Computer exercise C2b

(5 % of Final Grade)

May 2024

Deadlines: see the webpage of the course.
(This is an individual assignment!)

Please choose between PART A and PART B!

PART A

Based on Lecture 13, May 8 by Felix Lucka (CWI) on Inverse PDE problems.

In PART A you are going to solve an inverse problem for the one-dimensional heat-squation:

$$u_t(x,t) = \kappa\ u_{xx}(x,t),, \quad \kappa > 0$$

with boundary conditions $u(0,t) = u(1,t) = 0$ and initial condition $u(x,0) = u_0(x)$. The goal is to estimate $u_0(x)$ given measurements $d(x) = u(x,T)$ for some given T. For this purpose, first, discretize the heat equation using central differences in space and forward in time. You may wish to change the parameters (step sizes and final time) in the displayed Matlab-code in Figure 1.

(a) Discretize the heat equation using central differences in space and forward in time. Give an expression for the resulting recursion for $\mathbf{u}_n$.

(b) Formulate a least-squares data-fitting problem for finding the initial condition $\mathbf{u}_0$ and give an expression for its gradient. *Hint: formulate the objective function as* $\phi(\mathbf{u}_0) = \|Q\mathbf{u}_0 - \mathbf{d}\|_2^2$.

(c) Compute the solution using a steepest-descent iteration

$$\mathbf{u}_0^{k+1} = \mathbf{u}_0^k - \alpha \nabla \phi(\mathbf{u}_0^k),$$

for the data that is generated by the accompanying Matlab script; see the code in Figure (you could create a Python version in a similar way):

```
% parameters of the setting
T     = 1;
kappa = 1;

% discretization
dx    = 1e-2;
dt    = 1e-2;
x     = (dx:dx:(1-dx))';
n     = length(x);

% initial condition
u0    = exp(-1e2 * (x-0.5).^2);

% discrete Laplacian
L = spdiags(ones(n,1) * [1,-2,1]/dx^2, [-1:1], n, n);

% Forward Euler
u = u0;
for k=1:ceil(T/dt)
   u = u + dt * kappa * L * u;
end

% get data
d = u;
```

Figure 1: A Matlab version of a typical code for PART A.

PART B

Based on Lecture 14, May 15 by Chiheb ben Hammouda on stochastic DE/PDEs.

The geometric Brownian motion S solves

$$\begin{aligned} dS(t) =& rS(t)dt + \sigma S(t)dW(t), \\ S(0) =& S_0, \end{aligned} \tag{1}$$

The exact solution for (1) is: $S(t) = \exp\left((r - \sigma^2/2)t + \sigma W(t)\right)\ S_0$.

1. Compute a numerical approximation to the option value:

$$f(0, S_0) \equiv e^{-rT} E[g(S(T)) \mid S(0) = S_0],$$

by the Monte Carlo method (you may start with the attached initial code: `price_MC.m`) for the following cases:

- Case 1: $g(x) = \max(x - K, 0)$
- Case 2: $g(x) = \begin{cases} 1, & \text{if } \frac{K}{2} < x < K, \\ 0 & \text{otherwise,} \end{cases}$

Here we use $r = 0.04$, $\sigma = 0.4$, $T = \frac{1}{4}$ and $S_0 = K = 100$. Use TOL $= 0.1$ when controlling the error. For verification purposes, you may compare your error estimates with the true computational error. We have an exact solution for the two cases

(Case 1: $f(0, S_0) = 8.4333$, Case 2: $f(0, S_0) = 0.5145$).

2. Solve *Question* 1 by instead approximating the corresponding Black-Scholes PDE.

$$\begin{aligned} &\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{\sigma^2 S^2}{2}\frac{\partial^2 f}{\partial S^2} - rf = 0, \\ &f(T, S) = g(S). \end{aligned} \tag{2}$$

You may use and modify when necessary the attached Matlab Finite Difference program `fd1d.m`.

a Determine the option values with a precise accuracy $TOL = 10^{-k}$, $k = 1, 2, \ldots$. Explain how you estimate the numerical error, work with uniform discretizations to control it. Keep records of the Δt and Δs used, the amount of computational work and compare these results with the corresponding Monte Carlo ones.

(i) Perform the task using Backward Euler for the time stepping.

(ii) **(Bonus)** Perform the task using Crank-Nicholson[1] for the time stepping.

[1] See Exercise 4.1.

Matlab script: fd1d.m

```
% Finite difference schemes applied to european
% option pricing on a 1 asset BS model.
% Uses uniform meshes only.

% dS = r*S*dt + sigmap*S*dW
% Solves the parabolic Black-Scholes PDE
% D_t U + r*s D_s + 0.5*(sigmap*s)^2 * (D_s)^2 U - r* U= 0
% U(T,.) = payoff(.)
% at s = 0 we have a Dirichlet BC, U(t,0) = payoff(0)*exp(-r*(T-t))
% and at s = smax we have a non reflecting BC.
%--------------------------------------
% Input:
%Ns     = 40;  % Number of subintervals in s
%Nt     = 40;  % Number of subintervals in t
%smax   = 10;  % Maximum value of s in the discretization
%want_plot = 1% if we want plots during the run
%--------------------------------------
% Output:
% contr_p: approximate price for the given contract
%          defined in data.m
%  U     : vector of approximate prices in the sint abscisae
%
% function [contr_p,U,sint] = fd1dcn(Ns,Nt,smax,want_plot);

function [contr_p,U,sint] = fd1d(Ns,Nt,smax,want_plot)

% define parameters from the model
%data;
T = 1;sigmap =0.15;r=0.05;payoff = inline('max(x-10,0)','x');S0 = 10;
%--------------------------------------
% space grid
sv = linspace(0,smax,Ns+1)';
Np = Ns+1; % Number of points in the s grid
ds = sv(2)-sv(1);
dt = T/Nt;
% Define Finite difference operator
% Build time operator

sint = sv(2:Np-1);

diagon =                -   ((sigmap*sint).^2)/(ds^2)  -r;
supdia =  .5*(r*sint)/ds +.5*((sigmap*sint).^2)/(ds^2);
subdia = -.5*(r*sint)/ds +.5*((sigmap*sint).^2)/(ds^2);
At = spdiags([subdia diagon supdia],[0:2],Np-2,Np);

%Set right BC (extrapolation)
At(Np-2,Np-2:Np-1) = At(Np-2,Np-2:Np-1) + At(Np-2,Np)*[-1 2];
```

```
%Set left  BC
aleft =  At(1,1)*payoff(0); % info for left BC.
At = At(:,2:Np-1);
% time step operator
% Crank Nicholson
Bdt = (speye(Np-2,Np-2) - 0.5*dt* At);
% impose final payoff
U  = payoff(sint);

if want_plot,figure,  plot(sint,U); end
% Carry out the time stepping
for time_step = 1:Nt,
  tn = T- time_step*dt;
  rhs = U;
  rhs(1) = rhs(1) + dt *aleft*0.5*(exp(-r*(T-tn))+exp(-r*(T-tn+dt)));
  U = Bdt\((speye(Np-2,Np-2) + 0.5*dt* At)*rhs);
  if want_plot,plot(sint,U), hold on, pause(0.1),end
end
if want_plot,plot(sint, payoff(sint),'r'),end

% Approximate price
contr_p = interp1(sint,U,S0,'cubic');
if want_plot,plot(S0,contr_p,'*g'),end


disp(['Nt = ',num2str(Nt),', Ns = ',num2str(Ns),', ' ...
    'price = ',num2str(contr_p)])
```

Matlab script: price_MC.m

```
% Parameters setup
S0 = 100; % Initial stock price
K = 100;  % Strike price
r = 0.04; % Risk-free rate
sigma = 0.4; % Volatility
T = 1/4;    % Time to maturity
N = 100; % Number of Monte Carlo simulations


% Simulate stock price at final time using GBM model
Z = randn(N,1); % Standard normal random variables
% GBM formula for S(T)
ST = S0 * exp((r - 0.5 * sigma^2) * T + sigma * sqrt(T) * Z);

% Case 1: Payoff for a call option
payoff1 = max(ST - K, 0);



% Calculate the discounted expected payoffs
optionValue1 = exp(-r * T) * mean(payoff1);
```