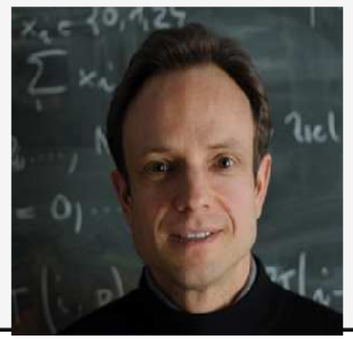
ETH Lecture 401-0674-00L Numerical Methods for Partial Differential Equations

Homework Problem Video Tutorial

Problem 6-1: Implicit Two-Stage Radau Runge-Kutta Single-Step Methods for Parabolic IBVPs

Prof. R. Hiptmair, SAM, ETH Zurich

Date: April 19, 2019



IBVP, strong form: ($\dot{u}$ ← heat source)

$$\begin{aligned} \dot{u} - \Delta u &= f(x,t) && \text{in } \Omega\times]0,1[\,, \\ u &= 0 && \text{on } \partial\Omega\times]0,1[\,, \\ u &= 0 && \text{on } \Omega\times\{0\}\,, \end{aligned} \tag{6.1.1}$$

($u = 0$ on $\partial\Omega\times]0,1[$ ← homogeneous Dirichlet b.c.)

$$f(x,t) = \begin{cases} 1 & \text{, if } \left\| x - \frac{1}{2}\begin{pmatrix}\cos\pi t\\ \sin\pi t\end{pmatrix}\right\| < \frac{1}{2}, \\ 0 & \text{elsewhere.} \end{cases} \tag{6.1.2}$$

Approach: MOL + $L(\pi)$-stable RK-SSM

a, Spatial variational formulation

Test with $v \in H^1_0(\Omega)$ + integrate + i.b.p.

$u : [0,T] \to H^1_0(\Omega)$

$$\int_\Omega \frac{\partial u}{\partial t}(x,t)\, v(x)\,\mathrm{d}x + \int_\Omega \operatorname{grad} u(x,t)\cdot\operatorname{grad} v(x)\,\mathrm{d}x = \int_\Omega f(x,t)\, v(x)\,\mathrm{d}x \tag{6.1.4}$$

($\frac{\partial u}{\partial t}$: just a parameter at this stage; $\forall v \in H^1_0(\Omega)$)

b, Derive the discrete evolution operator as introduced in [Lecture → Def. 6.2.6.2] for the 2-stage Radau method, a Runge-Kutta single-step method (RK-SSM) defined by the Butcher scheme

$A = (a_{ij})$

$$\begin{array}{c|c} \mathbf{c} & \mathcal{A} \\ \hline & \mathbf{b}^T \end{array} \;\hat{=}\; \begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array}\,, \tag{Lecture → Eq. (6.2.7.52)}$$

when applied to the linear scalar ordinary differential equation $\dot{y} = -y$.

(2)

Definition 6.2.6.31. General Runge-Kutta method →

For coefficients $b_i, a_{ij} \in \mathbb{R}$, $c_i := \sum_{j=1}^{s} a_{ij}$, $i,j = 1,\dots,s$, $s \in \mathbb{N}$, the discrete evolution $\Psi^{s,t}$ of an s-stage Runge-Kutta single step method (RK-SSM) for the ODE $\dot{\mathbf{u}} = \mathbf{f}(t,\mathbf{u})$, is defined by

$$\mathbf{k}_i := \mathbf{f}(t + c_i\tau, \mathbf{u} + \tau\sum_{j=1}^{s} a_{ij}\mathbf{k}_j)\,, \quad i = 1,\dots,s \quad , \quad \Psi^{t,t+\tau}\mathbf{u} := \mathbf{u} + \tau\sum_{i=1}^{s} b_i\mathbf{k}_i\,.$$

The $\mathbf{k}_i \in V_0$ are called increments.

increment equation

discrete evolution op.

MOL-ODE:

$$\mathbf{M}\left\{\frac{d}{dt}\vec{\mu}(t)\right\} + \mathbf{A}\vec{\mu}(t) = \vec{\varphi}(t) \quad \Leftrightarrow \quad \dot{\vec{\mu}} = \underbrace{\mathbf{M}^{-1}(\vec{\varphi}(t) - \mathbf{A}\vec{\mu}(t))}_{=\mathbf{f}(t,\vec{\mu})}\,. \tag{7.1.4.4}$$

LSE for increments

(7.1.7.7) $\Leftrightarrow$ $(\mathbf{I}_s \otimes \mathbf{M} + \tau\mathfrak{A} \otimes \mathbf{A})\begin{bmatrix}\vec{\kappa}_1\\ \vdots \\ \vec{\kappa}_s\end{bmatrix} = \begin{bmatrix}\vec{\varphi}(t_j + c_1\tau) - \mathbf{A}\vec{\mu}^{(j)}\\ \vdots \\ \vec{\varphi}(t_j + c_s\tau) - \mathbf{A}\vec{\mu}^{(j)}\end{bmatrix}\,.$ (6.2.7.9)

Kronecker product

For $\dot{y} = -y$ ▷

$$\kappa_1 = -y - \tau\frac{5}{12}\kappa_1 + \tau\frac{1}{12}\kappa_2\,, \tag{6.1.5}$$

$$\kappa_2 = -y - \tau\frac{3}{4}\kappa_1 - \tau\frac{1}{4}\kappa_2\,. \tag{6.1.6}$$

$$y^{(j+1)} = y^{(j)} + \tau\frac{3}{4}\kappa_1 + \tau\frac{1}{4}\kappa_2$$
$$= y^{(j)}\left(1 - \frac{\tau(1+\frac{\tau}{6})}{(1+\tau\frac{5}{12})(1+\tau\frac{1}{4}) + \frac{\tau^2}{16}}\right)\,. \tag{6.1.8}$$

$\Leftrightarrow$ $\Psi^{t,t+\tau}$

c) → Implement this

d) → Maximal error over all timesteps

ℓ	m_ℓ	error ε_ℓ	estimated rate
1	10	5.20365e-05	*
2	20	6.86977e-06	2.92119
:	40	8.84939e-07	2.95661
:	80	1.12381e-07	2.97718
:	160	1.41621e-08	2.98829
:	320	1.77756e-09	2.99407
:	640	2.22655e-10	2.99701
!	1280	2.78612e-11	2.99848
	2560	3.48362e-12	2.9996
	5120	4.36833e-13	2.99543

Rate of cvg.

$$\approx -\frac{\log \varepsilon_\ell/\varepsilon_{\ell-1}}{\log m_\ell/m_{\ell-1}}$$

△ rate = 3 ⇔ order = 3

③ c, 2-stage Radau discrete evolution for MOL-ODE

$$(7.1.7.7) \quad \Leftrightarrow \quad (\mathbf{I}_s \otimes \mathbf{M} + \tau \mathcal{A} \otimes \mathbf{A}) \begin{bmatrix} \vec{\kappa}_1 \\ \vdots \\ \vec{\kappa}_s \end{bmatrix} = \begin{bmatrix} \vec{\varphi}(t_j + c_1\tau) - \mathbf{A}\vec{\mu}^{(j)} \\ \vdots \\ \vec{\varphi}(t_j + c_s\tau) - \mathbf{A}\vec{\mu}^{(j)} \end{bmatrix}. \qquad (6.2.7.9)$$

$$\begin{bmatrix} \mathbf{M} + \tau\frac{5}{12}\mathbf{A} & \tau\frac{-1}{12}\mathbf{A} \\ \tau\frac{3}{4}\mathbf{A} & \mathbf{M} + \tau\frac{1}{4}\mathbf{A} \end{bmatrix} \begin{bmatrix} \vec{\kappa}_1 \\ \vec{\kappa}_2 \end{bmatrix} = \begin{bmatrix} \vec{\varphi}(t_j + \frac{1}{3}\tau) - \mathbf{A}\vec{\mu}^{(j)} \\ \vec{\varphi}(t_j + \tau) - \mathbf{A}\vec{\mu}^{(j)} \end{bmatrix}. \qquad (6.1.15)$$

$$\left(\underbrace{\begin{bmatrix} 1 & 1 \\ \cdot & \cdot \end{bmatrix} \otimes M}_{=: M_\otimes} + \tau \underbrace{\begin{bmatrix} 5/12 & -1/12 \\ 3/4 & 1/4 \end{bmatrix} \otimes A}_{=: A_\otimes} \right)$$

```
class Radau3MOLTimestepper {
public:
  Radau3MOLTimestepper() = delete;
  Radau3MOLTimestepper(const Radau3MOLTimestepper &) = delete;
  Radau3MOLTimestepper(Radau3MOLTimestepper &&) = delete;
  Radau3MOLTimestepper &operator=(const Radau3MOLTimestepper &) =
     delete;
  Radau3MOLTimestepper &operator=(Radau3MOLTimestepper &&) = delete;

  explicit Radau3MOLTimestepper(
     const lf::assemble::DofHandler &dofh);
  virtual ~Radau3MOLTimestepper() = default;

  Eigen::VectorXd discreteEvolutionOperator(double tau, double time,
     const Eigen::VectorXd &mu) const;
  // plus appropriate data
};
```

f, Using the LEHRFEM++ library, in the file `radau_three_timestepping.cc` implement a function

```
Eigen::VectorXd rhsVectorheatSource(
   const lf::assemble::DofHandler &dofh,
   double time);
```

} ⇒ computes $\vec{\varphi}(t)$

that computes the right-hand-side vector arising from the $\mathcal{S}_1^0(\mathcal{M})$-Galerkin finite element discretization of the linear functional $v \mapsto \int_\Omega f(\boldsymbol{x},t)\,\mathrm{d}\boldsymbol{x}$ on the right-hand-side of the spatial variational formulation of (6.1.1), where $f(\boldsymbol{x},t)$ is given by (6.1.2). As usual, the argument `dofh` supplies information about the mesh and the local-to-global index mapping. Of course, local quadrature should has to be employed and we opt for the 2D trapezoidal rule

$$\int_K \psi(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \frac{|K|}{\sharp\mathcal{V}(K)} \sum_{\ell=1}^{\sharp\mathcal{V}(K)} \psi(\boldsymbol{a}_K^\ell) \qquad (6.1.17)$$

for a cell $K \in \mathcal{M}$ with vertices $\boldsymbol{a}_K, \ldots, \boldsymbol{a}_K^{\sharp\mathcal{V}(K)}$ and area $|K|$.

As part of the specification we demand that all entries of the return vector corresponding to basis functions associated with nodes on $\partial\Omega$ are zero.

$\vec{\varphi}(t) = [\, f(\underline{x}^i, t) \cdot A_i \,]_{i=1}^{N}$, $N = \#\mathcal{V}(\mathcal{M})$

$A_i \hat{=}$ 1/3 · Sum of area of all triangles adjacent to $\underline{x}^i$ $\qquad \mathcal{V}(\mathcal{M}) = \{\underline{x}^i\}$

▷ Implementation: cell-oriented assembly

↓

Need: ENTITY_VECTOR_PROVIDER

C++ code 6.1.19: Sub-problem (6-1.f): `Eval()` method returning element vector ➙ GITLAB

```
template <typename FUNCTOR>
Eigen::Vector3d TrapRuleLinFEElemVecProvider<FUNCTOR>::Eval(
    const lf::mesh::Entity &tria) {
  Eigen::Vector3d ElemVec;
  /* SOLUTION_BEGIN */
  // Throw error in case no triangular cell
  LF_VERIFY_MSG(tria.RefEl() == lf::base::RefEl::kTria(),
                "Unsupported cell type " << tria.RefEl());
  // Obtain vertex coordinates of the triangle in a 2x3 matrix
  const auto corners{lf::geometry::Corners(*(tria.Geometry()))};
  // Compute the scaling factor for the local load vector
  const double area_third = lf::geometry::Volume(*(tria.Geometry())) / 3.0;
  LF_ASSERT_MSG((corners.cols() == 3) && (corners.rows() == 2),
                "Invalid vertex coordinate " << corners.rows() << "x"
                                             << corners.cols() << " matrix");
  ElemVec = Eigen::Vector3d(area_third * f_(corners.col(0)),
                            area_third * f_(corners.col(1)),
                            area_third * f_(corners.col(2)));
  /* SOLUTION_END */
  return ElemVec;
}  // TrapRuleLinFEElemVecProvider<FUNCTOR>::Eval
```

Element vector

Study the documentation of **lf::assemble::COOMatrix** and try to understand the following function, which follows the implementation of `lf::assemble::fix_flagged_solution_components()`.

h)

C++11 code 6.1.21: Function `dropMatrixRowsColumns` for the isolation of solution components

```
template <typename SCALAR, typename SELECTOR>
void dropMatrixRowsColumns(SELECTOR &&selectvals,
                           COOMatrix<SCALAR> &A) {
  const lf::assemble::size_type N(A.cols());
  LF_ASSERT_MSG(A.rows() == N, "Matrix must be square!");
  A.setZero([&selectvals](gdof_idx_t i, gdof_idx_t j) {
    return (selectvals(i).first || selectvals(j).first);
  });
  for (lf::assemble::gdof_idx_t dofnum = 0; dofnum < N; ++dofnum) {
    const auto selval{selectvals(dofnum)};
    if (selval.first) {
      A.AddToEntry(dofnum, dofnum, 1.0);
    }
  }
}
```

▷ Sets all off-diagonal entries of selected rows/columns to zero

▷ Sets corresponding diagonal entries → 0

h) Initialization of timestepper class

```
class Radau3MOLTimestepper {
public:
  Radau3MOLTimestepper() = delete;
  Radau3MOLTimestepper(const Radau3MOLTimestepper &) = delete;
  Radau3MOLTimestepper(Radau3MOLTimestepper &&) = delete;
  Radau3MOLTimestepper &operator=(const Radau3MOLTimestepper &) =
      delete;
  Radau3MOLTimestepper &operator=(Radau3MOLTimestepper &&) = delete;

  explicit Radau3MOLTimestepper(
      const lf::assemble::DofHandler &dofh);
  virtual ~Radau3MOLTimestepper() = default;

  Eigen::VectorXd discreteEvolutionOperator(double tau, double time,
      const Eigen::VectorXd &mu) const;
  // plus appropriate data
};
```

Precompute & store $A_\otimes$, $M_\otimes$ after assembly of A, M & removing "d.o.f. on $\partial\Omega$".

⑤

j)

```
class Radau3MOLTimestepper {
public:
  Radau3MOLTimestepper() = delete;
  Radau3MOLTimestepper(const Radau3MOLTimestepper &) = delete;
  Radau3MOLTimestepper(Radau3MOLTimestepper &&) = delete;
  Radau3MOLTimestepper &operator=(const Radau3MOLTimestepper &) =
     delete;
  Radau3MOLTimestepper &operator=(Radau3MOLTimestepper &&) = delete;

  explicit Radau3MOLTimestepper(
     const lf::assemble::DofHandler &dofh);
  virtual ~Radau3MOLTimestepper() = default;

  Eigen::VectorXd discreteEvolutionOperator(double tau, double time,
     const Eigen::VectorXd &mu) const;
  // plus appropriate data
};
```

$\leftrightarrow$ $\Psi^{t,t+\tau}\vec{\mu}$

$$\begin{bmatrix} \mathbf{M}+\tau\frac{5}{12}\mathbf{A} & \tau\frac{-1}{12}\mathbf{A} \\ \tau\frac{3}{4}\mathbf{A} & \mathbf{M}+\tau\frac{1}{4}\mathbf{A} \end{bmatrix} \begin{bmatrix} \vec{\kappa}_1 \\ \vec{\kappa}_2 \end{bmatrix} = \begin{bmatrix} \vec{\varphi}(t_j+\frac{1}{3}\tau) - \mathbf{A}\vec{\mu}^{(j)} \\ \vec{\varphi}(t_j+\tau) - \mathbf{A}\vec{\mu}^{(j)} \end{bmatrix}. \tag{6.1.15}$$

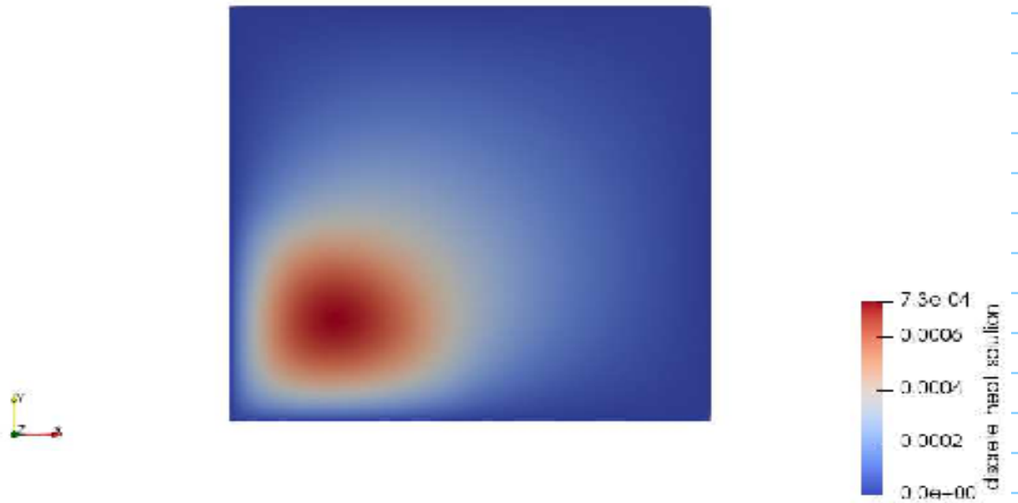
$M_\otimes + \tau A_\otimes$ ← compute through Eigen's sparse matrix arithmetic

← then solve LSE

$\vec{\varphi}$ has to be compiled anew in each timestep.

▽ No uniform timestep τ assumed.

l) Visualize solution at final time



m) Elaborate how the implementation of the 2-stage Radau RK-SSM timestepping for the method-of-linear ODE for (6.1.1) can be made more efficient when a uniform timestep is used.

↓

τ the same $\forall$ timesteps

▷ In the constructor precompute $M_\otimes + \tau A_\otimes$, & pre-LU factor it.