Numerical Modelling in FORTRAN day 6

Paul Tackley, 2020

Today‘s Goals

1. Learn about **binary input/output**
2. Practice, practice! Combine advection-diffusion and Poisson solvers to make a **convection simulation program**

Projects: start thinking about

Agree topic with me
before final lecture

Reminder: Debugging your code: Useful compiler options

- Examples with many checks switched on:
  - gfortran –fbacktrace –fcheck=all –finit-real=snan -ffpe-trap=invalid,zero,overflow –O0 program.f90
  - ifort –traceback –check uninit,bounds –init=snan,arrays –fpe0 –O0 program.f90

- Note: The program will run slowly with these options switched on. Use only when developing/debugging your program. Then compile using –O2 or –Ofast.
- Note: instead of typing all this every time use e.g. alias gf="gfortran –fbacktrace … –O0 "

Binary versus text (ascii) input/output
aka “unformatted” and “formatted”

- Binary I/O is
  - **faster** (data goes directly between memory and file, no conversion to/from text)
  - more **compact** (e.g., takes ~12 characters to write a 4-byte number with full precision)
  - more **accurate** (full accuracy retained, nothing lost in conversion)
- but...
  - more difficult to read files into other applications
  - different byte orders for different CPUs (most- or least-significant byte first)

Opening an unformatted (binary) file

- Same as text file except use **form= ‘unformatted’ ,** e.g.,
- open(1,file= ‘myfile’ ,form= ‘unformatted’)
- Optional keywords:
  - access= ‘direct’ or ‘sequential’ or ‘stream’ ! see next slide
  - recl=N ! record length:see next slide
  - status= ‘old’ or ’ new’ or ‘replace’ or ’ scratch’ or ‘unknown’
  - position= ‘asis’ or ’ rewind’ or ’ append’
  - action= ‘read’ or ‘write’ or ‘readwrite’
  - err=label ! if error, goes to label
  - iostat=ios ! ios=0 if successful

3 types of unformatted file

- access= '**sequential**' *(default)*
  - data is written or read sequentially as 'records': no jumping back and forth except with **rewind()** and **backspace()**
- access= '**direct**'
  - data is written in fixed-length records with length defined by **recl=N**
  - any record can be directly read or rewritten in any order
- access='**stream**' *(recommended, f95-)*
  - As sequential but without records. Similar to C. Can jump to any position using **pos=N**
    - Units are normally bytes. Pos=1 is the beginning of the file.
  - Can also be used with formatted (text) files.

write and **read** to binary file

- Basically the same as for formatted files except you don't specify a format, e.g.,
  - write (1) a,b,i
  - read (1,err=10,end=20,iostat=ios) a,b,i
  - read (1,rec=recordnumber) stuff ! for direct access files
  - read (1,pos=2020) stuff ! for stream files
- err, end, and iostat are useful for handling errors (otherwise the code will stop with an error message)

Inquire(): useful for obtaining information about a file

- inquire(file='Myfile.txt',exist=i) ! Does it exist? (i=true/false)
- inquire(directory='results',exist=i) ! Same for directory
- inquire(unit=3,pos=ipos) ! ipos=position of next read/write
- Many more options: iostat, opened, number, named, access, sequential, direct, form, formatted, unformatted, recl, nextrec, blank, position, action, read, write, readwrite, delim, pad

Binary files & Matlab

- Write from Fortran:

```
open(1,file='T.bin',form='unformatted',access='stream')
write(1) nx,ny,T
close(1)
```

- Read into Matlab:

```
fid = fopen('T.bin');
nx = fread(fid,1,'int32');
ny = fread(fid,1,'int32');
T = reshape(fread(fid,nx*ny,'single'),nx,ny);
fclose(fid);
contourf(T')
```

Pay attention to precision! (32/64 bits)

Command-line arguments

e.g. to pass the name of the parameter file to your program:

```
myprogram parB1.txt
myprogram parB10.txt
myprogram parB100.txt
```

Use the built-in (since f2003) routines command_argument_count, get_command_argument & get_command

Command-line arguments

- To read an optional parameter file name:

```
character(len=100):: fname='parameters.txt'

if (command_argument_count()>0) &
     call get_command_argument(1,fname)
print*,fname
```

```
>a.out
 parameters.txt

>a.out test.txt
 test.txt
```

get_command_argument(num,value,length,status)

num = which one (integer)

value = requested argument (string) (optional)

length = length of argument (integer) (optional)

status = 0 success, -1 value too short, 1 not possible

Now back to iterative solvers and advection-diffusion…

Goal for today's exercise: Calculate velocity field from temperature field =>**convection**

Non-dimensional equations for convection

Conservation of mass & energy (as in advection-diffusion exercise)

$$\nabla \cdot \vec{v} = 0$$

$$\frac{\partial T}{\partial t} = \nabla^2 T - \vec{v} \cdot \nabla T$$

Conservation of momentum for a highly-viscous fluid (Stokes equation)

$$-\nabla P + \nabla^2 \vec{v} = -Ra.T\hat{y}$$

Only one parameter: the Rayleigh number

$$Ra = \frac{\rho g \alpha \Delta T D^3}{\eta \kappa}$$

(ρ=density, g=gravity, α=thermal expansivity, ΔT=temperature drop, D=depth, η=viscosity, κ=thermal diffusivity)

Solving the Stokes equation

For highly viscous flow (e.g., Earth's mantle) with constant viscosity (*P*=pressure, *Ra*=Rayleigh number):

$$-\nabla P + \nabla^2 \vec{v} = -Ra.T\hat{y} \qquad \text{(y points up)}$$

Substituting the streamfunction for velocity, we get:

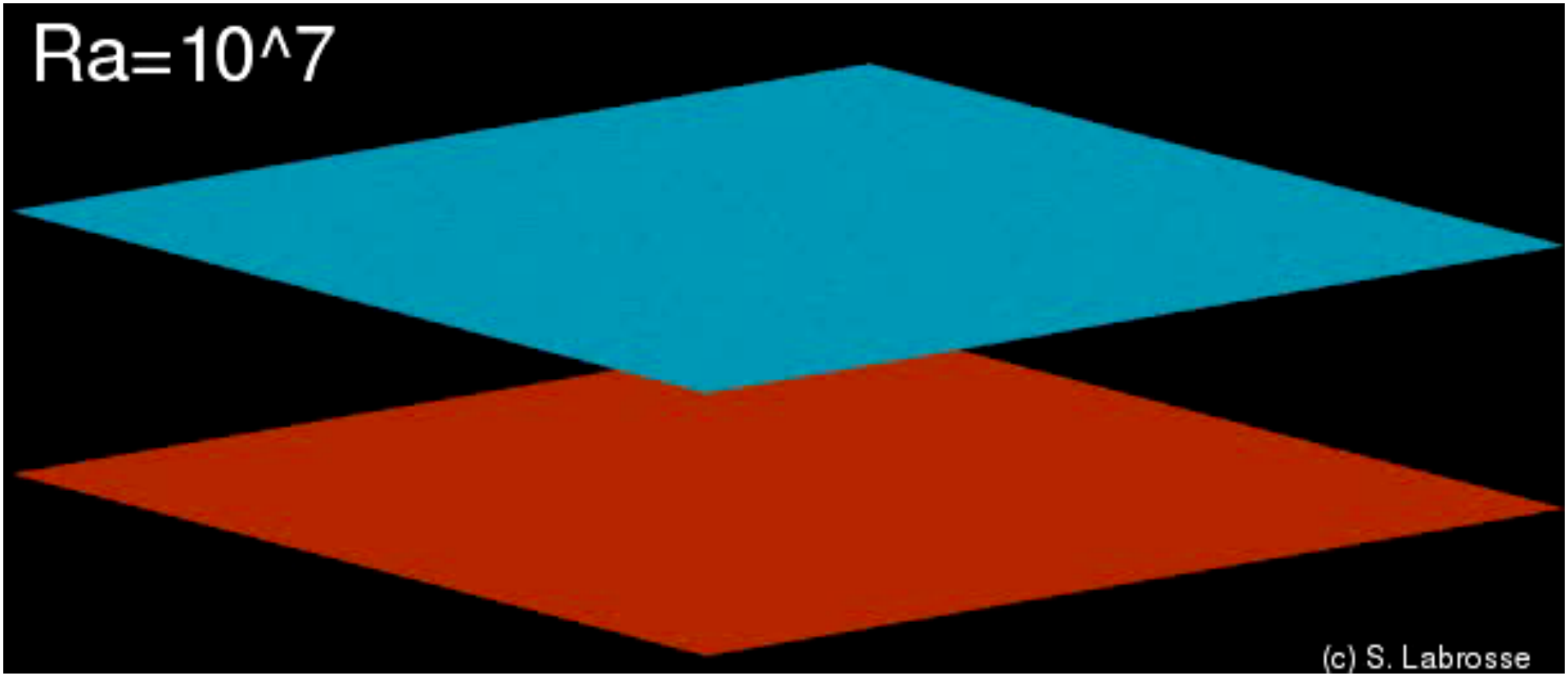
$$\left(v_x, v_y\right) = \left(\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x}\right) \qquad \nabla^4 \psi = Ra \frac{\partial T}{\partial x}$$

writing as 2 Poisson equations:

$$\nabla^2 \psi = \omega \qquad \nabla^2 \omega = Ra \frac{\partial T}{\partial x}$$

the **streamfunction-vorticity** formulation

Example of convection (in 3-D)



Mathematical aside

$-\nabla P + \nabla^2 \vec{v} = -Ra.T\hat{y}$ Is a vector equation with x and y parts:

$$-\frac{\partial P}{\partial x} + \nabla^2 v_x = 0 \quad (1) \qquad -\frac{\partial P}{\partial y} + \nabla^2 v_y = -Ra.T \quad (2)$$

Take $\frac{\partial(1)}{\partial y} - \frac{\partial(2)}{\partial x} \Rightarrow$ $\nabla^2 \left(\frac{\partial v_x}{\partial y} - \frac{\partial v_y}{\partial x} \right) = Ra \frac{\partial T}{\partial x}$

Use streamfunction: $\nabla^2 \left(\frac{\partial^2 \psi}{\partial y^2} + \frac{\partial^2 \psi}{\partial x^2} \right) = \nabla^2 \nabla^2 \psi = Ra \frac{\partial T}{\partial x}$

This is the z-component of the curl of the momentum equation:

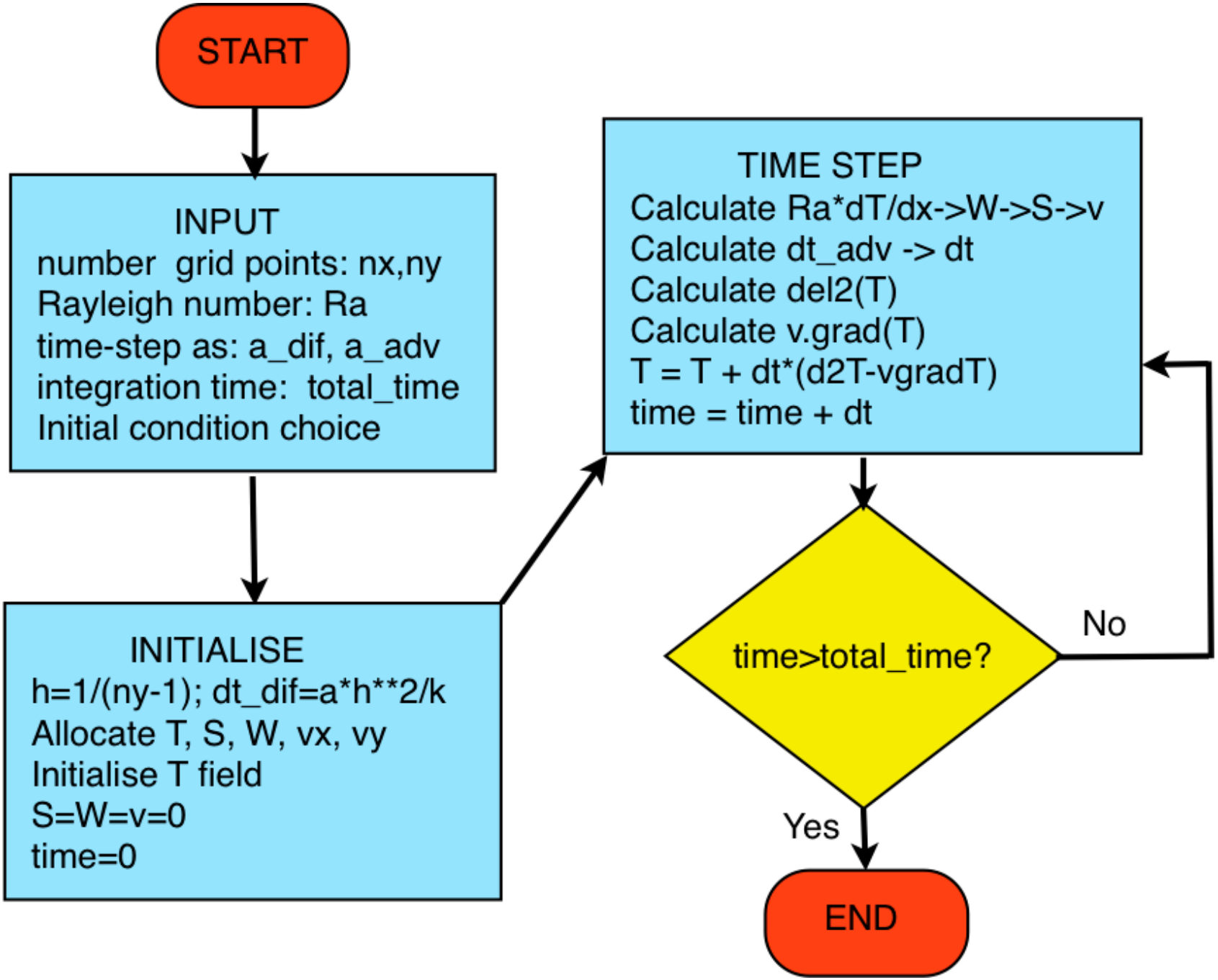
$$\nabla \times \mathbf{F} = \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z} \right) \hat{\mathbf{x}} + \left(\frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x} \right) \hat{\mathbf{y}} + \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \right) \hat{\mathbf{z}}$$

Combine advection-diffusion and Poisson solver routines

- Initialise
  - T=random or cosine(x)
  - S=W=0 (*only to begin with; not during time stepping*)
  - grid spacing h=1/(ny-1) and diffusive timestep
- Timestep until desired time is reached
  - calculate right-hand side=Ra*dT/dx
  - Poisson solve to get W from rhs
  - Poisson solve to get S from W
  - calculate V from S
  - calculate advective timestep and min(adv,diff)
  - take advection and diffusion time step
  - t=t+dt: have we finished? If not, loop again.

Use multigrid solver

Use existing adv-diff routines


START
INPUT
number grid points: nx,ny
Rayleigh number: Ra
time-step as: a_dif, a_adv
integration time: total_time
Initial condition choice
INITIALISE
h=1/(ny-1); dt_dif=a*h**2/k
Allocate T, S, W, vx, vy
Initialise T field
S=W=v=0
time=0
TIME STEP
Calculate Ra*dT/dx->W->S->v
Calculate dt_adv -> dt
Calculate del2(T)
Calculate v.grad(T)
T = T + dt*(d2T-vgradT)
time = time + dt
time>total_time?
No
Yes
END

Program definition

- Input parameters: (red=new things)
  - nx,ny : number of grid points (y=vertical, grid spacing dx=dy=h)
  - Ra: Rayleigh number (typical range 1e3-1e7)
  - total_time: large enough to 'settle down' (steady-state not expected)
  - err: maximum error in Poisson solves, i.e. (residue_rms/rhs_rms<err)
  - a_adv, a_dif: the usual time-stepping parameters
  - Tinit: initial T, "random" or "cosine" T(x)=0.5*(1+cos(3*pi*x/W)), W=width
- Boundary conditions
  - T: as in advdif, T=1 at bottom, 0 at top, dT/dx=0 at sides
  - S and W: 0 at all boundaries
- Example structure
  - Module with Poisson solver routines
  - Module with delsq, v_gradT, S_to_V routines
  - Main program

Exercise

- Get everything together and run two test cases using the given parameter files. **Read the parameter file name as a command-line argument!** (slides 11-12)

Test case 1

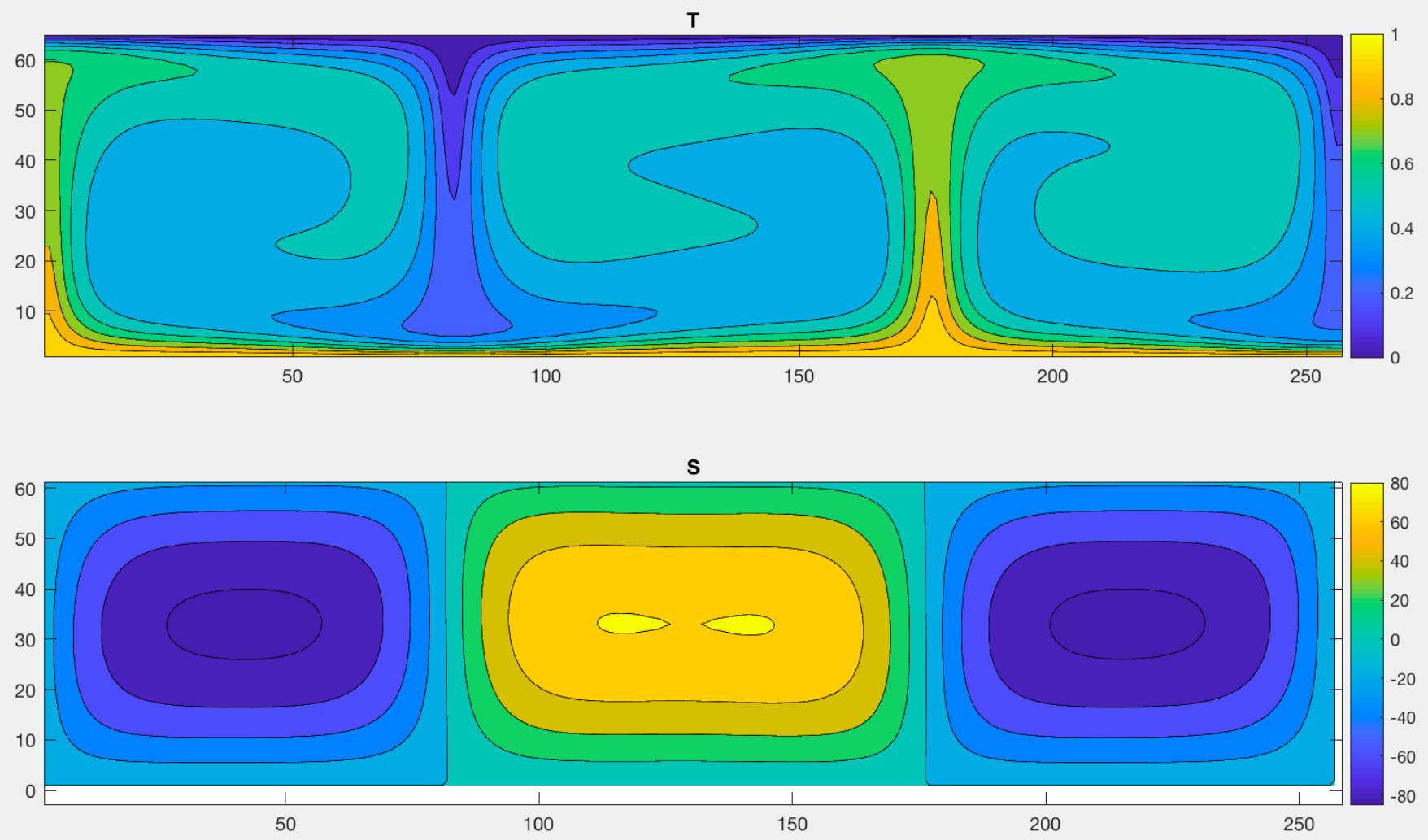
```
&inputs
 ny=65
 total_time=0.1
 Ra=1.e5
 err=1.e-3
 a_dif=0.23
 a_adv=0.4
 nx=257, Tinit='cosine'
 /
```

Test case 2

```
&inputs
 ny=65
 total_time=0.1
 Ra=1.e5
 err=1.e-3
 a_dif=0.23
 a_adv=0.4
 nx=1025, Tinit='random'
 /
```

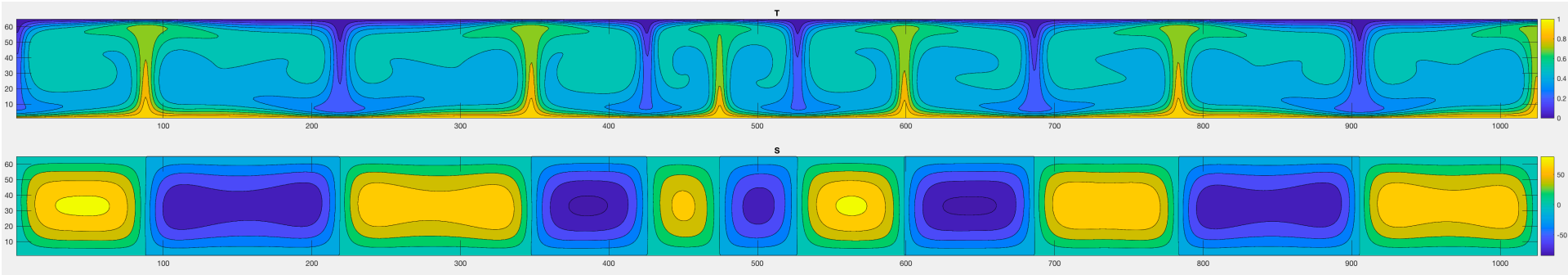
- Email f90 files and a plot of each case.
- Due in 2 weeks: 16.11.2020

Test 1 (width=4, cosine start) should look very similar to:



3 cells, upwelling-downwelling-upwelling-downwelling

Test 2 (width=16, random start) should look qualitatively similar to:



It won't look exactly the same due to the random initial temperature field, but there should be a similar spacing of upwellings & downwellings. It is interesting to see how many convection 'cells' you obtain. Here there are 11.

For bonus points (0.25 each):

- Save the T field at different times during the run and make an animation
- Use a derived variable type to hold all fields and grid information and simplify subroutine arguments, as in Class 5 slides 9-10