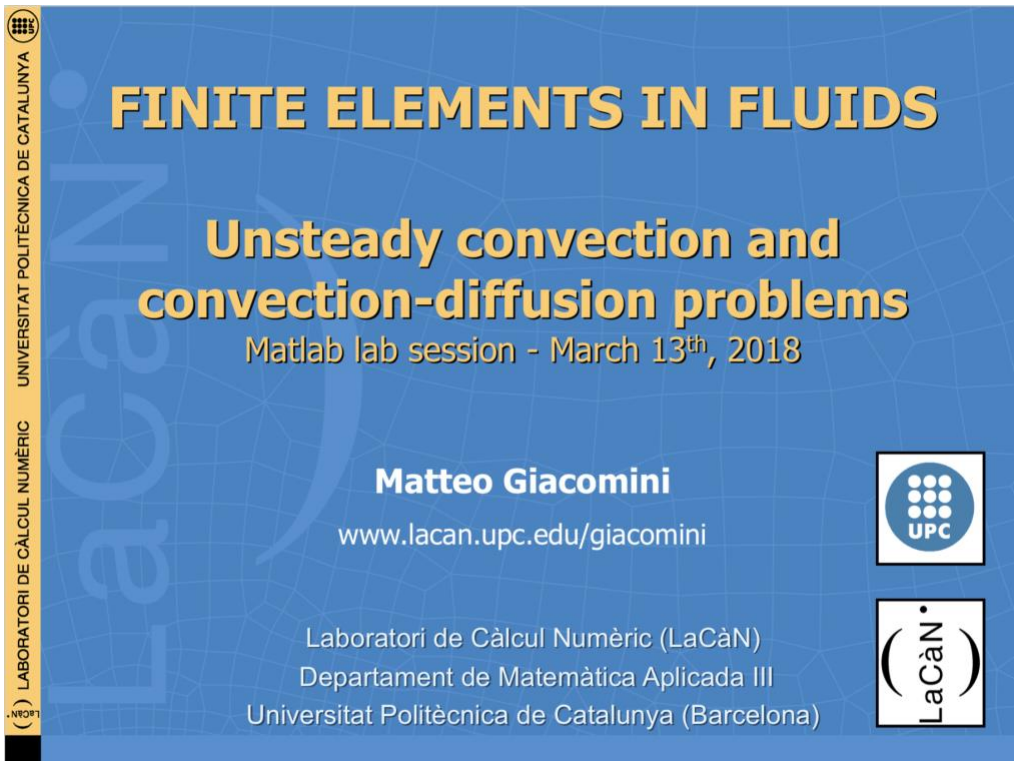


Unsteady Convection & Diffusion

NUMERICAL EXAMPLES

Iberico Leonardo, Juan Diego | Finite Elements in Fluids | 02/04/2018

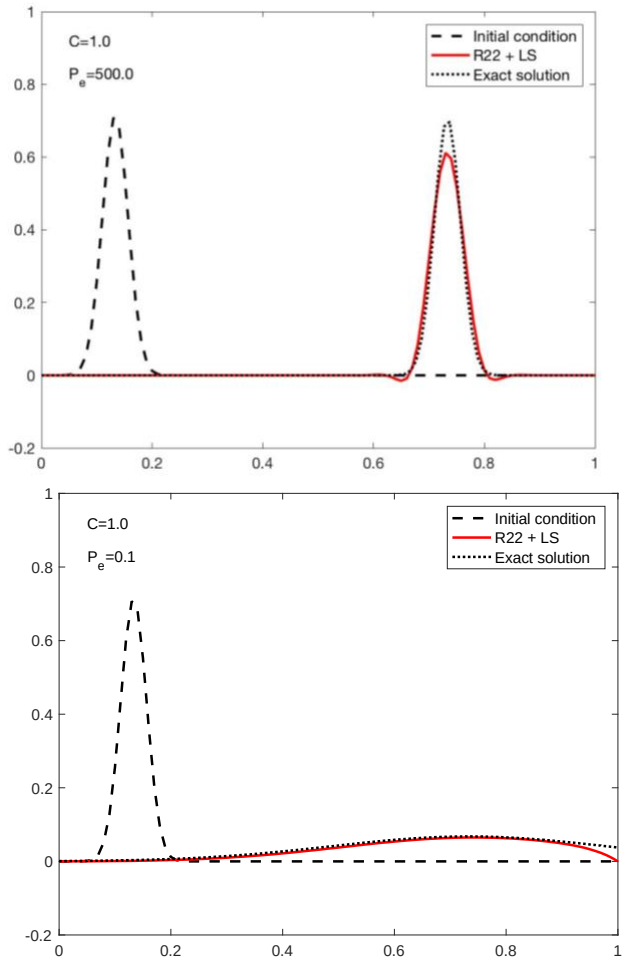
Example: The convection-diffusion case

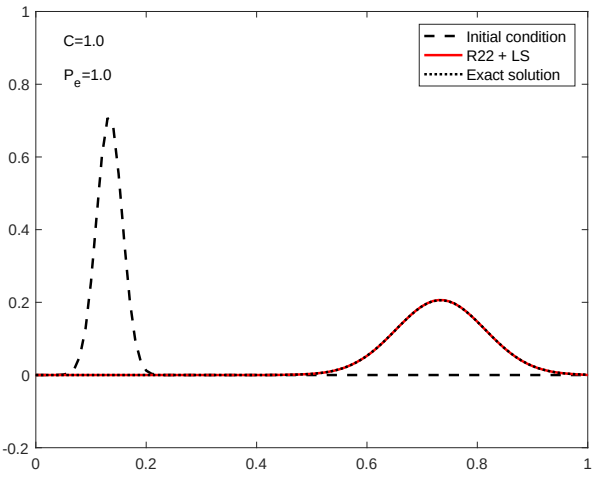
The following report it is focus unsteady convection-diffusion problem. The aim of the document it is to show the different behavior of the available schemes (time and space integrators), to solve the problem.

The solution is for 1D dimension, where the initial condition is propagation of a cosine profile.

EXERCISE GAUSSIAN HILL

- Test the problem for different values of viscosity proposed and comment the numerical results.





As it can be seen, the influence of the diffusion parameter on the equation acts on the high of the hill, making it decrease and wider as the diffusion parameter increase.

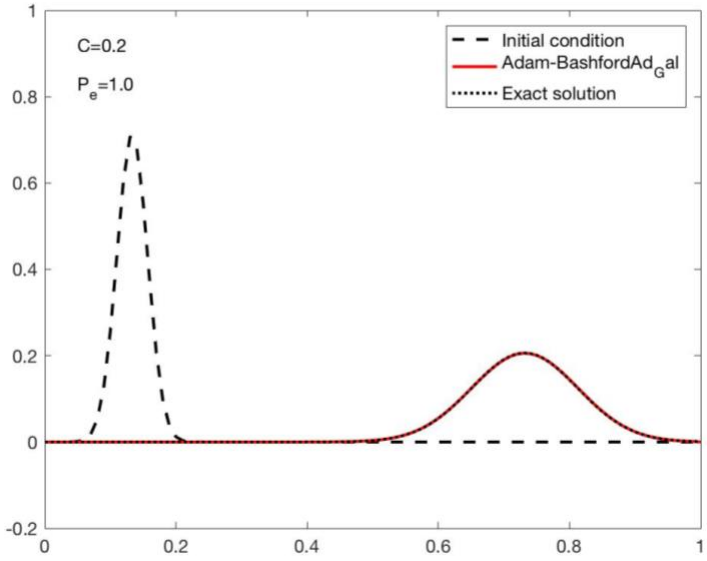
- Implement the Adams-Bashforth scheme
  The implementation was done in a way to include in the same function booth discretization in time-space Adam Bashforth with Galerkin.

```
function [Sol,Atot,F] = Ad_Gal(T,s,a,nu,f,K,M,G,xnode,dt,nstep,c,Accd1,bccd1)
% Sol = Ad_Gal(T,s,a,nu,f,K,M,G,xnode,dt,nstep,c,Accd1,bccd1)
% This function computes solution Sol at each time step using Galerkin formulation
%
% Input
```

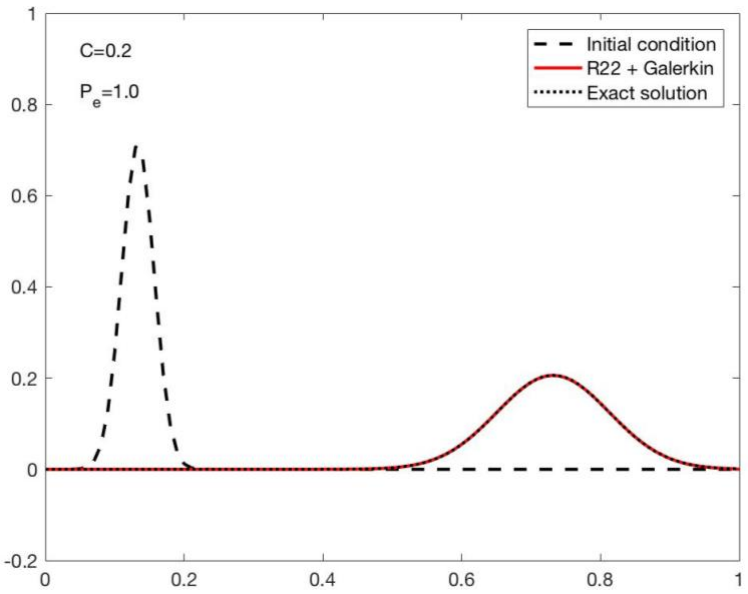
```
%   T,s:           time-integration matrices
%   tau:           stabilization matrix
%   a,nu:          problem coefficients
%   f,K,M,G:       matrices obtained by discretizing the different terms of the PDE
using FEM
%   xnode:         vector of nodal coordinates
%   dt:            time step
%   nstep:         number of time steps to be computed
%   c:             initial condition
%   Accd1, bccd1:  matrices to impose boundary conditions using lagrange
multipliers
%

%%%%%%%%%T and s are W and w%%%%%%%%%%

% Number of points
npoin = size(xnode,2);

% Integration matrix
[n,m] = size(T);
Id = eye(n,m);

% Computation of the matrix necessary to obtain solution at each time-step: A du =
F
Kt  = a*G + nu*K;

A = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(i,j)*M  + dt*T(i,j)*Kt];
    end
    A = [A; row];
end

Mf = M*f;

nccd = size(Accd1,1);
Accd = []; bccd = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(i,j)*Accd1];
    end
    Accd = [Accd; row];
    bccd = [bccd; bccd1];
end

nccd = n*nccd;
Atot = [A Accd'; Accd zeros(nccd)];

% Factorization of matrix Atot
[L,U] = lu(Atot);

Sol = c;
```

```
% Loop to compute the transient solution
for i=1
    aux = dt*(-Kt*c + Mf);
    F = [];
    for i =1:n
        F = [F; s(i)*aux];
    end
    F = [F;bccd*0];
    dc  = U\(L\F);
    dc = reshape(dc(1:n*npoin),npoin,n);
    c = c + sum(dc,2);
    Sol = [Sol c];
end

% Second step Adam-Bashforth
T = 0;
sn = 3/2;
sn0 = -1/2;
[n,m] = size(T);
Id = eye(n,m);


A = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(i,j)*M  + dt*T(i,j)*Kt];
    end
    A = [A; row];
end

Mf = M*f;

nccd = size(Accd1,1);
Accd = []; bccd = [];
for i = 1:n
    row = [];
    for j = 1:m
        row = [row, Id(i,j)*Accd1];
    end
    Accd = [Accd; row];
    bccd = [bccd; bccd1];
end

nccd = n*nccd;
Atot = [A Accd'; Accd zeros(nccd)];

% Factorization of matrix Atot
[L,U] = lu(Atot);


% Loop to compute the transient solution
for i=2:nstep
    auxn = dt*(-Kt*Sol(:,i) + Mf);
    auxn0 = dt*(-Kt*Sol(:,i-1) + Mf);
```

```
%      F=[];
    for i = 1:n
        F = [F(105:end); sn(i)*auxn + sn0(i)*auxn0];
    end
    F = [F;bccd*0];
    dc  = U\(L\F);
    dc = reshape(dc(1:n*npoin),npoin,n);
    c = c + sum(dc,2);
    Sol = [Sol c];
End
```



Adam Bashforth scheme shows a good accuracy following almost the exact solution. But the range of stability it is limited by the courant number <1/6.



On this side, the R22 Padé it has the same accuracy apparently as the Adam Bashforth, but taking care this is an implicit method, with higher computational cost but with wide range of Courant number.