The 1D diffusion equation

The famous diffusion equation, also known as the heat equation, reads

$$ \frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} $$

Here,

- \( u(x,t) \): unknown
- \( \alpha \): diffusion coefficient

Alternative, compact notation:

$$ u_t = \alpha u_{xx} $$

The initial-boundary value problem for 1D diffusion

The 1D diffusion equation

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

$$ u(x,0) = I(x), \quad x \in [0,L] $$ (2)

$$ u(0,t) = 0, \quad t > 0 $$ (3)

$$ u(L,t) = 0, \quad t > 0 $$ (4)

Mesh in time:

$$ 0 = t_0 < t_1 < \cdots < t_{N_{t}} - 1 < t_{N_{t}} = T $$ (5)

Mesh in space:

$$ 0 = x_0 < x_1 < \cdots < x_{N_{x} - 1} < x_{N_{x}} = L $$ (6)

Uniform mesh with constant mesh spacings \( \Delta t \) and \( \Delta x \):

$$ x_i = i\Delta x, \quad i = 0, \ldots, N_{x}, \quad t_n = n\Delta t, \quad n = 0, \ldots, N_{t} $$ (7)

The discrete solution

The numerical solution is a mesh function:

$$ u^n_i \approx u(x_i,t_n) $$

Finite difference stencil (or scheme): equation for \( u^n_i \) involving neighboring space-time points

Step 1: Discretizing the domain

Mesh in time:

$$ 0 = t_0 < t_1 < \cdots < t_{N_{t}} - 1 < t_{N_{t}} = T $$ (5)

Mesh in space:

$$ 0 = x_0 < x_1 < \cdots < x_{N_{x} - 1} < x_{N_{x}} = L $$ (6)

Uniform mesh with constant mesh spacings \( \Delta t \) and \( \Delta x \):

$$ x_i = i\Delta x, \quad i = 0, \ldots, N_{x}, \quad t_n = n\Delta t, \quad n = 0, \ldots, N_{t} $$ (7)

Step 2: Filling the equation at the mesh points

Require the PDE (1) to be fulfilled at arbitrary interior mesh points \((x_i,t_n)\) holds so

$$ \frac{\partial}{\partial t} u(x_i,t_n) = \alpha \frac{\partial^2}{\partial x^2} u(x_i,t_n) $$ (8)

Applies to all interior mesh points: \( i = 1, \ldots, N_{x} - 1 \) and \( n = 1, \ldots, N_{t} - 1 \)

For \( n = 0 \) we have the initial conditions \( u(x) = I(x) \) and \( u_t = 0 \)

At the boundaries \( i = 0, N_{x} \), we have the boundary condition \( u = 0 \).
The computational algorithm for the Forward Euler scheme

1. **Compute** \( u_0 = I(x_i), i = 0, \ldots, N_x \)
2. **For** \( n = 0, 1 \ldots \) can be updated in any sequence, but the time levels \( t_n \) must be updated in chronological order: \( t_n \) before \( t_{n+1} \).

**Notice**
- We visit one mesh point \((x_i, t_n)\) at a time, and we have an explicit formula for computing the associated \( u_{n+1} \) value.
- The spatial points can be updated in any sequence, but the time level \( t_n \) must be updated in chronological order: \( t_n \) before \( t_{n+1} \).

The finite difference stencil

\[
\begin{align*}
\frac{u_{n+1}^i - u_n^i}{\Delta t} &= \alpha \frac{u_{n+1}^i - 2u_n^i + u_{n-1}^i}{\Delta x^2} \\
\text{Initial condition: } u_n^0 &= I(x_i), i = 0, 1, \ldots, N_x
\end{align*}
\]

Observe:
- There is only one parameter, \( F \), in the discrete model: \( F \) lumps all mesh parameters \( \Delta t \) and \( \Delta x \), with the only physical parameter, the diffusion coefficient \( \alpha \).
- The value \( F \) and the smoothness of \( I(x) \) govern the quality of the numerical solution.

The Python implementation of the computational algorithm

```python
x = linspace(0, L, Nx+1)  # mesh points in space
dx = x[1] - x[0]
# mesh points in time
t = linspace(0, T, Nt+1)
dt = t[1] - t[0]
F = a*dt/dx**2
# Set initial condition u(x,0) = I(x)
for i in range(0, Nx+1):
    u_1[i] = I(x[i])
# Insert boundary conditions
u_1[0] = 0; u_1[Nx] = 0
# or more efficient switch of references
u_1, u = u, u_1
# Set initial condition u(x,0) = I(x)
for i in range(0, Nx+1):
    u_1[i] = I(x[i])
# Insert boundary conditions
u_1[0] = 0; u_1[Nx] = 0
# or more efficient switch of references
u_1, u = u, u_1
# Initialize u for the first time step
u_1[1] = u
# For more efficient switch of references
u_1, u = u, u_1
```

Solve the discretized PDE for the unknown \( u_{n+1}^i \):

\[
\frac{u_{n+1}^i - u_n^i}{\Delta t} = \alpha \frac{u_{n+1}^i - 2u_n^i + u_{n-1}^i}{\Delta x^2}
\]

where

\[
F = \frac{\alpha \Delta t}{\Delta x^2}
\]
Moving finite difference stencil

web page or a movie file.

Demo program

Program: diffu1D_u0.py
Produce animation on the screen
Each frame stored in tmp_frame%04d.png
les
tmp_frame0000.png, tmp_frame0001.png, ...

How to make movie file in modern formats:

Terminal> name=tmp_frame%04d.png
Terminal> fps=8 # frames per second in movie
Terminal> avconv -r $fps -i $name -vcodec flv movie.flv
Terminal> avconv -r $fps -i $name -vcodec libx64 movie.mp4
Terminal> avconv -r $fps -i $name -vcodec libvpx movie.webm
Terminal> avconv -r $fps -i $name -vcodec libtheora movie.ogg

Forward Euler applied to an initial plug profile

Nx = 50. The method results in a growing, unstable solution if F > 0.5.
Choosing F = 0.5 gives a sawtooth-like curve.
Link to movie file
Lowering F to 0.25 gives a smooth (expected) solution.
Link to movie file

Forward Euler applied to a Gaussian profile

 Nx = 50. F = 0.5.
Link to movie file

Backward Euler scheme

Backward difference in time, centered difference in space:

\[ D_t u = D_x D_x u \]n
i (12)

Written out:

\[ u^n_i - u^{n-1}_i \Delta t = \alpha (u^n_{i+1} - 2u^n_i + u^n_{i-1}) \Delta x^2 \]n
i (13)

Assumption: \( u^0_i \) is computed, but all quantities at the new time level \( t_n \) are unknown.

Notice

We cannot solve for \( u^n_i \) because the unknown value is coupled to two other unknown values: \( u^0_{i+1} \) and \( u^0_{i-1} \). That is, all the new unknown values are coupled to each other in a linear system of algebraic equations.

Let's write out the equations for \( N_x = 3 \)

Equation (13) written for \( i = 1, ..., N_x - 1 = 2 \) becomes

\[ \frac{u^n_i - u^{n-1}_i}{\Delta t} = \alpha \frac{u^n_{i+1} - 2u^n_i + u^n_{i-1}}{\Delta x^2} \]n
i (14)

\[ \frac{u^n_{i+1} - u^{n-1}_{i+1}}{\Delta t} = \alpha \frac{u^n_{i} - 2u^n_{i+1} + u^n_{i+2}}{\Delta x^2} \]n
i (15)

(The boundary values \( u^0_i \) and \( u^0_{N_x+1} \) are known as zero.)

Collecting the unknown values on the left-hand side and writing as \( 2 \times 2 \) matrix system:

\[
\begin{pmatrix}
1 + 2F & -F \\
-F & 1 + 2F
\end{pmatrix}
\begin{pmatrix}
u^n_1 \\
u^n_2
\end{pmatrix} = \begin{pmatrix}
u^{n-1}_1 \\
u^{n-1}_2
\end{pmatrix}
\]
Two classes of discretization methods: explicit and implicit

**Explicit**
Discretization methods that avoid linear systems and have an explicit formula for each new value of the unknown are called explicit methods.

**Implicit**
Discretization methods that lead linear systems are known as implicit methods.

The right-hand side is very sparse: a tridiagonal matrix

Two classes of discretization methods: explicit and implicit

Naive Python implementation with a dense 
(Nx + 1) × (Nx + 1) matrix

```python
x = linspace(0, L, Nx+1) # mesh points in space
t = linspace(0, T, N+1) # mesh points in time
u = zeros(Nx+1)
b = zeros(Nx+1)

# Set initial condition u(x,0) = I(x)
A[0,0] = A[Nx,Nx] = 1
for i in range(1, Nx):
    A[i,i] = 1 + 2*F
    A[i,i+1] = -F
    A[i,i-1] = -F

# Compute b and solve linear system
b[0] = b[Nx] = 0
u[:] = scipy.linalg.solve(A, b)
# Update u_1 before next step
u_1, u = u, u_1
```

The right-hand side

\[
A = \begin{pmatrix}
A_{00} & A_{01} & 0 & \cdots & \cdots & \cdots & 0 \\
A_{10} & A_{11} & 0 & \cdots & \cdots & \cdots & 0 \\
0 & A_{21} & A_{22} & 0 & \cdots & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\
0 & A_{i-2,i-1} & A_{i-1,i} & 0 & \cdots & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & A_{N-2,N-1} \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & A_{N-1,N}
\end{pmatrix}
\]  

(27)

Detailed expressions for the matrix entries

The nonzero elements are given by

\[
A_{i,i-1} = -F_0 \quad \text{for } i = 1, \ldots, N_x - 1
\]

(18)

\[
A_{i,i} = 1 + 2F_0 \quad \text{for } i = 1, \ldots, N_x
\]

(19)

\[
A_{i,i+1} = -F_0 \quad \text{for } i = 1, \ldots, N_x
\]

(20)

The linear system in matrix notation:

\[
AU = b, \quad U = (u_0, \ldots, u_{N_x})
\]

(21)

The equations for the boundary points correspond to

\[
A_{00} = 1, \quad A_{01} = 0, \quad A_{N_x,N_x-1} = 0, \quad A_{N_x,N_x} = 1
\]

(22)

\[
u_0 = 0, \quad u_{N_x} = u_{N_x-1}
\]

(23)

\[
b_0 = 0, \quad b_{N_x} = 0
\]

(24)

The linear system for a general \( N_x \)

\[
-F_0 u_{i-1} + (1 + 2F_0) u_i - F_0 u_{i+1} = u_{i-2}
\]

(16)

for \( i = 1, \ldots, N_x - 1 \).

What are the unknowns in the linear system?

- enter \( u_i \) for \( i = 1, \ldots, N_x - 1 \) (all internal spatial mesh points)
- on \( u_0, u_{N_x} \) (all spatial points)

The linear system at matrix row/s is:

\[
AU = b, \quad U = (u_0, \ldots, u_{N_x})
\]
A sparse matrix representation will dramatically reduce the computational complexity. With a dense matrix, the algorithm leads to \(O(N^3)\) operations. Utilizing the sparsity, the algorithm has complexity \(O(N)\)!

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

Computing the sparse matrix:

```python
# Precompute sparse matrix
diagonal[:] = 1 + 2*F
lower[:] = -F
upper[:] = -F
# Insert boundary conditions
diagonal[0] = 1
upper[0] = 0
diagonal[Nx] = 1
lower[-1] = 0
import scipy.sparse
A = scipy.sparse.diags(diagonals=[main, lower, upper],
offsets=[0, -1, 1], shape=(Nx+1, Nx+1), format='csr')
```

Backward Euler applied to a plug profile

\(N_x = 50, \; F = 0.5\).

Link to movie file

Backward Euler applied to a Gaussian profile

\(N_x = 50, \; F = 5\).

Link to movie file

Crank-Nicolson scheme

The PDE is sampled at points \((x_i, t_{n+1/2})\) (at the spatial mesh points, but in between two temporal mesh points).

\[
\frac{\partial}{\partial t} u(x_i, t_{n+1/2}) = \alpha \frac{\partial^2}{\partial x^2} u(x_i, t_{n+1/2})
\]

for \(i = 1, \ldots, N_x - 1\) and \(n = 0, \ldots, N_t - 1\).

Central differences in space and time:

\[
[D_t w = \alpha D_x D_x w]_{n+1/2}^{i+1/2}
\]

Averaging in time is necessary in the Crank-Nicolson scheme

Right-hand side term:

\[
\frac{1}{\Delta t} \left( u_{n+1/2}^{i+1/2} - 2 u_{n+1/2}^i + u_{n+1/2}^{i-1} \right)
\]

Problem: \(u_{n+1/2}^{i+1/2}\) is not one of the unknowns we compute.

Solution: replace \(u_{n+1/2}^{i+1/2}\) by an arithmetic average:

\[
\frac{u_{n+1/2}^{i+1/2}}{2} = \frac{1}{2} (u_{n}^i + u_{n+1}^{i+1})
\]

In compact notation (arithmetic average in time \(\bar{w}\)):

\[
[D_t w = \alpha D_x D_x \bar{w}]_{n+1/2}^{i+1/2}
\]
Crank-Nicolson scheme written out

\[
u_n^{i+1} - \frac{1}{\Delta t} F(u_n^{i+1} - 2u_n^{i} + u_n^{i-1}) = u_n^0 + \frac{1}{\Delta t} F(u_n^{i+1} - 2u_n^0 + u_n^{i})
\] (25)

Observe:

- The unknowns are \(u_n^0, u_n^{i+1}, u_n^{i-1}\)
- These unknowns are coupled so each other (a linear system)
- Must solve \(A\) at each time level

Now,

\[
A_{i,4} = -\frac{1}{\Delta x^2} F
\]
\[
A_{i,2} = \frac{1}{\Delta x^2} + F_0
\]
\[
A_{i,0} = -\frac{1}{\Delta x^2} F_0
\] (26)

for internal points. For boundary points,

\[
A_{i,4} = -F_0, \quad A_{i,2} = 1 + 2F_0, \quad A_{i,0} = -F_0
\]

for \(i = 1\) to \(N_x\).

The Laplace and Poisson equation

Laplace equation:

\[
\nabla^2 u = 0, \quad \text{1D: } u(x) = 0
\]

Poisson equation:

\[
-\nabla^2 u = f, \quad \text{1D: } u(x) = f(x)
\]

These are limiting behavior of time-dependent diffusion equations if

\[
\lim_{\Delta t \to 0} \frac{\partial u}{\partial t} = 0
\]

Then \(u_n = u_n^0 + 0\) is the limit \(t \to \infty\) reduces to

\[
u_n = \frac{f}{F_0}
\]

Crank-Nicolson applied to a Gaussian profile

\(N_x = 50\), \(F = 5\)

We can solve 1D Poisson/Laplace equation by going to infinity in time-dependent diffusion equations.

Looking at the numerical schemes, \(F \to \infty\) leads to the Laplace or Poisson equations (without \(F\) or \(f\), respectively).

Good news: choose \(F\) large in the 8E or CN schemes and one time step is enough to produce the stationary solution for \(t \to \infty\).
Damping of a discontinuity problem

Two pieces of a material, at different temperatures, are brought into contact at \( t = 0 \). Assume the end points of the pieces are kept at the initial temperature. How does the heat flow from the hot to the cold piece?

Or: A huge ion concentration on one side of a synapse in the brain (concentration discontinuity) is released and ions move by diffusion.

Properties of the solution

The PDE

\[ u_t = \alpha u_{xx} \]

admits solutions

\[ u(x,t) = Qe^{-\alpha k^2 t} \sin (kx) \]

Observations from this solution:

- The initial shape \( u(x) = Q \) \( \sin kx \) undergoes a damping \( e^{-\alpha k^2 t} \)
- The damping is very strong for short waves (large \( k \))
- The damping is weak for long waves (small \( k \))
- Consequence: \( u \) is smoothed with time

High frequency components of the solution are very quickly damped

Analysis of schemes for the diffusion equation

Solutions of diffusion problems are expected to be smooth. Can we understand when they are not?
Damping of a discontinuity: model

Solution
Assume a 1D model is sufficient (e.g., insulated rod):

\[ u(x, t) = \begin{cases} \text{U}_L, & x < L/2 \\ \text{U}_R, & x \geq L/2 \end{cases} \]

\[ \frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \quad u(0, t) = \text{U}_L, \quad u(L, t) = \text{U}_R \]

Damping of a discontinuity: Backward Euler scheme

Discrete model:

\[ [D_t u = \alpha D_x D_x]^n \]

results in a tridiagonal linear system

\[-F u^n_{i-1} + (1 + 2F) u^n_i - F u^n_{i+1} = u^{n-1}_i \]

where

\[ F = \alpha \frac{\Delta t}{\Delta x^2} \]

is the mesh Fourier number

Damping of a discontinuity: Forward Euler scheme

Discrete model:

\[ [D_t u = \alpha D_x D_x]^n \]

results in the explicit updating formula

\[ u^{n+1}_i = u^n_i + F \left( u^n_{i+1} - 2u^n_i + u^n_{i-1} \right) \]

Damping of a discontinuity: Crank-Nicolson scheme

Discrete model:

\[ [D_t u = \alpha D_x D_x]^n \]

results in a tridiagonal linear system
Damping of a discontinuity; Crank-Nicolson simulation

\[ F = 5 \]

Represents \( f(x) \) as a Fourier series

\[ f(x) = \sum_{k \in K} b_k e^{ikx} \]

The corresponding sum for \( u \)

\[ u(x,t) \approx \sum_{k \in K} b_k e^{-\alpha k^2 t} e^{ikx} \]

Such solutions are also accepted by the numerical schemes, but with a amplification factor \( A \) different from \( \exp(-\alpha k^2 t) \):

\[ u^n_q = A^n e^{ikq \Delta x} = A^n e^{ikx} \]

Analysis of the finite difference schemes

Stability:

- \( |A| < 1 \): decaying numerical solutions (as we want)
- \( A > 0 \): oscillating numerical solutions (as we do not want)

Accuracy:

- Compare numerical and exact amplification factor: \( A \) vs \( A^e = \exp(-\alpha k^2 \Delta t) \)

\[ \Delta t \leq \frac{\Delta x^2}{2\alpha} \]

Key spatial discretization quantity: the dimensionless \( p = \frac{1}{2} k \Delta x \)

Analysis of the Forward Euler scheme

\[ \left[ D_t^1 u = \alpha D_x D_x u \right]^n \]

Inserting

\[ u^n_q = A^n e^{ikq \Delta x} \]

leads to

\[ A = 1 - 4F \sin^2 \left( \frac{k \Delta x}{2} \right) \]

The complete numerical solution is

\[ u^n_q = (1 - 4F \sin^2 p)^-n e^{ikq \Delta x} \]

Results for stability

We always have \( A \leq 1 \). The condition \( A \geq -1 \) implies

\[ 4F \sin^2 p \leq 2 \]

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

\[ F \leq \frac{1}{2} \]

on

\[ \Delta t \leq \frac{\Delta x^2}{2\alpha} \]

Implications of the stability result

Less favorable criterion than for \( u_{tt} = c^2 u_{xx} \) taking \( \Delta x \) implies time step \( \frac{1}{2} \Delta t \) (to use \( \frac{1}{2} \Delta t \) as in a wave equation). Need very small time steps for fine spatial meshes.

Analysis of the Backward Euler scheme

\[ \left[ D_t^- u = \alpha D_x D_x u \right]^n \]

Inserting

\[ u^n_q = A^n e^{ikq \Delta x} \]

leads so

\[ A = 1 - 4F \sin^2 \left( \frac{k \Delta x}{2} \right) \]

The complete numerical solution is

\[ u^n_q = (1 - 4F \sin^2 p)^-n e^{ikq \Delta x} \]

Stability: We see that \( |A| < 1 \) for all \( \Delta t > 0 \) and that \( A > 0 \) (no oscillations)
Analysis of the Crank-Nicolson scheme

The scheme

\[ \Delta u = \alpha \Delta x \Delta t \frac{u^{n+1} - u^n}{\Delta t} \]

leads to

\[ A = 1 - 2F \sin^2 p \]

\[ u^{n+1} = (1 - 2F \sin^2 p)^n u^n \exp(-ik \Delta x) \]

Stability: The criteria \( A > 1 \) and \( A < 1 \) are fulfilled for any \( \Delta t > 0 \)

Summary of amplification factors

\[ A_w = \exp(-\alpha k^2 \Delta t) = \exp(-4Fp^2) \]

\[ A = 1 - 4F \sin^2 \left( \frac{k \Delta x}{2} \right) \quad \text{Forward Euler} \]

\[ A = (1 + 4F \sin^2 p)^{-1} \quad \text{Backward Euler} \]

\[ A = \frac{1}{2} \left( 1 - 2F \sin^2 p \right) \quad \text{Crank-Nicolson} \]

Note: \( A_w = \exp(-\alpha k^2 \Delta t) = \exp(-Fk^2 \Delta x^2) = \exp(-Fp^2) \).

Summary of accuracy of amplification factors; large time steps

Summary of accuracy of amplification factors; small time steps

Summary of accuracy of amplification factors; time steps around the Forward Euler stability limit

Observations

- The key spatial dissipation parameter is the dimensionless \( \rho = \frac{\phi \Delta x}{k} \).
- The key temporal dissipation parameter is the dimensionless \( F = \frac{\alpha \Delta t}{\Delta x^2} \).
- Imporant: \( \Delta t \) and \( \Delta x \) in combination with \( \alpha \) and \( k \) determine accuracy.
- Crank-Nicolson gives oscillations and too much damping of short waves for increasing \( F \).
- These waves will manifest themselves as high frequency oscillatory noise in the solution.
- Steep solutions will have short waves with significant (visible) amplitudes.
- All schemes fail to dampen short waves entirely.

The problem of correct damping for \( w = \omega x \) is partially manifested in the similar time discretization schemes for \( u'(t) = -\omega u(t) \).