FD methods for diffusion (I)

Yesterday we solved the transient (time-dependent) heat equation in 1D. In the absence of heat sources, the governing equation is

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial z^2} \tag{1}$$

if material parameters are homogeneous.

In *explicit* finite difference schemes, the temperature at time $n+1$ depends only on the already known temperature at time n. The explicit finite difference (FD) discretization of eq. (1) is

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2} \quad , \tag{2}$$

using central differences for the spatial derivatives (subscripts i are indicating the location in 1D, superscripts indicating the time).

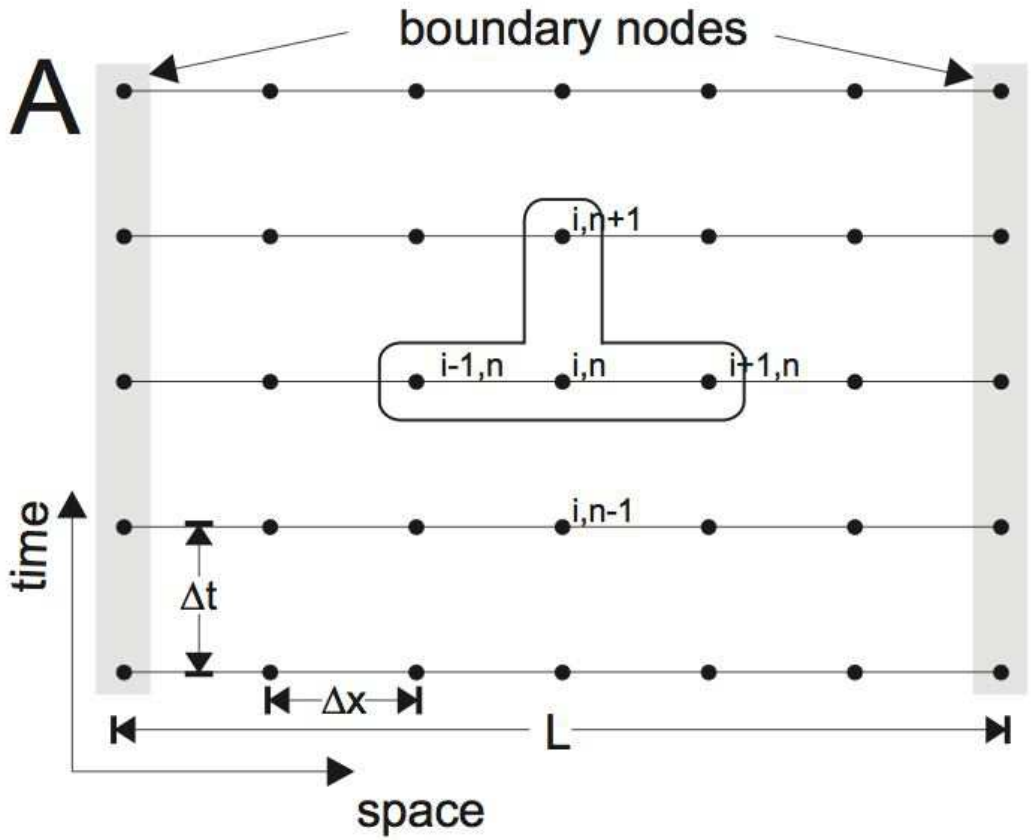
Equation (2) can be rearranged in the following manner (with all the quantities at time $n+1$ on the left and quantities at time n on the right hand side):

$$T_i^{n+1} = T_i^n + \kappa\,\Delta t\,\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2} \tag{3}$$

Since we know T_{i+1}^n, T_i^n and T_{i-1}^n, we can compute T_i^{n+1}. – This is schematically shown on the following figure A, and the corresponding algorithm is called a *forward time, centered space* (FTCS) for the way it is computed.

The major advantage of explicit finite difference methods is that they are relatively simple, only one solution for T needs to be stored, and the method is computationally fast for each time step. However, the main drawback is that stable solutions are obtained only when

$$0 < \frac{2\kappa\,\Delta t}{h^2} \leq 1 \qquad \text{or} \qquad \Delta t \leq \frac{h^2}{2\kappa} \qquad \text{for given } h\,. \tag{4}$$



Scheme of the explicit finite difference (FTCS) discretization.

If condition (4) is not satisfied, the solution becomes unstable, starts to wildly oscillate, or "blows up". The physically meaning of this stability condition is that the maximum time step needs to be smaller

than the time it takes for an small anomaly to diffuse across the grid (nodal) spacing h. The explicit solution, eq. (3), is an example of a *conditionally stable* method that only leads to well behaved solutions if a criterion like eq. (4) is satisfied.

Note that eq. (4) can only hold for $\kappa \Delta t > 0$; having a negative diffusivity, or using a time-reversed ($\Delta t < 0$) scheme, will inevitably lead to a blow up since small features will get amplified rather than smoothed out.

This is an issue if one wishes to reconstruct coupled diffusive-advective processes (such as mantle convection), going from the present-day temperature field back in time. We will revisit an FTCS scheme similar to eq. (2) for advection that involves single derivatives in space later. Unlike to eq. (2), the FTCS scheme for advection is *always* unstable. Even if the FTCS scheme for diffusion can be made stable, the stability condition leads to numerical convenience issues, since the stability condition effectively limits our available spatial resolution.

An alternative approach is an *implicit* finite difference scheme, where the spatial derivatives are evaluated at the *new* time step:

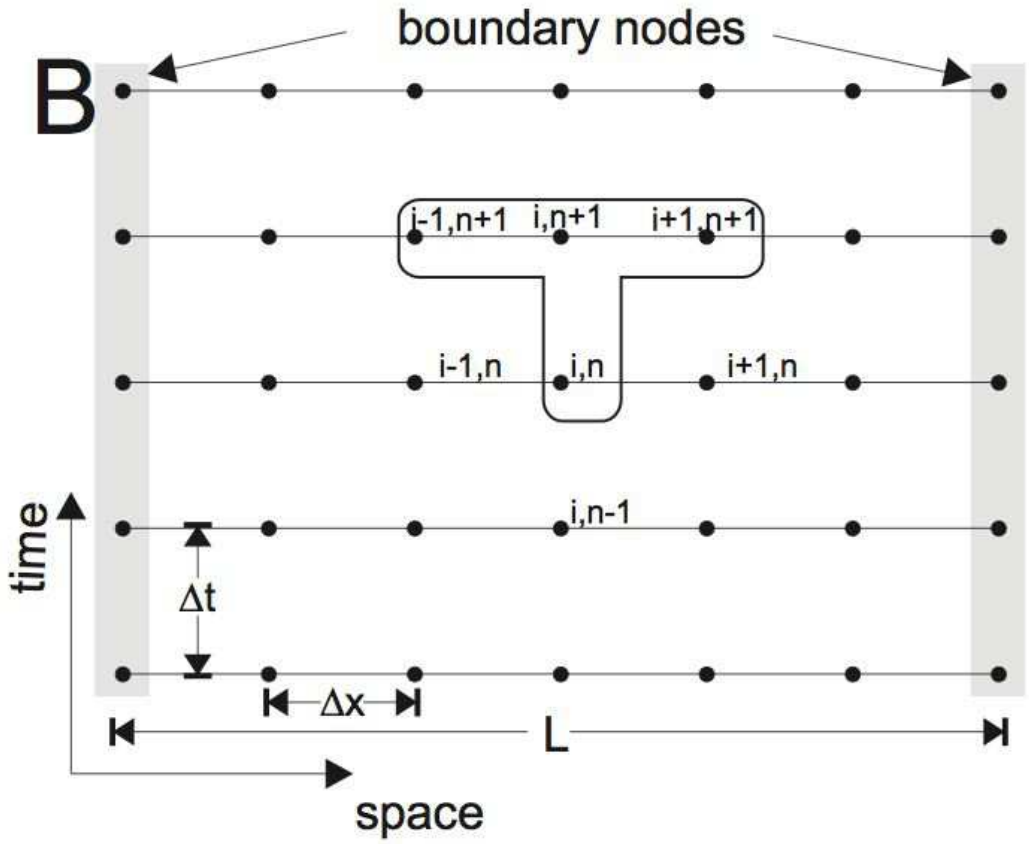


Figure B shows for example a *fully implicit* discretization scheme:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \kappa \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{h^2} \quad , \tag{5}$$

where, with other words, the time derivative is completely taken backward (BTCS).

For future notation, it is useful to rewrite eqs. (3) and (5) in *stencil* notation

$$T_i^{n+1} = T_i^n + \frac{\kappa \Delta t}{h^2} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} T_i^n \qquad \text{(FTCS)}$$

$$T_i^{n+1} = T_i^n + \frac{\kappa \Delta t}{h^2} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} T_i^{n+1} \qquad \text{(BTCS)}$$

where the stencil [...] is an operator that acts on a space point T_i and its two nearest neighbours. Unless it is specifically noted, the central point in this stencil is the coefficient of point T_i.

Inspection of the first expression shows that, as long as $\frac{\kappa \Delta t}{h^2} < \frac{1}{2}$, the FTCS scheme acts as a simple 3 point smoother where every point is replaced by some fraction of its previous value and some mixture of the values of its nearest neighbours:

$$T_i^{n+1} = \begin{bmatrix} \beta & (1-2\beta) & \beta \end{bmatrix} T_i^n \quad , \qquad \beta = \kappa \Delta t / h^2$$

For example, if $\beta = 1/4$, the stencil looks like [1/4 1/2 1/4] and therefore, after one pass of the algorithm, a unit spike on the grid gets reduced to half its height and smeared out over its two nearest neighbours. If $\beta > 1/2$, however, the scheme is clearly unstable, as a single step will make the temperature go negative.

The great advantage of the BTCS scheme is, instead, that it is always (unconditionally) stable. – However, this benefit comes with a clear drawback: The implementation of this scheme requires solving a system of equations at each time step !

This can be seen from the stencil notation for the BTCS scheme

$$\begin{bmatrix} -\beta & (1+2\beta) & -\beta \end{bmatrix} T_i^{n+1} = T_i^n$$

This turns out to be actually a system of linear equations of the form

$$\mathbf{A}\, x = b$$

where $\mathbf{A}$ is a *tridiagonal matrix* that is primarily zero except for the diagonal (which has the value $(1+2\beta)$) and one super and one sub diagonal of value $-\beta$. The vector $x = T^{n+1}$ corresponds to the array of temperature values at timestep $n+1$ and the vector b is the known array of temperatures at time n. Thus if we can invert $\mathbf{A}$ we can solve for T^{n+1} as

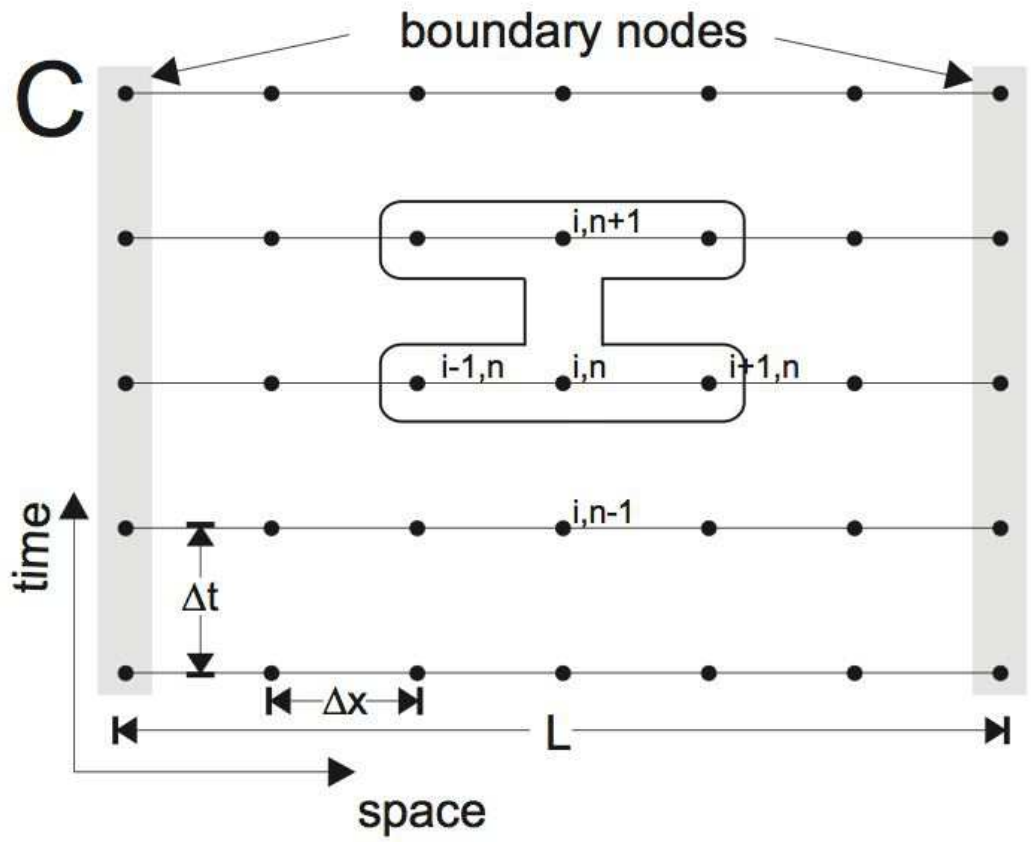
$$T^{n+1} = \mathbf{A}^{-1}\, T^n$$

Since the implementation of the BTCS scheme requires solving a system of equations at each time step, the computational effort per time the BTCS scheme is greater than the computational effort per time step for the FTCS scheme.

Its unconditional stability does not mean, however, that it is always accurate. Taking large time steps may result in an inaccurate solution for features with small spatial scales. For any application, it is therefore always a good idea to check results by decreasing the time step until the solution does not change anymore (this is called convergence check), and to ensure the method can deal with small and large scale features robustly at the same time.

It turns out that the fully implicit method is second order accurate in space but only first order accurate in time. It is, however, possible to formulate a scheme which is second order accurate both in time and in space (i.e. $O(h^2, \Delta t^2)$). One such scheme is the Crank-Nicholson scheme that is also unconditionally stable.

Crank-Nicholson method



Crank-Nicholson finite difference discretization.

The Crank-Nicholson scheme is a combination of both explicit and implicit methods. This “best of both worlds” is obtained by computing the average of the fully implicit and fully explicit schemes:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \frac{\kappa}{2} \left(\frac{(T_{i+1}^{n+1} - 2\,T_i^{n+1} + T_{i-1}^{n+1}) + (T_{i+1}^{n} - 2\,T_i^{n} + T_{i-1}^{n})}{h^2} \right)$$

This scheme should generally yield the best performance for any diffusion problem; it is second order in time and space accurate, because the averaging of fully explicit and fully implicit methods corresponds to evaluating the time derivative centered on $n + 1/2$. Such centered evaluation lead to second order accuracy also for the time derivative.

In stencil notation, it reads

$$\begin{bmatrix} -\beta & 2(1+\beta) & -\beta \end{bmatrix} T_i^{n+1} = \begin{bmatrix} \beta & 2(1-\beta) & \beta \end{bmatrix} T_i^{n} \quad . \qquad (6)$$

Equation (6) has the form

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

and can be solved by tridiagonal inversion. Note that, although its RHS is more complicated, it is still known. Once the boundary conditions are added to the matrices $\mathbf{A}$ and $\mathbf{B}$, the solution is obtained as

$$T^{n+1} = \mathbf{A}^{-1}\mathbf{B}\,T^{n}$$

The Crank-Nicholson scheme is also *unconditionally stable* (with all the previous caveats in place). For simple diffusional problems, this is probably your best choice.

Boundary conditions

Any PDE with second order derivatives (and above) must have boundary conditions specified at the edges of the domain.

Boundary conditions are extremely important and often control the behaviour of the solution. Unfortunately, they are also often the most difficult part of a problem to get to behave (many times the boundary conditions can be a source of instability even if the general scheme is stable). In discrete systems, boundary conditions are also required to make sure there are enough equations for unknowns, i.e. within the domain, all the points satisfy the stencil equation, however, at the edges T_1 and T_N we have to specify additional information to make up for the lack of the T_0 resp. T_{N+1} points.

For second order differential equations, one has basically four types of boundary conditions to deal with:

1. Dirichlet boundary conditions

The simplest boundary conditions are known as Dirichlet conditions and simply state the value at the boundary (T_1 or T_N) as a known function of time. In a 1D problem, if both boundaries are of Dirichlet type, then a unique solution exists and the equations are easy to solve as the only points that are unknown are the interior points.

2. Neumann boundary conditions

In addition to specifying the temperature at the boundary, one could also specify the heat flux (or temperature gradient) at that boundary. These conditions are known as Neumann conditions and produce a great deal more freedom in the behaviour of the governing equations. If a boundary has Neumann BC's then the temperature on that boundary is variable with time and requires an additional equation.

Numerical Implementation (BTCS)

Within Matlab, we declare matrix $\mathbf{A}$ to be sparse by initialization with the `sparse` function. This will ensure computationally fairly efficient internal treatment mithin Matlab. Once the coefficient matrix **A** and the right-hand-side vector **rhs** have been constructed, MATLAB functions can be used to obtain the solution **x** and you will not have to worry about choosing a proper matrix solver for now.

First, however, we have to construct the matrices and vectors. The coefficient matrix **A** can be constructed with a simple loop

```
» A = sparse(nz,nz);
» for i=2:nz-1
»     A(i,i-1) = - beta;
»     A(i,i) = (1 + 2 * beta);
»     A(i,i+1) = - beta;
» end
```

and the (Dirichlet) boundary conditions are set by

```
» A(1,1) = 1;
» A(nz,nz) = 1;
```

Once the coefficient matrix has been constructed, its structure can be visualized with the command

```
» spy(A)
```

Try it ! – for example py putting a “break-point” into the Matlab code after assembly.

The right-hand-side vector **rhs** can be constructed with

```
» rhs = zero(nz,1);
» rhs(2:nz-1) = Told(2:nz-1);
» rhs(1) = Tsurf; rhs(nz) = Tini;
```

The only thing that remains to be done is to solve the system of equations and find **x**. MATLAB does this with

```
» x = A\rhs;
```

The vector **x** is now filled with new temperature T^{n+1}, and we can go to the next time step.

Note that, for canstant $\Delta t, \kappa$, and h, the matrix **A** doest change with time. Therefore we have to form it only once in the program, which speeds up the code significantly. Only the vectors **rhs** and **x** need to be recomputed. – Having a constant matrix helps a lot for large systems because operations such as x = A\rhs can then be optimized further by storing **A** in a special form.

Exercise

1. Save the script `heat1D.m` from yesterday as `heat1Dimplicit.m`. Program the implicit finite difference scheme explained above. Compare the results with results from the explicit code.
2. Time-dependent, analytical solutions for the heat equation exist. For example, if the initial temperature distribution is

$$T(z, t = 0) = T_{max} \exp\left\{-\left(\frac{z}{\sigma}\right)^2\right\}$$

where T_{max} is the maximum amplitude of the temperature perturbation at $x = 0$ and σ its half-width of perturbance.
The solution is then

$$T(z, t) = \frac{T_{max}}{\sqrt{1 + 4t\kappa/\sigma^2}} \exp\left\{-\frac{z^2}{\sigma^2 + 4t\kappa}\right\}$$

for $T = 0$ at infinity. Program the analytical solution and compare it with the numerical solution with the same initial condition.

3. Program the Crank-Nicholson method (*cf.* Figure C) for the Lord Kelvin problem. This scheme should generally yield the best performance for any diffusion problem, it is second order time and space accurate, because the averaging of fully explicit and fully implicit methods.

   There is an example code `cranknicholson.m` with documentation `cranknicholson.pdf` available at the course website – but for a different problem – try to follow the implementation.

4. Bonus question: What modifications of the code have to be made if the thermal conductivity k in the heat equation is not a constant ? Assume for simplicity that the grid spacing h is constant.