5 Finite differences: and what about 2D?

The transient heat equation with sources/sinks in 2D is given by

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

where, ρ is density, c_p heat capacity, k thermal conductivity and Q radiogenic heat production. If the thermal conductivity is spatially constant, we can rewrite the equation to

$$\frac{\partial T}{\partial t} = \kappa\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial z^2}\right) + \frac{Q}{\rho c_p} \tag{2}$$

5.1 Explicit method

The simplest way to discretize equation 2 is to employ an explicit discretization scheme

$$\frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\Delta t} = \kappa\left(\frac{T_{i,j+1}^{n} - 2T_{i,j}^{n} + T_{i,j-1}^{n}}{\Delta x^2} + \frac{T_{i+1,j}^{n} - 2T_{i,j}^{n} + T_{i-1,j}^{n}}{\Delta z^2}\right) + \frac{Q_{i,j}^{n}}{\rho c_p} \tag{3}$$

rearranging gives

$$T_{i,j}^{n+1} = T_{i,j}^{n} + \frac{\kappa\Delta t}{\Delta x^2}\left(T_{i,j+1}^{n} - 2T_{i,j}^{n} + T_{i,j-1}^{n}\right) + \frac{\kappa\Delta t}{\Delta z^2}\left(T_{i+1,j}^{n} - 2T_{i,j}^{n} + T_{i-1,j}^{n}\right) + \frac{Q_{i,j}^{n}\Delta t}{\rho c_p} \tag{4}$$

Boundary conditions can be set the usual way. A constant temperature on the left-hand side of the domain (at $j = 1$), for example, is given by

$$T_{i,j=1} = T_{left} \tag{5}$$

A constant flux on the same boundary is set through fictious boundary points

$$\frac{\partial T}{\partial x} = c_1 \tag{6}$$

$$\frac{T_{i,2} - T_{i,0}}{2\Delta x} = c_1 \tag{7}$$

and $T_{i,0}$ can be eliminated by using equation 4. A (major) disadvantage of these explicit schemes is that it is only stable if

$$\frac{\kappa\Delta t}{\min(\Delta x^2, \Delta z^2)} \leq 0.5 \tag{8}$$

5.2 Fully implicit method

A way around these stability problems is to employ an implicit discretization scheme (remember the 1D exercises?). The fully implicit discretization scheme of equation 2 is

$$\frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\Delta t} = \kappa\left(\frac{T_{i,j+1}^{n+1} - 2T_{i,j}^{n+1} + T_{i,j-1}^{n+1}}{\Delta x^2} + \frac{T_{i+1,j}^{n+1} - 2T_{i,j}^{n+1} + T_{i-1,j}^{n+1}}{\Delta z^2}\right) + \frac{Q_{i,j}^{n}}{\rho c_p} \tag{9}$$

rearranging to put terms with $n+1$ on the left-hand-side and terms with n on the rhs gives

$$-s_z T_{i+1,j}^{n+1} + (1 + 2s_z + 2s_x)T_{i,j}^{n+1} - s_z T_{i-1,j}^{n+1} - s_x T_{i,j+1}^{n+1} - s_x T_{i,j-1}^{n+1} = T_{i,j}^{n} + \frac{Q_{i,j}^{n}\Delta t}{\rho c_p} \tag{10}$$

where $s_x = \kappa\Delta t/\Delta x^2$ and $s_z = \kappa\Delta t/\Delta z^2$.

As in the 1D case, we have to write these equations in a matrix **A** and a vector **rhs** (and use $\mathbf{c} = \mathbf{A}\backslash\mathbf{rhs}$ to solve for T^{n+1}). From a practical point of view, this is a bit more complicated than in the 1D case, since we have to deal with book-keeping issues. As a first step the nodes have to be numbered continuously (see figure 2 for an example). The derivative versus x-direction is fairly similar to the 1D case, e.g. (fig. 2)

$$\frac{\partial^2 T}{\partial x^2}|_{i=3,j=4} = 1/\Delta x^2 \left(T_{19} - 2T_{18} + T_{17}\right) \tag{11}$$

The derivative versus z-direction is given by (fig. 2).

$$\frac{\partial^2 T}{\partial z^2}|_{i=3,j=4} = 1/\Delta z^2 \left(T_{25} - 2T_{18} + T_{11}\right) \tag{12}$$

If n_x are the number of gridpoints in x-direction and n_z the number of points in z-direction, we can write equations 11 and 12 in a more general way as:

$$\frac{\partial^2 T}{\partial x^2}|_{i,j} = 1/\Delta x^2 \left(T_{(i-1)n_x+j+1} - 2T_{(i-1)n_x+j} + T_{(i-1)n_x+j-1}\right) \tag{13}$$

$$\frac{\partial^2 T}{\partial z^2}|_{i,j} = 1/\Delta z^2 \left(T_{in_x+j} - 2T_{(i-1)n_x+j} + T_{(i-2)n_x+j}\right) \tag{14}$$

In matrix format this gives something like

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & .. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .. & 0 & 0 \\ 0 & 1 & .. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .. & 0 & 0 \\ : & : & & & & & & & & & & & & \\ 0 & 0 & & -s_z & .. & -s_x & (1+2s_x+2s_z) & -s_x & .. & -s_z & 0 & & 0 & 0 \\ 0 & 0 & & 0 & -s_z & .. & -s_x & (1+2s_x+2s_z) & -s_x & .. & -s_z & & 0 & 0 \\ : & : & & & & & & & & & & & : & : \\ 0 & 0 & .. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .. & 1 & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .. & 0 & 1 \end{pmatrix} \tag{15}$$

Note that we now have 5 diagonals filled with numbers as opposed to 3 diagonals in the 1D case. The coefficient matrix **c** is given by

$$\mathbf{c} = \begin{pmatrix} T_1^{n+1} = T_{1,1} \\ T_2^{n+1} = T_{1,2} \\ : \\ T^{n+1}_{(i-1)n_x+j} = T_{i,j} \\ T^{n+1}_{(i-1)n_x+j+1} = T_{i,j+1} \\ : \\ T^{n+1}_{n_x n_z - 1} = T_{n_z,n_x-1} \\ T^{n+1}_{n_x n_z} = T_{n_z,n_x} \end{pmatrix} \tag{16}$$

and the rhs-vector is given by (ignoring radioactive heat!)

$$\mathbf{rhs} = \begin{pmatrix} T_{bottom} \\ T_{bottom} \\ : \\ T^n_{(i-1)n_x+j} \\ T^n_{(i-1)n_x+j+1} \\ : \\ T_{top} \\ T_{top} \end{pmatrix} \tag{17}$$

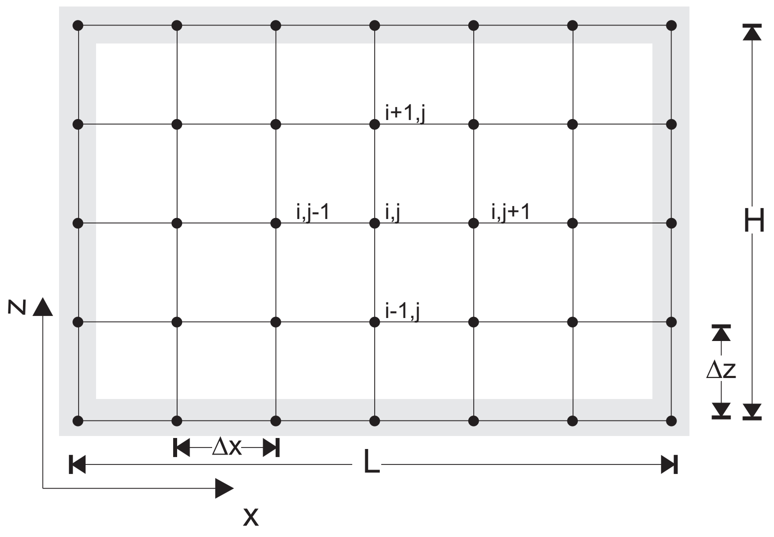

Figure 1: Finite difference discretization in 2D

5.3 Other methods

The fully implicit method discussed above works fine, but is only first order accurate in time. A simple modification is to employ a Crank-Nicolson timestep discretization which is second order accurate in time. I never saw a case where this really makes a big difference, but it is mathematically better and doesn't cost much in terms of additional programming, so you may consider using it for diffusion-type equations.

A different, and more serious, issue is the fact that the cost of solving $\mathbf{c} = \mathbf{A}\backslash\mathbf{rhs}$ is a strong function of the size of $\mathbf{A}$. This size depends on the number of gridpoints in x- (n_x) and z-direction (n_z). For a 2D problem with $n_x \times n_z$ internal points, $(n_x \times n_z)^2 \times (n_x \times n_z)^2$ equations have to be solved at every timestep. This quickly fills the computer memory (especially if going to 3D cases).

For the special case of the temperature equation, different techniques have therefore been developed. One such technique, is the socalled alternating direction implicit (ADI) method. It basically consists in solving the 2D equations half explicit and half implicit along 1D profiles (what you do is the following: (1) discretize equation 2 implicitly in the x-direction and explicit in the z-direction. (2) solve it for time $n + 1/2$, and (3) repeat the same but with an implicit discretization in the z-direction). Compared to the other method it is fast!! However, ADI-methods only work if the governing equations have time-derivatives, and unfortunately this is often not the case in geodynamics. In the exercises, we therefore focuss on the fully implicit formulation. If, however, you have to write a thermal solver at some point, you may strongly consider to use the ADI method (which is still very fast in 3D).

5.4 Exercises

In the first two exercises you're gonna program the diffusion equation in 2D both with an explicit and an implicit discretization scheme. The problem considered is that of the thermal structure of a lithosphere of 100 km thickness, with an initial linear thermal gradient of 13 K/km. Suddenly a plume with T=1500 C impings at the bottom of the lithosphere. What happen with the thermal structure of the lithosphere? A related (structural geology) problem is that of the cooling of batholites (like the ones in the Sierra Nevada).

1. Fill in the question marks in the script "heat2Dexplicit.m" (fig. 4), by programming the explicit finite difference scheme. Employ zero flux boundary conditions $\frac{\partial T}{\partial x} = 0$ on the left and on the

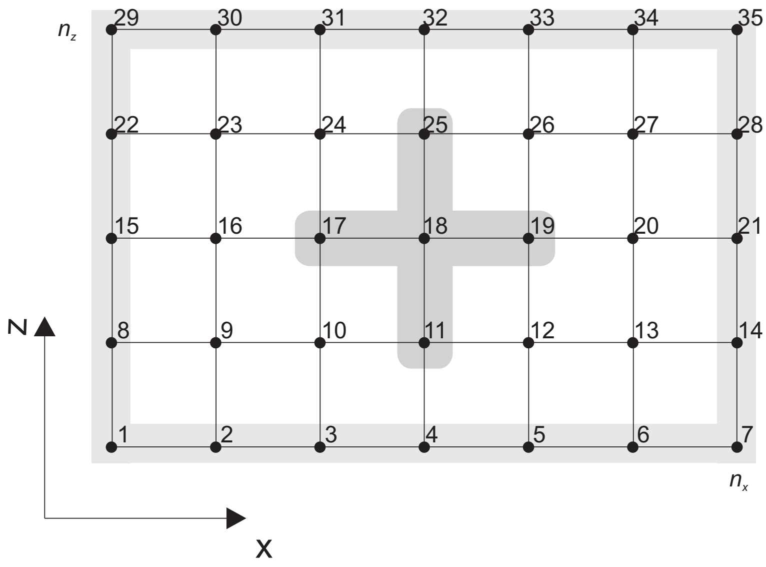

Figure 2: Numbering scheme for a 2D grid with $n_x = 7$ and $n_z = 5$.

right-side of the domain, and constant temperature conditions on the top and bottom. Ignore the effects of radioactive heat.

2. Finish the code "heat2Dimplicit.m", by programming the implicit finite difference approximation of the 2D temperature equation.

3. A simple (time-dependent) analytical solution for the temperature equation exists for the case that the initial temperature distribution is

$$T(x,z,t=0) = T_{max} \exp\left[\frac{-(x^2+z^2)}{\sigma^2}\right] \tag{18}$$

   where T_{max} is the maximum amplitude of the temperature perturbation at $(x,z) = (0,0)$ and σ it's half-width. The solution is than

$$T(x,z,t) = \frac{T_{max}}{1+4t\kappa/\sigma^2} \exp\left[\frac{-(x^2+z^2)}{\sigma^2+4t\kappa}\right] \tag{19}$$

   Program the analytical solution and compare it with the numerical solution with the same initial condition.

4. Bonus question 1: Add the effects of radioactive heat to the explicit/implicit equations above. Use Turcotte and Schubert (1981) or google to find typical values of Q, ρ, c_p for rocks.

5. Bonus question 2: write a code for the thermal equation with variable thermal conductivity k (equation 1). Assume that the grid spacing Δx is constant. This type of code is not only relevant for thermal problems, but also for problems like (1) hydrogeological problems (Darcy flow; how much do I have to pump to get drinking water?, or: how far did the chemical waste go into the aquifer?), (2) fluid movements through the crust and through fault zones (which is related to the creation of ore deposits), (3) magma migration through the mantle, (4) geochemistry and mineral reactions at grain-boundary scale, (5) aftershocks and fluids (well, depends whom you ask...).

6. Bonus question 3: write a code for the thermal equation with variable thermal conductivity k (equation 2), and with variable x- and z-spacing, variable density ρ and variable heat capacity c_p. Include source/sink terms.

```
%heat2D_explicit.m
%  1th exercise
% Solves the 2D heat equation with an explicit finite difference scheme

clear

%Physical parameters
L       =   150e3;      %   Width of lithosphere     [m]
H       =   100e3;      %   Height of lithosphere    [m]
Tbot    =   1300;       %   Temperature of bottom lithosphere  [C]
Tsurf   =   0;          %   Temperature of country rock        [C]
Tplume  =   1500;       %   Temperature of plume               [C]
kappa   =   1e-6;       %   Thermal diffusivity of rock        [m2/s]
Wplume  =   25e3;       %   Width of plume                     [m]
day     =   3600*24;    %   # seconds per day
year    =   365.25*day; %   # seconds per year

% Numerical parameters
nx      =   101;         %   # gridpoints in x-direction
nz      =   51;         %   # gridpoints in z-direction
nt      =   500;          %   Number of timesteps to compute
dx      =   L/(nx-1);   %   Spacing of grid in x-direction
dz      =   H/(nz-1);   %   Spacing of grid in z-direction
[x2d,z2d] = meshgrid(-L/2:dx:L/2, -H:dz:0);  % create grid

% Compute stable timestep
dt          = min([dx,dz])^2/kappa/4;

% Setup initial linear temperature profile
T       =   abs(z2d./H)*Tbot;

% Imping plume beneath lithosphere
ind      =  find(abs(x2d(1,:)) <= Wplume/2);
T(1,ind) =  Tplume;

time    =   0;
for n=1:nt

    % Compute new temperature
    Tnew = zeros(nz,nx);
    sx  = kappa*dt/dx^2;
    sz  = kappa*dt/dz^2;
    for j=2:nx-1
        for i=2:nz-1
            Tnew(i,j) = ????;
        end
    end

    % Set boundary conditions
    Tnew(1,:)   =  T(1 ,: );
    Tnew(nz,:)  =  ?;
    for i=2:nz-1
        Tnew(i,1) = ?
        Tnew(i,nx) = ?
    end

    T           =   Tnew;
    time        =   time+dt;

    % Plot solution every 50 timesteps
    if (mod(n,50)==0)
        figure(1), clf
        pcolor(x2d/1e3,z2d/1e3,Tnew); shading interp, colorbar
        hold on
        contour(x2d/1e3,z2d/1e3,Tnew,[100:100:1500],'k');

        xlabel('x [km]')
        ylabel('z [km]')
        zlabel('Temperature [^oC]')
        title(['Temperature evolution after ',num2str(time/year/1e6),' Myrs'])

        drawnow
    end

end
```

Figure 3: MATLAB script heat2D_explicit.m to solve the 2D heat equation.

```
%heat2D_implicit.m
%  2nd exercise
% Solves the 2D heat equation with an implicit finite difference scheme

clear

%Physical parameters
L       =   150e3;      %   Width of lithosphere    [m]
H       =   100e3;      %   Height of lithosphere   [m]
Tbot    =   1300;       %   Temperature of bottom lithosphere  [C]
Tsurf   =   0;          %   Temperature of country rock        [C]
Tplume  =   1500;       %   Temperature of plume               [C]
kappa   =   1e-6;       %   Thermal diffusivity of rock        [m2/s]
Wplume  =   25e3;       %   Width of plume                     [m]
day     =   3600*24;    %   # seconds per day
year    =   365.25*day; %   # seconds per year
dt      =   100e6*year;   %   timestep

% Numerical parameters
nx      =   51;         %   # gridpoints in x-direction
nz      =   51;         %   # gridpoints in z-direction
nt      =   100;          %   Number of timesteps to compute
dx      =   L/(nx-1);   %   Spacing of grid in x-direction
dz      =   H/(nz-1);   %   Spacing of grid in z-direction
[x2d,z2d] = meshgrid(-L/2:dx:L/2, -H:dz:0);  % create grid


% Setup initial linear temperature profile
T       =   abs(z2d./H)*Tbot;

% Imping plume beneath lithosphere
ind      =  find(abs(x2d(1,:)) <= Wplume/2);
T(1,ind) =  Tplume;

% Setup numbering
num      =  1;
for i=1:nz
    for j=1:nx
        Number(i,j) = num;
        num = num+1;
    end
end

% Construct the A matrix
A       = sparse(nx*nz,nx*nz);
sx      = kappa*dt/dx^2;
sz      = kappa*dt/dz^2;
for i = 2:nz-1
    for j = 2:nx-1
        ii          = Number(i,j);
        A( ii, Number(i+1,j  ))  =   ??;
        A( ii, Number(i  ,j+1))  =   ??;
  ??
    end
end

% Set lower and upper BC
for j = 1:nx
??
end

% Set left and right BC
for i = 1:nz
??
end


time    =   0;
for n=1:nt

    % Compute rhs
    rhs   = zeros(nx*nz,1);
    for i = 1:nz
        for j = 1:nx
            ii      = Number(i,j);
          ??
        end
    end

    % Compute solution vector
    Tnew_vector =   A\rhs;

    % Create 2D matrix from vector
    Tnew        =   Tnew_vector(Number);
    T           =   Tnew;
    time        =   time+dt;

    % Plot solution every 50 timesteps
    if (mod(n,10)==0)
        figure(1), clf
        pcolor(x2d/1e3,z2d/1e3,Tnew); shading interp, colorbar
        hold on
        contour(x2d/1e3,z2d/1e3,Tnew,[100:100:1500],'k');

        xlabel('x [km]')
        ylabel('z [km]')
        zlabel('Temperature [^oC]')
        title(['Temperature evolution after ',num2str(time/year/1e6),' Myrs'])

        drawnow
    end

end
```

Figure 4: MATLAB script heat2D_implicit.m to solve the 2D heat equation.