Numerical Modelling in FORTRAN day 11

Paul Tackley, 2014

Today's Goals

1. **Useful libraries and other software**
2. **Implicit time stepping**

Useful libraries and other software

- For standard operations such as matrix solution of systems of linear equations, taking fast-Fourier transforms, etc., it is convenient to download suitable routines from some free library, rather than writing your own from scratch.
- There are also more specialised free programs available for some applications
- A list is at: http://www.lahey.com/other.htm
- Much is available at: http://www.netlib.org/

Some well-known libraries

- lapack95: common linear algebra problems: equations, least squares, eigenvalues etc.
  – http://www.netlib.org/lapack95/
  – also a parallel version scalapack http://www.netlib.org/scalapack/
- MPI: for running programs on multiple CPUs. 2 common versions are
  – MPICH: http://www.mpich.org/
  – Open MPI: http://www.open-mpi.org/
- PETSc: Portable, Extensible Toolkit for Scientific Computation: http://www.mcs.anl.gov/petsc/
- ETH has a site license for the commercial NAG mathematical library: see ides.ethz.ch

Implicit time-stepping

Diffusion equation again

$$\frac{\partial T}{\partial t} = \nabla^2 T$$

So far we have used **explicit** time stepping

physical equation

$$\frac{\partial T}{\partial t} = \nabla^2 T$$

explicit => calculate derivatives at old time

$$\frac{T_{new} - T_{old}}{\Delta t} = \nabla^2 T_{old}$$

$$T_{new} = T_{old} + \Delta t \nabla^2 T_{old}$$

each Tnew can be calculated from already-known Told

But: the value of del2(T) changes over dt

Implicit: calculate derivatives at new time

$$\frac{T_{new} - T_{old}}{\Delta t} = \nabla^2 T_{new}$$

Put knowns on right-hand side, unknowns on LHS

$$T_{new} - \Delta t \nabla^2 T_{new} = T_{old}$$

More complicated to find Tnew: del2 term links all Tnew points.
Why use it? Because the timestep is not limited: stable for any dt

任何一个时间dt都可以

Eqn looks like Poisson's equation: we can modify existing solver

How about accuracy?

- Simple implicit and explicit methods are both first order accurate. means T changes linearly
- Several related methods give second-order accuracy, including predictor-corrector, Runge-Kutta and semi-implicit.
- Of these, only the semi-implicit method has an unlimited time step, so let's focus on that!
- The semi-implicit method uses the average of derivatives at the beginning and end of the time step

semi-implicit diffusion

$$\frac{T_{new} - T_{old}}{\Delta t} = \frac{\nabla^2 T_{new} + \nabla^2 T_{old}}{2}$$

Now generalised equation to deal with all 3 cases:
explicit (beta=0), implicit (beta=1), semi-implicit (beta=0.5)

$$\frac{T_{new} - T_{old}}{\Delta t} = \beta \nabla^2 T_{new} + (1 - \beta) \nabla^2 T_{old}$$

Put knowns on right-hand side, unknowns on LHS

$$\boxed{T_{new} - \beta \Delta t \nabla^2 T_{new} = T_{old} + \Delta t (1 - \beta) \nabla^2 T_{old}}$$

Rearrange to look like Poisson

$$T_{new} - \beta \Delta t \nabla^2 T_{new} = T_{old} + \Delta t (1-\beta) \nabla^2 T_{old}$$

$$\left(\nabla^2 - 1/(\beta \Delta t)\right) T_{new} = -\frac{1}{\beta \Delta t}\left[T_{old} + \Delta t (1-\beta) \nabla^2 T_{old}\right]$$

So, modify the Poisson solver to solve $\left(\nabla^2 - C\right) T = f$

Finite-difference ‘modified Poisson’

$$\left(\nabla^2 - C\right)T = f$$

$$\frac{T_{i-1,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1} - (4 + Ch^2)T_{i,j}}{h^2} = f_{i,j}$$

Iterative correction:

$$R_{i,j} = \frac{T_{i-1,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1} - (4 + Ch^2)T_{i,j}}{h^2} - f_{i,j}$$

$$T^{n+1} = T^n + \frac{\alpha h^2}{\left(4 + Ch^2\right)} R$$

9th homework

- Modify your favourite program (either *diffusion*, *advection-diffusion*, *infinite-Pr convection* or *low-Pr convection*) to include implicit diffusion
- Test for different values of beta (0, 0.5 and 1.0) run the same cases
- If beta≥0.5, ignore the diffusion timestep constraint!
- Due date: 19 December (2 weeks from now)

Implementing implicit diffusion

- Modify your Poisson solver (iterations and multigrid) passing the constant 'C' in as an extra argument.
  - C=0 should give same result as before. Useful for streamfunction calculation.
- Add 'beta' to the namelist inputs
- Modify diffusion calculation
  - if beta>0 call multigrid, otherwise explicit like before
  - use beta when calculating rhs term
  - ignore diffusive timestep if beta≥0.5
- Run some tests with beta=0, 0.5 or 1.0 and see if you can tell the difference. It will have more effect at low Ra (or B), when diffusion dominates

Example: Low Pr convection

Temperature equation with variable beta:

$$T_{new} - \beta\Delta t\nabla^2 T_{new} = T_{old} + \Delta t\left((1-\beta)\nabla^2 T_{old} - \left(\vec{v}\cdot\nabla T\right)_{old}\right)$$

Rearrange:

$$\left(\nabla^2 - 1/(\beta\Delta t)\right)T_{new} = -\frac{1}{\beta\Delta t}\left[T_{old} + \Delta t\left((1-\beta)\nabla^2 T_{old} - \left(\vec{v}\cdot\nabla T\right)_{old}\right)\right]$$

Call the new 'modified Poisson' solver $\left(\nabla^2 - C\right)u = f$

With (u=T)

$$C = 1/(\beta\Delta t);\ f = -\frac{1}{\beta\Delta t}\left[T_{old} + \Delta t\left((1-\beta)\nabla^2 T_{old} - \left(\vec{v}\cdot\nabla T\right)_{old}\right)\right]$$

Example: Low Pr convection (2)

Vorticity equation with variable beta:

$$\omega_{new} - \Pr \beta \Delta t \nabla^2 \omega_{new} = \omega_{old} + \Delta t \left(\Pr(1-\beta)\nabla^2 \omega_{old} - \left(\vec{v}\cdot\nabla\omega\right)_{old} - Ra \Pr \frac{\partial T_{old}}{\partial x} \right)$$

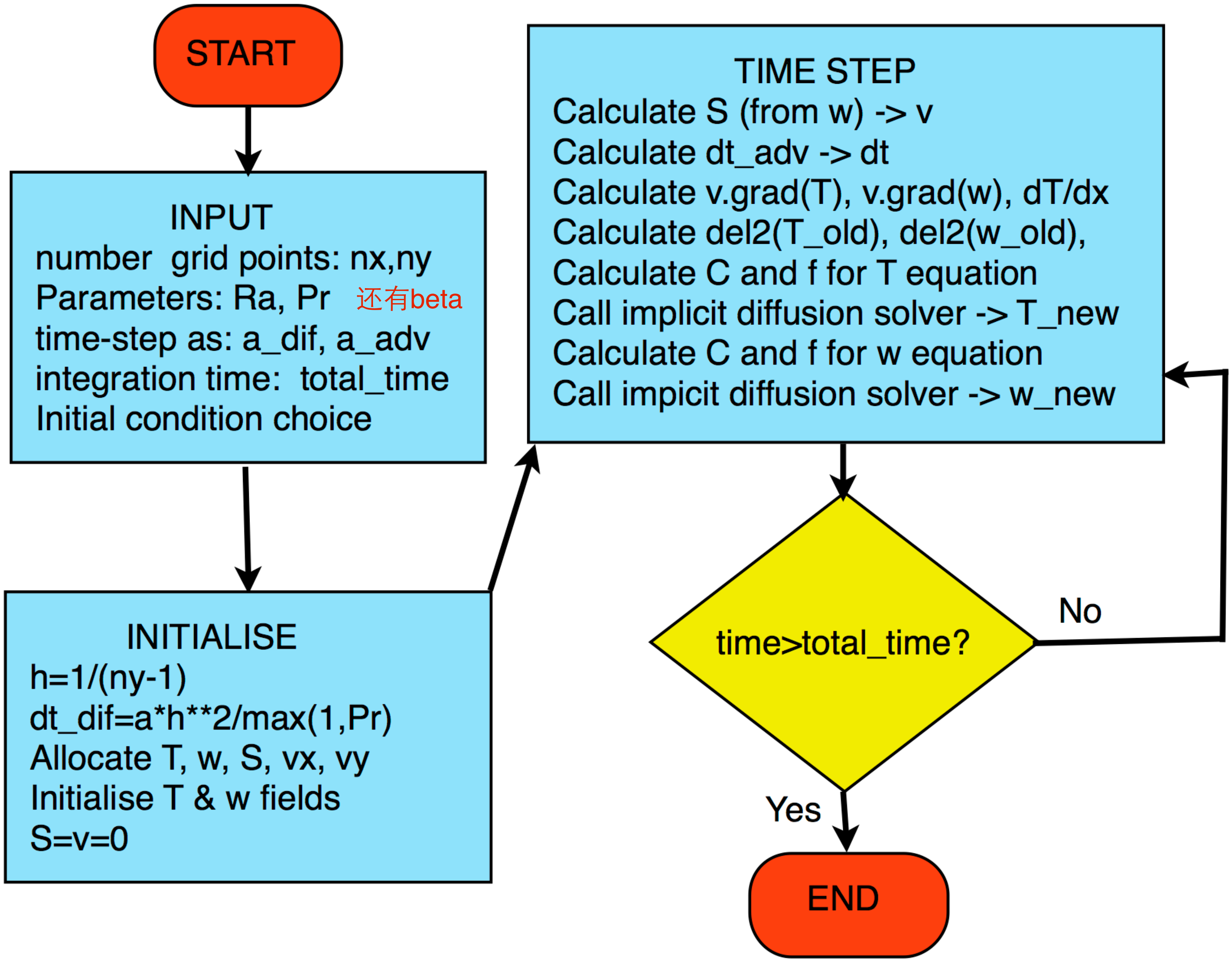
Rearrange:

$$\left(\nabla^2 - \frac{1}{\Pr \beta \Delta t}\right)\omega_{new} = -\frac{1}{\Pr \beta \Delta t}\left[\omega_{old} + \Delta t \left(\Pr(1-\beta)\nabla^2 \omega_{old} - \left(\vec{v}\cdot\nabla\omega\right)_{old} - Ra \Pr \frac{\partial T_{old}}{\partial x} \right)\right]$$

Call the new ‘modified Poisson’ solver $\left(\nabla^2 - C\right)u = f$

With (u=w)

$$C = \frac{1}{\Pr \beta \Delta t}; \;\; f = -\frac{1}{\Pr \beta \Delta t}\left[\omega_{old} + \Delta t \left(\Pr(1-\beta)\nabla^2 \omega_{old} - \left(\vec{v}\cdot\nabla\omega\right)_{old} - Ra \Pr \frac{\partial T_{old}}{\partial x} \right)\right]$$


START
INPUT
number grid points: nx,ny
Parameters: Ra, Pr
还有beta
time-step as: a_dif, a_adv
integration time: total_time
Initial condition choice
INITIALISE
h=1/(ny-1)
dt_dif=a*h**2/max(1,Pr)
Allocate T, w, S, vx, vy
Initialise T & w fields
S=v=0
TIME STEP
Calculate S (from w) -> v
Calculate dt_adv -> dt
Calculate v.grad(T), v.grad(w), dT/dx
Calculate del2(T_old), del2(w_old),
Calculate C and f for T equation
Call implicit diffusion solver -> T_new
Calculate C and f for w equation
Call impicit diffusion solver -> w_new
time>total_time?
No
Yes
END

Note about boundary conditions

- If your existing iteration routines assume boundary conditions are 0 all round, they need modifying for T field.
- T boundary conditions are 0 at top, dT/dx=0 at sides, and
  - On fine grid: 1 at bottom
  - On coarse grids: 0 at bottom
- Either pass a boundary condition switch into the iteration routines, or write different routines for S and T fields.

Hand in

- Source code
- Results of test case(s) run using different values of beta and (if beta>0.5) different time steps including ones > diffusive time step