540 Geodynamics – University of Southern California, Los Angeles

Computational Geodynamics: Advection equations and the art of numerical modeling

Sofar we mainly focussed on diffusion equation in a non-moving domain. This is maybe relevant for the case of a dike intrusion or for a lithosphere which remains undeformed. However there are also cases where material moves. An example is a plume rising through a convecting mantle. The plume is hot and hence its density is low compared to the colder mantle around it. The hot material rises with a given velocity (when term-projects are finished we will have codes that give us those velocities for a given density distribution). If the numerical grid remains fixed in the background, the hot temperatures should be moved to different gridpoints (see fig. 1 for an illustration of this effect).

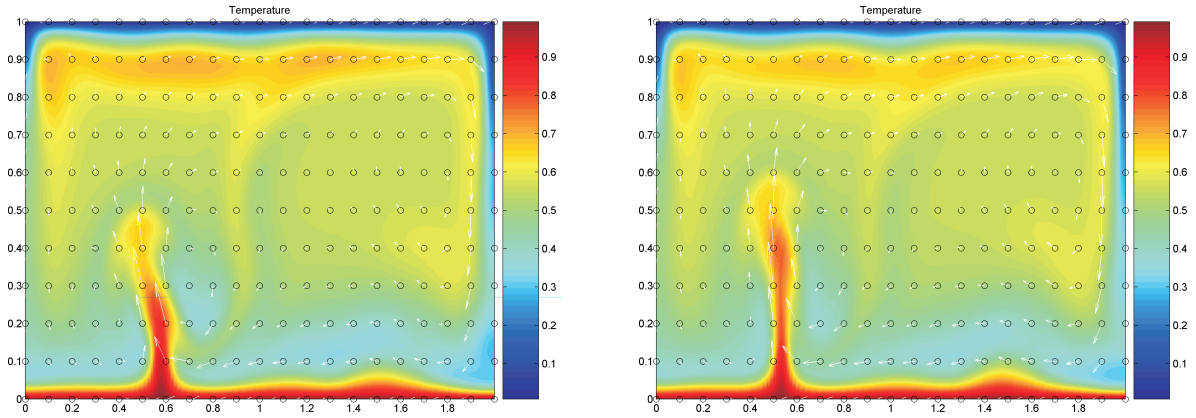

Figure 1: Snapshots of a bottom heated thermal convection model with a Rayleigh-number of 5×10^5 and constant viscosity (no internal heating). Temperature is advected through a fixed (eulerian) grid (circles) with a velocity (arrows) that is computed with a Stokes solver.

Mathematically, the temperature equation gets an additional term and in 1-D the diffusion-advection equation becomes (see equation cheat sheet)

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

or in 2-D

$$\rho c_p \left(\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} + v_z \frac{\partial T}{\partial z} \right) = \frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial z} \left(k \frac{\partial T}{\partial z} \right) \tag{2}$$

where v_x, v_z are velocities in x-,respectively z-direction.
If diffusion is absent (i.e. $k = 0$), the advection equations are

$$\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} = 0 \tag{3}$$

and

$$\frac{\partial T}{\partial t} + v_x \frac{\partial T}{\partial x} + v_z \frac{\partial T}{\partial z} = 0 \tag{4}$$

These equations should also be solved if you want to compute the settling of a heavy particle in a fluid, or in weather prediction codes. In the current lecture we're going to look at some options on how to solve these equations with a finite difference scheme on a fixed grid. Even though the equations are fairly simple, it is far from simple to solve them accurately. If not done carefully, you'll end up with strong numerical artifacts such as wiggles and numerical diffusion.

```
% advection_central_1D
%

clear all

nx      =   201;
W       =    40;     % width of domain
Vel     =   -4;      % velocity
sigma   =   1;
Ampl    =   2;
nt      =   500;    % number of timesteps
dt      =   1e-2;

dx      =   W/(nx-1);
x       =    0:dx:W;

% Initial gaussian T-profile
xc      =   20;
T       =    Ampl*exp(-(x-xc).^2/sigma^2);

% Velocity
Vx      =   ones(1,nx)*Vel;

% Central finite difference discretization
for itime=1:nt

    % central fin. diff
    for ix=2:nx-1
        Tnew(ix) = ??
    end

    Tnew(1)      =   T(1);
    Tnew(nx)     =   T(nx);
    T            =    Tnew;
    time         =   itime*dt;

    % Analytical solution for this case
    T_anal       =    Ampl*exp(-(x-xc-time*Vel).^2/sigma^2);

    figure(1),clf, plot(x,T,x,T_anal),  legend('Numerical','Analytical')
    drawnow
end
```

Figure 2: MATLAB script to be used with exercise 1.

FTCS method

In a 1-D case the simplest way to discretize equation 3 is by employing a central finite difference scheme (also called the forward-time, central space or FTCS scheme):

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

Exercise 1

- Program the FTCS method in the code of figure 2 and watch what happens. Change the sign of the velocity. Change the timestep and gridspacing and compute the nondimensional parameter $\alpha = |v_x|\Delta t/\Delta x$. When do unstable results occur?

As you saw from the exercise, the FTCS method does not work... In fact it is a nice example of a scheme that looks nice (and logical!) on paper, but sucks [you can actually mathematically prove this by a socalled von Neuman stability analysis, if you're interested you should read chapter 5 of Marc Spiegelman's script on the course webpage; it's related to the fact that this scheme produces anti-diffusion, which is numerically instable].

Lax method

Some day, long ago, there was a guy called Lax and he thought: well let's replace the T_i^n in the time-derivative of equation 5 with $(T_{i+1}^n + T_{i-1}^n)/2$. The resulting equation is

$$\frac{T_i^{n+1} - (T_{i+1}^n + T_{i-1}^n)/2}{\Delta t} = -v_{x,i} \frac{T_{i+1}^n - T_{i-1}^n}{2\Delta x} \tag{6}$$

Exercise 2

- Program the Lax method by modifying the script of the last exercise. Play with different velocities and compute the Courant number α, which is given by the following equation:

$$\alpha = \frac{\Delta x}{\Delta t |v_x|} \tag{7}$$

  Is the numerical scheme stable for all Courant numbers?

As you saw from the exercise the Lax method doesn't blow up, but does have a lot of numerical diffusion. So it's an improvement but it's still not perfect, since you may loose the plumes of figure 1 around mid-mantle purely due to numerical diffusion (and than you'll probably find some fancy geochemical explanation for this...). I hope that you also slowly start to see that numerical modeling is a bit of an art...

Upwind scheme

The next scheme that people came up with is the so-called upwind scheme. Here, the spatial finite difference scheme depends on the sign of the velocity:

$$\frac{T_i^{n+1} - T_i^n}{\Delta t} = -v_{x,i} \begin{cases} \frac{T_i^n - T_{i-1}^n}{\Delta x}, \text{ if } v_{x,i} > 0 \\ \frac{T_{i+1}^n - T_i^n}{\Delta x}, \text{ if } v_{x,i} < 0 \end{cases} \tag{8}$$

Exercise 3

- Program the upwind scheme method. Play with different velocities and compute the Courant number α Is the numerical scheme stable for all Courant numbers?

Also the upwind scheme suffers from numerical diffusion.

Sofar we employed explicit discretizations. You're probably wondering whether implicit discretizations will save us again this time. Bad news: doesn't work (try if you want). So we have to come up with something else.

Staggered Leapfrog

The explicit discretizations discussed sofar were second order accurate in time, but only first order in space. We can also come up with a scheme that is second order in time and space

$$\frac{T_i^{n+1} - T^{n-1}}{2\Delta t} = -v_{x,i} \frac{T_{i+1}^n - T_{i-1}^n}{2\Delta x} \tag{9}$$

the difficulty in this scheme is that we'll have to store two timestep levels: both T^{n-1} and T^n (but i'm sure you'll manage).

Exercise 4

- Program the staggered leapfrog method (assume that at the first timestep $T^{n-1} = T^n$). Play around with different values of the Courant number α. Also make the width of the gaussian curve smaller.

The staggered leapfrog method works quite well regarding the amplitude and wavespeed as long as α is close to one. If however $\alpha << 1$ and the lengthscale of the to-be-transported quantity is small compared to the number of gridpoints (e.g. we have a thin plume), numerical oscillations again occur. This is typically the case in mantle convection simulations (see Fig. 1), so again we have a numerical scheme that's not great for mantle convection simulations.

MPDATA

This is a technique that is frequently applied in (older) mantle convection codes. The basic idea is based on an idea of Smolarkiewicz and is basically a modification of the upwind scheme, where some iterative corrections are applied. The results are quite ok. However it's somewhat more complicated to implement (the system that's solved is actually nonlinear and requires iterations, which we did not discuss yet). Moreover we still have a restriction on the timestep (given by the Courant criterium). For this reason we won't go into detail here (see Spiegelman's lecture notes, chapter 5 if you're interested).

Semi-Lagrangian

So what we want is a scheme that doesn't blow up, has only small numerical diffusion and that is not limited by the Courant criterium. It actually exists and is called the semi-lagrangian method. I'm a big fan of it and so is Marc Spiegelman (see his course notes), and people that do climate modelling or numerical weather prediction. It's however a bit weird and has little to do with the finite difference schemes we discussed sofar. Since this scheme is however the one that's maybe most important for practical purposes we will go in more detail here.

Basic idea

The basic idea of the semi-lagrangian method is illustrated in figure 3A and is given by the following
For each point i

1. Assume that the future velocity $v_x(n+1,i)$ is known. Under the assumption that the velocity at the old timestep is almost identical to the future velocity ($v_x(n+1,i) \cong v_x(n,i)$) *and* that velocity doesn't vary spatially ($v_x(n,i-1) \cong v_x(n,i) \cong v_x(n,i+1)$), we can compute the location $\mathbf{X}$ where the particle came from by $\mathbf{X}= x(i) - \Delta t v_x(n+1,i)$

2. Interpolate your temperature from gridpoints to the location $\mathbf{X}$ at time n. Use cubic interpolation (in MATLAB use the command interp1(x,T, x_point, 'cubic') for interpolation). Now you simply say that $T(n+1,i) = T(n,\mathbf{X})$, and you're done. Note that this scheme assumes that no heat-sources were active during the advection of T from $T(n,\mathbf{X})$ to $T(n+1,i)$. If heat sources are present *and* are spatially variable, some extra stuff needs to be done (read Marc's lecture notes).

Exercise 5

- Program the semi-lagrangian advection scheme illustrated in figure 3A. Is there a Courant criterium for stability?

Some improvements

The algorithm described in figure 3A illustrates the basic idea of the semi-lagrangian scheme. However it has two problems. First it assumes that velocity is spatially constant (which is clearly not the case in mantle convection simulations). Second, it assumes that velocity doesn't change between time n and $n+1$. We can overcome both problems by using a more accurate timestepping algorithm. An example is an iterative mid-point scheme which works as follows (see figure 3B):
For each point i

1. Use the velocity $v_x(n+1,i)$ to compute the location $\mathbf{x}$ at time $n+1/2$ (in other words: take half a timestep backwards in time).

2. Find the velocity at the location $\mathbf{x}$ at half timestep $n+1/2$. Assume that the velocity at the half-timestep can be computed as $v_x(n+1/2,i) = \frac{v_x(n+1,i)+v_x(n,i)}{2}$. Use linear interpolation for the spatial interpolation of velocity $v_x(n+1/2,\mathbf{x})$.

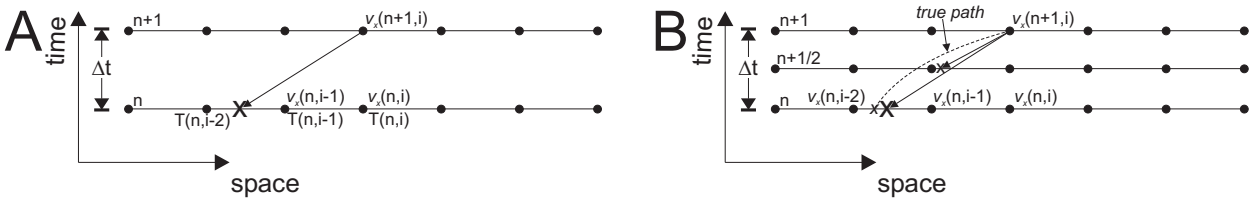

Figure 3: Basics of the semi-lagrangian method. See text for explanation.

3. Go back to point 1, but use the velocity $v_x(n+1/2,\mathbf{x})$ instead of $v_x(n+1,i)$ to move the point $x(n+1,i)$ backwards in time. Repeat this process a number of times (for example 5 times). This gives a fairly accurate centered velocity.

4. Compute the location $\mathbf{X}$ with the centered velocity $v_x(n+1/2,\mathbf{x})$, with $\mathbf{X}= x(i)-\Delta t v_x(n+1/2,\mathbf{x})$.

5. Use cubic interpolation to find the temperature at point $\mathbf{X}$.

Other methods are also possible. A popular one is the fourth order Runga-Kutta method, which is implemented similarly (but takes a bit more work).

Exercise 6

- Program the semi-lagrangian advection scheme with the centered midpoint method as illustrated in figure 3B. If you don't understand the recipe above check Marc's lecture notes, page 67 for a MATLAB pseudocode. Some care has to be taken if point $\mathbf{X}$ is outside of the computational domain, since MATLAB will return NaN for the velocity (or temperature) of this point. Use the velocity $v_x(n+1,i)$ in this case. A pseudocode is given by
  if isnan(Velocity)
  Velocity = Vx(i)
  end

2D advection

Since I'm convinced that the semi-lagrangian method is the only good advection algorithm (except in the case of pseudospectral methods), this is the one we're going to implement in 2D.
Assume that velocity is given by

$$v_x(x,z) = z \tag{10}$$
$$v_z(x,z) = -x \tag{11}$$

Moreover, assume that the initial temperature distribution is gaussian and given by

$$T(x,z) = 2\exp\left(\frac{\left((x+0.25)^2+z^2\right)}{0.1^2}\right) \tag{12}$$

with $x \in [-0.5,0.5]$, $z \in [-0.5,0.5]$.

Exercise 7

- Program advection in 2D using the semi-lagrangian advection scheme with the centered midpoint method. Use the MATLAB routine interp2 for interpolation and employ linear interpolation for velocity and cubic interpolation for temperature. A MATLAB script that'll get you started is shown on figure 4.

```
% semi_lagrangian_2D: 2D semi-lagrangian  with center midpoint time stepping method
%

clear all

W        =    40;      % width of domain
sigma    =   .1;
Ampl     =   2;
nt       =    500;    % number of timesteps
dt       =    5e-1;

% Initial grid and velocity
[x,z]    =    meshgrid(-0.5:.025:0.5,-0.5:.025:0.5);
nz       =    size(x,1);
nx       =    size(x,2);

% Initial gaussian T-profile
T        =    Ampl*exp(-((x+0.25).^2+z.^2)/sigma^2);

% Velocity
Vx       =    z;
Vz       =    -x;

for itime=1:nt

    Vx_n     = Vx;                      %    Velocity at time=n
    Vx_n1    = Vx;                      %    Velocity at time=n+1
%     Vx_n1_2 = ??;                      %    Velocity at time=n+1/2
%     Vz_n     = ??;                      %    Velocity at time=n
%     Vz_n1    = ??;                      %    Velocity at time=n+1
%     Vz_n1_2 = ??;      %    Velocity at time=n+1/2
    Tnew     =    zeros(size(T));
    for ix=2:nx-1
        for iz=2:nz-1

            Vx_cen = Vx(iz,ix);
            Vz_cen = Vz(iz,ix);
   %          for ??
%                 X =?
%                 Z = ?

                %linear interpolation of velocity
%                 Vx_cen = interp2(x,z,?,?, ?, 'linear');
%                 Vz_cen = interp2(x,z,?,?, ?, 'linear');

                if isnan(Vx_cen)
                    Vx_cen =    Vx(iz,ix);
                end
                if isnan(Vz_cen)
                    Vz_cen =    Vz(iz,ix);
                end


%             end
%             X = ?;
%             Z = ?;

            % Interpolate temperature on X
%             T_X = interp2(x,z,?,?,?, 'cubic');
            if isnan(T_X)
                T_X = T(iz,ix);
            end

            Tnew(iz,ix) = T_X;
        end

    end

    Tnew(1,:)    =    T(1,:);
    Tnew(nx,:)   =    T(nx,:);
    Tnew(:,1)    =    T(:,1);
    Tnew(:,nx)   =    T(:,nx);

    T            =    Tnew;
    time         =    itime*dt;


    figure(1),clf
    pcolor(x,z,T), shading interp, hold on, colorbar
    contour(x,z,T,[.1:.1:2],'k'),
    hold on, quiver(x,z,Vx,Vz,'w')
    axis equal, axis tight
    drawnow
    pause
end
```

Figure 4: MATLAB script to be used with exercise 7.

Advection and diffusion: operator splitting

Sofar we learned that the semi-lagrangian methods are probably the best methods for advection dominated problems. We also learned how to solve the diffusion equation in 1-D and 2-D. In geodynamics, we often want to solve the coupled advection-diffusion equation, which is given by equation 1 in 1-D and by equation 2 in 2-D. We can solve this pretty easily by taking the equation apart and by computing the advection part separately from the diffusion part. This is called operator-splitting, and what's done in 1-D is the following. First solve the advection equation

$$\frac{\tilde{T}^{n+1} - T^n}{\Delta t} + v_x \frac{\partial T}{\partial x} = 0 \tag{13}$$

for example by using a semi-lagrangian advection scheme. Then solve the diffusion equation

$$\rho c_p \frac{\partial \tilde{T}}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial \tilde{T}}{\partial x}\right) + Q \tag{14}$$

and you're done (note that I assumed that Q is spatially constant; if not you should consider to slightly improve the advection scheme by introducing source terms) . In 2D the scheme is identical (but you'll have to find your old 2D codes...).

Exercise 8

- Program diffusion-advection in 2D using the semi-lagrangian advection scheme coupled with an implicit 2D diffusion code (from last week's exercise). Base your code on the script of figure 4.