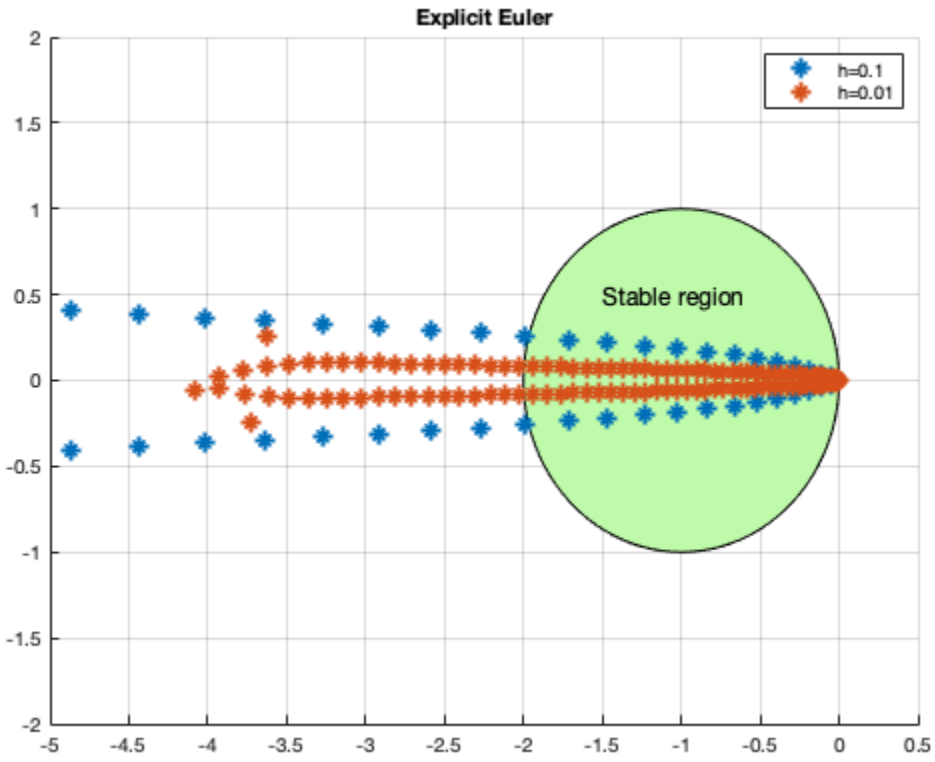
Practica_20_Casas_Mercade

Table of Contents

Section A)

```
close all
clear all
clc

N=128;
haches=[0.1, 0.01];
D1=dftdiffmat1(N);
D2=dftdiffmat2(N);
Burger=@(f)(0.1*D2*f+f.*D1*f);
for h=haches
    x=(2*pi/N)*[0:N-1]';
    f0=exp(-10.*(cos(x/2)).^2);
    f = ExplicitEuler(f0, h,Burger ,100);
end

figure;
%The stability region of the Explicit Euler (RK1) is a circle of
 radius 1
%centered at z=-1
pos = [-2 -1 2 2]; %Com si fos un rectangle
r = rectangle('Position',pos,'Curvature',[1 1]);
r.FaceColor = [0.752, 0.984, 0.674];
text(-1.5,0.5,  ['Stable region'], 'FontSize', 12)
title('Explicit Euler')
axis([-5 0.5 -2 2])
grid on
hold on

%Now we compute the jacobian evaluated at the initial condition f0
DF = jaco(Burger, f0);
[evec,eval]=eig(DF);
z=diag(eval).*haches;

plot(z,'*')
legend('h=0.1','h=0.01')
hold on

% This integrations is not stable neither for h=0.1 nor h=0.01 because
```

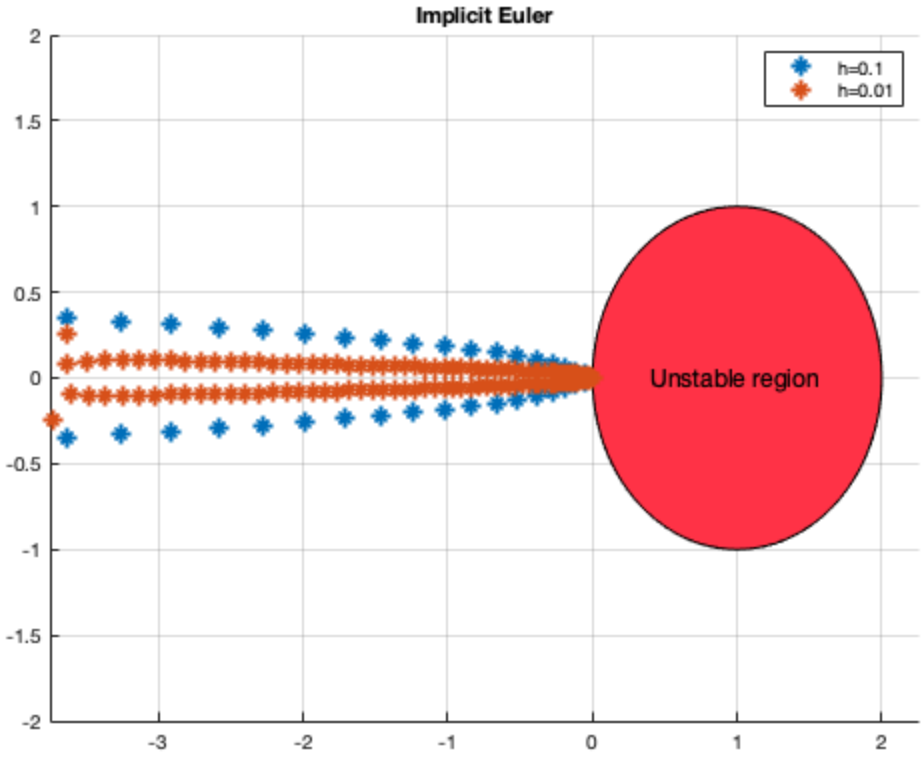
```
% in both cases there are points outside the stability region of
 euler's
% explicit method.
```

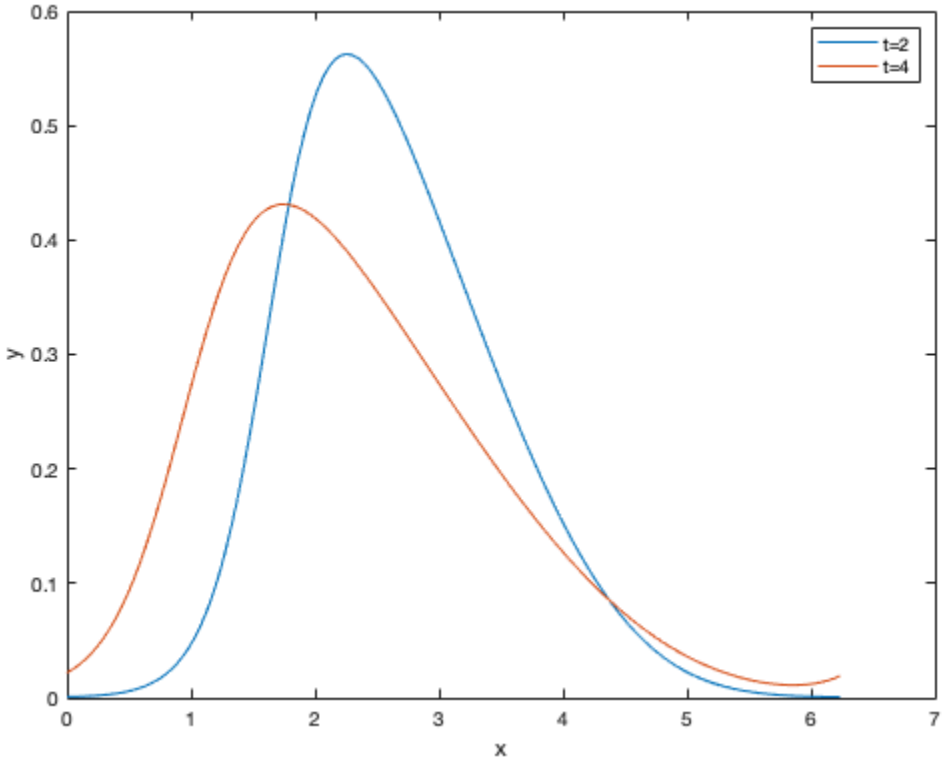


Section B)

```
h = 0.1;
fBurger=@(t,f)(0.1*D2*f+f.*D1*f);


figure;
%The stability region of the Explicit Euler (RK1) is a complex circle
 of radius 1
% centered at z=1
pos = [0 -1 2 2]; %Com si fos un rectangle
r = rectangle('Position',pos,'Curvature',[1 1]);
r.FaceColor = [1, 0.2, 0.274];
text(0.4,0,  ['Unstable region'], 'FontSize', 12)
title('Implicit Euler')
grid on
axis([-3.75 2.25 -2 2])
hold on

%Now we compute the jacobian evaluated at the initial condition f0
DF = jaco(Burger, f0);
[evec,eval]=eig(DF);
```

```
z=diag(eval).*haches;
plot(z,'*')
legend('h=0.1','h=0.01')
hold on


figure;
times = [2,4];
h = 0.1;
Results=[];
for t = times
    steps = t/h;
    [time, result] = bdf1(fBurger, 0, h, f0, steps);
    Results=[Results, result(:,end)];
end

plot(x, Results);
legend('t=2','t=4')
xlabel('x')
ylabel('y')


% It is straightforward to check that his problem is stiff. It is
 mainly
% because the the right-most real value of the eigenvalues is "far"
 from
% the left-most value on the complex plane. Due to this, we will need
 to
% use a very small timestep h to fit all the eigenvalues into the
 stable
% zone of the eulers explicit method (and all the others explicit
% methods). For a fixed value of the time, make h very small (almost
 zero)
% leads to a very large number (tending to infinity) of evaluations,
 thus
% to a high computational cost. In order to avoid this problem it is
 better
% in this cases to use Implicit Euler's methods for which all the
 Re(z)<0
% plane is an stable region.

Warning: Imaginary parts of complex X and/or Y arguments ignored
```


Implicit Euler
h=0.1
h=0.01
Unstable region
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-3
-2
-1
0
1
2


t=2
t=4
y
x
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5
6
7

Codes used

```
%{
% Code 26: Fourier 2nd Ord Differentiation Matrix % Input: N (even)
% Output: NxN Diff. Matrix D2
function D2 = dftdiffmat2(N)
% Segona derivada d'una funcio 2pi periodica.
% Cal multiplicar aquesta matriu per una malla de la funcio que volem
% diferenciar avalauda en els nodes xj = 2pi/N.
WN = exp(-i*2*pi/N) ; D2 = zeros(N) ; k = [0:N-1];
for j = 0:N-1
    for l = 0:N-1
        D2(j+1,l+1) = -sum(((k-N/2).^2).*WN.^(-(k-N/2)*(j-l)))/N;
    end
end
%}

%{
% Code 25: Fourier 1st Ord Differentiation Matrix % Input: N (even)
% Output: NxN Diff. Matrix D1 %ALVARO
function D1 = dftdiffmat1(N)
% Deriva una funcio 2pi periodica
% Cal multiplicar aquesta matriu per una malla de la funcio que volem
% diferenciar avalauda en els nodes xj = 2pi/N
WN = exp(-i*2*pi/N) ; D1 = zeros(N) ; k = [0:N-1];
for j = 0:N-1
    for l = 0:N-1
        D1(j+1,l+1) = (i/N)*sum((k-N/2).*WN.^(-(k-N/2)*(j-l)));
    end
end
%}

%{
% Code 27: BDF1 (implicit Euler time-stepper)
% Solution for u_t = f(t,u) with u(t0) = v0 (n-dimensional)
% Input: fun (function name) ; t0 (initial time)
% h (time-step) ; v0 (initial condition) ; N (no. steps)
% Output: T (time vector: 1 x N+1)
% Y (solution matrix: n x N+1)
function [T,Y] = bdf1(fun,t0,h,v0,N)
T = zeros(1,N+1); n = length(v0); I = eye(n);
Y = zeros(length(v0),N+1); DF = zeros(n); H = sqrt(eps)*I;
T(1) = t0; Y(:,1) = v0;
for j = 1:N
    z0 = Y(:,j) ; tplus = t0 + h*j; T(j+1) = tplus;
    dz = 1; z = z0 ;
    while norm(dz) > 1e-12
        f1 = feval(fun,tplus,z);
        for kk = 1:n
            f2 = feval(fun,tplus,z+H(:,kk));
            Df(:,kk) = (f2 - f1)/H(kk,kk);
        end
        dz = -(I-h*Df)\(z-z0-h*f1); z = z + dz;
```

```
    end
    Y(:,j+1) = z ;
end
%}

%{
function DF = jaco(F, x)
    % This code give you the Jacobian matrix of the function F
 evaluated in x.
    % The Jacobian matrix is (n,m) meanwhile the sixe of F is n and
 the size of
    % x is m.

    % F son les n funcions escalars

    % Input: F(x):R^m ---> R^n
    % x: (m x 1)-vector ; F: (n x 1)-vector
    % Output: DF(x) (n x m) Jacobian matrix at x
    % [df1/dx1 ... df1/dxm]
    % [ ...           ... ]
    % [dfn/dx1 ... dfn/dxm]

    % El mateix q que faria la funcio jacobian(F, v) del matlab

    f1 = feval(F, x); m = length(x); n = length(f1);

    h = sqrt(eps); H = eye(m) * h; % eps = epsilon minima maquina.

    DF = zeros(n, m); % Matriu a omplir

    for j = 1:m

        f2 = feval(F, x + H(:, j)); % Seleccionem una fila de H per
 fer la derivada direccional, fent una variacio epsilon

        DF(:, j) = (f2 - f1) / h; % Apliquem definiicio derivada
 direccional
    end

end
%}
```

Published with MATLAB® R2018b