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

Step 1: Discretizing the domain

Mesh in time:
\[
0 = t_0 < t_1 < t_2 < \cdots < t_{Nt-1} < t_{Nt} = T
\]
Mesh in space:
\[
0 = x_0 < x_1 < x_2 < \cdots < x_{N_x-1} < x_{N_x} = L
\]
Uniform mesh with constant mesh spacings \( \Delta t \) and \( \Delta x \):
\[
x_i = i\Delta x, \ i = 0, \ldots, N_x, \ t_n = n\Delta t, \ n = 0, \ldots, N_t
\]

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
\[
\frac{\partial}{\partial t} u(x_i, t_n) = \alpha \frac{\partial^2 u}{\partial x^2}(x_i, t_n)
\]
Applicable 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 condition \( u = I(x) \) and \( u_t = 0 \)
At the boundaries \( i = 0, N_x \) we have the boundary condition \( u = 0 \)
Step 3: Replacing derivatives by finite differences

Use a forward difference in time and a centered difference in space (Forward Euler scheme):

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

Write out,

$$u[n+1] - u[n] = \alpha \frac{u[n+2] - 2u[n] + u[n-1]}{\Delta x^2}$$

Initial condition: $u[n](i) = I(i), i = 0, 1, \ldots, N_x$.

The mesh Fourier number

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

Observe

There is only one parameter $F$, in the discrete model. $F$ lumps 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 computational algorithm for the Forward Euler scheme

- Compute $u[n](i) = I(i), i = 0, 1, \ldots, N_x$.
- For $n = 0, 1, \ldots, N_t$:
  - Compute $u[n+1]$ for all internal spatial points $i = 1, \ldots, N_x - 1$.
  - Set the boundary values $u[n+1] = 0$ for $i = 0$ and $i = N_x$.

Notice

We use one mesh point 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 $u[n]$ must be updated in chronological order $u[n-1]$ before $u[n+1]$.

Step 4: Formulating a recursive algorithm

- Nature of the algorithm: compute $u$ in space as $t = \Delta t, 2\Delta t, 3\Delta t, \ldots$
- Two simultaneous equations are set up for the general discrete equation:
  - $u[n]$ and $u[n+1]$ are already computed for $i = 0, \ldots, N_x$, and $u[n+1]$ is the unknown quantity.

Solve the discrete PDE for the unknown $u[n+1]$:

$$u[n+1] = u[n] + F(u[n+2] - 2u[n] + u[n-1])$$

where

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

The finite difference stencil

The Python implementation of the computational algorithm

```python
x = linspace(0, L, Nx+1)  # mesh points in space
dx = x[1] - x[0]
# Insert boundary conditions
u[0] = 0; u[Nx] = 0
# Update u_1 before next step
u_1[:] = u
# or more efficient switch of references
#u_1, u = u, u_1
```

Stencil at interior point

For $n$ in range(0, Nt):

```python
    # Set initial condition $u(n)(i) = I(i)$
    for i in range(0, Nx+1):
        u[i] = I[i]
    # Set initial conditions
    u[0] = u[0]  # u[i+1] = 2u[i] - u[i-1]
    # Insert boundary conditions
    u[0] = 0; u[Nx] = 0
    # Update u_1 before next step
    u_1[:] = u
```

For more efficient switch of references:

```python
    u_1, u = u, u_1
```
Moving finite difference stencil

- Page 1:
  - How to make movie file in modern formats:
    - Terminal:
      - name=tmp_frame%04d.png
      - fps=8 # frames per second in movie
      - avconv -r $fps -i $name -vcodec flv movie.flv
      - avconv -r $fps -i $name -vcodec libx64 movie.mp4
      - avconv -r $fps -i $name -vcodec libvpx movie.webm
      - avconv -r $fps -i $name -vcodec libtheora movie.ogg

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

Forward Euler applied to an initial plug profile

- Forward Euler scheme
  - Backward difference in time, centered difference in space:
    \[
    (D_t - \alpha D_x^2)u[n] = \left( \frac{u[n] - u[n-1]}{\Delta t} \right) = \alpha \left( \frac{u[n+1] - 2u[n] + u[n-1]}{\Delta x^2} \right)
    \]
  - Notice: We cannot solve for \( u[n] \) because that unknown value is coupled to two other unknown values: \( u[n-1] \) and \( u[n+1] \). 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 \( i = 1, \ldots, N_x - 1 = 1, 2 \) becomes
      \[
      \begin{align*}
      u_1[n] - u_1[n-1] &= \alpha (u_2[n] - 2u_1[n] + u_0[n]) \\
      u_2[n] - u_2[n-1] &= \alpha (u_3[n] - 2u_2[n] + u_1[n])
      \end{align*}
      \]
    - (The boundary values \( u_0[n] \) and \( u_3[n] \) are known as zeros.)
    - Collecting the unknown new values on the left-hand side and writing as a \( 2 \times 2 \) matrix system:
      \[
      \begin{pmatrix}
      1 + 2F & -F \\
      -F & 1 + 2F
      \end{pmatrix}
      \begin{pmatrix}
      u_1[n]
      u_2[n]
      \end{pmatrix}
      = \begin{pmatrix}
      u_1[n-1]
      u_2[n-1]
      \end{pmatrix}
      \]
### Two classes of discretization methods: explicit and implicit

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

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

### A is very sparse: a tridiagonal matrix

\[
A = \begin{pmatrix}
A_{00} & A_{01} & 0 & \cdots & \cdots & \cdots & 0 \\
A_{10} & A_{11} & A_{12} & 0 & \cdots & \cdots & \cdots \\
0 & A_{21} & A_{22} & A_{23} & \cdots & \cdots & \cdots \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
0 & \cdots & \cdots & 0 & A_{j-2,j-1} & A_{j-1,j} & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & 0 & A_{N-1,N-2} & A_{N-1,N} & 0 \\
\end{pmatrix}
\]

### Detailed expressions for the matrix entries

The nonzero elements are given by

\[
A_{i-1,i} = u_i - u_{i-1} \quad (22)
\]

### The right-hand side

\[
b = \begin{pmatrix}
b_0 \\
b_1 \\
b_2 \\
b_3 \\
\vdots \\
b_N \\
\end{pmatrix}
\]

\[
b_0 = 0 \\
b_i = u_{i-1}^0, \quad i = 1, \ldots, N \quad (23)
\]

### Naive Python implementation with a dense \((N + 1) \times (N + 1)\) matrix

```python
x = linspace(0, L, N+1) # mesh points in space
n = x[-1] - x[0]
t = linspace(0, T, M+1) # mesh points in time
u = zeros(x.shape)
u_1 = zeros(x.shape)

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

for i in range(0, N+1):
    A[1,i-1] = -F0
    A[1,i] = 1 + 2*F0
    A[1,i+1] = -F0

b[0] = b[0] = b
A_0 = -F0
A_1 = 1 + 2*F0
A_N = -F0

u_1, u = u, u_1

# Update u_1 before next step
u[0] = scipy.linalg.solve(A, b)

for i in range(1, N):
    # Set initial condition u(x,0) = I(x)
    b = f(x[i])
    A[1,i-1] = -F0
    A[1,i] = 1 + 2*F0
    A[1,i+1] = -F0

    A_0 = -F0
    A_1 = 1 + 2*F0
    A_N = -F0

    u_1, u = u, u_1

    # Update u_1 before next step
    u = scipy.linalg.solve(A, b)
```

### The linear system for a general \(N\)

\[
-F_0 u_{i-1} + (1 + 2 F_0) u_i - F_0 u_{i+1} = u_i^0 - u_i^0 \quad (16)
\]

for \(i = 1, \ldots, N \quad (16)\)

What are the unknowns in the linear system?
- \(u_i^0\) for \(i = 1, \ldots, N - 1\) (all internal spatial mesh points)
- \(u_0^0, u_N^0\) (all spatial points)

The linear system is made non-singular:

\[
A U = b, \quad U = \{u_0^0, \ldots, u_N^0\}
\]
A sparse matrix representation will dramatically reduce the computational complexity.

With a dense matrix, the algorithm leads to $O(N^3)$ operations. Utilizing sparsity, the algorithm has complexity $O(N \times N)$. Scipy enables storage and calculations with the three nonzero diagonals only.

# Representation of sparse matrix and right-hand side

diagonal = zeros(Nx+1)  
lower = zeros(Nx)        
upper = zeros(Nx)        
b = zeros(Nx+1)          

# 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')

# Set initial condition

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

for n in range(0, Nt):
    b = u_1  
b[0] = b[-1] = 0.0  

u[:]= scipy.sparse.linalg.spsolve(A, b)

# Switch variables before next step

u_1, u = u, u_1

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:

$$[D_t u = \alpha D_x D_x u_{n+1/2}]$$

Right-hand side term:

$$\frac{1}{\Delta x^2} \left( u_{n+1/2}^{n+1} - 2 u_{n+1/2}^n + u_{n+1/2}^{n-1} \right)$$

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

Solution: replace $u_{n+1/2}^{n+1}$ by an arithmetic average:

$$u_{n+1/2}^{n+1} \approx \frac{1}{2} (u_{n+1/2}^n + u_{n+1/2}^{n+1})$$

In compact notation (arithmetic average in time $u_{n+1/2}$):

$$[D_t u = \alpha D_x D_x u_{n+1/2}]$$
In the time-dependent diffusion equations, we can solve 1D Poisson/Laplace equations by going to infinity in time-dependent diffusion equations if \( \lim_{t \to \infty} \frac{\partial u}{\partial t} = 0 \). These are limiting behavior of time-dependent diffusion equations:

- Laplace equation:
  \[ \nabla^2 u = 0 \]

- Poisson equation:
  \[ -\nabla^2 u = f \]

Looking at the numerical schemes, \( F \to \infty \) leads to the Laplace or Poisson equations (without \( f \) for \( \nabla^2 u = 0 \)).

Good news: choose \( F \) large in the BE or CN schemes and one time step is enough to produce the stationary solution for \( t \to \infty \).

The numerical schemes, \( F \to \infty \) leads to the Laplace or Poisson equations (without \( f \) for \( \nabla^2 u = 0 \)).

Good news: choose \( F \) large in the BE or CN schemes and one time step is enough to produce the stationary solution for \( t \to \infty \).
**Extensions**

These extensions are performed exactly as for a wave equation as they only affect the spatial derivatives (which are the same as in the wave equation).

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

Future versions of this document will for completeness and independence of the wave equation document feature info on the FEM, contact at \( t = 0 \).

**Properties of the solution**

The PDE

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

admits solutions

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

Observations from this solution:

- The initial shape \( f(x) = \sin(kx) \) 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?

**Example**

Test problem:

\[
\begin{align*}
\frac{\partial u}{\partial t} &= \frac{\partial^2 u}{\partial x^2}, & x \in (0,1), \ t \in (0,T] \\
u(0,t) &= u(1,t) = 0, & t \in (0,T] \\
u(x,0) &= \sin(\pi x) + 0.1 \sin(100 \pi x) \\
\end{align*}
\]

Exact solution:

\[
u(x,t) = e^{-\pi^2 t} \sin(\pi x) + 0.1 e^{-\pi^2 10^4 t} \sin(100 \pi x)
\]

**Damping of a discontinuity, problem**

Problem

Two pieces of a material, at different temperatures, are brought in 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.
Damping of a discontinuity; model
Solution
Assume a 1D model is sufficient (e.g., an insulated rod):
\[
u(x,0) = \begin{cases} U_L, & x < L/2 \\ U_R, & x \geq L/2 \end{cases} \]
\[
\partial u \partial t = \alpha \partial^2 u \partial x^2, \quad u(0,t) = U_L, \quad u(L,t) = U_R
\]

Damping of a discontinuity; Backward Euler scheme
Discrete model:
\[
[D - \Delta t \partial u \partial t = \alpha \partial^2 u \partial x^2 ]_n^i
\]
results in a (tridiagonal) linear system
\[
-Fu_{n-1}^i + (1 + 2F)u_n^i - Fu_{n+1}^i = u_{n-1}^i
\]
where
\[
F = \frac{\alpha \Delta t}{\Delta x^2}
\]
is the mesh Fourier number

Damping of a discontinuity; Forward Euler scheme
Discrete model:
\[
[D + \Delta t \partial u \partial t = \alpha \partial^2 u \partial x^2 ]_n^i
\]
results in the explicit updating formula
\[
u_{n+1}^i = u_n^i + F \left( u_{n+1}^i - 2u_n^i + u_{n-1}^i \right)
\]

Damping of a discontinuity; Crank-Nicolson scheme
Discrete model:
\[
[D\partial u \partial t = \alpha \partial^2 u \partial x^2 ]_n^i
\]
results in a tridiagonal linear system

Damping of a discontinuity; Backward Euler simulation
\[ F = \frac{1}{2} \]
Movie

Damping of a discontinuity; Forward Euler simulation
\[ F = \frac{1}{2} \]
Movie

Damping of a discontinuity; Crank-Nicolson scheme
Movie
Damping of a discontinuity; Crank-Nicolson simulation

\[ F = 5 \]

Movie

Fourier representation

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

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

The corresponding sum for \( u \) is

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

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

\[ u_n q = A_n e^{ikq \Delta x} \]

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 \( A_{\text{exact}} = \exp(-\alpha k^2 \Delta t) \)

Results for stability

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

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

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

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

or

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

Analysis of the Forward Euler scheme

\[ [D^+ u = \alpha D_x D_x u]_n \]

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

\[ u_n q = (1 - 4F \sin^2 \rho) e^{ikq \Delta x}, \quad \rho = k \Delta x/2 \]

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

Analysis of the Backward Euler scheme

\[ [D^- u = \alpha D_x D_x u]_n \]

Inserting

\[ u_n q = A_n e^{ikq \Delta x} \]

leads to

\[ A = (1 + 4F \sin^2 \rho)^{-1} \]

The complete numerical solution is

\[ u_n q = (1 + 4F \sin^2 \rho)^{-1} e^{ikq \Delta x} \]

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

\[ D_{t} u = \alpha D_{x} D_{x} u \]

leads to

\[ A = 1 - 2 F \sin^{2} \beta \]

\[ u_{n+1}^{a} = \left( 1 - 2 F \sin^{2} \beta \right) u_{n+1}^{a} \exp \left( -i k \Delta x \right) \]

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

Summary of amplification factors; small time steps

\[ A_{E} = \exp \left( -\alpha k^{2} \Delta t \right) = \exp \left( -4 F \Delta t \right) \]

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

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

\[ A = \frac{1 - 2 F \sin^{2} \beta}{1 + 2 F \sin^{2} \beta} \quad \text{Crank-Nicolson} \]

Notes: \( A_{E} = \exp \left( -\alpha k^{2} \Delta t \right) = \exp \left( -F \Delta t \right) = \exp \left( -4 F \Delta t \right) \)

Summary of accuracy of amplification factors; large time steps

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

Summary of accuracy of amplification factors; small time steps

Observations

- The key spatial discretisation parameter is the dimensionless \( \beta = \frac{\pi}{k} \Delta x \)
- The key temporal discretisation parameter is the dimensionless \( F = \alpha \Delta t / \Delta x^{2} \)
- Impartial: \( \Delta t \) and \( \Delta x \) in combination with \( \alpha \) and \( k \) determine accuracy
- Crank-Nicolson gives oscillations and too much damping of short waves for increasing \( F \)
- Short waves will manifest themselves as high frequency oscillatory noise in the solution
- Steep solvers will have short waves with significant (visible) amplitudes
- All solvers fail to dampen short waves enough

The problem of correct damping for \( u_{E} = u_{E} \Delta \) is partially manifested in the similar time discretisation schemes for \( u'(t) = -\alpha u(t) \).