Series 3 Warmup

Numerical methods for PDEs
Last edited: April 7, 2017
Due date: None at 23:59

Template codes are available on the course's webpage at `https://moodle-app2.let.ethz.ch/course/view.php?id=3089`.

This is a warmup problem. You do **NOT need to hand in** this problem.

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 $x_1, \ldots, x_N$, 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?