# 1 Finite difference example: 1D implicit heat equation

## 1.1 Boundary conditions – Neumann and Dirichlet

We solve the transient heat equation

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) \tag{1}$$

on the domain $-L/2 \leq x \leq L/2$ subject to the following boundary conditions for fixed temperature

$$\begin{aligned} T(x = -L/2, t) &= T_{left} \\ T(x = L/2, t) &= T_{right} \end{aligned} \tag{2}$$

with the initial condition

$$T(x < -W/2, x > W/2, t = 0) = 300 \tag{3}$$

$$T(-W/2 \leq x \leq W/2, t = 0) = 1200, \tag{4}$$

where we have again assumed a hot dike intrusion for $-W/2 \leq x \leq W/2$.

Boundary conditions (BCs, see also sec. **??**) for PDEs that specify values of the solution function (here $T$) to be constant, such as eq. (2), are called *Dirichlet boundary conditions*. We can also choose to specify the gradient of the solution function, *e.g.* $\partial T/\partial x$ (*Neumann boundary condition*). This gradient boundary condition corresponds to heat flux for the heat equation and we might choose, *e.g.*, zero flux in and out of the domain (isolated BCs):

$$\begin{aligned} \frac{\partial T}{\partial x}(x = -L/2, t) &= 0 \\ \frac{\partial T}{\partial x}(x = L/2, t) &= 0. \end{aligned} \tag{5}$$

## 1.2 Solving an implicit finite difference scheme

As before, the first step is to discretize the spatial domain with $n_x$ finite difference points. The implicit finite difference discretization of the temperature equation within the medium where we wish to obtain the solution is eq. (**??**). Starting with fixed temperature BCs (eq. 2), the boundary condition on the left boundary gives

$$T_1 = T_{left} \tag{6}$$

and the one on the right

$$T_{n_x} = T_{right}. \tag{7}$$

Eqs. (**??**), (6), and (7) can be written in matrix form as

$$A\boldsymbol{x} = \boldsymbol{b}. \tag{8}$$

For a six-node grid, for example, the coefficient matrix $A$ is

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -s & (1+2s) & -s & 0 & 0 & 0 \\ 0 & -s & (1+2s) & -s & 0 & 0 \\ 0 & 0 & -s & (1+2s) & -s & 0 \\ 0 & 0 & 0 & -s & (1+2s) & -s \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \tag{9}$$

the unknown temperature vector $\boldsymbol{x}$ is

$$\boldsymbol{x} = \begin{pmatrix} T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \\ T_5^{n+1} \\ T_6^{n+1} \end{pmatrix}, \tag{10}$$

and the known right-hand-side vector $\boldsymbol{b}$ is

$$\boldsymbol{b} = \begin{pmatrix} T_{left} \\ T_2^n \\ T_3^n \\ T_4^n \\ T_5^n \\ T_{right} \end{pmatrix}. \tag{11}$$

Note that matrix $A$ will have a unity entry on the diagonal and zero else for each node where Dirichlet (fixed temperature) boundary conditions apply; see derivation below and eqs. (**??**) and (**??**) for how to implement Neumann boundary conditions.

Matrix $A$ also has an overall peculiar form because most entries off the diagonal are zero. This "sparseness" can be exploited by specialized linear algebra routines, both in terms of storage and speed. By avoiding computations involving zero entries of the matrix, much larger problems can be handled than would be possible if we were to store the full matrix. In particular, the fully implicit FD scheme leads to a "tridiagonal" system of linear equations that can be solved efficiently by LU decomposition using the Thomas algorithm (*e.g.* *Press et al.*, 1993, sec. 2.4).

## 1.3 Matlab implementation

Within Matlab, we declare matrix $A$ to be sparse by initializing it with the `sparse` function. This will ensure a computationally efficient internal treatment within Matlab. Once the

coefficient matrix $A$ and the right-hand-side vector $\boldsymbol{b}$ have been constructed, MATLAB functions can be used to obtain the solution $\boldsymbol{x}$ 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(nx,nx);
for i=2:nx-1
      A(i,i-1) = -s;
      A(i,i ) = (1+2*s);
      A(i,i+1) = -s;
end
```

and the boundary conditions are set by:

```
A(1 ,1 ) = 1;
A(nx,nx) = 1;
```

(Exercise: Improve on the loop formulation for $A$ assembly by using Matlab vector functionality.)

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

```
>>spy(A)
```

(Try it, for example by putting a "break-point" into the Matlab code below after assembly.) The right-hand-side vector $\boldsymbol{b}$ can be constructed with

```
b = zeros(nx,1);
b(2:nx-1) = Told(2:nx-1);
b(1) = Tleft; b(nx) = Tright;
```

The only thing that remains to be done is to solve the system of equations and find $\boldsymbol{x}$. MATLAB does this with

```
x = A\b;
```

The vector $\boldsymbol{x}$ is now filled with new temperatures $T^{n+1}$, and we can go to the next time step. Note that, for constant $\Delta t$, $\kappa$, and $\Delta x$, the matrix $A$ does not change with time. Therefore we have to form it only once in the program, which speeds up the code significantly. Only the vectors $\boldsymbol{b}$ and $\boldsymbol{x}$ need to be recomputed. (Note: Having a constant matrix helps a lot for large systems because operations such as `x = A\b` can then be optimized further by storing $A$ in a special form.)

## 1.4 Exercises

1. Save the script `heat1Dexplicit.m` from last section as `heat1Dimplicit.m`. Program the implicit finite difference scheme explained above. Compare the results with results from last section's explicit code.

2. Time-dependent, analytical solutions for the heat equation exists. For example, if the initial temperature distribution (initial condition, IC) is

$$T(x, t = 0) = T_{max} \exp\left(-\left(\frac{x}{\sigma}\right)^2\right) \tag{12}$$

where $T_{max}$ is the maximum amplitude of the temperature perturbation at $x = 0$ and $\sigma$ its half-width of the perturbance (use $\sigma < L$, for example $\sigma = W$). The solution is then

$$T(x, t) = \frac{T_{max}}{\sqrt{1 + 4t\kappa/\sigma^2}} \exp\left(\frac{-x^2}{\sigma^2 + 4t\kappa}\right) \tag{13}$$

(for $T = 0$ BCs at infinity). (See *Carslaw and Jaeger*, 1959, for useful analytical solutions to heat conduction problems).

Program the analytical solution and compare the analytical solution with the numerical solution with the same initial condition. Compare results of the implicit and FTCS scheme used last section to the analytical solution near the instability region of FTCS,

$$s = \frac{\kappa \Delta t}{(\Delta x)^2} < \frac{1}{2}. \tag{14}$$

*Note:* Eq. (13) can be derived using a *similarity variable*, $\tilde{x} = x/x_c$ with $x_c \propto \sqrt{\kappa t}$. Looks familiar?

3. A steady-state temperature profile is obtained if the time derivative $\partial T/\partial t$ in the heat equation (eq. **??**) is zero. There are two ways to do this.

   (a) Wait until the temperature does not change anymore.

   (b) Write down a finite difference discretization of $\partial^2 T/\partial x^2 = 0$ and solve it. (See the limit case consideration above.)

   Employ both methods to compute steady-state temperatures for $T_{left} = 100^\circ$ and $T_{right} = 1000^\circ$. Derive the analytical solution and compare your numerical solutions' accuracies. Use the implicit method for part (a), and think about different boundary conditions, and the case with heat production.

4. Apply no flux boundary conditions at $|x| = L/2$ and solve the dike intrusion problem in a fully implicit scheme. Eqs. (**??**) and (**??**) need to replace the first and last columns of your $A$ matrix.

5. Derive and program the Crank-Nicolson method (*cf.* Figure **??**C). This "best of both worlds" method is obtained by computing the average of the fully implicit and fully explicit schemes:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = \frac{\kappa}{2}\left(\frac{\left(T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}\right) + \left(T_{i+1}^n - 2T_i^n + T_{i-1}^n\right)}{(\Delta x)^2}\right). \tag{15}$$

   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 to obtain the time derivative corresponds to evaluating the derivative centered on $n + 1/2$. Such centered evaluation also lead to second order accuracy for the spatial derivatives.

   Compare the accuracy of the Crank-Nicolson scheme with that of the FTCS and fully implicit schemes for the cases explored in the two previous problems, and for ideal values of $\Delta t$ and $\Delta x$, and for large values of $\Delta t$ that are near the instability region of FTCS.

   *Hint*: Proceed by writing out eq. (15) and sorting terms into those that depend on the solution at time step $n + 1$ and those at time step $n$, as for eq. (**??**).

6. Bonus question: Write a code for the thermal equation with variable thermal conductivity $k$: $\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)$. Assume that the grid spacing $\Delta x$ is constant. For extra bonus, allow for variable grid spacing and variable conductivity.

# Bibliography

Carslaw, H. S., and J. C. Jaeger (1959), *Conduction of Heat in Solids*, 2nd ed., Oxford University Press, London, p. 243.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1993), *Numerical Recipes in C: The Art of Scientific Computing*, 2 ed., Cambridge University Press, Cambridge.