Prof. Dr. Karsten Urban
Kristina Steih

Institute for Numerical Mathematics
Ulm University
Summer Term 2014

Numerical Finance – Sheet 11

(Exercise Class July 23th, 2014)

Programming Exercise 1: Parabolic FEM 2D **(16 points)**

Solve the 2D heat equation

$$\begin{aligned} u_t - \Delta u &= f && \text{on } \Omega = (0,1), \\ u(t,x) &= g(t,x), && \text{on } \Gamma = \partial\Omega, \\ u(0,x) &= u_0(x), && \forall\, x \in \Omega. \end{aligned}$$

using linear Finite Elements in space. Unlike in 1D, where we have first used the simplified structure *SimpleParabolicFEM1D*, we will now in 2D consider directly the full structure `ParabolicFEM2D` that allows a time-dependent assembly of stiffness matrix and right-hand side (compare Prog. Ex. 3 on Sheet 9).
Hence, you will find the following on the homepage:

- `TimedepPDE2D`: An extension of the class `PDE2D` that has the member `current_time` and the function `set_time()` to change this variable.
- `ParabolicFEM2D`: The FEM model class that can solve parabolic PDEs using a general θ-scheme. As in 1D, we can now assemble time-dependent systems $A = A_h(t)$ and $F = b_h(t)$ by calling in each time step

  ```
  pde2d.set_time(t);
  assemble_A();
  ```

- As in 1D, our PDE problem is now called `TimedepPoisson2D` and is derived from `TimedepPDE2D`. Note that all that has changed are the signatures of the internal functions pointers for f and g – these functions now take two parameters (time and space). They are called with the member variable `current_time` and the argument `x`.
  Hence, you can copy everything you have done in `Poisson2D` and only have to adapt the function calls of `f` and `d_bc`.

Complete the missing parts, so that you can solve a general θ-scheme.

Use your implementation to solve the heat equation for

a) $f(t,x) = 1$, $g(t,x) = 0$, $u_0(x) = 0$.

b) $f(t,x) = 10$, $g(t,x) = (1-t)\|x\|_1$, $u_0(x) = \|x\|_1$.

Visualize your solutions $u = u(t,x)$.

Hints:

- ***Mass matrix assembly:*** The mass matrix M_h is again assembled completely inside the function `ParabolicFEM2D::assemble_M()`. As for the stiffness matrix, this assembly is now done on the reference element, using the formula for integration by substitution:

$$M^{(m)}_{x_r^{(m)},x_i^{(m)}} = \int_{T_m} \psi_{x_r^{(m)}}(x)\psi_{x_i^{(m)}}(x)dx = \int_{\hat{T}} N_r(\hat{x})N_i(\hat{x})|\det S|d\hat{x}.$$

  The integrals can again be computed by hand. Keep in mind that you only have an entry $M_{i,j}$ if neither x_i nor x_j are on the Dirichlet boundary.

- ***Sparse Matrix:*** The sparse matrix class has been extended. Look at *sparsematrix.h* and *operators.h* to find the additional operators and functions (e.g. for +, =, multiplication with a scalar, clear the matrix data etc.).

- ***Plot script:*** In order to visualize the solution, you might want to plot a sequence of 3D plots. On the homepage, you find an example for a gnuplot script that produces such a *gif* file from data that has been written by the function `ParabolicFEM2D::write_solution_on_grid`. In addition to the data file name, you have to specify the number of timesteps and elements in the grid, as well as the plot ranges (in the square brackets). This script only works with gnuplot version 4.4 or newer.
  Usage: Either call directly from the command line
  `gnuplot < plot_u_2d_parab.gp`
  or start gnuplot and then call
  `gnuplot> load ‘‘plot_u_2d_parab.gp’’`

Programming Exercise 2: American Options 1D (19 points)

Compute the value of an American Put option with $\sigma = 0.6$, $r = 0.06$, $T = 1$, $K = 12$. Recall that this means solving the inequality

$$V_t(S,t) + \frac{\sigma^2}{2}S^2V_{SS}(S,t) + rSV_S(S,t) - rV(S,t) \leq 0$$

with $V \geq (K-S)$. As before, we can transform this to the heat inequality using the transformations

$$S = Ke^x, \quad t = T - \frac{\tau}{\frac{1}{2}\sigma^2}, \quad q = \frac{2r}{\sigma^2}, \quad v(\tau,x) := V\left(T - \frac{2\tau}{\sigma^2}, Ke^x\right)$$

$$y(x,\tau) = \frac{1}{K}\exp\left(\tfrac{1}{2}(q-1)x + \left(\tfrac{1}{4}(q-1)^2 + q\right)\tau\right)v(\tau,x),$$

This leads to the linear complementary problem

$$\begin{aligned} y_\tau - y_{xx} &\geq 0, \\ y - g &\geq 0 \\ (y_\tau - y_{xx})(y-g) &= 0 \\ y(x,0) &= g(x,0), \\ \lim_{x\to\pm\infty} y(\tau,x) &= \lim_{x\to\pm\infty} g(\tau,x), \end{aligned}$$

where

$$g(\tau,x) = \exp\left\{\frac{1}{4}\big((q-1)^2+4q\big)\tau\right\}\max\left\{e^{\frac{1}{2}(q-1)x} - e^{\frac{1}{2}(q+1)x}, 0\right\} \qquad \text{(Put)},$$

$$g(\tau,x) = \exp\left\{\frac{1}{4}\big((q-1)^2+4q\big)\tau\right\}\max\left\{e^{\frac{1}{2}(q+1)x} - e^{\frac{1}{2}(q-1)x}, 0\right\} \qquad \text{(Call)}.$$

To implement this, use Programming Exercise 3 from Sheet 9 as a starting point. From there, you will have to do the following steps:

- Implement the projected SOR method to solve the linear complementary problem. In contrast to the cg or gmres implementation. you can assume that the SOR algorithm handles only sparse matrices. Recall that you cannot use the ()-operator on sparse matrices without knowing if there ist a matrix entry, since you would create a matrix a full matrix in the process. Implement the SOR algorithm such that it can handle sparse matrices of arbitrary structure!

- Copy the class `parabolicfem1d.h` and name it `parabolicineqfem1d.h`. For the linear complimentary problem, we have to make minor changes in the class:
  - Add the time-dependent obstacle function $g(\tau, x)$.
  - Adjust the solve-routine, using e.g. $\omega = 1.25$.

- Adjust the main program.

- Transform the solution back into Black-Scholes coordinates and plot the obstacle, the solution $V(t, S)$ and the contact point into one plot.