Discretization of transient groundwater flow

```
clear, clc
%set_demo_defaults()
```

The transient evolution of the head in a confined aquifer is given by an **I**nitial and **B**oundary **V**alue **P**roblem (**IBVP**)

PDE: $S_s \frac{\partial h}{\partial t} - \nabla \cdot [K \nabla h] = f_s$ on $\mathbf{x} \in \Omega$

BC: $h(\mathbf{x}, t) = h_b(\mathbf{x}, t)$ on $\mathbf{x} \in \partial\Omega_D$ (Dirichlet boundary)

$\mathbf{q} \cdot \hat{\mathbf{n}}_i = q_b(\mathbf{x}, t)$ on $\mathbf{x} \in \partial\Omega_N$ (Neumann boundary)

IC: $h(\mathbf{x}, 0) = h_0(\mathbf{x})$ on $\mathbf{x} \in \Omega$

For the moment we assume that S_s and K are independent of the head, so that the problem is **linear**. However, both S_s and K can vary with location due to differences in material. The discretization of the divergence of the fluxes/spatial derivatives is identical to the steady problem so that

$$-\nabla \cdot [K \nabla h] \approx \mathbf{L}\mathbf{u},$$

where `L=-D*Kd*G` and `h` the vector of unknown heads. Below we'll discuss a common time discretization.

Theta Method

To discretize the time derivative we use a finite difference

$$\frac{\partial h}{\partial t} \approx \frac{\mathbf{h}^{n+1} - \mathbf{h}^n}{\Delta t}$$

where the superscript n denotes the time level and $\Delta t = t^{n+1} - t^n$. So the discretized equations becomes

$$\mathbf{M}(\mathbf{h}^{n+1} - \mathbf{h}^n) + \Delta t\, \mathbf{L}\mathbf{h}^? = \Delta t\, \mathbf{f}_s,$$

where $\mathbf{M}$ is a N by N diagonal *mass matrix* with the values of S_s in each cell on the diagonal, where N is the number of unknowns. The properties of the time integrations are determined by choosing the time level at which $\mathbf{h}^?$ is evaluated. In the theta-method we introduce a paramter θ that allows us to determine the time level as follows

$\mathbf{h}^? = \theta \mathbf{h}^n + (1-\theta)\mathbf{h}^{n+1}$ (note some textbooks flip in this definition, we are following Iserles)

Substituting into the discrete equation

$$\mathbf{M}(\mathbf{h}^{n+1} - \mathbf{h}^n) + \Delta t\, \mathbf{L}(\theta \mathbf{h}^n + (1-\theta)\mathbf{h}^{n+1}) = \Delta t\, \mathbf{f}_s$$

Collecting the unknown terms on the left and the known terms on the right

$$\mathbf{IM}\, \mathbf{h}^{n+1} = \mathbf{EX}\, \mathbf{h}^n + \Delta t\, \mathbf{f}_s$$

where we have the following implicit and explicit matrices

$$\mathbf{IM} = \mathbf{M} + (1-\theta)\Delta t\,\mathbf{L},$$

$$\mathbf{EX} = \mathbf{M} - \theta\Delta t\,\mathbf{L}.$$

Note that the specifics of the PDE and the coordinate system are hiden in **L**, therefore the theta-method can be applied to **any** transient linear equation.

```
clf
rho = 1e3;
cp = 2e3;
kappa = 4;
Diff = kappa/(rho*cp);

Grid.xmin = 0; Grid.xmax = 1; Grid.Nx = 50;
Grid = build_grid(Grid);
[D,G,I] = build_ops(Grid);
L = -D*Diff*G; M = I;
IM = @(theta,dt) M + (1-theta)*dt*L;
EX = @(theta,dt) M - theta*dt*L;
```

Amplification Matrix

In the absence of boundary conditions or source terms driving the problem, we expect head variations to decline and the head to become constant across the domain. For $\mathbf{f}_s = 0$ we can write

$$\mathbf{h}^{n+1} = \mathbf{IM}^{-1}\mathbf{EX}\,\mathbf{h}^n$$

$$\mathbf{h}^{n+1} = \mathbf{A}\,\mathbf{h}^n$$

where $\mathbf{A} = \mathbf{IM}^{-1}\mathbf{EX}$ is the amplification matrix. The solution at the n-th timestep is therefore

$$\mathbf{h}^n = \mathbf{A}^n\mathbf{h}^0$$

where $\mathbf{h}^0$ is the initial condition. Using the spectral decomposition of symmetric matrices we can write

$$\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1}$$

where $\mathbf{Q}$ is the square matrix where the eigenvectors of $\mathbf{A}$ are the columns and $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues λ of $\mathbf{A}$. Since $\mathbf{Q}^{-1}\mathbf{Q} = \mathbf{I}$, we see that the n-th power of a matrix is

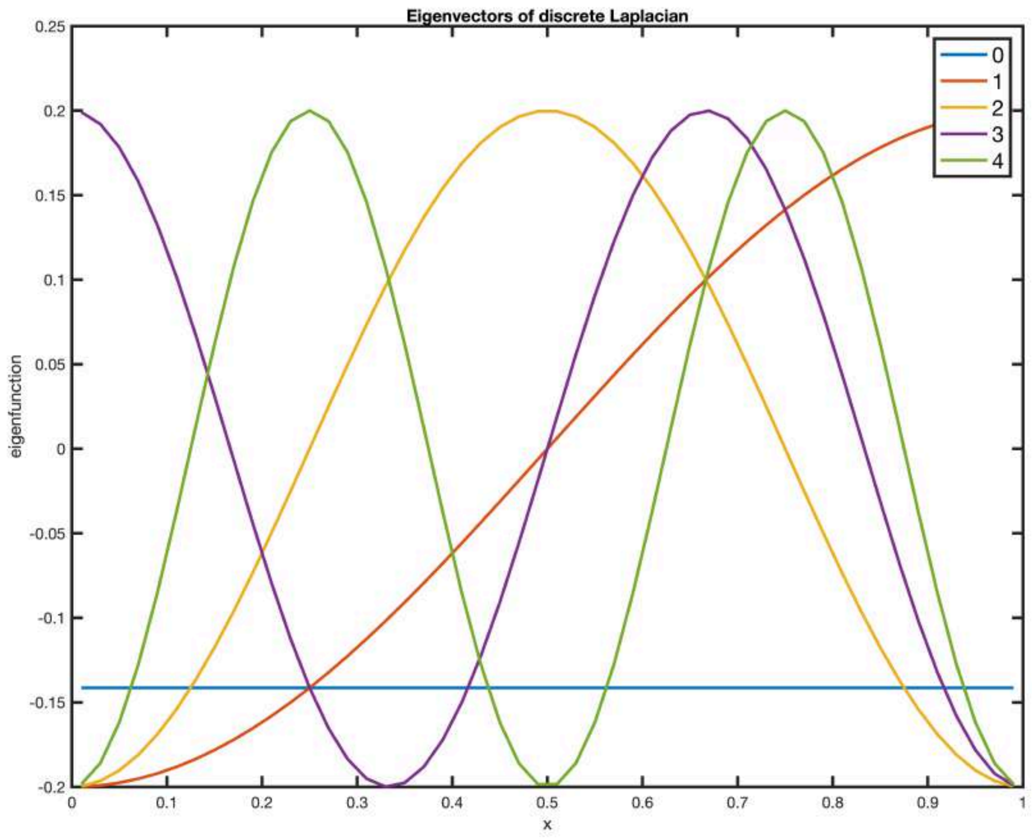
$$\mathbf{A}^n = \mathbf{Q}\boldsymbol{\Lambda}^n\mathbf{Q}^{-1},$$

is simply computed by raising the eigenvalues to the n-th power.

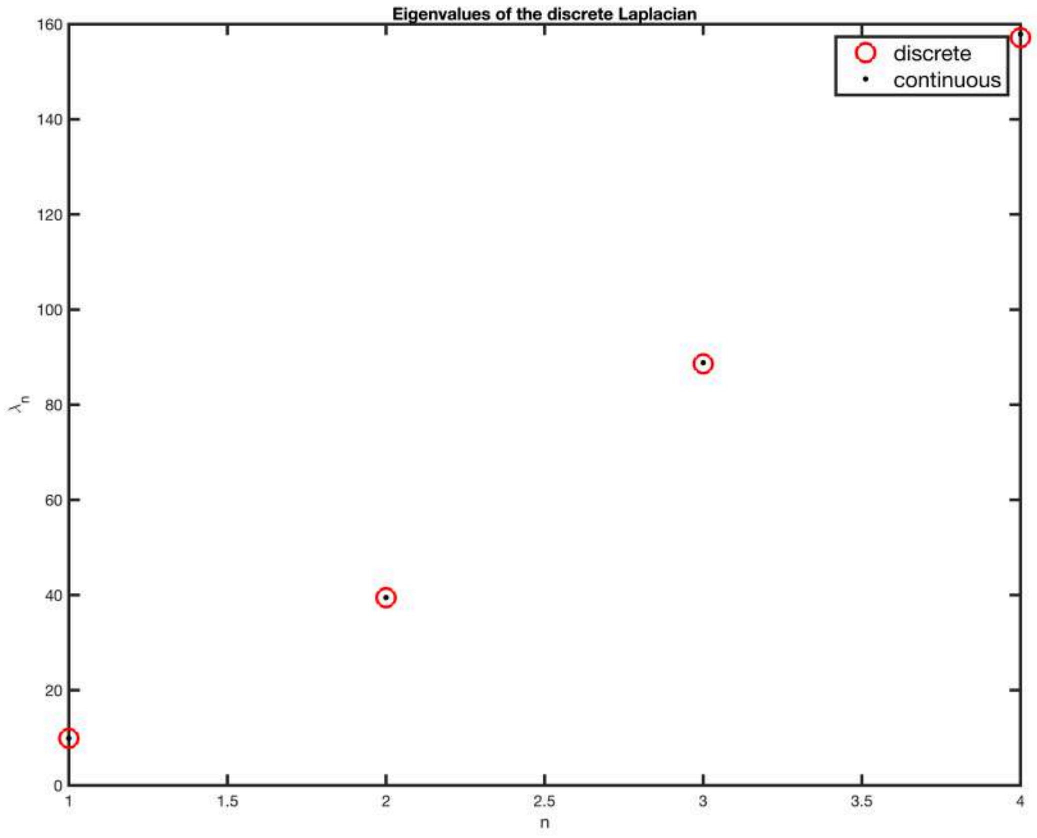
Example: Spectral decomposition of the discrete Laplacian

```
[Q,Lambda] = eig(full(L)); lam = diag(Lambda);
figure('position',[10 10 900 600])
subplot 121
clf
```

```
for i=1:5
    plot(Grid.xc,Q(:,i)), hold on
end
title('Eigenvectors of discrete Laplacian','fontsize',18)
xlabel('x','fontsize',22)
ylabel('eigenfunction','fontsize',14)
legend('0','1','2','3','4','fontsize',14)
pbaspect([1 .8 1])
```



```
subplot 122
clf
scale = (pi/Grid.Lx).^2/lam(2); % I am not sure why we need this scaling
for n = 1:4;
    plot(n,scale*lam(n+1),'ro'), hold on
    plot(n,(n*pi/Grid.Lx).^2,'k.')
end
title('Eigenvalues of the discrete Laplacian','fontsize',18)
xlabel 'n'
ylabel '\lambda_n'
legend('discrete','continuous','fontsize',14)
pbaspect([1 .8 1])
```



In GEO 366M Mathematical Methods in Geophysics you will learn/have learned that the eigenvalues, λ_n, and eigenvectors, $\varphi_n(x)$, of the Laplacian, $\nabla^2(\bullet)$ are

1. $\lambda_n = \left(\frac{n\pi}{H}\right)^2$
2. $\varphi_n = \cos\left(\sqrt{\lambda_n}\,x\right)$

In GEO 366M you will also show that the analytic solution to a heat conduction problem has the form

$T(x,t) = \sum_{n=0}^{\infty} a_n \varphi_n(x) \mathrm{e}^{-\lambda_n D_t t}$,

where the coefficients a_n depend of the initial condition and $D_t = \frac{\kappa}{\rho c_p}$ is the thermal diffusivity. The general solution shows that increasing oscillatory functions decay increasingl rapidly. This makes physical sense, a diffusive process like heat conduction smoothes the solution rapidly.

Similarly the evolution of the discrete solution, **u**, will depend on the eigenvalues of the amplification matrix **A** (different from **L**!). For a stable numerical method all $\lambda_n(\mathbf{A}) \leq |1|$. Below we show that the magnitude of the eigenvalues of **A** changes significantly with θ.

Note: For some reasons the eigenvalues need to be scaled - not sure why that is. Also there is an error in the eigenvalues that increases with n. So if the larger ones are off just increase Nx until the respective eigen function is resolved!

Properties of the Theta-Method

Forward Euler (FE): $\theta = 1$

In this case, we have the following matrices

$\mathbf{IM} = \mathbf{M}$,

$\mathbf{EX} = \mathbf{M} - \Delta t\,\mathbf{L}$,

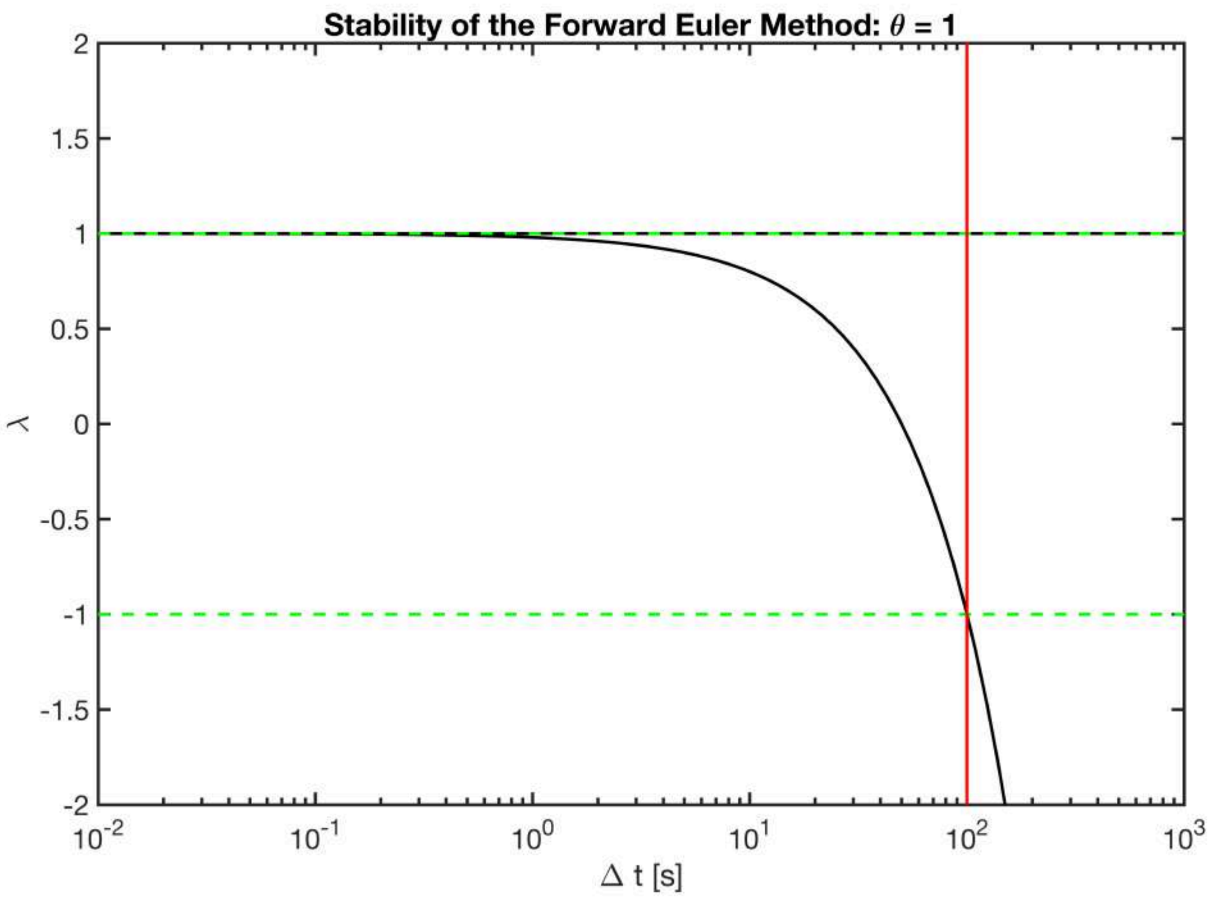
since $\mathbf{M}$ is diagonal no matrix has to be inverted to compute the next timestep and the method is called *explicit*. Explicity timesteps are simple to implement and computationally cheap. However, we will show below that typically need very many of them.

Relative to the transport, **Lu**, which is evaluated at t^n, the time derivative is discretized with a one-sided derivative. Therefore, it can be shown that the Forward Euler Method is first-order accurate, i.e, $\mathcal{O}(\Delta t)$.

To understand the stability of the FE-method we plot $\min_n(\lambda_n)$ and $\max_n(\lambda_n)$ of the amplification matrix, $\mathbf{A}$,as function of Δt.

```
theta = 1;
dt_max = Grid.dx^2/(2*Diff);
dt_vec = logspace(-2,3,1e2);

figure('position',[10 10 900 600])
for i = 1:length(dt_vec)
    A = inv(IM(theta,dt_vec(i)))*EX(theta,dt_vec(i));
    lam = eig(full(A));
    lam_max_FE(i) = max(lam);
    lam_min_FE(i) = min(lam);
end
semilogx(dt_vec,lam_max_FE,'k'), hold on
semilogx(dt_vec,lam_min_FE,'k')
semilogx(dt_vec,ones(size(dt_vec)),'g--','linewidth',2)
semilogx(dt_vec,-ones(size(dt_vec)),'g--','linewidth',2)
semilogx(dt_max*[1 1],[-2 2],'r','linewidth',2), hold off
ylim([-2 2])
xlabel '\Delta t [s]'
ylabel('\lambda')
title 'Stability of the Forward Euler Method: \theta = 1'
```



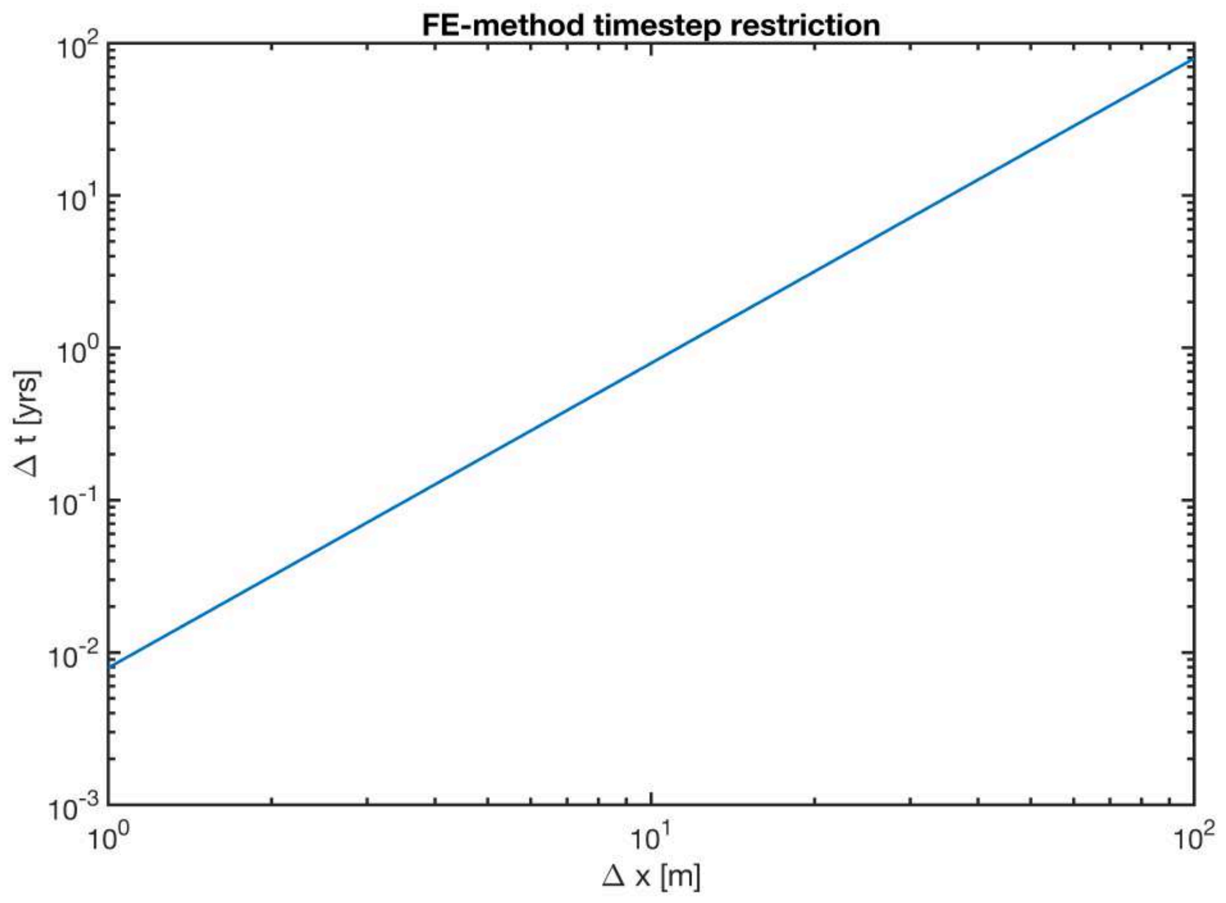
As Δt increases the stability condition, $\lambda_n(\mathbf{A}) \leq |1|$, so that the FE-method is only *conditionally stable*. The FE-method is only stable (does not blow up) under the condition

$\Delta t_N \leq \frac{\Delta x^2}{2D_t}$ (Neumann condition)

where $D_t = \frac{\kappa}{\rho c_p}$ is the thermal diffusivity. A smart chap called John Von Neumann figure this one out - personal hero of mine. The physical interpretation of this condition is that Δt_N is the time required for the head to diffuse across one grid block.

For thermal conduction the timestep limit declines rapidly with grid spacing

```
dx_vec = logspace(0,2,1e2);
s2yr = 60^2*24*365;
figure('position',[10 10 900 600])
loglog(dx_vec,dx_vec.^2/(2*Diff)/s2yr);
xlabel '\Delta x [m]', ylabel '\Delta t [yrs]'
title 'FE-method timestep restriction'
```



Therefore, the Forward Euler method is typically not efficient for solving the heat equation.

Backward Euler (BE): $\theta = 0$

In this case, we have the following matrices

$\mathbf{IM} = \mathbf{M} + \Delta t\,\mathbf{L}$,

$\mathbf{EX} = \mathbf{M}$

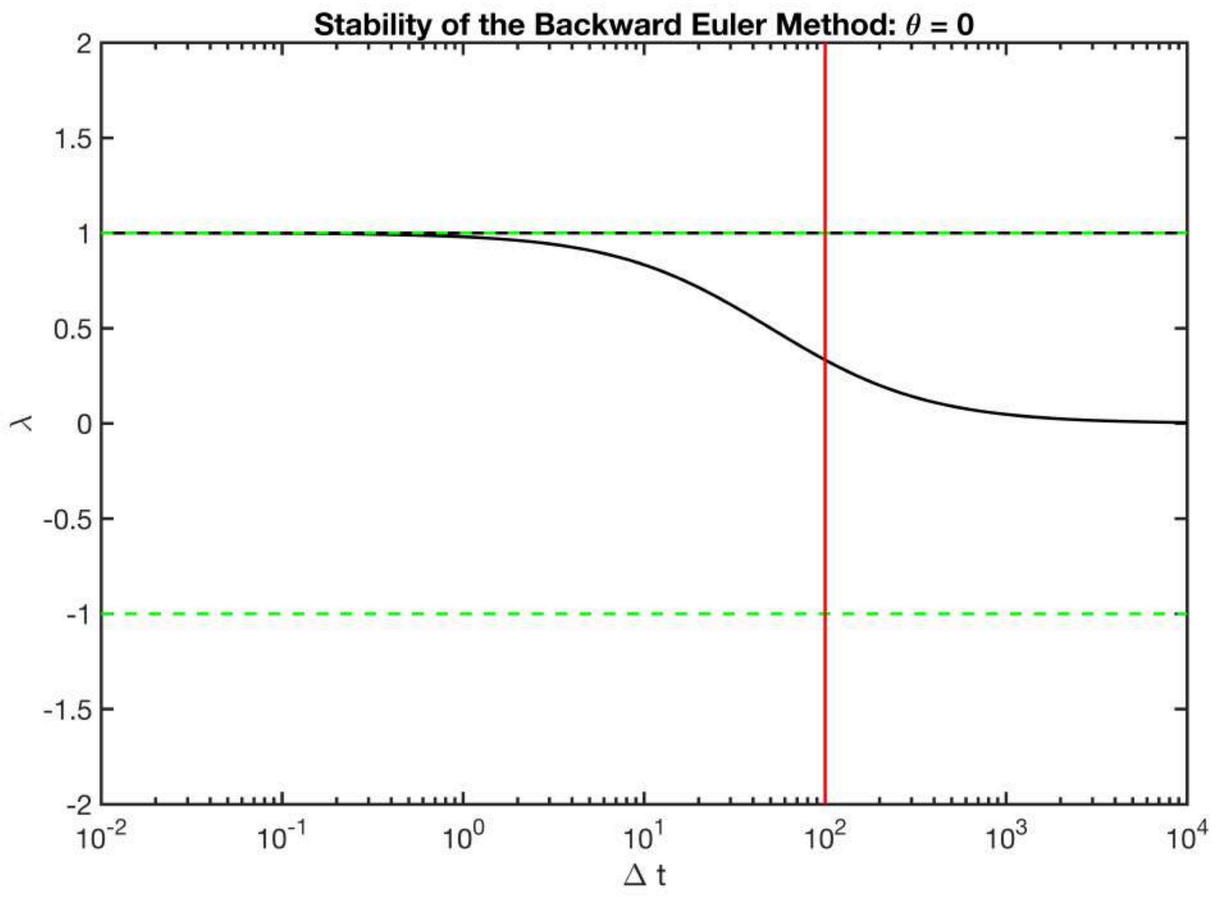
so that $\mathbf{IM}$ is not diagonal and a linear system must be solved at every time step. A timestepping methos that requires solution of a linear system is called *implicit*.

Relative to the transport, **Lu**, which is evaluated at t^{n+1}, the time derivative is discretized with a one-sided derivative. Therefore, it can be shown that the Backward Euler Method is also first-order accurate, i.e, $\mathcal{O}(\Delta t)$.

To understand the stability of the BE-method we plot $\min_n(\lambda_n)$ and $\max_n(\lambda_n)$ of the amplification matrix, **A**,as function of Δt.

```
theta = 0;
dt_vec = logspace(-2,4,1e2);
```

```
figure('position',[10 10 900 600])
for i = 1:length(dt_vec)
    A = inv(IM(theta,dt_vec(i)))*EX(theta,dt_vec(i));
    lam = eig(full(A));
    lam_max_FE(i) = max(lam);
    lam_min_FE(i) = min(lam);
end
semilogx(dt_vec,lam_max_FE,'k'), hold on
semilogx(dt_vec,lam_min_FE,'k')
semilogx(dt_vec,ones(size(dt_vec)),'g--','linewidth',2)
semilogx(dt_vec,-ones(size(dt_vec)),'g--','linewidth',2)
semilogx(dt_max*[1 1],[-2 2],'r','linewidth',2), hold off
ylim([-2 2])
xlabel '\Delta t'
ylabel('\lambda')
title 'Stability of the Backward Euler Method: \theta = 0'
```



We see that the stability condition is always satisfied and it can be shown that $0 \leq \lambda_n(\mathbf{A}) \leq |1|$ for the BE-method. This implies that the BE-method is *unconditionally stable*. This means you can take timesteps as large as you like and the computation will not blow up, but it will -of course - become increasingly less accurate. In general, the BE-method is the most robust and should be the first choice if you solve complicated problems.

Crank-Nicholson (CN): $\theta = \frac{1}{2}$

In this case, we have the following matrices

$$\mathbf{IM} = \mathbf{M} + \frac{\Delta t}{2}\mathbf{L},$$

$$\mathbf{EX} = \mathbf{M} - \frac{\Delta t}{2}\mathbf{L}.$$

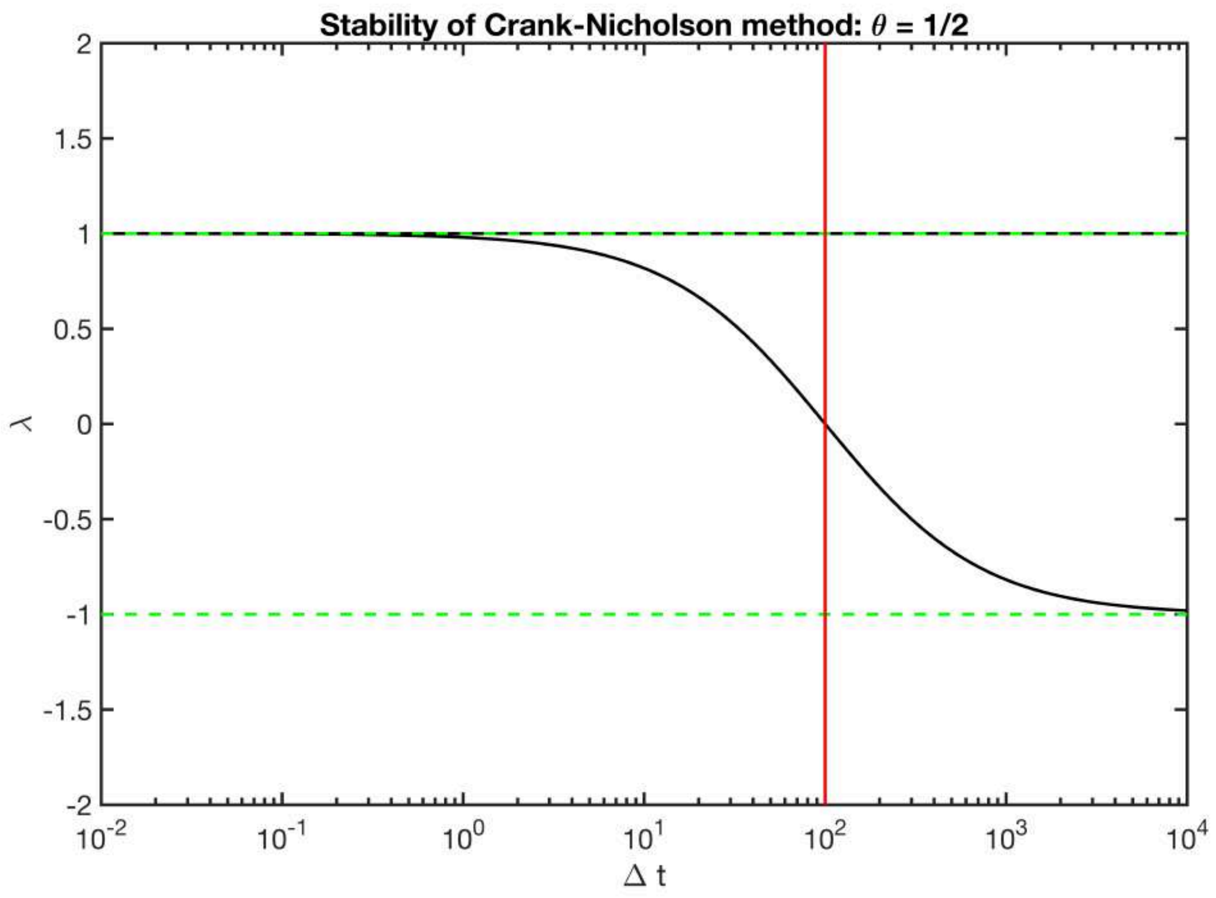
Similar to the BE-method the CN-method is implicit and each time step requires the inversion of a linear system.

Beacuse, the transport term, Lu, is evaluated at $t^{n+1/2}$the time-discretization is a central finite difference. Therefore, it can be shown that the CN-method is also first-order accurate, i.e, $\mathcal{O}(\Delta t^2)$.

To understand the stability of the CN-method we plot $\min_n(\lambda_n)$ and $\max_n(\lambda_n)$ of the amplification matrix, **A**,as function of Δt.

```
theta = 0.5;
dt_vec = logspace(-2,4,1e2);

figure('position',[10 10 900 600])
for i = 1:length(dt_vec)
    A = inv(IM(theta,dt_vec(i)))*EX(theta,dt_vec(i));
    lam = eig(full(A));
    lam_max_FE(i) = max(lam);
    lam_min_FE(i) = min(lam);
end
semilogx(dt_vec,lam_max_FE,'k'), hold on
semilogx(dt_vec,lam_min_FE,'k')
semilogx(dt_vec,ones(size(dt_vec)),'g--','linewidth',2)
semilogx(dt_vec,-ones(size(dt_vec)),'g--','linewidth',2)
semilogx(dt_max*[1 1],[-2 2],'r','linewidth',2), hold off
ylim([-2 2])
xlabel '\Delta t'
ylabel('\lambda')
title 'Stability of Crank-Nicholson method: \theta = 1/2'
```



We can see that the stability condition, $\lambda_n(\mathbf{A}) \leq |1|$, is satisfied for all Δt, so that the CN-method is unconditionally stable. However, note that the eigenvalues become negative at the Neumann condition for the FE-method. Once the amplification matrix has negative eigenvalues, oscillation *may* occur in the CN-method. The oscillations are not unstable, i.e., they don't blow up, they just make you think you screwed up. In my experience, no oscillations occur if the IC and the BC's don't introduce kinks in the solution. However, I often solve problems with kinks in the initial condition and so I typically use the BE-method.

In general, first solve the problem with the BE-method. Only once everything works, switch to the CN-method to gain the additional accuracy. If the switch introduces oscillations, you have to reduce the timestep or smooth the IC. Sometimes it is sufficient to solve one time step with the BE method, which gets rid of all kinks, and then switch to the CN-method.

Solving the transient problem

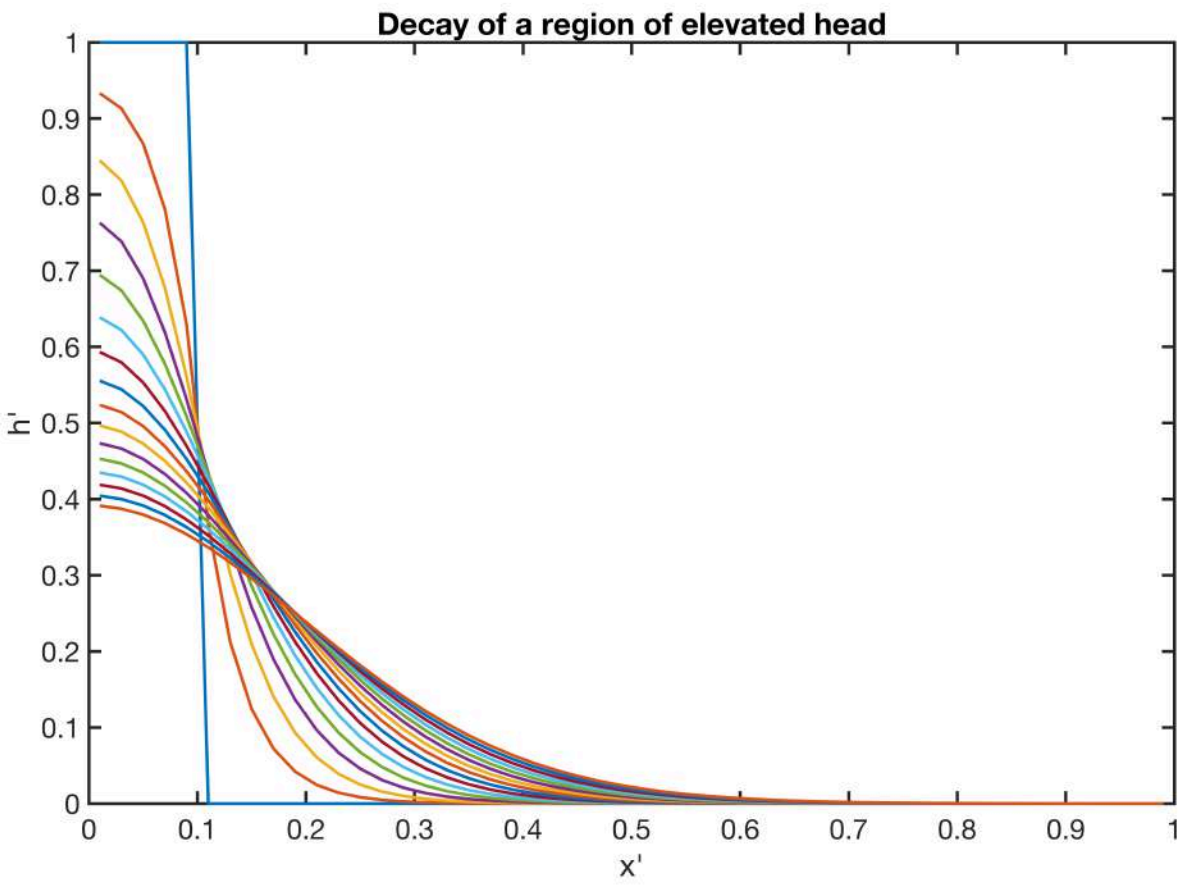
This is just a simple example, the decay of local region of elevated head.

```
tmax = 1e4;
Nt = 15;
dt = tmax/Nt;
fprintf('dt/dt_N = %3.2e.\n',dt/dt_max)
```

```
dt/dt_N = 6.67e+00.
```

```
theta = 0;
Im = IM(theta,dt); Ex = EX(theta,dt);
h = zeros(Grid.Nx,1); h(Grid.xc<=.1) = 1;

time = 0;
figure('position',[10 10 900 600])
plot(Grid.xc,h,'-'), hold on
for i = 1:Nt
    time = time + dt;
    h = solve_lbvp(Im,Ex*h,[],[],I);
    plot(Grid.xc,h,'-'), drawnow
end
hold off
title 'Decay of a region of elevated head'
xlabel 'x''', ylabel 'h'''
```



Oscillation in CN-Method

The Crank-Nicholson method can lead to oscillations when the higher-order assumed in the temporal discretization is violated by a discontinuous initial or boundary condition, as illustrated by the three examples below.

```
theta = .5;
```

```
h0 = zeros(Grid.Nx,1); h0(Grid.xc<=.1)  = 1;
% u = cos(2*pi*Grid.xc);
```

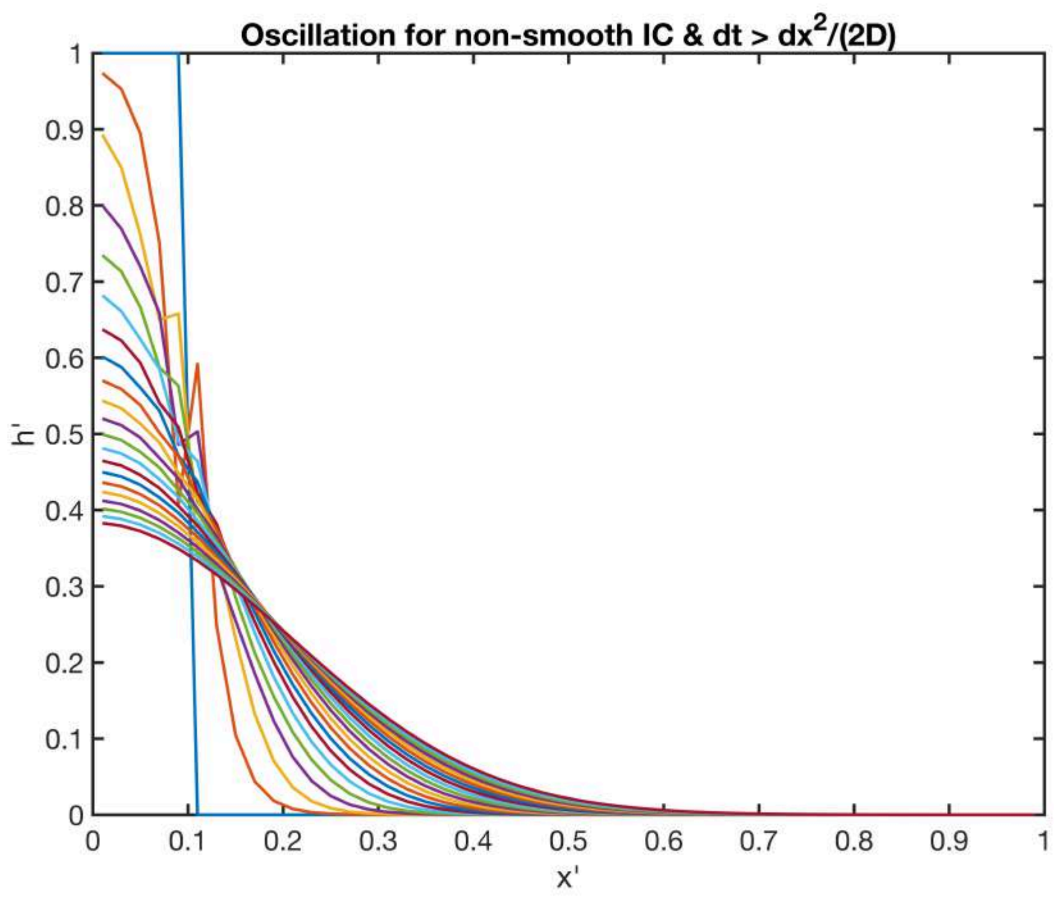
Exmple 1: If the initial condition is not smooth, an oscillation-free solution is only guaranteed for $dt \leq dx^2/(2D)$. For larger time steps the solution can become oscillatory at early times as illustrated in the example below. In contrast to the explicit timestep these oscillation do not grow exponentially because $|\lambda| \leq 1$(see above). As shown in the example these oscillations disappear over time as the solution becomes more smooth.

```
dt = 5*dt_max; % Nx = 50
Nt = ceil(tmax/dt);
fprintf('dt/dt_N = %3.2e.\n',dt/dt_max)
```

```
dt/dt_N = 5.00e+00.
```

```
Im = IM(.5,dt); Ex = EX(.5,dt); h = h0;

figure('position',[10 10 900 600])
plot(Grid.xc,h,'-'), hold on
for i = 1:Nt
    h = solve_lbvp(Im,Ex*h,[],[],I);
    plot(Grid.xc,h,'-'), drawnow
end
hold off
title 'Oscillation for non-smooth IC & dt > dx^2/(2D)'
xlabel 'x'''
ylabel 'h'''
pbaspect([1 .8 1])
```



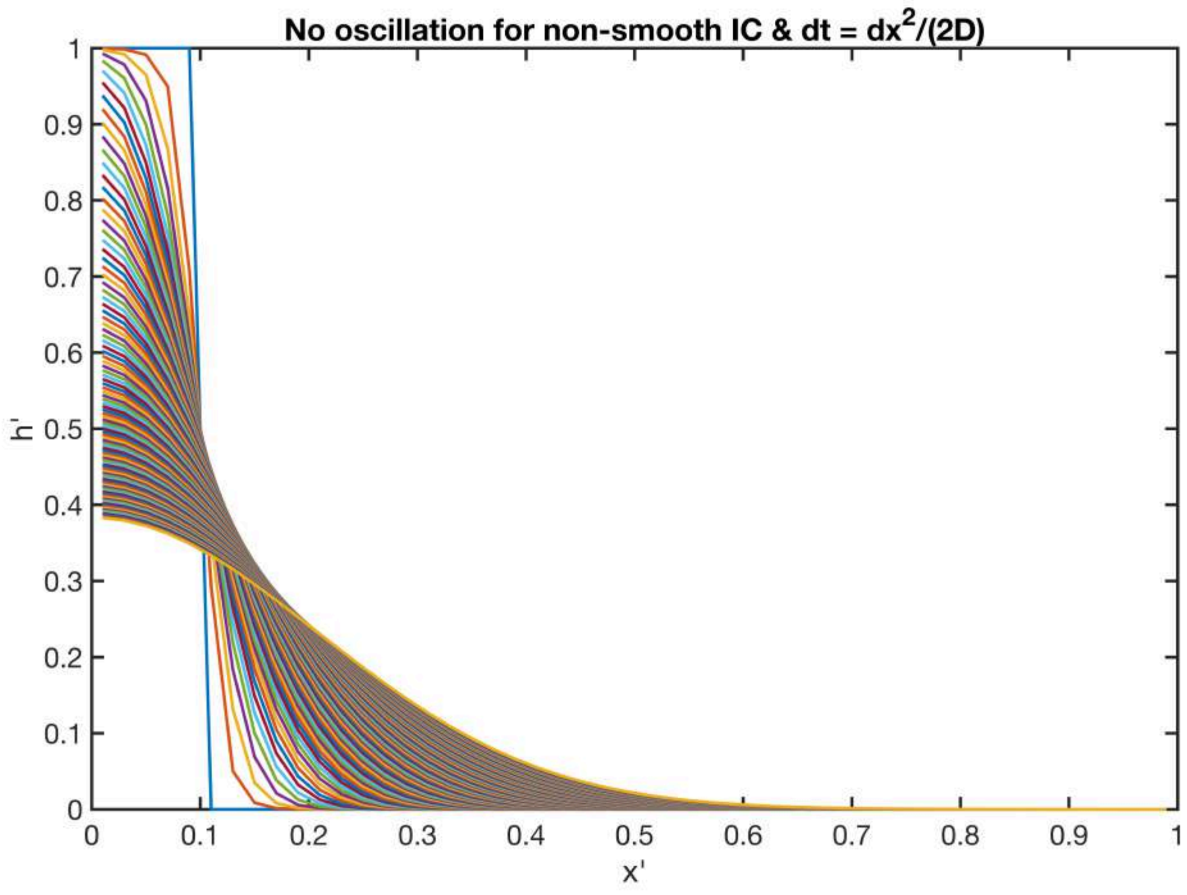
Example 2: Below is an example with the citical timeste, $dt = dx^2/(2D)$, where the solution remains smooth despite the discontinuous initial condition.

```
dt = 1*dt_max;
Nt = ceil(tmax/dt);
fprintf('dt/dt_N = %3.2e.\n',dt/dt_max)
```

```
dt/dt_N = 1.00e+00.
```

```
Im = IM(.5,dt); Ex = EX(.5,dt); h = h0;

figure('position',[10 10 900 600])
plot(Grid.xc,h,'-'), hold on
for i = 1:Nt
    time = time + dt;
    h = solve_lbvp(Im,Ex*h,[],[],I);
    plot(Grid.xc,h,'-'), drawnow
end
hold off
title 'No oscillation for non-smooth IC & dt = dx^2/(2D)'
xlabel 'x'''
ylabel 'h'''
```



Example 3: Here we illustrate that the solution remais smooth if the initial condition is smooth, even if the timestep exceeds the conditon $dt > dx^2/(2D)$. Of course, the same is true for smaller timesteps.

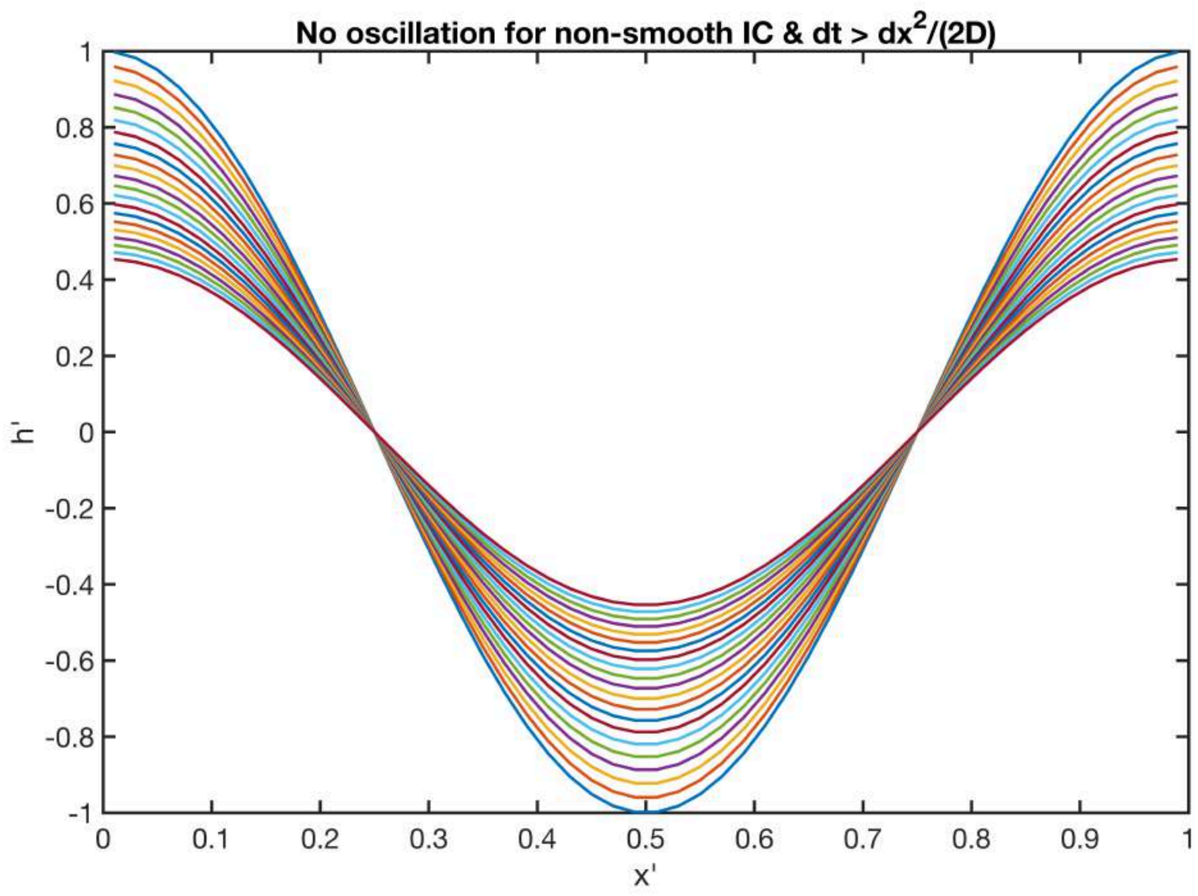
```
dt = 5*dt_max;
Nt = ceil(tmax/dt);
fprintf('dt/dt_N = %3.2e.\n',dt/dt_max)
```

```
dt/dt_N = 5.00e+00.
```

```
Im = IM(.5,dt); Ex = EX(.5,dt);
h = cos(2*pi*Grid.xc);

figure('position',[10 10 900 600])
plot(Grid.xc,h,'-'), hold on
for i = 1:Nt
    time = time + dt;
    h = solve_lbvp(Im,Ex*h,[],[],I);
    plot(Grid.xc,h,'-'), drawnow
end
hold off
title 'No oscillation for non-smooth IC & dt > dx^2/(2D)'
xlabel 'x'''
```

```
ylabel 'h'''
```



Auxillary functions

set_defaults()

```
function [] = set_defaults()
    set(0, ...
    'defaultaxesfontsize',   18, ...
    'defaultaxeslinewidth',   2.0, ...
    'defaultlinelinewidth',   2.0, ...
    'defaultpatchlinewidth',  2.0,...
    'DefaultLineMarkerSize', 12.0);
end
```