Numerical Modelling in **FORTRAN**

day 8

Paul Tackley, 2019
Today‘s Goals

1. Learn about pointers, generic procedures and operators (examples of overloading)
2. Write low-Prandtl number convection program.
Projects: start thinking about

Agree topic with me before final lecture
Registered for project
Project
(optional, 1 KP)

1. Chosen topic, agreed upon with me (suggestions given, also ask the advisor of your MSc or PhD project).
   - Due end of Semesterprüfung (14 Feb 2020)
   - Start planning soon!
Generic procedures
(i.e., using a generic name to access different procedures)

• e.g., the intrinsic (built-in) \texttt{sqrt} function. There are several versions: \texttt{sqrt}(real), \texttt{dsqrt}(double precision), \texttt{csqrt}(complex). Use the generic name and the correct one will automatically be used.

• You can define the same thing. Define similar sets of procedures then define a generic \texttt{interface} to them (see example next slide).

• Easiest way: in a \texttt{module}
generic interface apbxc
to functions rapbxc and iapbxc

A program using this generic function

module goodstuff
    implicit none

    interface apbxc ! define generic interface
        module procedure rapbxc,iapbxc
    end interface

contains

    real function rapbxc(a,b,c) ! actual function
        real,intent(in):: a,b,c
        rapbxc = a+b*c
    end function rapbxc

    integer function iapbxc(a,b,c)
        integer,intent(in):: a,b,c
        iapbxc = a+b*c
    end function iapbxc

end module goodstuff

!---------------------------------
program generic
    use goodstuff
    implicit none
    real:: a=1.2,b=3.4,c=5.6
    integer:: i=2,j=3,k=4

    print*,rapbxc(a,b,c) ! real arguments
    print*,iapbxc(i,j,k) ! integer arguments
end program generic
Overloading

- **Overloading** means that one operator or procedure name is used to refer to several procedures: which one is used depends on the variable types.

- **Examples**
  - `apbxc` is overloaded with `rapbxc` and `iapbxc`
  - `*,+,-,/` are overloaded with integer, real, complex versions in different precisions

- You can overload existing operators, or define new overloaded operators or procedures
Overloading existing + and −

Useful for defined types

```
module coordstuff
  implicit none

  type point  ! defined type
    real:: x,y,z
  end type point

  interface operator (+) ! new version of +
    module procedure pointplus
  end interface

  interface operator (-) ! new version of −
    module procedure pointminus
  end interface

  contains

  function pointplus(a,b)
    type(point):: a,b
    type(point),intent(in):: a,b
    pointplus%x = a%x + b%x
    pointplus%y = a%y + b%y
    pointplus%z = a%z + b%z
  end function pointplus

  function pointminus(a,b)
    type(point):: a,b
    type(point),intent(in):: a,b
    pointminus%x = a%x - b%x
    pointminus%y = a%y - b%y
    pointminus%z = a%z - b%z
  end function pointminus
end module coordstuff
```

```
program test
  use coordstuff
  type(point):: p1,p2,p3
  p1%x=1.2; p1%y=0. ; p1%z=3.1
  p2%x=0. ; p2%y=1.2; p2%z=1.7

  p3 = p1 - p2   ! using overloaded −
  print*,p3
  p3 = p1 + p2   ! using overloaded +
  print*,p3
end program test
```
Defining new operator \texttt{.distance.}.

```fortran
module coordsagain
    implicit none

    type point
        real:: x,y,z
    end type point

    interface operator (.distance.) ! new operator
        module procedure pointseparation
    end interface

    contains

    real function pointseparation(a,b)
        type(point), intent(in):: a,b
        pointseparation = sqrt( &
            (a%x-b%x)**2+(a%y-b%y)**2+(a%z-b%z)**2)
    end function pointseparation

end module coordsagain

!-----------------------------------------
program test
    use coordsagain
    type(point):: p1,p2
    real:: d
    p1%x=1.2; p1%y=0. ; p1%z=3.1
    p2%x=0. ; p2%y=1.2; p2%z=1.7

    d = p1.distance.p2         ! using new operator
    print*,d
end program test
```
Overloading "=". Useful for conversion between different types: in this case point to real.

Must use subroutine with 1st argument intent(out) and 2nd argument intent(in).

```fortran
module coords3
  implicit none
  type point               ! defined type
    real:: x,y,z
  end type point
  interface assignment (=) ! new "="
    module procedure absvec  ! converts
      ! point to real
  end interface
contains
  subroutine absvec(a,b)   ! calculates distance
    real,intent(out):: a    ! from origin
    type(point),intent(in):: b
    a = sqrt(b%x**2+b%y**2+b%z**2)
  end subroutine absvec
end module coords3
--------------------
program test
  use coords3
  type(point):: p1
  real:: d
  p1%x=1.2; p1%y=0. ; p1%z=3.1

  d = p1      ! using new =,
p  print*,d
end program test
```
Combine operators into 1 module

- Generic procedures
- New operators
- Overloaded operators (e.g., +, -, =)
The interfaces

module coords
  implicit none

  type point
    ! defined type
    real:: x, y, z
  end type point

interface apbxc  ! define generic interface
  module procedure raphbc, iapbxc
end interface

interface operator (.distance.) ! new operator
  module procedure pointseparation
end interface

interface operator (+)  ! new version of +
  module procedure pointplus
end interface

interface operator (-)  ! new version of -
  module procedure pointminus
end interface

interface assignment (=)  ! new "="
  module procedure absvec ! converts
    ! point to real
end interface

contains
The procedures

real function rapbxc(a,b,c) ! real version of fn
real,intent(in):: a,b,c
rapbxc = a+b*c
end function rapbxc

integer function iapbxc(a,b,c) ! int version
integer,intent(in):: a,b,c
iapbxc = a+b*c
end function iapbxc

real function pointseparation(a,b) ! for .distance.
type(point),intent(in):: a,b
pointseparation = sqrt( &
    (a%x-b%x)**2+(a%y-b%y)**2+(a%z-b%z)**2)
end function pointseparation

function pointplus(a,b) ! for +
type(point):: pointplus
type(point),intent(in):: a,b
pointplus%x = a%x + b%x
pointplus%y = a%y + b%y
pointplus%z = a%z + b%z
end function pointplus

function pointminus(a,b) ! for -
type(point):: pointminus
type(point),intent(in):: a,b
pointminus%x = a%x - b%x
pointminus%y = a%y - b%y
pointminus%z = a%z - b%z
end function pointminus

subroutine absvec(a,b) ! for = (distance
real,intent(out):: a ! from origin)
type(point),intent(in):: b
a = sqrt(b%x**2+b%y**2+b%z**2)
end subroutine absvec

end module coords
program test
  use coords
  real:: a=1.2,b=3.4,c=5.6,d
  integer:: i=2,j=3,k=4
  type(point):: p1,p2,p3
  p1%x=1.2; p1%y=0. ; p1%z=3.1
  p2%x=0. ; p2%y=1.2; p2%z=1.7

  print*,apbxc(a,b,c)  ! real arguments
  print*,apbxc(i,j,k)  ! integer arguments

  p3 = p1 - p2       ! using overloaded -
  p3 = p1 + p2       ! using overloaded +

  d = p1              ! using overloaded =,

  d = p1.distance.p2 ! using new operator
end program test
Programming:
Finite Prandtl number convection (i.e., almost any fluid)

Ludwig Prandtl (1875-1953)
Values of the Prandt number $Pr$

$$Pr = \frac{\nu}{K}$$

- Viscous diffusivity
- Thermal diffusivity

- Liquid metals: 0.004-0.03
- Air: 0.7
- Water: 1.7-12
- Rock: $\sim 10^{24} !!!$ (effectively infinite)
Finite-Prandtl number convection

• Existing code assumes infinite Prandtl number
  – also known as Stokes flow
  – appropriate for highly-viscous fluids like rock, honey etc.

• Fluids like water, air, liquid metal have a lower Prandtl number so equations must be modified
Applications for finite Pr

- Outer core (geodynamo)
- Atmosphere
- Ocean
- Anything that’s not solid like the mantle
Equations

- Conservation of mass (= ‘continuity’)
- Conservation of momentum (‘Navier-Stokes’ equation: \( F=ma \) for a fluid)
- Conservation of energy

Claude Navier (1785-1836)  
Sir George Stokes (1819-1903)
Finite Pr Equations

Navier-Stokes equation: \( F = ma \) for a fluid

\[
\rho \left( \frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} \right) = -\nabla P + \rho \nu \nabla^2 \vec{v} + 2 \rho \hat{\Omega} \times \vec{v} + g \rho \alpha T \hat{y}
\]

Coriolis force

Valid for constant viscosity only

“ma”

continuity and energy equations same as before

\[
\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T = \kappa \nabla^2 T + Q \quad \nabla \cdot \vec{v} = 0
\]

\( \rho = \) density, \( \nu = \) kinematic viscosity, \( g = \) gravity, \( \alpha = \) thermal expansivity
Non-dimensionalise the equations

- Reduces the number of parameters
- Makes it easier to identify the dynamical regime
- Facilitates comparison of systems with different scales but similar dynamics (e.g., analogue laboratory experiments compared to core or mantle)
Non-dimensionalise to thermal diffusion scales

• Lengthscale $D$ (depth of domain)
• Temperature scale (T drop over domain)
• Time to $\frac{D^2}{\kappa}$
• Velocity to $\frac{\kappa}{D}$
• Stress to $\frac{\rho \nu \kappa}{D^2}$
Nondimensional equations

\[ \nabla \cdot \vec{v} = 0 \]

\[ \frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T = \nabla^2 T \]

\[ \frac{1}{\text{Pr}} \left( \frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} \right) = -\nabla P + \nabla^2 \vec{v} + \frac{1}{\text{Ek}} \vec{\Omega} \times \vec{v} + \text{Ra} \dot{T} \hat{y} \]

\[ \text{Pr} = \frac{\nu}{\kappa} \]

\[ \text{Ek} = \frac{\nu}{2\Omega D^2} \]

\[ \text{Ra} = \frac{g \alpha \nabla TD^3}{\nu \kappa} \]

Prandtl number  \hspace{1cm}  \text{Ekman number}  \hspace{1cm}  \text{Rayleigh number}
As before, use streamfunction

\[ v_x = \frac{\partial \psi}{\partial y} \quad \quad v_y = -\frac{\partial \psi}{\partial x} \]

Also simplify by assuming \( 1/Ek=0 \)
Eliminating pressure

• Take curl of 2D momentum equation: curl of grad=0, so pressure disappears
• Replace velocity by vorticity: \( \vec{\omega} = \nabla \times \vec{v} \)
• in 2D only one component of vorticity is needed (the one perpendicular to the 2D plane), \( \nabla^2 \psi = \omega_z \)

\[
\frac{1}{\text{Pr}} \left( \frac{\partial \omega}{\partial t} + v_x \frac{\partial \omega}{\partial x} + v_y \frac{\partial \omega}{\partial y} \right) = \nabla^2 \omega - Ra \frac{\partial T}{\partial x}
\]
=> the streamfunction-vorticity formulation

\[
\frac{1}{Pr} \left( \frac{\partial \omega}{\partial t} + v_x \frac{\partial \omega}{\partial x} + v_y \frac{\partial \omega}{\partial y} \right) = \nabla^2 \omega - Ra \frac{\partial T}{\partial x}
\]

\[
\nabla^2 \psi = -\omega \quad \left( v_x, v_y \right) = \left( \frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x} \right)
\]

\[
\frac{\partial T}{\partial t} + \vec{v} \cdot \nabla T = \nabla^2 T + Q
\]
Note: Effect of high Pr

\[
\frac{1}{Pr} \left( \frac{\partial \omega}{\partial t} + v_x \frac{\partial \omega}{\partial x} + v_y \frac{\partial \omega}{\partial y} \right) = \nabla^2 \omega - Ra \frac{\partial T}{\partial x}
\]

If Pr->infinity, left-hand-side=>0 so equation becomes Poisson like before:

\[
\nabla^2 \omega = Ra \frac{\partial T}{\partial x}
\]
Taking a timestep

(i) Calculate $\psi$ from $\omega$ using: \[ \nabla^2 \psi = \omega \]

(ii) Calculate $\mathbf{v}$ from $\psi$: \[ (v_x, v_y) = \left( \frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x} \right) \]

(iii) Time-step $\omega$ and $T$ using explicit finite differences:
\[
\frac{\partial T}{\partial t} = -v_x \frac{\partial T}{\partial x} - v_y \frac{\partial T}{\partial y} + \nabla^2 T
\]
\[
\frac{\partial \omega}{\partial t} = -v_x \frac{\partial \omega}{\partial x} - v_y \frac{\partial \omega}{\partial y} + \Pr \nabla^2 \omega - Ra \Pr \frac{\partial T}{\partial x}
\]
T time step is the same as before

\[
\frac{T_{\text{new}} - T_{\text{old}}}{\Delta t} = -v_x \frac{\partial T_{\text{old}}}{\partial x} - v_y \frac{\partial T_{\text{old}}}{\partial y} + \nabla^2 T_{\text{old}}
\]

\[
T_{\text{new}} = T_{\text{old}} + \Delta t \left( \nabla^2 T_{\text{old}} - v_x \frac{\partial T_{\text{old}}}{\partial x} - v_y \frac{\partial T_{\text{old}}}{\partial y} \right)
\]

\[\omega_{\text{new}} = \omega_{\text{old}} + \Delta t \left( \Pr \nabla^2 \omega_{\text{old}} - v_x \frac{\partial \omega_{\text{old}}}{\partial x} - v_y \frac{\partial \omega_{\text{old}}}{\partial y} - Ra \Pr \frac{\partial T_{\text{old}}}{\partial x} \right)\]
START

INPUT
number grid points: nx, ny
Parameters: Ra, Pr
time-step as: a_dif, a_adv
integration time: total_time
Initial condition choice

TIME STEP
Calculate S (from w) -> v
Calculate dt_adv -> dt
Calculate del2(T), del2(w), dT/dx
Calculate v.grad(T) & v.grad(w)
T = T + dt*(d2T-vgradT)
w = w+dt*(Pr.d2w-vgradw-RaPrdTdx)
time = time + dt

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>total_time?

END
Stability condition

Diffusion: \[ dt_{\text{diff}} = a_{\text{diff}} \frac{h^2}{\max(Pr,1)} \]

Advection: \[ dt_{\text{adv}} = a_{\text{adv}} \min \left( \frac{h}{\max \text{val}(abs(vx))}, \frac{h}{\max \text{val}(abs(vy))} \right) \]

Combined: \[ dt = \min(dt_{\text{diff}}, dt_{\text{adv}}) \]
Modification of previous convection program

- Replace Poisson calculation of $w$ with time-step, done at the same time as $T$ time-step
- Get a compiling code!
- Make sure it is stable and convergent for values of $Pr$ between 0.001 and 10
- Hand in your code, and your solutions to the test cases in the following slides
- Due date: 2 December
Test cases

• All have nx=257, ny=65, Ra=1e6, total_time=0.1, Tinit=‘cosine’, initial W=0, unless otherwise stated

• In addition to the fields, plot and hand in a graph of maximum velocity vs. time for each case.
Standard parameter file

On web site, lowPrandt_parameters.txt

&inputs
Pr=0.01 ! <----- CHANGE ONLY THIS
nx=257  ny=65
total_time=0.1
Ra=1.e6
err=1.e-3
a_dif=0.15  a_adv=0.4
Tinit='cosine'
/

Empty parameter file

For testing that you can set default values for Input parameters.
On web site, lowPrandt_parameters_EMPTY.txt
Check for uninitialized variables

- Code should assume sensible values of input parameters, because they are optional in namelist blocks
  - Test using empty namelist block
- All arrays should be initialized (typically to 0)
- Test using recommended debugging options
$Pr=10$
Pr=1

Heat Transfer

Velocity

Stream Function

Graph of $V_{max}$ vs Time
Pr=0.1

The images show the temperature (T), velocity (w), and salinity (s) distributions with contour plots. The graphs below depict the maximum velocity ($V_{max}$) over time.
Pr = 0.01
Hand in

• Source files
• Images and graphs for test cases
• Make sure it compiles with
  – “gfortran --fcheck=all --finit-real=snan --ffpe-trap=invalid,zero,overflow --O0”
• Make sure it runs with both the downloaded 'full' parameter file and the 'empty' parameter file