Series 6

ETH zürich

Computational Methods for
Engineering Applications
Last edited: November 30, 2020
Due date: Dec 13 at 23:59

Template codes are available on the course's webpage at `https://moodle-app2.let.ethz.ch/course/view.php?id=13412`.

Exercise 1 Transient heat equation in 1D

We consider the following one-dimensional, time dependent heat equation:

$$\frac{\partial u}{\partial t}(x,t) - \frac{\partial^2 u}{\partial x^2}(x,t) = 0, \qquad (x,t) \in (0,1) \times (0,T), \tag{1}$$

$$u(0,t) = g_L(t), \quad u(1,t) = g_R(t), \qquad t \in [0,T], \tag{2}$$

$$u(x,0) = u_0(x), \qquad x \in [0,1], \tag{3}$$

where $T > 0$ is the final time, and $g_L, g_R : [0,T] \longrightarrow \mathbb{R}$ are Dirichlet boundary conditions.

We first discretize the above equation with respect to the spatial variable, using *centered finite differences*.

To this aim, we subdivide the interval $[0,1]$ in $N+1$ subintervals of equal length, where N is the number of *interior* grid points $x_1, \dots, x_N$, and $x_0 = 0$, $x_{N+1} = 1$.

The space discretization leads to a *semidiscrete* system of equations associated to (1):

$$\frac{\partial \boldsymbol{u}}{\partial t}(t) + \mathbf{A}\boldsymbol{u}(t) = \boldsymbol{G}(t), \tag{4}$$

where $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\boldsymbol{u} = \{u_i\}_{i=1}^N$ denotes the approximate values of the solution at the interior grid points. $\boldsymbol{G} : [0,T] \longrightarrow \mathbb{R}^N$ is a source term coming from the boundary conditions.

Hint: $\boldsymbol{G}$ appears from the fact that the discretization for u_1 and u_N includes respectively $u_0 = g_L(t)$ and $u_{N+1} = g_R(t)$

1a)

Denote by h the mesh width, that is $h = \frac{1}{N+1}$. Write down the matrix $\mathbf{A}$ and the vector $\boldsymbol{G}(t)$ explicitly.

To fully discretize (1), we still need to apply a time discretization to (4).

1b)

Apply the *forward Euler* scheme to (4), denoting by $\boldsymbol{u}^k = \left\{u_i^k\right\}_{i=1}^N$ the approximate value of the vector $\boldsymbol{u}$ at time k, for $k = 0, \ldots, K$, and by $\Delta t = \frac{T}{K}$ the time step. How does the update formula at each time step look like?

1c)

In the template file heat_1dfd.cpp, implement the function

```
void createPoissonMatrix(SparseMatrix& A, int N),
```

where `typedef Eigen::SparseMatrix<double> SparseMatrix`. This function computes the matrix $\mathbf{A}$ from (4). Here the input parameter N denotes the number of *interior* grid points. Assume that the size of the input matrix A has not been initialized.

Hint: You can copy the routine directly from the solution to an old assignment and do very small modifications to obtain the desired matrix!

1d)

In the template file heat_1dfd.cpp, implement the function

```
void explicitEuler(Eigen::MatrixXd  & u, Vector & time, const Vector u0, double dt, double T,
                   int N, const std::function<double(double)>& gL,
                   const std::function<double(double)>& gR)
```

(with `typedef Eigen::VectorXd Vector`). The input and output parameters are specified in the template file.

1e)

With the help of the script `sol_movie.m` provided in the handout, observe a movie of the approximate solution to (1) when using the forward Euler scheme. Set the parameters to $T = 0.3$, $\Delta t = 0.0002$, $N = 40$ and $u_0(x) = 1 + \min(2x, 2 - 2x)$ the hat function, $x \in [0, 1]$. Take $g_L(t) = g_R(t) = \exp(-10t)$. What happens to the energy of the system? How does it change if $g_L(t) = 0$, $g_R(t) = 0$? And if $g_L(t) = 1$, $g_R(t) = 0$?

1f)

We now consider an implicit timestepping. Namely, we derive the Crank-Nicolson scheme. Start with the semidiscrete formulation (4) and integrate over $[t^k, t^{k+1}]$. Use the trapezoidal rule for the integrals involving $\mathbf{A}\boldsymbol{u}$ and $G(t)$, and the approximation $\boldsymbol{u}^k \approx \boldsymbol{u}(t^k)$. Write down the system of equations to be solved at each timestep (this should agree with the Crank-Nicolson scheme stated in the script).

1g)

In the template file heat_1dfd.cpp, implement the function

```
void CrankNicolson(Eigen::MatrixXd & u, Vector & time, const Vector u0, double dt, double T,
int N)
```

(with `typedef Eigen::VectorXd Vector`). The input and output parameters are specified in the template file.

Hint: In this exercise, you may want to compute $I - M$, where M is a certain sparse matrix and I is the identity. Due to Eigen typecasting, if I is not explicitly defined as a sparse matrix (e.g. it is generated with Eigen::MatrixXd::Identity), $I - M$ will not be a sparse matrix, and sparse solvers will not work. There are several ways to go around this; a simple one is to define I as sparse too with:

```
SparseMatrix I(N,N);
I.setIdentity();
```

1h)

With the help of the script `sol_movie.m` provided in the handout, observe a movie of the approximate solution to (1) when using the Crank-Nicolson timestepping scheme. Set the parameters as in subproblem **1e)**. Concerning the energetic behavior of the system, you should observe the same qualitative behavior as in subproblem **1h)**.

1i)

Compute an approximate solution to (1) with both the forward Euler and the Crank-Nicolson schemes. Set the parameters to $T = 0.3$, $N = 20$, $\Delta t = 0.001$ and $u_0(x) = 1 + \min(2x, 2 - 2x)$, $x \in [0, 1]$, $g_L(t) = g_R(t) = \exp(-10t)$. Use now the script `sol_movie.m` provided in the handout to observe the movie for each of the two solutions. Repeat the experiment with $N = 20$, $\Delta t = 0.01$ and with $N = 5$, $\Delta t = 0.01$. What do you observe?

1j)

Give an explanation for the observations from subproblem **1i)**. Which condition has to be fulfilled by Δt when using the explicit Euler scheme?

Exercise 2 Linear transport equation in 1D

Consider the linear transport equation in one dimension with initial data u_0:

$$\frac{\partial u}{\partial t}(x,t) + a(x)\frac{\partial u}{\partial x}(x,t) = 0, \qquad (x,t) \in (x_l, x_r) \times \mathbb{R}, \tag{5}$$

$$u(x,0) = u_0(x), \qquad x \in [x_l, x_r], \tag{6}$$

with $a : \mathbb{R} \longrightarrow \mathbb{R}$. We neglect boundary conditions for now.

2a)

Derive the equation for the characteristics. Assuming $a(x) = \frac{1}{2}$, draw manually or produce a plot of the characteristic lines in the (x,t)-plane. How would they look for $a(x) = sin(2\pi x)$?

Hint: For the second question, you should use a numerical ODE solver.

2b)

Explain why the solution u to (5) is constant along the characteristics.

We now want to compute an approximate solution to (5). For time discretization, we will always use the *forward Euler* scheme, while for space discretization we consider two different finite difference schemes: centered, and upwind.

2c)

In the template file linear_transport.cpp, implement the function

```
void CenteredFD(Eigen::MatrixXd & u, Vector & time, const Vector u0, double dt, double T,
                int N, const std::function<double(double)>& a),
```

that computes the approximate solution to (5) using the *forward Euler* scheme for time discretization and *centered finite differences* for space discretization. The arguments of the function CenteredFD are specified in the template file. The input argument N denotes the number of grid points *including the boundary points*, i.e. `x[0] = xL`, `x[N-1] = xR`. Take $x_l = 0$, $x_r = 5$.

Outflow boundary conditions are a simple way of modeling a larger physical domain. For a given discretization $x_l = x_1 < x_2 < \cdots < x_N = x_r$, we consider two additional **ghost points**: x_0 and x_{N+1}. We consider that at every time step n, $u_0^n = u_1^n$ and $u_{N+1}^n = u_N^n$. This allows information to flow out of the domain, but not in. Assume outflow boundary conditions.

2d)

In the template file linear_transport.cpp, implement the function

```
void UpwindFD(Eigen::MatrixXd & u, Vector & time, const Vector u0, double dt, double T,
              int N, const std::function<double(double)>& a, double xL, double xR)
```

that computes the approximate solution to (5) using the *forward Euler* scheme for time discretization and *upwind finite differences* for space discretization. The arguments of the function UpwindFD are specified in the template file. The input argument N denotes the number of grid points *including the boundary points*, i.e. `x[0] = xL`, `x[N-1] = xR`. Take $x_l = 0$, $x_r = 5$.

No restriction is imposed on the sign of velocity function a; your code **must** be able to handle positive and negative values.

Assume again outflow boundary conditions.

2e)

Run the function main contained in the file linear_transport.cpp. As input parameters, set: $T = 2$, $N = 101$, $\Delta t = 0.002$ and $a(x) = 2 + sin(2\pi x)$. The initial condition has been set to

$$u_0(x) = \begin{cases} 0 & \text{if } x < 0.25 \text{ or } x > 0.75 \\ 2 & \text{if } 0.25 \leq x \leq 0.75. \end{cases}$$

Use the file `sol_movie.m/.py` to observe movies of the solutions obtained using upwind finite differences, and centered differences. Repeat the same using now a velocity with changing signs, $a(x) = sin(2\pi x)$. Answer the following questions:

- The solutions obtained with which finite difference schemes make sense?
- Based on physical considerations, explain the reason why some schemes fail to give a meaningful solution.
- For the schemes that work, what happens to the energy of the system?

2f)

Run the function main contained in the file linear_transport.cpp using $\Delta t = 0.002$, $\Delta t = 0.01$, $\Delta t = 0.015$ and $\Delta t = 0.05$, and the other parameters as in the previous subtask (with $a = \sin(2\pi x)$). Running the routine `sol_movie.m`, observe the results that you obtain in the four cases when using the upwind finite difference scheme. You can see that in some cases the solution is meaningful, while in the others the energy explodes and the solution is unphysical. Why does this happen? Which condition should the time step Δt fulfill in order to have stability?

2g)

We want to extend equation 5 to include an additional source term f:

$$\frac{\partial u}{\partial t}(x,t) + a(x)\frac{\partial u}{\partial x}(x,t) = f(x,t), \qquad (x,t) \in (x_l, x_r) \times \mathbb{R}, \tag{7}$$

$$u(x,0) = u_0(x), \qquad x \in [x_l, x_r], \tag{8}$$

$f : (x_l, x_r) \times \mathbb{R} \to \mathbb{R}$ represents a known source (or sink) of u. Characteristic curves are still defined as the curves which verify the ODE $x'(t) = a(x(t), t)$.

Is the solution to eq. 7 constant along characteristic curves?

2h)

In the template file linear_transport.cpp, implement the function

```
void UpwindFDwithSource(Eigen::MatrixXd & u, Vector & time, const Vector u0, double dt, double T,
                        int N, const std::function<double(double)>& a, double xL, double xR,
                        const std::function<double(double,double)>& f)
```

that computes the approximate solution to (7), including the source term. All the considerations in exercise **2c)** apply.

2i)

Test your function UpwindFDwithSource with the following data:

- $a(x) = 1 + 8\exp(-\frac{x}{2})$
- $f(x,t) = 200\ \exp(-1000(x-1)^2)\sin(2\pi t)^{16}$
- $x_l = 0$, $x_r = 5$, $T = 2$.
- $u_0(x)$ as in task **2e)**

This provides a simple model of a dam break over the course of an (initially dry) river. The slope of the river, and thus water velocity, is highest at the origin, and lowest at the end, following an exponential profile. The riverbed is also subjected to periodic rains in the upper part of its course. Here u represents water height in arbitrary units.