Finite difference methods for diffusion processes

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Dec 12, 2012

Note: **VERY PRELIMINARY VERSION**

Contents

1 A simple 1D diffusion equation

We consider a diffusion problem for a quantity $u(x,t)$:

$$\frac{\partial u}{\partial t} = \alpha\frac{\partial^2 u}{\partial x^2}, \quad x\in(0,L),\ t\in(0,T] \tag{1}$$

$$u(x,0) = I(x), \qquad x\in[0,L] \tag{2}$$

$$u(0,t) = 0, \qquad t>0, \tag{3}$$

$$u(L,t) = 0, \qquad t>0\,. \tag{4}$$

Equation (1) is known as a one-dimensional *diffusion equation*, also often referred to as a *heat equation*. With only a first-order derivative in time, only one *initial condition* is needed, while the second-order derivative in time leads to a demand for two *boundary conditions*. The parameter α must be given and is referred to as the *diffusion coefficient*.

Diffusion equations like (1) have a wide range of applications throughout physical, biological, and financial sciences. One of the most common applications is propagation of heat, where $u(x,t)$ represents the temperature of some substance at point x and time t. Chapter **??** goes into several widely occurring applications.

1.1 Forward Euler scheme

The first step in the discretization procedure is to replace the domain $[0,L] \times [0,T]$ by a set of mesh points. Here we apply equally spaced mesh points

$$x_i = i\Delta x, \quad i = 0, \ldots, N_x,$$

and

$$t_n = n\Delta t, \quad n = 0, \ldots, N\,.$$

Moreover, u_i^n denotes the numerical approximation of $u(x_i, t_n)$. Approximating the PDE (1) at a mesh point (x_i, t_n) leads to

$$\frac{\partial}{\partial t}u(x_i,t_n) = \alpha\frac{\partial^2}{\partial x^2}u(x_i,t_n), \tag{5}$$

Let us first replace the time derivative by a forward difference and the spatial derivative by a central difference:

$$[D_t^+ u = \alpha D_x D_x u]_i^n\,. \tag{6}$$

Written out,

$$\frac{u^{n+1}-u^n}{\Delta t} = \alpha\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}\,. \tag{7}$$

As usual, we anticipate that u_i^n is already computed such that u_i^{n+1} is the only unknown, which we can easily solve for:

$$u_i^{n+1} = u_i^n + \alpha\frac{\Delta t}{\Delta x^2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right)\,. \tag{8}$$

The computational algorithm is then

1. compute $u_i^0 = I(x_i)$for $i = 0, \ldots, N_x$
2. for $n = 0, 1, \ldots, N$:
   (a) apply (8) for all the internal spatial points $i = 1, \ldots, N_x - 1$

(b) set the boundary values $u_i^{n+1} = 0$ for $i = 0$ and $i = N_x$

The algorithm is compactly fully specified in Python:

```
x = linspace(0, L, Nx+1)   # mesh points in space
dx = x[1] - x[0]
t = linspace(0, T, N+1)    # mesh points in time
dt = t[1] - t[0]
C = a*dt/dx**2
u   = zeros(Nx+1)
u_1 = zeros(Nx+1)

# Set initial condition u(x,0) = I(x)
for i in range(0, Nx+1):
    u_1[i] = I(x[i])

for n in range(0, N):
    # Compute u at inner mesh points
    for i in range(1, Nx):
        u[i] = u_1[i] + C*(u_1[i-1] - 2*u_1[i] + u_1[i+1])

    # Insert boundary conditions
    u[0] = 0;  u[Nx] = 0

    # Update u_1 before next step
    u_1[:]= u
```

1.2 Backward Euler Scheme

We now apply a backward difference in time in (5), but the same central difference in space:

$$[D_t^- u = D_x D_x u]_i^n, \tag{9}$$

which written out reads

$$\frac{u_i^n - u^n - 1_i}{\Delta t} = \alpha \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}\,. \tag{10}$$

Now we assume u_i^{n-1} is computed, but all quantities at the "new" time level n are unknown. This time it is not possible to solve with respect to u_i^n because this value couples to its neighbors in space, u_{i-1}^n and u_{i+1}^n, which are also unknown. Let us examine this fact for the case when $N_x = 3$. Equation (10) written for $i = 1, \ldots, Nx - 1 = 1, 2$ becomes

$$\frac{u_1^n - u_1^{n-1}}{\Delta t} = \alpha \frac{u_2^n - 2u_1^n + u_0^n}{\Delta x^2} \tag{11}$$

$$\frac{u_2^n - u_2^{n-1}}{\Delta t} = \alpha \frac{u_3^n - 2u_2^n + u_1^n}{\Delta x^2} \tag{12}$$

The boundary values u_0^n and u_3^n are known as zero. Collecting the unknown new values u_1^n and u_2^n on the left-hand side gives

$$\left(1 + 2\alpha\frac{\Delta t}{\Delta x^2}\right) u_1^n - \alpha\frac{\Delta t}{\Delta x^2} u_2^n = u^n - 1_1, \tag{13}$$

$$-\alpha\frac{\Delta t}{\Delta x^2} u_2^n + \left(1 + 2\alpha\frac{\Delta t}{\Delta x^2}\right) u_2^n = u^n - 1_2\,. \tag{14}$$

This is a coupled 2×2 system of algebraic equations for the unknowns u_1^n and u_2^n. Discretization methods that lead to a coupled system of equations for the unknown function at a new time level are said to be *implicit methods*. The counterpart, *explicit methods*, refers to discretization methods where there is a simple explicit formula for the values of the unknown function at each of the spatial mesh points at the new time level. From an implementational point of view, implicit methods are more comprehensive to code since they require the solution of coupled equations, i.e., a matrix system, at each time level.

In the general case, (10) gives rise to a coupled $(Nx-1) \times (Nx-1)$ system of algebraic equations for all the unknown u_i^n at the interior spatial points $i = 1, \ldots, Nx-1$. Collecting the unknowns on the left-hand side, and introducing the quantity

$$C = \alpha\frac{\Delta t}{\Delta x^2}, \tag{15}$$

(10) can be written

$$-Cu_{i-1}^n + (1+2C)\, u_i^n - Cu_{i+1}^n) = u_{i-1}^n, \tag{16}$$

for $i = 1, \ldots, Nx-1$. One can either view these equations as a system for where the u_i^n values at the internal grid points, $i = 1, \ldots, N_x - 1$, are unknown, or we may append the boundary values u_0^n and $u_{N_x}^n$ to the system. In the latter case, all u_i^n for $i = 0, \ldots, N_x$ are unknown and we must add the boundary equations to the $N_x - 1$ equations in (16):

$$u_0^n = 0, \tag{17}$$

$$u_{N_x}^n = 0\,. \tag{18}$$

A coupled system of algebraic equations can be written on matrix form, and this is important if we want to call up ready-made software for solving the system. The equations (16) and (17)–(18) correspond to the matrix equation

$$AU = b$$

where $U = (u_0^n, \ldots, u_{N_x}^n)$, and the matrix A has the following structure:

$$A = \begin{pmatrix} A_{0,0} & A_{0,1} & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ A_{1,0} & A_{1,1} & 0 & \ddots & & & & & \vdots \\ 0 & A_{2,1} & A_{2,2} & A_{2,3} & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & A_{i,i-1} & A_{i,i} & A_{i,i+1} & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & \ddots & \ddots & A_{N_x-1,N_x} \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & A_{N_x,N_x-1} & A_{N_x,N_x} \end{pmatrix} \tag{19}$$

The nonzero elements are given by

$$A_{i,i-1} = -C \tag{20}$$
$$A_{i,i} = 1 + 2C \tag{21}$$
$$A_{i,i+1} = -C \tag{22}$$

for the equations for internal points, $i = 1, \ldots, N_x - 1$. The equations for the boundary points correspond to

$$A_{0,0} = 1, \tag{23}$$
$$A_{0,1} = 0, \tag{24}$$
$$A_{N_x,N_x-1} = 0, \tag{25}$$
$$A_{N_x,N_x} = 1\,. \tag{26}$$

The right-hand side b is written as

$$b = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_i \\ \vdots \\ b_{N_x} \end{pmatrix} \tag{27}$$

with

$$b_0 = 0, \tag{28}$$
$$b_i = u_i^{n-1}, \quad i = 1, \ldots, N_x - 1, \tag{29}$$
$$b_{N_x} = 0\,. \tag{30}$$

We observe that the matrix A contains quantities that do not change in time. Therefore, A can be formed once and for all before we enter the recursive formulas for the time evolution. The right-hand side b, however, must be updated at each time step. This leads to the following computational algorithm, here sketched with Python code:

```
x = linspace(0, L, Nx+1)   # mesh points in space
dx = x[1] - x[0]
t = linspace(0, T, N+1)    # mesh points in time
u   = zeros(Nx+1)
u_1 = zeros(Nx+1)

# Data structures for the linear system
A = zeros((Nx+1, Nx+1))
b = zeros(Nx+1)

for i in range(1, Nx):
    A[i,i-1] = -C
    A[i,i+1] = -C
    A[i,i] = 1 + 2*C
A[0,0] = A[Nx,Nx] = 1

# Set initial condition u(x,0) = I(x)
for i in range(0, Nx+1):
    u_1[i] = I(x[i])

import scipy.linalg

for n in range(0, N):
    # Compute b and solve linear system
    for i in range(1, Nx):
        b[i] = -u_1[i]
    b[0] = b[Nx] = 0
    u[:] = scipy.linalg.solve(A, b)

    # Update u_1 before next step
    u_1[:] = u
```

1.3 Sparse matrix implementation

We have seen from (19) that the matrix A is tridiagonal. The code segment above used a full, dense matrix representation of A, which stores a lot of values we know are zero beforehand, and worse, the solution algorithm computes with all these zeros. With $N_x + 1$ unknowns, the work by the solution algorithm is $\frac{1}{3}(N_x+1)^3$ and the storage requirements $(N_x+1)^2$. By utilizing the fact that A is tridiagonal and employing corresponding software tools, the work and storage demands can be proportional to N_x only.

The key idea is to apply a data structure for a tridiagonal or sparse matrix. The `scipy.sparse` package has relevant utilities. For example, we can store the nonzero diagonals of a matrix. The package also has linear system solvers that operate on sparse matrix data structures. The code below illustrates how we can store only the main diagonal and the upper and lower diagonals.

```
# Representation of sparse matrix and right-hand side
main  = zeros(Nx+1)
lower = zeros(Nx-1)
upper = zeros(Nx-1)
b     = zeros(Nx+1)

# Precompute sparse matrix
main[:] = 1 + 2*C
lower[:] = -C  #1
upper[:] = -C  #1
# Insert boundary conditions
main[0] = 1
main[Nx] = 1

A = scipy.sparse.diags(
    diagonals=[main, lower, upper],
    offsets=[0, -1, 1], shape=(Nx+1, Nx+1),
    format='csr')
print A.todense()

# Set initial condition
for i in range(0,Nx+1):
    u_1[i] = I(x[i])

for n in range(0, N):
    b = u_1
    b[0] = b[-1] = 0.0  # boundary conditions
    u[:] = scipy.sparse.linalg.spsolve(A, b)
    u_1[:] = u
```

The `scipy.sparse.linalg.spsolve` function utilizes the sparse storage structure of `A` and performs in this case a very efficient Gaussian elimination solve.

1.4 The θ Rule

The θ rule provides a family of finite difference approximations in time. Applied to the 1D diffusion problem we have

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha \left(\theta \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} + (1-\theta)\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \right) .$$

This scheme also leads to a matrix system with entries $1 + 2C\theta$ on the main diagonal of the matrix, and $-C\theta$ on the super- and sub-diagonal. The right-hand side entry b_i is

$$b_i = u_i^n + C(1-\theta)\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} .$$

1.5 The Laplace and Poisson equation as the steady state limit

The Laplace equation, $\nabla^2 u = 0$, or the Poisson equation, $-\nabla^2 u = f$, occur in numerous applications throughout science and engineering. We can solve

1D variants of the Laplace equations with the listed software, because we can interpret $u_{xx} = 0$ as the limiting solution of $u_t = \alpha u_{xx}$ when u reach a steady state limit where $u_t = 0$. Technically in a program, we just take one large time step, or equivalently, set α to a large value. All we need is to have C large. As $C \to \infty$, we can from the schemes see that we get

$$\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} = 0,$$

which is nothing but the discretization $[D_x D_x u]_i^{n+1} = 0$ of $u_{xx} = 0$.

The Backward Euler scheme can solve the limit equation directly and hence produce a solution of the 1D Laplace equation. With the Forward Euler scheme we must do the time stepping since $C > 1/2$ is illegal and leads to instability. We may interpret this time stepping as solving the equation system from u_{xx} by iterating on a time pseudo time variable.

1.6 Extensions

- Variable coefficients
- Neumann and Robin conditions
- 2D and 3D

2 Analysis of schemes for the diffusion equation

2.1 Properties of the solution

A particular characteristic of diffusive processes, governed by an equation like

$$u_t = \alpha u_{xx}, \tag{31}$$

is that the initial shape $u(x, 0) = I(x)$ spreads out in space with time, along with a decaying amplitude. For example, (31) admits a solution of the form

$$u(x, t) = Qe^{-at} \sin(kx), . \tag{32}$$

The parameters Q and k can be freely chosen, while inserting (32) in (31) gives the constraint

$$a = -\alpha k^2 .$$

A very important feature is that the initial shape $I(x) = Q \sin kx$ undergoes a damping $\exp(-\alpha k^2 t)$, meaning that rapid oscillations in space, corresponding to large k, are very much faster dampened than slow oscillations in space, corresponding to small k. This feature leads to a smoothing of the initial condition with time.

The following examples illustrates the damping properties of the diffusion equation. We consider the problem

$$\begin{aligned} u_t &= u_{xx}, \quad x \in (0,1),\ t \in (0,T], \\ u(0,t) &= u(1,t) = 0, \quad t \in (0,T], \\ u(x,0) &= \sin(\pi x) + 0.1\sin(100\pi x)\,. \end{aligned}$$

The initial condition has been chosen such that adding two solutions like (32) constructs an analytical solution to the problem:

$$u(x,t) = e^{-\pi^2 t}\sin(\pi x) + 0.1e^{-\pi^2 10^4 t}\sin(100\pi x)\,. \tag{33}$$

Figure 1 illustrates the rapid damping of rapid oscillations $\sin(100\pi x)$ and the very much slower damping of the slowly varying $\sin(\pi x)$ term. After about $t = 0.5 \cdot 10^{-4}$ the rapid oscillations do not have a visible amplitude, while we have to wait until $t \sim 0.5$ before the amplitude of the long wave $\sin(\pi x)$ becomes invisible.

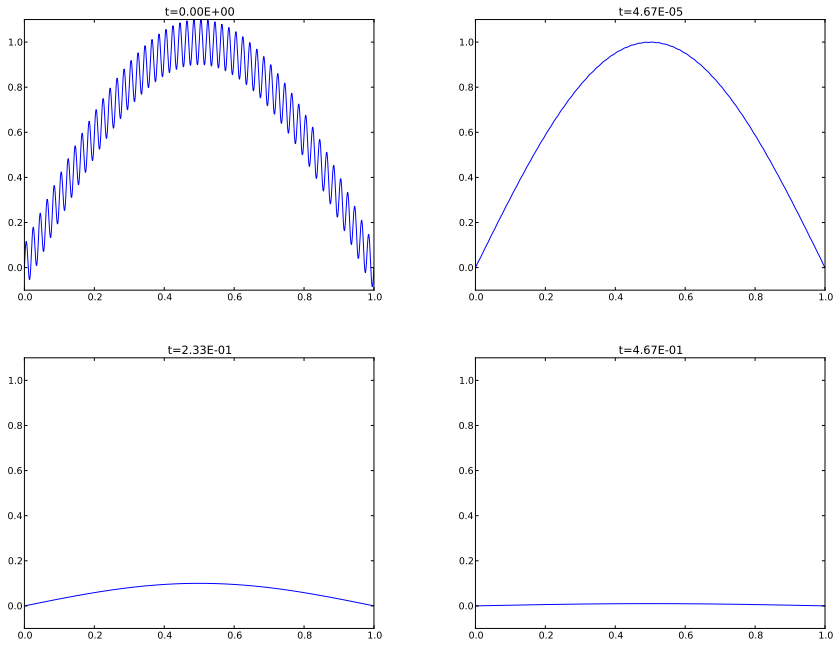

Figure 1: Evolution of the solution of a diffusion problem: initial condition (upper left), 1/100 reduction of the small waves (upper right), 1/10 reduction of the long wave (lower left), and 1/100 reduction of the long wave (lower right).

The relevance of studying the behavior of a particular solution of the form (32) is related to a Fourier representation of the solution. We first express the initial condition as a Fourier series

$$I(x) \approx \sum_{k \in K} b_k e^{ikx}, \tag{34}$$

where $i = \sqrt{-1}$ and K is a set of k values needed to build $I(x)$ with sufficient accuracy from basic sinusoidal components e^{ikx}. Instead of using a specific sine or cosine function for the spatial variation, we use a complex exponential function to ease the hand calculations later. Letting K contain infinitely many k values makes the sum converge to $I(x)$ under reasonable assumptions on the smoothness of $I(x)$.

The corresponding solution u can now be expressed as

$$u(x,t) \approx \sum_{k \in K} b_k e^{-\alpha k^2 t} e^{ikx}, \, . \tag{35}$$

Note that (33) is a special case of (35) where $K = \{\pi, 100\pi\}$, $b_\pi = 1$, and $b_{100\pi} = 0.1$.

2.2 Analysis of the finite difference schemes

We have seen that a general solution of the diffusion equation can be built as a linear combination of basic components

$$e^{-\alpha k^2 t} e^{ikx} \, .$$

A fundamental question is whether such components are also solutions of the finite difference schemes. This is indeed the case, but the amplitude $\exp(-\alpha k^2 t)$ might be modified (which also happens when solving the ODE counterpart $u' = -\alpha u$). We therefore look for numerical solutions of the form

$$u_q^n = A^n e^{ikq\Delta x} = A^n e^{ikx}, \tag{36}$$

where A must be determined by inserting the component into an actual scheme.

Stability. The exact wave component decays according to $\exp(-\alpha k^2 t)$. We should therefore require $|A| < 1$ to have a decaying numerical solution as well. However, if $-1 \leq A < 0$, A^n will change sign from time level to time level, and we get stable, non-physical oscillations in the numerical solutions that are not present in the exact solution.

Accuracy. To determine how accurately a finite difference scheme treats one wave component (36), we see that the basic deviation from the exact solution is reflected in how well A^n approximates $\exp(-\alpha k^2 t) = \exp(-\alpha k^2 n \Delta t)$, or how well the numerical amplification factor A approximates the exact amplification factor $A_\mathrm{e} = \exp(-\alpha k^2 \Delta t)$.

2.3 Analysis of the Forward Euler scheme

The Forward Euler finite difference scheme for $u_t = \alpha u_{xx}$ can be written as

$$[D_t^+ u = \alpha D_x D_x u]_q^n \,.$$

Inserting a wave component (36) in the scheme demands calculating the terms

$$e^{ikq\Delta x}[D_t^+ A]^n = e^{ikq\Delta x} A^n \frac{A-1}{\Delta t},$$

and

$$A^n D_x D_x [e^{ikx}]_q = A^n \left(-e^{ikq\Delta x} \frac{4}{\Delta x^2} \sin^2\left(\frac{k\Delta x}{2}\right)\right) \,.$$

Inserting these terms in the discrete equation and dividing by $A^n e^{ikq\Delta x}$ leads to

$$\frac{A-1}{\Delta t} = -\alpha \frac{4}{\Delta x^2} \sin^2\left(\frac{k\Delta x}{2}\right),$$

and consequently

$$A = 1 - 4C \sin^2\left(\frac{k\Delta x}{2}\right), \tag{37}$$

where C is a constant introduced for convenience:

$$C = \frac{\alpha \Delta t}{\Delta x^2} \,. \tag{38}$$

The complete numerical solution is then

$$u_q^n = \left(1 - 4C \sin^2\left(\frac{k\Delta x}{2}\right)\right)^n e^{ikq\Delta x} \,. \tag{39}$$

Stability. We easily see that $A \leq 1$, but $A < -1$ might be a possibility that will lead to growth of a numerical wave component. The criterion $A \geq -1$ implies

$$4C \sin^2(p/2) \leq 2 \,.$$

The worst case is when $\sin^2(p/2) = 1$, so a sufficient criterion for stability is

$$C \leq \frac{1}{2}, \tag{40}$$

or expressed as a condition on Δt:

$$\Delta t \leq \frac{\Delta x^2}{2\alpha} \,. \tag{41}$$

Note that halving the spatial mesh size, $\Delta x \rightarrow \frac{1}{2}\Delta x$, requires Δt to be reduced by a factor of 1/4. The method hence becomes very expensive for fine spatial meshes.

Accuracy. Since A is expressed in terms of C and the parameter we now call $p = k\Delta x$, we also express A_{e} by C and p:

$$A_{\mathrm{e}} = \exp\left(-\alpha k^2 \Delta t\right) = \exp\left(-Cp^2\right).$$

Computing the Taylor series expansion of A/A_{e} in terms of C can easily be done with aid of `sympy`:

```
def A_exact(C, p):
    return exp(-C*p**2)

def A_FE(C, p):
    return 1 - 4*C*sin(p/2)**2

from sympy import *
C, p = symbols('C p')
A_err_FE = A_FE(C, p)/A_exact(C, p)
print A_err_FE.series(C, 0, 6)
```

The result is

$$\frac{A}{A_{\mathrm{e}}} = 1 - 4C\sin^2(p/2) + Cp^2 - 4C^2p^2\sin^2(p/2) + \frac{1}{2}C^2p^4 + \cdots$$

Recalling that $C = \alpha\Delta t/\Delta x$, $p = k\Delta x$, and that $\sin^2(p/2) \leq 1$, we realize that the dominating error terms are at most

$$1 - 4\alpha\frac{\Delta t}{\Delta x^2} + \alpha\Delta t - 4\alpha^2\Delta t^2 + \alpha^2\Delta t^2\Delta x^2 + \cdots.$$

2.4 Analysis of the Backward Euler scheme

Discretizing $u_t = \alpha u_{xx}$ by a Backward Euler scheme,

$$[D_t^- u = \alpha D_x D_x u]_q^n,$$

and inserting a wave component (36), leads to calculations similar to those arising from the Forward Euler scheme, but since

$$e^{ikq\Delta x}[D_t^- A]^n = A^n e^{ikq\Delta x}\frac{1 - A^{-1}}{\Delta t},$$

we get

$$\frac{1 - A^{-1}}{\Delta t} = -\alpha\frac{4}{\Delta x^2}\sin^2\left(\frac{k\Delta x}{2}\right),$$

and then

$$A = \left(1 + 4C\sin^2(p/2)\right)^{-1} . \tag{42}$$

The complete numerical solution can be written

$$u_q^n = \left(1 + 4C\sin^2(p/2)\right)^{-n} e^{ikq\Delta x} . \tag{43}$$

Stability. We see from (42) that $0 < A < 1$, which means that all numerical wave components are stable and non-oscillatory for any $\Delta t > 0$.

2.5 Analysis of the Crank-Nicolson scheme

The Crank-Nicolson scheme can be written as

$$[D_t u = \alpha D_x D_x \overline{u}^x]_q^{n+\frac{1}{2}},$$

or

$$[D_t u]_q^{n+\frac{1}{2}} = \frac{1}{2}\alpha\left([D_x D_x u]_q^n + [D_x D_x u]_q^{n+1}\right) .$$

Inserting (36) in the time derivative approximation leads to

$$[D_t A^n e^{ikq\Delta x}]^{n+\frac{1}{2}} = A^{n+\frac{1}{2}} e^{ikq\Delta x}\frac{A^{\frac{1}{2}} - A^{-\frac{1}{2}}}{\Delta t} = A^n e^{ikq\Delta x}\frac{A-1}{\Delta t} .$$

Inserting (36) in the other terms and dividing by $A^n e^{ikq\Delta x}$ gives the relation

$$\frac{A-1}{\Delta t} = -\frac{1}{2}\alpha\frac{4}{\Delta x^2}\sin^2\left(\frac{k\Delta x}{2}\right)(1+A),$$

and after some more algebra,

$$A = \frac{1 - 2C\sin^2(p/2)}{1 + 2C\sin^2(p/2)} . \tag{44}$$

The exact numerical solution is hence

$$u_q^n = \left(\frac{1 - 2C\sin^2(p/2)}{1 + 2C\sin^2(p/2)}\right)^n e^{ikp\Delta x} . \tag{45}$$

Stability. The criteria $A > -1$ and $A < 1$ are fulfilled for any $\Delta t > 0$.

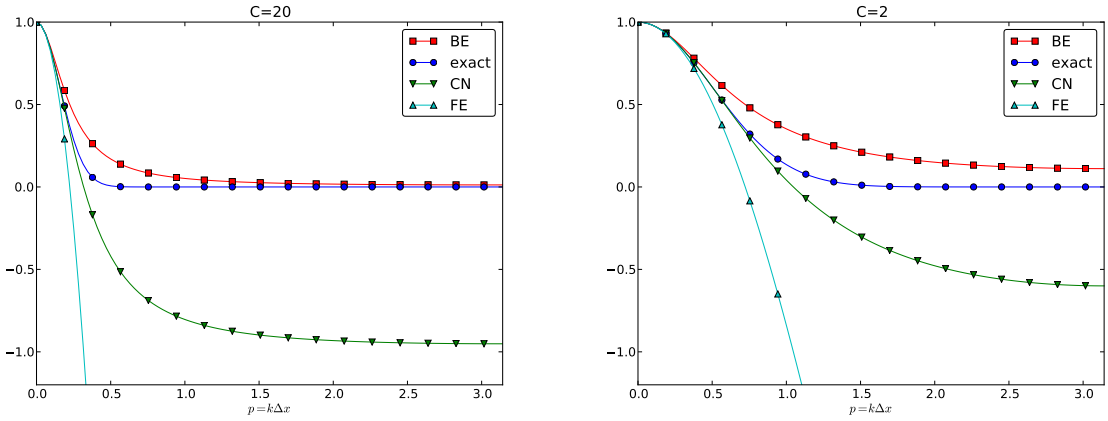

Figure 2: Error in amplification factor for large time steps.

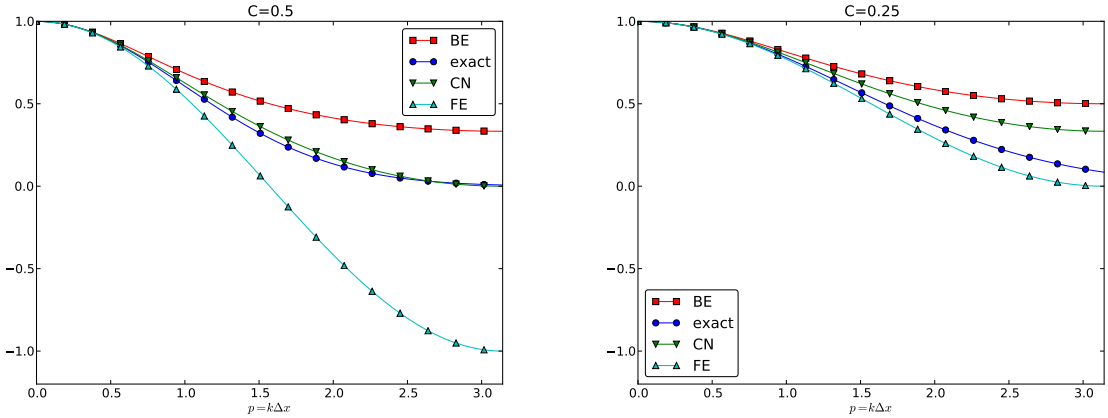

Figure 3: Error in amplification factor for time steps around the Forward Euler stability limit.

2.6 Summary of accuracy of amplification factors

We can plot the various amplification factors against $p = k\Delta x$ for different choices of the C parameter. Figures 2, 3, and 4 show how long and small waves are damped by the various schemes compared to the exact damping. As long as all schemes are stable, the amplification factor is positive, except for Crank-Nicolson when $C > 0.5$.

The effect of negative amplification factors is that A^n changes sign from one time level to the next, thereby giving rise to oscillations in time in an animation of the solution. We see from Figure 2 that for $C = 20$, waves with $p \geq \pi/2$ undergo a damping close to -1, which means that the amplitude does not decay and that the wave component jumps up and down in time. For $C = 2$ we have a damping of a factor of 0.5 from one time level to the next, which is very much

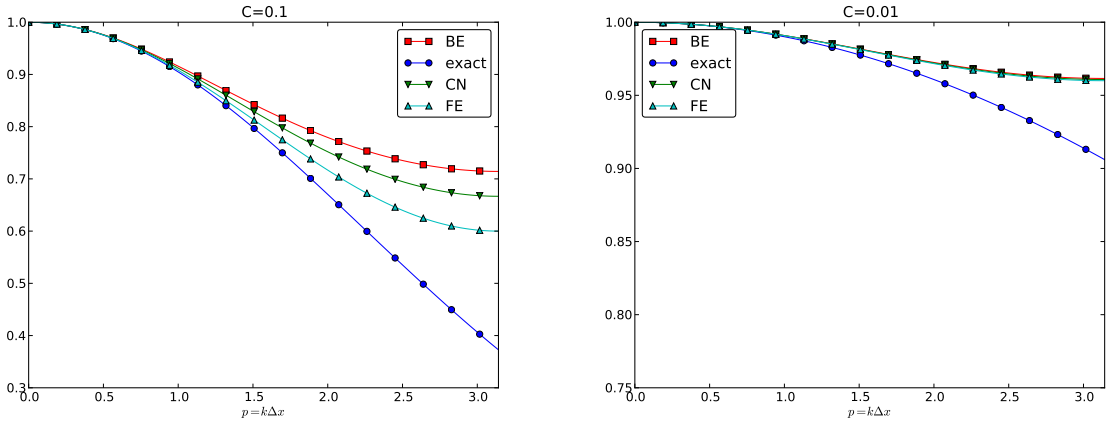

Figure 4: Error in amplification factor for small time steps.

smaller than the exact damping. Short waves will therefore fail to be effectively dampened. These waves will manifest themselves as high frequency oscillatory noise in the solution.

A value $p = \pi/2$ corresponds to four mesh points per wave length of e^{ikx}, while $p = \pi$ implies only two points per wave length, which is the smallest number of points we can have to represent the wave on the mesh.

To demonstrate the oscillatory behavior of the Crank-Nicolson scheme, we choose an initial condition that leads to short waves with significant amplitude. A discontinuous $I(x)$ will in particular serve this purpose.

Run $C =$ .....

Index