Table of Contents

Practical 10 - Cortés & García

```
clear all; close all;
format long;

f_ini = @(x) exp(-10*(cos(x/2)).^2); % Definie initial condition

N = 128;
xj = (2*pi/N)*(0:N-1)'; % Fourier grid for discretization
```

(a) Explicit Euler

Let us have $\vec{f}(t)$ a vector of size N s.t. $f_j(t) = f(\frac{2\pi j}{N}, t)$. Now, the given equation tells us that:

$$\dot{f}_j = \sum_{l=0}^{N-1} \mathbf{D}^{(2)}_{jl} f_l(t) + f_j \sum_{l=0}^{N-1} \mathbf{D}^{(1)}_{jl} f_l(t)$$

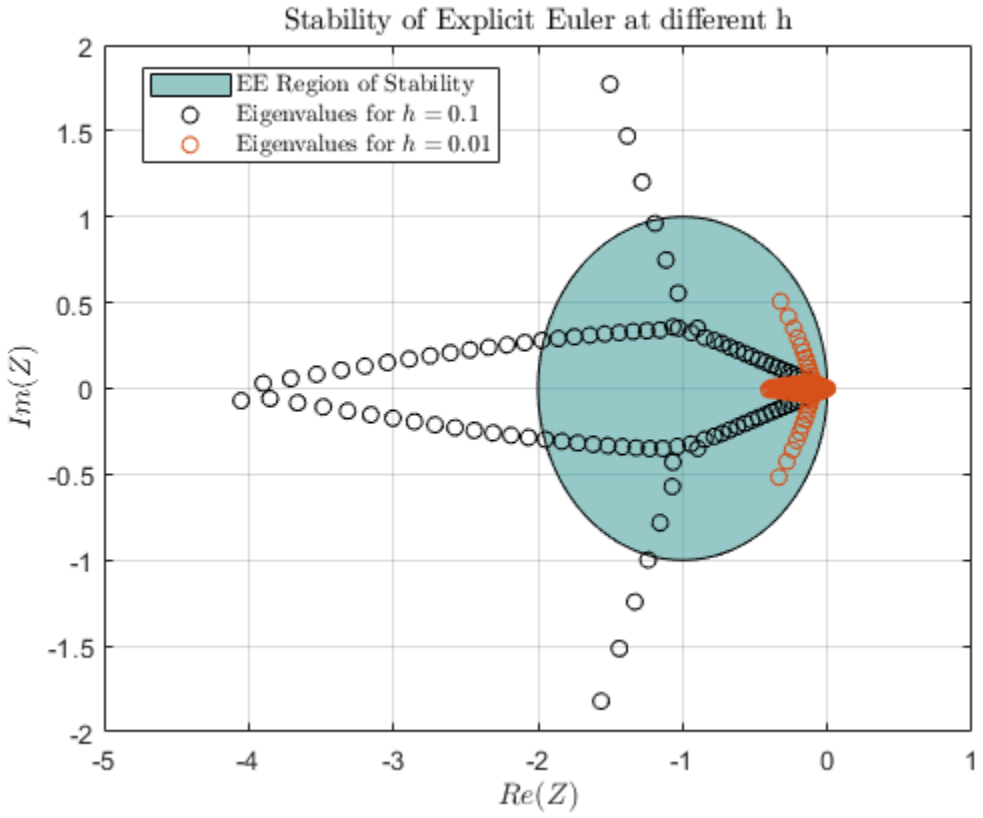
This means we can write

$$\dot{\vec{f}} = \vec{F}(\vec{f})$$

To compute this function $\vec{F}$, we will need to compute the Fourier differentiation matrices for size N, $\mathbf{D}^{(1)}$ and $\mathbf{D}^{(2)}$. Then, as it is asked in the statement, we will compute the Jacobian of the function $\vec{F}$. The Explicit Euler (EE) method converges when $z_j = h\lambda_j$, where h is the time-step used and λ_j the j-th eigenvalue of the Jacobian of $\vec{F}$, are all inside a circle of radius 1 centered in $(-1, 0)$.

```
% Compute differentiation matrices
D1 = dftdiffmat1(N);
D2 = dftdiffmat2(N);

% Build F function
F = @(f) 0.01*D2*f + f.*(D1*f);

% Find initial condition (t=0)
f_0 = f_ini(xj);

% Find eigenvalues of the Jacobian
DF0 = jac(F, f_0);
eigenval = eig(DF0);
```

```
th = 0:0.01:2*pi;
circle = [sin(th)-1 ; cos(th)]; % Compute stability region

% Plot stability region and z_j for both time-steps
figure(1)
plot(circle(1, :), circle(2, :), 'Color',[0 0.5 0.5], 'LineWidth',0.8)
fill(circle(1, :), circle(2, :),[0 0.5 0.5],'FaceAlpha',0.4)
hold on
plot(0.1*eigenval,'ko')
plot(0.01*eigenval,'o', 'Color', [0.85 0.325 0.1]  )
axis([-5 1 -2 2])
hold off
legend('EE Region of Stability', 'Eigenvalues for
 $h=0.1$', 'Eigenvalues for
 $h=0.01$', 'interpreter', 'LaTeX','location','best')
xlabel('$Re(Z)$','interpreter','latex')
ylabel('$Im(Z)$','interpreter','latex')
title('Stability of Explicit Euler at different
 h','interpreter','latex')
grid on
```

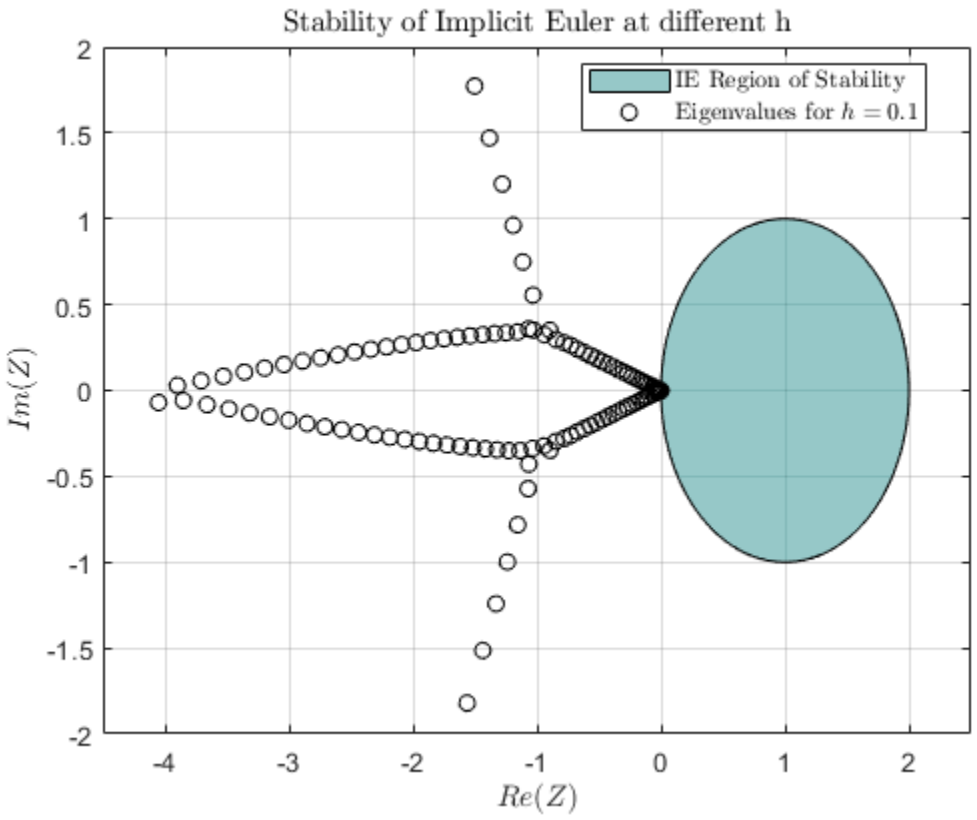


In Figure 1 we can observe the delimitation of the stability region of the EE method and the values obtained for z_j for both $h = 0.1$ and $h = 0.01$. We can see that in the first case, the method will not be stable, for there are many values of z_j that fall out of the stability region. However, the method is stable for $h = 0.01$, since we can clearly see that all the values of z_j fall inside the stability region.

(b) Implicit euler

For the Implicit Euler method (IE), teh stability region is different than for the EE method. Instead, now it consists of the complex plane, excluding the circle of radius 1 centered in $(1, 0)$. If we use the previously computed eigenvalues and plot them with the new stability region we will be able to see wether this method will be stable or not for $h = 0.1$.

```
circle = [sin(th)+1 ; cos(th)];

% Plot stability region of the IE method
figure(2)
plot(circle(1, :), circle(2, :), 'Color',[0 0.5 0.5], 'LineWidth',0.8)
fill(circle(1, :), circle(2, :),[0 0.5 0.5],'FaceAlpha',0.4)
hold on
plot(0.1*eigenval,'ko')
axis([-4.5 2.5 -2 2])
hold off
legend('IE Region of Stability', 'Eigenvalues for
 $h=0.1$', 'interpreter', 'LaTeX','location','best')
xlabel('$Re(Z)$','interpreter','latex')
ylabel('$Im(Z)$','interpreter','latex')
title('Stability of Implicit Euler at different
 h','interpreter','latex')
grid on
```

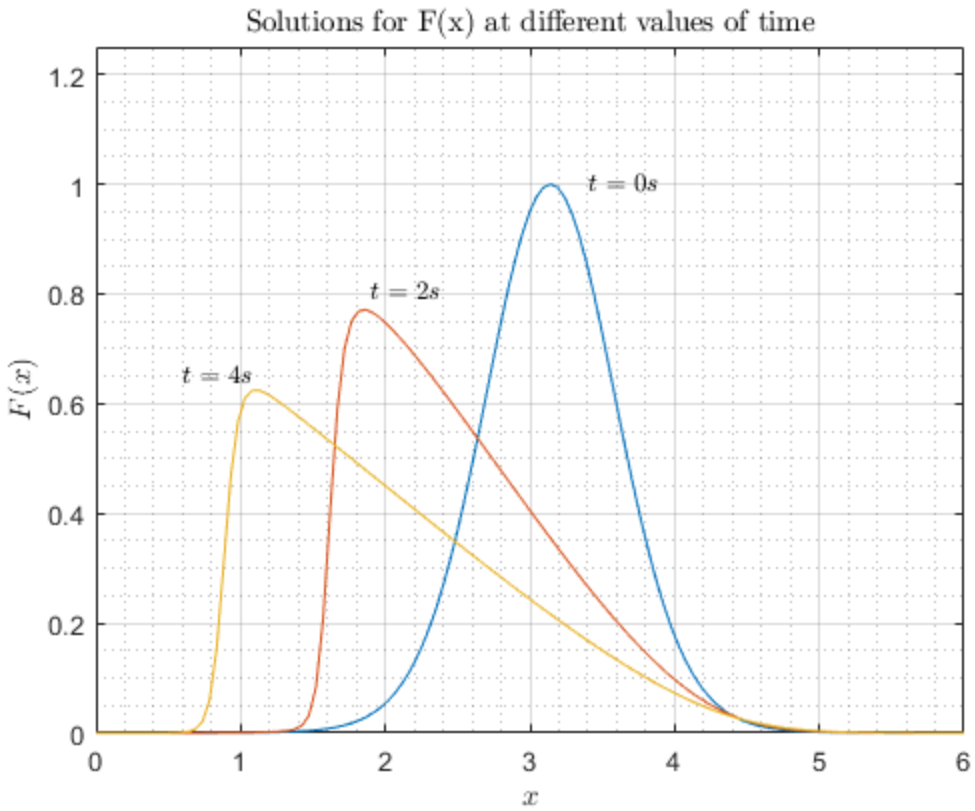


After seeing this plot, we conclude this method is stable for $h = 0.1$. Then, we can compute and plot the solution at the requested values.

To compute and plot the evolution of the function through time with respect to x by using the BDF1 method, we need a function that depends both on t and the values f_j.

```
Ft = @(t, f) F(f);

[T,Y] = bdf1(Ft, 0, 0.1, f_0, N);

figure(3)
plot(xj, Y(:, 1)) % t = 0 = (1-1)*0.1
hold on
plot(xj, Y(:, 21)) % t = 2 = (20-1)*0.1
plot(xj, Y(:, 41)) % t = 4 = (40-1)*0.1
hold off
axis([0 6 0 1.25])
ylabel('$F(x)$','interpreter','latex')
xlabel('$x$','interpreter','latex')
title('Solutions for F(x) at different values of
 time','interpreter','latex')
text(3.4,1,'$t = 0s$', 'Interpreter', 'latex')
text(1.9,0.8,'$t = 2s$', 'Interpreter', 'latex')
text(0.6,0.65,'$t = 4s$', 'Interpreter', 'latex')
grid(gca,'minor')
grid on
```

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



As it was said in class, we can say that the obtained result is pretty good: we observe the behaviour of a wave, advancing toward the negative values of x.

Auxiliary Codes

Code 19: Computation of the Jacobian J 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

```
function DF = jac(F,x)
    f1 = feval(F,x);
    n = length(f1);
    m = length(x);

    DF = zeros(n,m);
    H = sqrt(eps)*eye(m);

    for j = 1:m
        f2 = feval(F,x+H(:,j));
        DF(:,j) = (f2 - f1)/H(j,j);
    end
end

% Code 25: Fourier 1st Ord Differentiation Matrix
% Input: N (even)
% Output: NxN Diff. Matrix D1
function D1 = dftdiffmat1(N)
```

```
    WN = exp(-1i*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) = (1i/N)*sum((k-N/2).*WN.^(-(k-N/2)*(j-l)));
        end
    end
end

% Code 26: Fourier 2nd Ord Differentiation Matrix
% Input: N (even)
% Output: NxN Diff. Matrix D2
function D2 = dftdiffmat2(N)
    WN = exp(-1i*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
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
end
```

Published with MATLAB® R2020b