S. Mishra
K. O. Lye
L. Scarabosio

Autumn Term 2015

Computational Methods for Engineering Applications II

ETH Zürich
D-MATH

Homework Problem Sheet 5

This assignment sheet contains some tasks marked as **Core problems**. If you hand them in (see deadline at the end of the problem sheet), these tasks will be corrected. Full mark for the total of the core problems in all assignments will give a 20% bonus on the total points in the final exam. This is really a bonus, which means that at the exam you can still get the highest grade without having the bonus points (of course then you need to score more points at the exam).

The total number of points for the Core problems of this sheet is **3 points**. (The total number of points over all assignments is 20).

Problem 5.1 Transient heat equation in 1D

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

$$\frac{\partial u}{\partial t}(x,t) - \frac{\partial^2 u}{\partial x^2}(x,t) = 0, \qquad (x,t) \in (0,1)\times(0,T), \tag{5.1.1}$$

$$u(0,t) = u(1,t) = 0, \qquad t \in [0,T], \tag{5.1.2}$$

$$u(x,0) = u_0(x), \qquad x \in [0,1], \tag{5.1.3}$$

where $T > 0$ is the final time.

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 (5.1.1):

$$\frac{\partial \boldsymbol{u}}{\partial t}(t) + \mathbf{A}\boldsymbol{u}(t) = 0, \tag{5.1.4}$$

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.

(5.1a) Denote by h the mesh width, that is $h = \frac{1}{N+1}$. Write down the matrix $\mathbf{A}$ explicitly.

To fully discretize (5.1.1), we still need to apply a time discretization to (5.1.4).

(5.1b) Apply the *forward Euler* scheme to (5.1.4), 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 $k = 0, \dots, K$, and by $\Delta t = \frac{T}{K}$ the time step. How does the update formula at each time step look like?

(5.1c) 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 (5.1.4). 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!

(5.1d) (**Core problem**) 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)
```

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

(5.1e) With the help of the script `sol_movie.m` provided in the handout, observe a movie of the approximate solution to (5.1.1) when using the forward Euler scheme. Set the parameters to $T = 0.3$, $\Delta t = 0.0002$, $N = 40$ and $u_0(x) = \min(2x, 2 - 2x)$ the hat function, $x \in [0, 1]$. What happens to the energy of the system?

(5.1f) We now consider an implicit timestepping. Namely, we derive the Crank-Nicolson scheme. Start with the semidiscrete formulation (5.1.4) and integrate over $[t^k, t^{k+1}]$. Use the trapezodial rule for the integral involving $\mathbf{A}\boldsymbol{u}$ 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).

(5.1g) (**Core problem**) 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.

(5.1h) With the help of the script `sol_movie.m` provided in the handout, observe a movie of the approximate solution to (5.1.1) when using the Crank-Nicolson timestepping scheme. Set the parameters as in subproblem (5.1e). Concerning the energetic behavior of the system, you should observe the same qualitative behavior as in subproblem (5.1h).

(5.1i) Compute an approximate solution to (5.1.1) with both the forward Euler and the Crank-Nicolson schemes. Set the parameters to $T = 0.3$, $N = 20$, $\Delta t = 0.001$ and $u_0(x) = \min(2x, 2 - 2x)$, $x \in [0, 1]$. 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?

(5.1j) Give an explanation for the observations from subproblem (5.1i). Which condition has to be fulfilled by Δt when using the explicit Euler scheme?

Problem 5.2 Linear transport equation in 1D

Consider the linear transport equation in one dimension with *periodic boundary conditions* and initial data u_0:

$$\frac{\partial u}{\partial t}(x,t) + a\frac{\partial u}{\partial x}(x,t) = 0, \qquad (x,t) \in (0,1) \times \mathbb{R}, \tag{5.2.1}$$

$$u(0,t) = u(1,t), \quad \frac{\partial u}{\partial x}(0,t) = \frac{\partial u}{\partial x}(1,t), \qquad t \in \mathbb{R}, \tag{5.2.2}$$

$$u(x,0) = u_0(x), \qquad x \in [0,1], \tag{5.2.3}$$

with $a \in \mathbb{R}$.

(5.2a) Derive the equation for the characteristics. Assuming $a = \frac{1}{2}$, draw manually or produce a plot of the characteristic lines in the (x,t)-plane.

(5.2b) Explain why the solution u to (5.2.1) is constant along the characteristics. Would this still be true if the right-hand side in (5.2.1) is not zero?

We now want to compute an approximate solution to (5.2.1). For time discretization, we will always use the *forward Euler* scheme, while for space discretization we consider three different finite differences: upwind, central and downwind finite differences.

(5.2c) (**Core problem**) 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, double a),
```

that computes the approximate solution to (5.2.1) 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. Pay attention that this time the input argument N denotes the number of grid points *including the boundary points*.

(5.2d) In the template file linear_transport.cpp, implement the function

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

that computes the approximate solution to (5.2.1) using the *forward Euler* scheme for time discretization and *downwind finite differences* for space discretization. The arguments of the function DownwindFD are specified in the template file.

(5.2e) 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, double a),
```

that computes the approximate solution to (5.2.1) using the *forward Euler* scheme for time discretization and *centered finite differences* for space discretization. The arguments of the function DownwindFD are specified in the template file.

(5.2f) Run the function main contained in the file linear_transport.cpp. As input parameters, set: $T = 2$, $N = 101$, $\Delta t = 0.02$ and $a = 1$. 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` to observe movies of the solutions obtained using upwind finite differences, downwind finite differences and centered differences. Repeat the same using now a negative velocity $a = -1$. 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?

(5.2g) Run the function main contained in the file linear_transport.cpp using $\Delta t = 0.002$, $\Delta t = 0.01$, $\Delta t = 0.011$ and $\Delta t = 0.05$, and the other parameters as in the previous subtask (with $a = 1$). 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?

Problem 5.3 Upwind finite differences for Burgers' equation in 1D

We consider the so-called Burgers' equation with Dirichlet boundary conditions:

$$\frac{\partial u}{\partial t}(x,t) + \frac{\partial}{\partial x}\left(\frac{u^2(x,t)}{2}\right) = 0, \qquad (x,t) \in (0,1) \times \mathbb{R}, \tag{5.3.1}$$

$$u(0,t) = u_0(0), \qquad t \in \mathbb{R}, \tag{5.3.2}$$

$$u(0,x) = u_0(x), \qquad x \in [0,1], \tag{5.3.3}$$

with some initial data $u_0 = u_0(x) \geq 0$. At the right boundary $x = 1$, we use a non-reflecting boundary condition.

Burgers' equation is an example of a *nonlinear* hyperbolic partial differential equation.

(5.3a) Burgers' equation is a transport equation with nonconstant velocity $a = a(u)$, where u is the solution to (5.3.1). Explain this statement and write explicitly the velocity at which waves travel.

(5.3b) Consider an equispaced grid on the interval $[0,1]$. Furthermore, assume that $u_0(x) \geq 0$ for every $x \in [0,1]$. Then the upwind finite difference scheme with forward Euler time stepping reads:

$$\frac{u_j^{k+1} - u_j^k}{\Delta t} + \frac{(u_j^k)^2 - (u_{j-1}^k)^2}{2\Delta x} = 0, \quad j = 1, \ldots, N, k = 0, \ldots, K-1, \tag{5.3.4}$$

(for some $K \in \mathbb{N}$) where Δt denotes the time step, Δx denotes the mesh width, and u_j^k is the approximate value of the solution at time $k\Delta t$ at the point $j\Delta x$, $j = 0, \ldots, N$, $k = 0, \ldots, K$.

Complete the template file burgers_upwind.cpp implementing the function

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

that computes the approximate solution to (5.3.1) using the scheme given in (5.3.4). The arguments of the function BurgersUpwind are specified in the template file.

(5.3c) Run the function main contained in the template file burgers_upwind.cpp with the following parameters: final time $T = 2$, time step $\Delta t = 0.001$, $N = 101$ gridpoints (including those on the boundary) and initial data

$$u_0(x) = \begin{cases} 1 - 2x & \text{if } x \leq \frac{1}{2} \\ 0 & \text{else,} \end{cases} \tag{5.3.5}$$

already implemented in the template file as U0_s. Use the routine `sol_movie.m` provided in the handout to see a movie of the solution. Describe the behavior observed.

(5.3d) Repeat the experiment of subproblem (5.3c), now with the initial data

$$u_0(x) = \begin{cases} 0 & \text{if } x \leq \frac{1}{2} \\ 1 & \text{else,} \end{cases} \tag{5.3.6}$$

already implemented in the template file as U0_r (all the other parameters as in the previous subproblem). Describe the behavior observed in this case.

Published on December 9.

To be submitted on December 16.

Last modified on December 10, 2015