf{M}\vec{\xi})^{\frac{1}{2}}$ and $\left\|\vec{\xi}\right\|_{\mathbf{A}} := (\vec{\xi}^\top\mathbf{A}\vec{\xi})^{\frac{1}{2}}$ of a coefficient vector $\vec{\mu}$ become for the transformed vector $\vec{\eta} := \mathbf{T}^\top\mathbf{M}\vec{\mu}$?

Solution: First, since $\vec{\eta} = \mathbf{T}^\top\mathbf{M}\vec{\mu}$, we have $\vec{\mu} = \mathbf{M}^{-1}(\mathbf{T})^{-\top}\vec{\eta}$, so

$$\begin{aligned}\|\vec{\mu}\|_{\mathbf{M}} &= (\vec{\mu}^\top\mathbf{M}\vec{\mu})^{1/2} = \left((\mathbf{M}^{-1}\mathbf{T}^{-\top}\vec{\eta})^\top\mathbf{M}(\mathbf{M}^{-1}\mathbf{T}^{-\top}\vec{\eta})\right)^{1/2}\\ &= (\vec{\eta}^\top\mathbf{T}^{-1}(\mathbf{M}^{-1})^\top\mathbf{M}\mathbf{M}^{-1}\mathbf{T}^{-\top}\vec{\eta})^{1/2}\\ &= (\vec{\eta}^\top\mathbf{T}^{-1}\mathbf{M}^{-1}\mathbf{T}^{-\top}\vec{\eta})^{1/2} = \|\vec{\eta}\|,\end{aligned}$$

and

$$\begin{aligned}\|\vec{\mu}\|_{\mathbf{A}} &= (\vec{\mu}^\top\mathbf{A}\vec{\mu})^{1/2} = \left((\mathbf{M}^{-1}\mathbf{T}^{-\top}\vec{\eta})^\top\mathbf{A}(\mathbf{M}^{-1}\mathbf{T}^{-\top}\vec{\eta})\right)^{1/2}\\ &= (\vec{\eta}^\top\mathbf{T}^{-1}(\mathbf{M}^{-1})^\top\mathbf{A}\mathbf{M}^{-1}\mathbf{T}^{-\top}\vec{\eta})^{1/2}\\ &= (\vec{\eta}^\top\mathbf{T}^{-1}\mathbf{M}^{-1}\mathbf{A}\mathbf{T}\mathbf{T}^{-1}\mathbf{M}^{-1}\mathbf{T}^{-\top}\vec{\eta})^{1/2} = (\vec{\eta}^\top\mathbf{D}\vec{\eta})^{1/2}.\end{aligned}$$

(12.2d) Express $\mathsf{m}(u_N, u_N)$ and $\mathsf{a}(u_N, u_N)$ through the coefficient vector $\vec{\boldsymbol{\mu}}$ associated with a function $u_N \in V_{0,N}$ from the discrete Galerkin trial/test space and through the Galerkin matrices $\mathbf{A}$ and $\mathbf{M}$. How does (12.2.2) read when stated in terms of matrices and coefficient vectors? What is the relationship of γ and the generalized eigenvalues λ_i?

Solution: We get

$$\mathsf{m}(u_N, u_N) = \mathsf{m}\left(\sum_{i=1}^N \mu_i b_N^i, \sum_{j=1}^N \mu_j b_N^j\right) = \sum_{i=1}^N \sum_{j=1}^N \mu_i \mu_j \mathsf{m}(b_N^i, b_N^j) = \|\vec{\boldsymbol{\mu}}\|_{\mathbf{M}}^2$$
$$\mathsf{a}(u_N, u_N) = \mathsf{a}\left(\sum_{i=1}^N \mu_i b_N^i, \sum_{j=1}^N \mu_j b_N^j\right) = \sum_{i=1}^N \sum_{j=1}^N \mu_i \mu_j \mathsf{a}(b_N^i, b_N^j) = \|\vec{\boldsymbol{\mu}}\|_{\mathbf{A}}^2$$

It follows that (3) can be rewritten as

$$\exists \gamma > 0 : \qquad \vec{\boldsymbol{\mu}}^\top \mathbf{A} \vec{\boldsymbol{\mu}} \geq \gamma \vec{\boldsymbol{\mu}}^\top \mathbf{M} \vec{\boldsymbol{\mu}}, \quad \forall \vec{\boldsymbol{\mu}} \in \mathbb{R}^N.$$

Using the results from the previous points, we have that

$$\vec{\boldsymbol{\mu}}^\top \mathbf{A} \vec{\boldsymbol{\mu}} \geq \gamma \vec{\boldsymbol{\mu}}^\top \mathbf{M} \vec{\boldsymbol{\mu}} \iff \vec{\boldsymbol{\eta}}^\top \mathbf{D} \vec{\boldsymbol{\eta}} \geq \gamma \vec{\boldsymbol{\eta}}^\top \vec{\boldsymbol{\eta}}$$

which translates to

$$\lambda_i \boldsymbol{\eta}_i^2 \geq \gamma \boldsymbol{\eta}_i^2, \forall i \in \{1, .., N\}$$

From here it follows that

$$\gamma \leq \min\{\lambda_1, ..., \lambda_N\}$$

(12.2e) In the sequel we assume a uniform timestep $\tau > 0$. Show that

$$\left\|\vec{\boldsymbol{\mu}}^{(j)}\right\|_{\mathbf{M}} \leq \frac{1}{1+\gamma\tau} \left\|\vec{\boldsymbol{\mu}}^{(j-1)}\right\|_{\mathbf{M}}, \tag{12.2.4}$$

where $\gamma > 0$ is the constant from (12.2.2).

HINT: There are two ways to tackle the problem: rephrase (12.2.4) in terms of $\vec{\boldsymbol{\eta}}^{(k)}$ and look at the implicit Euler method for these transformed coefficient vectors, see subproblem (12.2b). This is another application of the diagonalization technique.

Alternatively, you may use the Cauchy-Schwarz inequality

$$\vec{\boldsymbol{\xi}}^\top \mathbf{M} \vec{\boldsymbol{\zeta}} \leq \left\|\vec{\boldsymbol{\xi}}\right\|_{\mathbf{M}} \left\|\vec{\boldsymbol{\zeta}}\right\|_{\mathbf{M}}, \tag{12.2.5}$$

and the implicit Euler recursion formula from subproblem (12.2a)

Solution: We already know that

$$\left\|\vec{\boldsymbol{\mu}}^{(j)}\right\|_{\mathbf{M}} \leq \frac{1}{1+\gamma\tau} \left\|\vec{\boldsymbol{\mu}}^{(j-1)}\right\|_{\mathbf{M}} \Leftrightarrow \left\|\vec{\boldsymbol{\eta}}^{(j)}\right\| \leq \frac{1}{1+\gamma\tau} \left\|\vec{\boldsymbol{\eta}}^{(j-1)}\right\|$$

From the subproblem (12.2b) we know that

$$\vec{\boldsymbol{\eta}}^{(j)} - \vec{\boldsymbol{\eta}}^{(j-1)} = -\tau D \vec{\boldsymbol{\eta}}^{(j)} \Leftrightarrow \vec{\boldsymbol{\eta}}^{(j)} = (1+\tau D)^{-1} \vec{\boldsymbol{\eta}}^{(j-1)}$$

Since $\gamma \leq \{\lambda_1, ..., \lambda_N\}$ it follows that $(1+\tau D)^{-1} < (1+\tau\gamma I_N)^{-1}$. Therefore

$$\left\|\vec{\boldsymbol{\eta}}^{(j)}\right\| = \left\|(1+\tau D)^{-1} \vec{\boldsymbol{\eta}}^{(j-1)}\right\| \leq \frac{1}{1+\tau\gamma} \left\|\vec{\boldsymbol{\eta}}^{(j-1)}\right\|$$

(12.2f) Now show (12.2.4) with $\|\cdot\|_{\mathbf{M}}$ replaced with $\|\cdot\|_{\mathbf{A}}$.

HINT: Here it is recommended to use diagonalization and the result of subproblem (12.2c).

Solution:

$$\begin{aligned}
\left\|\vec{\boldsymbol{\mu}}^{(j)}\right\|_{\mathbf{A}} \leq \frac{1}{1+\gamma\tau}\left\|\vec{\boldsymbol{\mu}}^{(j-1)}\right\|_{\mathbf{A}} &\iff \left((\vec{\boldsymbol{\eta}}^{(j)})^\top D\vec{\boldsymbol{\eta}}^{(j)}\right)^{1/2} \leq \frac{1}{1+\gamma\tau}\left((\vec{\boldsymbol{\eta}}^{(j-1)})^\top D\vec{\boldsymbol{\eta}}^{(j-1)}\right)^{1/2} \\
&\iff \left\|\sqrt{D}\boldsymbol{\eta}^{(j)}\right\| \leq \frac{1}{1+\gamma\tau}\left\|\sqrt{D}\boldsymbol{\eta}^{(j-1)}\right\| \\
&\iff \sqrt{\lambda_i}\left\|\boldsymbol{\eta}_i^{(j)}\right\| \leq \frac{1}{1+\gamma\tau}\sqrt{\lambda_i}\left\|\boldsymbol{\eta}_i^{(j-1)}\right\|, \quad \forall i \in \{1, N\} \\
&\iff \left\|\boldsymbol{\eta}_i^{(j)}\right\| \leq \frac{1}{1+\gamma\tau}\left\|\boldsymbol{\eta}_i^{(j-1)}\right\|, \quad \forall i \in \{1, N\}
\end{aligned}$$

which follows directly from subproblem (12.2e).

(12.2g) What is the relationship of the estimates obtained in subproblems (12.2e) and (12.2f) with [NPDE, Lemma 6.1.22].

HINT: Bound $\left\|\vec{\boldsymbol{\mu}}^{(j)}\right\|_{\mathbf{M}}$ and $\left\|\vec{\boldsymbol{\mu}}^{(j)}\right\|_{\mathbf{A}}$ in terms of $\left\|\vec{\boldsymbol{\mu}}^{(0)}\right\|_{\mathbf{M}}$ and $\left\|\vec{\boldsymbol{\mu}}^{(0)}\right\|_{\mathbf{A}}$, respectively. Then, for fixed t consider $\tau \to 0$ and $j \approx t/\tau \to \infty$. Remember the limit

$$\mathrm{e}^t = \lim_{n\to\infty}(1+\tfrac{t}{n})^n\,, \quad t \in \mathbb{R}\,.$$

Solution: [NPDE, Lemma 6.1.22] tells us that

$$\|u(t)\|_{L^2(\Omega)} \leq \mathrm{e}^{-\gamma t}\|u_0\|_{L^2(\Omega)}, \qquad |u(t)|_{H^1(\Omega)} \leq \mathrm{e}^{-\gamma t}|u_0|_{H^1(\Omega)}$$

Writing the approximate solution $u_N(t) = \sum_{i=1}^N \mu_i b_N^i$ we get

$$\|u(t)\|_{L^2(\Omega)} = \|\vec{\boldsymbol{\mu}}(t)\|_{\mathbf{M}}, \qquad |u_N(t)|_{H^1(\Omega)} = \|\vec{\boldsymbol{\mu}}(t)\|_{\mathbf{A}}$$

From the previous point we have that

$$\left\|\vec{\boldsymbol{\mu}}^{(j)}\right\|_{\mathbf{M}} \leq \frac{1}{1+\gamma\tau}\left\|\vec{\boldsymbol{\mu}}^{(j-1)}\right\|_{\mathbf{M}} = \left(\frac{1}{1+\gamma\tau}\right)^j \left\|\vec{\boldsymbol{\mu}}^{(0)}\right\|_{\mathbf{M}}$$

We now consider that the interval $(0,t) = (0,t_j)$ has been discretized into n intervals, with $\tau = t/n$ it follows that

$$\|\vec{\boldsymbol{\mu}}(t)\|_{\mathbf{M}} \leq \left(\frac{1}{1+\gamma t/n}\right)^n \|\vec{\boldsymbol{\mu}}(0)\|_{\mathbf{M}} = \left(1-\frac{\gamma t}{n+\gamma t}\right)^n \|\vec{\boldsymbol{\mu}}(0)\|_{\mathbf{M}} \leq \left(1-\frac{\gamma t}{n}\right)^n \|\vec{\boldsymbol{\mu}}(0)\|_{\mathbf{M}}$$

which becomes for $n \to \infty$

$$\|\vec{\boldsymbol{\mu}}(t)\|_{\mathbf{M}} \leq \mathrm{e}^{-\gamma t}\|\vec{\boldsymbol{\mu}}(0)\|_{\mathbf{M}}$$

In the same way,

$$\|\vec{\boldsymbol{\mu}}(t)\|_{\mathbf{A}} \leq \mathrm{e}^{-\gamma t}\|\vec{\boldsymbol{\mu}}(0)\|_{\mathbf{A}}$$

It follows that the relations from subproblems (12.2e) and (12.2f) converge to the ones from [NPDE, Lemma 6.1.22] in the limit $n \to \infty$.

Problem 12.3 Radiative Cooling

This problem is dedicated to the full spatial and temporal discretization of a 2nd-order parabolic evolution problem, see [NPDE, Section 6.1]. It will also ask for implementation in C++ in later sub-problems.

The evolution of the temperature distribution $u = u(\boldsymbol{x}, t)$ in a homogeneous "2D body" (occupying the space $\Omega \subset \mathbb{R}^2$) with convective cooling ($\rightarrow$ [NPDE, Ex. 2.7.5]) is modelled by the linear second-order parabolic initial-boundary value problem (IBVP) with flux (spatial) boundary conditions ($\rightarrow$ [NPDE, § 6.1.9])

$$\begin{aligned} \frac{\partial u}{\partial t} - \Delta u &= 0 && \text{in } \Omega \times [0,T], \\ -\mathbf{grad}\, u \cdot n &= cu && \text{on } \partial\Omega \times [0,T], \\ u(\boldsymbol{x},0) &= u_0(\boldsymbol{x}) && \text{in } \Omega, \end{aligned} \tag{12.3.1}$$

with $c > 0$.

We pursue a method of lines approach, see [NPDE, Section 6.1.4]. For the spatial Galerkin semi-discretization of (12.3.1) we employ linear finite elements on a triangular mesh $\mathcal{M}$ of Ω (FE space $\mathcal{S}_1^0(\mathcal{M})$) with polygonal boundary approximation.

(12.3a) Derive the spatial variational formulation of the form $\mathsf{m}(\dot{u}, v) + \mathsf{a}(u, v) = \ell(v)$ for (12.3.1), with suitable bilinear forms a and m, and linear form ℓ. Do not forget to specify the function spaces for $u(t, \cdot)$ and the test function v.

HINT: Combine the considerations leading to [NPDE, Eq. (6.1.14)] with the approach explained in [NPDE, Ex. 2.9.6].

Solution: In order to derive the spatial Formulation of (12.3.1), we multiply with a test function v (does not depend on time!) and integrate over Ω:

$$\begin{aligned} &\int_\Omega \dot{u}\, v \, \mathrm{d}\boldsymbol{x} + \int_\Omega \operatorname{grad} u \cdot \operatorname{grad} v \, \mathrm{d}\boldsymbol{x} - \int_{\partial\Omega} v \operatorname{grad} u \cdot \boldsymbol{n} \, \mathrm{d}\boldsymbol{x} = 0 \\ &\quad \Rightarrow \underbrace{\int_\Omega \dot{u}\, v \, \mathrm{d}\boldsymbol{x}}_{m(\dot{u},v)} + \underbrace{\int_\Omega \operatorname{grad} u \cdot \operatorname{grad} v \, \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} cu\, v \, \mathrm{d}\boldsymbol{x}}_{a(u,v)} = \underbrace{0}_{\ell(v)} \end{aligned}$$

For any $t \in [0,T]$ the function spaces are

$$u(t,\cdot) \in H^1(\Omega) =: U \quad , \quad v \in H^1(\Omega) =: V \ .$$

(12.3b) Argue why the total thermal energy

$$E(t) := \int_\Omega u(\boldsymbol{x}, t) \, \mathrm{d}\boldsymbol{x} \ ,$$

decreases with time, if $u_0(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \Omega$.

HINT: Appeal to the heat conduction background to justify the assumption that $u(\boldsymbol{x}, t) \geq 0$ for all $(\boldsymbol{x}, t)$. Use test function $v \equiv 1$ in the variational formulation.

Solution: The quantity u specifies the temperature distribution. We track the temporal evolution of the diffusion of the temperature. These two physical effects can never lead to negative values

in the temperature distribution except if there were negative values in the start. The boundary condition models convective cooling (→ [NPDE, Ex. 2.7.5]) where our domain Ω is embeded in a fluid at bulk temperature 0. So if $u \geq 0$ on the boundary then the 2D body will be cooled. The convective cooling dissapears as $u \to 0$ (it would even warm the 2D body for negative temperature $u < 0$).

Thus $u(\boldsymbol{x}, t) \geq 0$ and

$$E(t) := \int_\Omega u(\boldsymbol{x}, t) \, \mathrm{d}\boldsymbol{x} = \int_\Omega |u(\boldsymbol{x}, t)| \, \mathrm{d}\boldsymbol{x} \geq 0.$$

We do a similar computation to [NPDE, Lemma 6.1.22], but we don't need to use the function w that is introduced there, we can do the calculation directly

$$\frac{\mathrm{d}}{\mathrm{d}t} E(t) = \frac{\mathrm{d}}{\mathrm{d}t} \int_\Omega u(\boldsymbol{x}, t) \, \mathrm{d}\boldsymbol{x} = \frac{\mathrm{d}}{\mathrm{d}t} m(u, 1) = -a(u, 1) = -c \underbrace{\int_{\partial\Omega} u \, \mathrm{d}\boldsymbol{x}}_{\geq 0} \leq 0.$$

Where in the last step we used the positivity of u. The case $\frac{\mathrm{d}}{\mathrm{d}t} E(t) = 0$ appears only for $u \equiv 0$.

(12.3c) Compute the local mass matrix $\mathbf{M}_{\widehat{K}}$ corresponding to $\mathsf{m}(\cdot, \cdot)$ and the local stiffness matrix $\mathbf{A}_{\widehat{K}}$ corresponding to $\mathsf{a}(\cdot, \cdot)$ for the unit triangle $\widehat{K}$ with vertices $\binom{0}{0}$, $\binom{1}{0}$ and $\binom{0}{1}$. Assume that the edge connecting $\binom{0}{0}$ and $\binom{1}{0}$ forms part of $\partial\Omega$ and that the coefficient c is constant along this edge.

HINT: See [NPDE, Def. 3.6.35] for the definition of local matrices, and [NPDE, Section 3.3.5] for concrete formulas. [NPDE, Lemma 3.6.61] for $d = 2$ may also come handy.

Solution: The mass and stiffness matrices are

$$\mathbf{M} = \frac{1}{24} \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}, \qquad \mathbf{A} = \frac{1}{2} \begin{pmatrix} 2 & -1 & -1 \\ -1 & 1 & \\ -1 & & 1 \end{pmatrix} + \frac{c}{6} \begin{pmatrix} 2 & 1 & \\ 1 & 2 & \\ & & \end{pmatrix}.$$

(12.3d) Now we turn to the full spatial semi-discretization of (12.3.1). Template files for the new classes you will need to write are available in the lecture `svn` repository

`assignments_codes/assignment12/Problem3`

In `EvSolver.hpp`, implement the method

```
template <class Function>
computeTriplets_(Function const& c, Triplet &tripA, Triplet &tripM)
```

that computes the triplet vectors tripM and tripA for $\mathbf{M}, \mathbf{A} \in \mathbb{R}^{N \times N}$, $N := \dim \mathcal{S}_1^0(\mathcal{M})$, for the semi-discrete evolution

$$\mathbf{M} \frac{\mathrm{d}}{\mathrm{d}t} \vec{\boldsymbol{\mu}}(t) + \mathbf{A} \vec{\boldsymbol{\mu}}(t) = 0 \tag{12.3.2}$$

resulting from the $\mathcal{S}_1^0(\mathcal{M})$-based finite element semi-discretization of (12.3.1), when standard nodal bassis functions are used. Here, the argument `c` supplies the value of c.

HINT: The use of index-value triplets for the initialization of sparse matrices in the C++ template library Eigen is explained in [NPDE, Rem. 3.6.45]. [NPDE, Code 3.6.49] provides an example for the use of triplets.

HINT: Do not use the result of sub-problem (12.3c) (which is only valid for a very special triangle). Use your already implemented `DofHandler`, `MatrixAssembler`, `VectorAssembler`, `BoundaryDofs`, and local assemblers, developed in subproblems 7.4, 8.1, and 8.2 (also available in the corresponding solution folders).

HINT: Since we are dealing with Robin B.C., your bilinear form should have a term which is integrating over the boundary. This means the local assembler has to take edges as element type instead of triangles, reason why you cannot use your analytical implementations. This was already done in subproblem (8.2g). Solution to subproblem (8.1e) might also be useful.

Solution:

Listing 12.5: Implementation of `computeTriplets_()`

```
112 template <class DofHandler>
113 template <class Function>
114 void EvSolver<DofHandler>::computeTriplets_(Function const& c,
115                                             Triplets &tripA,
116                                             Triplets &tripM){
117
118   NPDE15::LBoundaryDofs<DofHandler> get_bnd_dofs(dofh);
119   // obtain index vectors corresponding to boundary
120   IndexVector robin_dofs;
121   get_bnd_dofs(robin_dofs);
122   // Create matrix Assembler
123   NPDE15::MatrixAssembler<DofHandler> matAssembler(dofh);
124   // fill triplets for a( , ) considering boundary term
125   matAssembler(tripA, NPDE15::AnalyticalLocalLaplace());
126   matAssembler(tripA, NPDE15::LocalMass(c), robin_dofs);
127   // fill triplets for m( , )
128   matAssembler(tripM, NPDE15::AnalyticalLocalMass());
129 }
```

(12.3e) In [NPDE, Ex. 6.1.85] you learned about the L-stable SDIRK-2 implicit 2-stage Runge-Kutta method described by the Butcher scheme

$$\begin{array}{c|cc} \lambda & \lambda & 0 \\ 1 & 1-\lambda & \lambda \\ \hline & 1-\lambda & \lambda \end{array} \qquad \lambda := 1 - \tfrac{1}{2}\sqrt{2}\,. \tag{12.3.3}$$

Derive the recursion obtained by applying the SDIRK-2 method to the linear scalar ODE $\dot{y} = -\gamma y$, $\gamma \in \mathbb{R}$.

Solution: Here $\dot{y} = -\gamma y$. SDIRK-2 single step method

$$\kappa_1 = -\gamma y - \gamma\tau\lambda\kappa_1 \tag{12.3.4}$$

$$\kappa_2 = -\gamma y - \gamma\tau(1-\lambda)\kappa_1 - \gamma\tau\lambda\kappa_2 \tag{12.3.5}$$

solving this system, we get

$$\kappa_1 = \frac{-\gamma y}{1+\lambda\tau}, \qquad \kappa_2 = \frac{-\gamma y(1+2\lambda\tau-\tau)}{(1+\lambda\tau)^2} \tag{12.3.6}$$

Finally, updating the solution, we get

$$Y^{j+1} = Y^j + \tau(1-\lambda)\kappa_1 + \tau\lambda\kappa_2 \tag{12.3.7}$$

$$= Y^j\left(1 - \frac{\tau\gamma(1+\lambda^2\tau)}{(1+\lambda\tau)^2}\right) \tag{12.3.8}$$

(12.3f) Determine the order of the SDIRK-2 single step method empirically by applying it to the initial value problem $\dot{y} = -y$ on $[0, 2]$, $y(0) = 1$. To that end write a short code in MATLAB or C++.

Solution: The order is 2 as expected. See Listing 12.6 and Listing 12.7.

Listing 12.6: Implementation for subproblem (12.3f)

```
%Radau Convergence analysis

N = [20 40 80 160 320 740 1480];

err = ones(1,length(N));
err2 = ones(1,length(N));
Rate_conv = ones(1,(length(N) - 1));

for i = 1:length(N)
    [Y,err(i)] = RadCol_Scalar(N(i));
    if (i > 1)
        Rate_conv(i - 1) =
           log(err(i)/err(i-1))/log(N(i)/N(i-1));
    end;
end;

figure(1)
loglog(N,err);
Rate_conv
```

Listing 12.7: Implementation for subproblem (12.3f)

```
function [Y,err] = RadCol_Scalar(N)

tau = (1/N);
T = (0:tau:1);
Y = ones(1,length(T));
Yex = ones(1,length(T));

l = 1 - sqrt(2)/2;

Y(1) = 1;
Yex(1) = 1;

for i = 2:length(Y)
```

```
    Y(i) = (1 - tau*(1 + l^2*tau)/(1+l*tau)^2)*Y(i-1);
    Yex(i) = Yex(1)*exp(-tau*(i-1));
end;

err = max(abs(Y - Yex));

end
```

(12.3g) Give a rigorous proof that the SDIRK-2 method is $\mathrm{L}(\pi)$-stable ($\rightarrow$ [NPDE, Def. 6.1.84]).

HINT: Of course, the recursion found in subproblem (12.3e) has to be used. Then the problem boils down to discussing the behavior of a rational function on the (positive) real axis.

Solution: We use the recursion found in subproblem (12.3e) to get

$$Y^{j+1} = \left(1 - \frac{\tau\gamma(1+\lambda^2\tau)}{(1+\lambda\tau)^2}\right)^j Y^0, \tag{12.3.9}$$

i.e. $\Psi_\gamma^\tau := \left(1 - \frac{\tau\gamma(1+\lambda^2\tau)}{(1+\lambda\tau)^2}\right)$.

We want to show that $\forall\gamma > 0$

$$\lim_{j\to\infty} \left(\Psi_\gamma^\tau\right)^j y_0 = 0, \quad \forall y_0 \in \mathbb{R}, \forall\tau > 0. \tag{12.3.10}$$

It suffices to prove that $\frac{\tau\gamma(1+\lambda^2\tau)}{(1+\lambda\tau)^2} > 0$, since this will guarantee that $\left|\Psi_\gamma^\tau\right| < 1$ and therefore $\lim_{j\to\infty}(\Psi_\gamma^\tau)^j = 0$. As $\tau, \gamma > 0$, it is clear the whole expression is indeed positive.

(12.3h) [NPDE, § 6.1.42] presents the linear system of equations for the increments of an s-stage Runge-Kutta method when applied to the semi-discrete evolution (12.3.2). State this linear system explicitly in block form for the special case of the SDIRK-2 method.

Solution: Formulas [NPDE, Eq. (6.1.43)] gives us two equations systems which we have to solve for the unknown vectors $\mathbf{k}_1$, $\mathbf{k}_2$

$$\begin{aligned} \mathbf{M}\mathbf{k}_1 + \tau\lambda\mathbf{A}\mathbf{k}_1 &= -\mathbf{A}\mu, \\ \mathbf{M}\mathbf{k}_2 + \tau(1-\lambda)\mathbf{A}\mathbf{k}_1 + \tau\lambda\mathbf{A}\mathbf{k}_2 &= -\mathbf{A}\mu. \end{aligned}$$

Then [NPDE, Eq. (6.1.44)] gives us an update formula for $\boldsymbol{\mu}$

$$\boldsymbol{\mu}^{j+1} = \boldsymbol{\mu}^j + \tau k_1(1-\lambda) + \tau k_2\lambda.$$

(12.3i) Write a method

```
template <class Function>
void solve(Vector & U, Function const& c);
```

in EvSolver that carries out m uniform timesteps of the in order to solve (12.3.1) over the time interval $[0, 1]$. The finite element Galerkin discretization from subproblem (12.3d) is used in space. The argument `U` is a column vector that passes the values of the initial temperature distribution in

the vertices of the mesh. The return value provides the basis coefficients of the approximation of $u(\cdot, 1)$ of u at $t = 1$. This function will be called within the main.

HINT: From [NPDE, Def. 6.1.39] and [NPDE, Eq. (6.1.41)] it should be clear how to obtain the linear systems of equations [NPDE, Eq. (6.1.43)], [NPDE, Eq. (6.1.44)] for the Runge-Kutta increments.

Solution: See Listing 12.8.

Listing 12.8: Implementation of `solve()`

```
66  template <class DofHandler>
67  template <class Function>
68  void EvSolver<DofHandler>::solve(Vector &U, Function const& c){
69
70    // compute triplets for m and a.
71    Triplets tripletsA, tripletsM;
72    computeTriplets_(c, tripletsA, tripletsM);
73
74    // compute auxiliary matrix M + tau*lambda*A from SDIRK-2 system
75    // on triplets level
76    Triplets auxtrip;
77    //int offset = N;
78    for (auto & itA : tripletsA)
79      auxtrip.push_back({itA.row(), itA.col(), lambda*Tau*itA.value()});
80    for (auto & itM : tripletsM)
81      auxtrip.push_back({itM.row(), itM.col(), itM.value()});
82
83    Matrix auxM(N, N);
84    auxM.setFromTriplets(auxtrip.begin(), auxtrip.end());
85    auxM.makeCompressed();
86
87    // RK coefficients
88    std::vector<double> b = {1-lambda, lambda};
89
90    // Assemble a's Galerkin Matrix (for RHS)
91    Matrix A(N,N);
92    A.setFromTriplets(tripletsA.begin(), tripletsA.end());
93    A.makeCompressed();
94
95    E[0] = average_(U);
96    for( int i = 0; i < m; i++){
97      // create RHS
98      Vector L = -A*U;
99      // create first increment vector
100     Vector k1 = L/auxM;
101     // substitute in RHS for second increment
102     L = L - Tau*(1-lambda)*A*k1;
103     // create second increment vector
104     Vector k2 = L/auxM;
105     // Update solution
106     U += Tau*(b[0]*k1 + b[1]*k2 );
107     E[i+1] = average_(U);
108     std::cout << "Time iter #" << i+1 << "\n";
109   }
110 }
```

(12.3j) In EvSolver, implement a method

```
double average_(Vector const& U);
```

that computes $\int_\Omega u \, \mathrm{d}\boldsymbol{x}$ for $u \in \mathcal{S}_1^0(\mathcal{M})$. The argument `U` passes the coefficients of u w.r.t. the standard nodal basis of $\mathcal{S}_1^0(\mathcal{M})$.

HINT: Your implementation for subproblem (7.3a) might be useful.

Solution:

Listing 12.9: Implementation of `average_()`

```
131 template <class DofHandler>
132 double EvSolver<DofHandler>::average_(Vector const& U){
133   double total = 0.0;
134   // Traverse the cells using EntityIterator of co-dimension 0
135   for (auto eit = gv.template begin<0>(); eit != gv.template end<0>();
        ++eit){
136     // get element's geometry
137     auto const& egeom = (*eit).geometry();
138     double f_int = 0.0;
139     for (int i = 0; i<egeom.corners(); i++){
140       // get global index of current vertex
141       auto global_id = dofh(*eit, i);
142       // get coefficient corresponding to given dof
143       auto f_val = U[global_id];
144       // add contribution
145       f_int+= 1.0/egeom.corners()*f_val*egeom.volume();
146     }
147     total += f_int;
148   } // end traversing grid cells
149   return total;
150 }
```

(12.3k) For the evolution problem (12.3.1) on $\Omega = (0,1)^2$ track the behavior of the thermal energy

$$E(t) = \int_\Omega u(\boldsymbol{x}, t) \, \mathrm{d}\boldsymbol{x} \tag{12.3.11}$$

over the period $[0, T]$ for $u_0 \equiv 1$, $\gamma = 1$. Use the fully discrete evolution implemented in EvSolver::solve() and extend it, so it also computes approximations for $E(t_k)$ for $k = 0, \ldots, m$ (t_k are the points of the equidistant temporal grid). Then implement a method Vector getE() which returns a Vector E containing said approximations.

Complete `main.cc` to compute $u(x, t)$ for $t = 1$ for the mesh supplied in the file `square_32.msh`. Make a plot of u for $t = 0$ and $t = 1$ using `Paraview`. Plot the approximation for $E(t)$ that you have computed as a function of t for $m = 100$.

HINT: The plot for E is depicted in Figure 12.2.

Solution:

The field plots at $t = 0$ and $t = 1$ are displayed in Figure 12.3. For the addition to `RadTEvl` see Listing 12.8, for the implementation of `main.cc` see Listing 12.10.

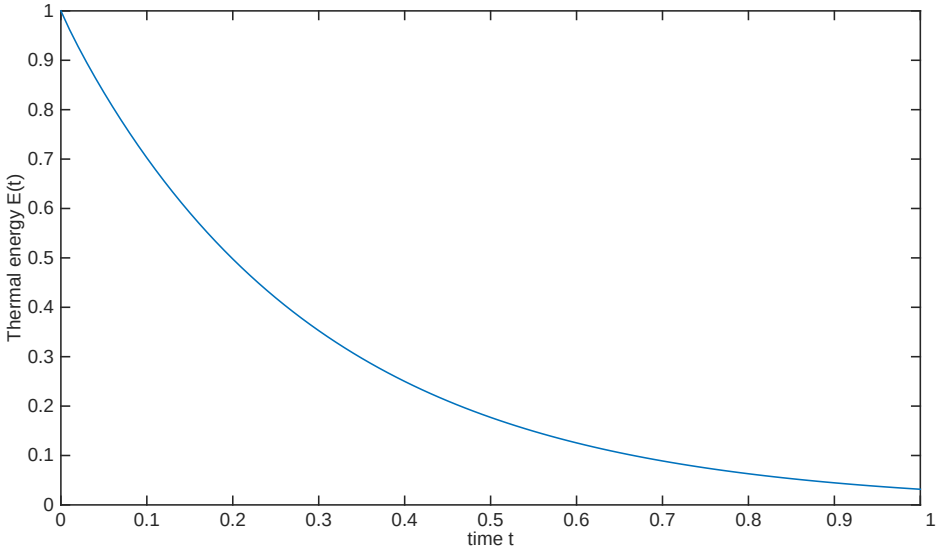
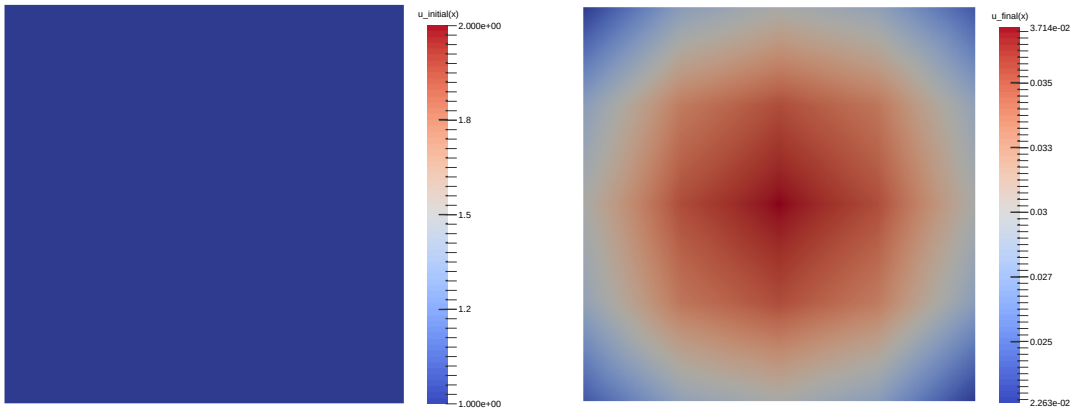


Figure 12.2: Result of subproblem (12.3k)

Listing 12.10: Implementation of `average_()`

```
35 int main(int argc, char *argv[]){
36   try{
37     // load the grid from file
38     std::string fileName = "square_32.msh";
39
40     // Declare and create mesh using the Gmsh file
41     Dune::GridFactory<GridType> gridFactory;
42     Dune::GmshReader<GridType>::read(gridFactory, fileName.c_str(), false,
       true);
43     std::unique_ptr<GridType> workingGrid(gridFactory.createGrid());
44     workingGrid->loadBalance();
45
46     // Get the Gridview
47     GridView gv = workingGrid->leafGridView();
48
49     // Initialize dof-handler
50     DofHandler dofh(gv);
51
52     unsigned N = dofh.size();
53     std::cout << "Solving for N =" << N << " unknowns.\n";
54
55     // initialize evolution solver (SDIRK-2) using m = 100
56     EvSolver<DofHandler> solver(dofh, 100);
57     Vector U(N); U.setOnes();
58     solver.solve(U, [](Coordinate const& x){return 1.0;});
59     Vector E = solver.getE();
60
61     // Write solutions to vtk format
62     Dune::VTKWriter<GridView> vtkwriter(gv);
63     std::stringstream outputName;
64     outputName << "solution";
65     Vector U0(N); U0.setOnes();
```



(a) Initial distribution at $t = 0$ (b) Final distribution at $t = 1$

Figure 12.3: Initial and Final distribution for subproblem (12.3k).

```
66      vtkwriter.addVertexData(U0, "u_initial(x)");
67      vtkwriter.addVertexData(U, "u_final(x)");
68      vtkwriter.write(outputName.str().c_str());
69
70      std::ofstream outE("Et.dat", std::ios::out | std::ios::binary );
71      outE << E;
72      outE.close( );
73
74      std::cout << "Done.\n";
75
76    }
```

Published on 13.05.2015.

To be submitted on 20.05.2015.

References

[NPDE] Lecture Slides for the course "Numerical Methods for Partial Differential Equations".SVN revision # 76119.

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

Last modified on May 28, 2015