Universität Konstanz *WS 11/12*
Fachbereich Mathematik und Statistik
S. Volkwein, O. Lass

Übungen zu **Modellreduktion mit Proper Orthogonal Decomposition**

http://www.math.uni-konstanz.de/numerik/personen/volkwein/teaching/

Submission: 18.11.2011, 10:00 o'clock
Codes by E-Mail and Reports together with the Homework

Program 1 (6 Points)

Given the linear heat equation

$$\begin{aligned} \dot{y}(x,t) &= c\Delta y(x,t) && \text{in} && \Omega \times (0,1], \\ y(x,t) &= 0 && \text{on} && \partial\Omega \times (0,1], \\ y(x,0) &= y_0 && \text{in} && \Omega, \end{aligned} \tag{1}$$

where $\Omega = [0,1] \times [0,1]$. Discretize (1) in space by using the finite difference method following the results obtained in Exercise 2. For the time discretization utilize the following method:

- implicit Euler method (IE)
- Crank Nicolson scheme (CN)
- Rannacher smoothing (RS), i.e. four half implicit Euler steps ($\Delta t/2$) followed by regular Crank Nicolson steps.

For simplicity we use equidistant timesteps in time. Structure your code as follows:

`main` ... main script file where all parameters are set and the solution is plotted.

`[A,h,X1,X2] = preparation(n)` ... Given the parameter n, number of inner points, this function return the discretization of the Laplace operator, the discretization size h and the discretization grid `X1` and `X2` (including boundary points).

`[Y,t] = solve_heat_fdm(c,A,h,tstep,Y0,method)` ... Solves the heat equation (1). The variables are `c` the diffusion coefficient, `tstep` the number of time steps, `Y0` the vector of the initial condition on the inner points and `method` to select a solver (`'IE'`,`'CN'`,`'RS'`). The return values are a matrix `Y` with columns containing the initial condition and solution to (1) in the inner points and `time` a vector containing the timesteps.

`YFDM = add_boundary(Y)` ... adds the boundary to the solution `Y`.

Do not introduce any further functions or variables and follow the syntax exactly and document your code well. Visualize the solution to your like. To test your code choose `n`$= 100$, `tsteps`$= 100$ and the following settings for c and y_0:

- $c = 0.01$ and $y_0(x) = \sin(2\pi x_1)\sin(\pi x_2)$

- $c = 0.05$ and $y_0(x) = \begin{cases} 1, & \text{on } (0.25, 0.75) \times (0.25, 0.75), \\ 0, & \text{otherwise.} \end{cases}$

- $c = 0.5$ and $y_0(x) =$ `1*(rand(size(x1)) < 0.001)`

How does the performance of the three methods differ? What do you observe? Submit your code as a ZIP or TAR/ZIP archive containing one folder named `your_lastname_prog01`.