STRUCTURE OF THE LEEDS DYNAMO CODE

Ashley P. Willis

a.p.willis@shef.ac.uk

Governing equations

$$\begin{aligned}(Ro\,\partial_t - E\,\nabla^2)\,\boldsymbol{u} &= \boldsymbol{N}_u - \boldsymbol{\nabla}\hat{p},\\ (\partial_t - \nabla^2)\,\boldsymbol{B} &= \boldsymbol{N}_B,\\ (\partial_t - q\nabla^2)\,C &= N_C\,.\end{aligned}$$

Toriodal-poloidal decomposition

$$\begin{aligned}\boldsymbol{u} &= \boldsymbol{\nabla}\times(T\boldsymbol{r}) + \boldsymbol{\nabla}\times\boldsymbol{\nabla}\times(P\boldsymbol{r}),\\ \boldsymbol{B} &= \boldsymbol{\nabla}\times(\mathcal{T}\boldsymbol{r}) + \boldsymbol{\nabla}\times\boldsymbol{\nabla}\times(\mathcal{P}\boldsymbol{r}).\end{aligned}$$

Numerical formulation

$$\begin{aligned}(\partial_t - \nabla^2)\mathcal{P} &= N_{\mathcal{P}}\\ (\partial_t - \nabla^2)\mathcal{T} &= N_{\mathcal{T}}\\ (Ro\,\partial_t - E\nabla^2)T &= N_T,\\ (Ro\,\partial_t - E\nabla^2)P &= g\\ -\nabla^2 g &= N_P\\ (\partial_t - q\nabla^2)\,C &= N_C.\end{aligned}$$

Model equation

$$(a\,\partial_t - b\nabla^2)f = N$$

- A lot of code-sharing.
- narrow f.d. stencil or higher-order approximation for a given width.
- good stability.
- Each 2nd-order + 2 BCs, except system

$$\left\{\begin{aligned}(Ro\,\partial_t - E\nabla^2)P &= g\\ -\nabla^2 g &= N_P\end{aligned}\right. \quad (r_i < r < r_o) \qquad P = \partial_r P = 0\ (r = r_i, r_o),$$

which has 4 BCs on P, none on g.

Green's function or influence-matrix method

$$\left\{ \begin{array}{rcl} (Ro\,\partial_t - E\nabla^2)P &=& g \\ -\nabla^2 g &=& N_P \end{array} \right. \quad (r_i < r < r_o) \qquad P = \partial_r P = 0 \; (r = r_i, r_o).$$

Write as system

$$\left\{ \begin{array}{rcl} \mathsf{X}\,P &=& g \\ \mathsf{Y}\,g &=& N \end{array} \right. \qquad P = 0,0 \quad \partial_r P = 0,0 \;\; (r = r_i, r_o).$$

Let $P = \bar{P} + a\,P_i + b\,P_o$, where

$$\left\{ \begin{array}{rcl} \mathsf{X}\,P_i &=& g_i \\ \mathsf{Y}\,g_i &=& 0 \end{array} \right. \qquad \begin{array}{l} \partial_r P_i = 0,0 \\ g_i = 1,0 \end{array}$$

$$\left\{ \begin{array}{rcl} \mathsf{X}\,P_o &=& g_o \\ \mathsf{Y}\,g_o &=& 0 \end{array} \right. \qquad \begin{array}{l} \partial_r P_o = 0,0 \\ g_o = 0,1 \end{array}$$

$$\left\{ \begin{array}{rcl} \mathsf{X}\,\bar{P} &=& \bar{g} \\ \mathsf{Y}\,\bar{g} &=& N \end{array} \right. \qquad \begin{array}{l} \partial_r \bar{P} = 0,0 \\ \bar{g} = 0,0 \end{array}$$

- P_i and P_o both precomputable.
- Only inverted for $\bar{P}$ each timestep.
- P satisfies two BCs automatically: $\partial_r P = 0,0$.
- Scalars a and b set by remaining two BCs: $P = 0,0$.
  (Requires inversion of a 2×2 matrix.)
- BCs on another variable or coupling BCs satisfied to numerical precision.
- Compuatational overhead is nominal.

Modules

Somewhere to gather together related data and functions that work on that data. Don't worry about syntax here!

```
!*****************************************************************
 module rotation
!*****************************************************************
   use parameters, timestep       ! use data from other modules
   implicit none
   save
                                  ! data other mods can inherit
   double precision :: rot_omega
   double precision :: rot_torque
                                                 ! private data
   double precision, private :: rot_inertia

 contains
!-----------------------------------------------------------------
!  initialise
!-----------------------------------------------------------------
   subroutine rot_precompute()
      rot_omega   = 0d0
      rot_inertia = (8d0*d_PI/15d0) * d_ICradius**5
   end subroutine rot_precompute

!-----------------------------------------------------------------
!  timestep
!-----------------------------------------------------------------
   subroutine rot_predictor()
      rot_omega   = rot_omega  &
         + (tim_dt/rot_inertia) * rot_torque
   end subroutine rot_predictor

!*****************************************************************
 end module rotation
!*****************************************************************
```

Modules:

codensity, (IC) rotation, velocity, magnetic	these variables stored here + predictor-corrector functions
timestep	functions for setting up matrices for model eqn $(a\partial_t - b\nabla^2)\, f = N$.
nonlinear	evaluate nonlinear (coupling) terms N.
mpi	`mpi_rnk`, `mpi_sze`
mesh	selection of radial points r_n
legendre	weights for transforms.
variables	definition of data types `(coll)`, `(spec)`, `(phys)`
transform	between spherical harmonic coeffs and data at points in physical space.

The 'main' program:

parameters	physical and numerical parameters
main	the main time-stepping loop
io	input / output

A 'derived' data type:

Define type:

```
type phys                                  !theta,phi,r
   double precision :: Re(10, 0:11, 5)
end type phys
```

Declare a variable: `type (phys) :: p`

Set value of an element: `p%Re(j,i,n) = 1d0`

- No need to specify the dimension of every variable with the same size
- Very handy when passing between functions.
- Fortran column dominant – keep 'close' data in first index, split over last index...

Legendre transform strongly linked to data types

$$A(\theta, \phi) = \sum_{m=0}^{M-1} \sum_{l=m}^{L-1} A_{lm} \, \hat{Y}_l^m(\theta, \phi)$$

$$A(r_n, \theta, \phi) = \sum_m \mathrm{e}^{im\phi} \sum_l A_{lmn} \, P_l^{|m|}(\cos\theta)$$

$$A(r_n, \theta_j, \phi) = \sum_m \mathrm{e}^{im\phi} A_{jmn}$$

$$A(r_n, \theta_j, \phi_i) = A_{jin}$$

`(coll)` $(A_n)_{lm}$ Data at collocation point for each harmonic.
All n on same CPU (split accross modes lm)

$\rightarrow$ transpose $\rightarrow$

`(spec)` $(A_{lm})_n$ All spectral coeffs on same CPU (split accross radial pts)

sum over l at each θ_j $\rightarrow (A_{jm})_n$
sum over m at ϕ_i (FFT) $\rightarrow$

`(phys)` $(A_{ij})_n$ data evaluated on points in physical space.

Strengths

- Short – 3000 lines excluding IO. Readable – modular, function-based.
- Data easily manipulated – function-based, netCDF.
- Fast in serial.
- Timestep control (improved by Chris Jones).
- Parallelised, linear scaling with number of CPUs.

Weakness

- Number of CPUs currently limited to number of radial points.
  (See index n on `(spec)`)

Possible way out:

$(A_n)_{lm}$; (transpose) $\rightarrow (A_l)_{mn}$; (sum over l) $\rightarrow (A_j)_{mn}$;
(transpose) $\rightarrow (A_m)_{jn}$; (FFT) $\rightarrow (A_i)_{jn}$.

The ‘derived’ types:

```
type spec
   double precision     :: Re(0:i_H1, i_pN)
   double precision     :: Im(0:i_H1, i_pN)
end type spec

type coll
   double precision     :: Re(i_N, 0:i_pH1)
   double precision     :: Im(i_N, 0:i_pH1)
end type coll

type phys
   double precision     :: Re(i_Th, 0:i_Ph-1, i_pN)
end type phys
```

Currently split over radial points, $n \in [1, N]$,
and harmonics lm mapped to a single index $nh \in [0 : H - 1]$.

```
nh  = -1
do m = 0, M-1
   do l = m, L-1
      nh = nh + 1
```