MANUAL FOR MODEL WALEFFE FLOW CODE

Code by Mat Chantry

Manual by Laurette Tuckerman

Model Waleffe Flow

We use a mean-poloidal-toroidal representation

$$\mathbf{u} = f(y)\mathbf{e}_x + g(y)\mathbf{e}_z + \nabla \times \psi(x,y,z)\mathbf{e}_y + \nabla \times \nabla \times \phi(x,y,z)\mathbf{e}_y,$$

where ψ and ϕ are periodic in x and z.

$$\begin{aligned}
\nabla \times \psi(x,y,z)\mathbf{e}_y &= \partial_x\psi\mathbf{e}_z - \partial_z\psi\mathbf{e}_x \\
\nabla \times \nabla \times \phi(x,y,z)\mathbf{e}_y &= \nabla(\nabla \cdot \phi\mathbf{e}_y) - \nabla^2(\phi\mathbf{e}_y) = \nabla(\partial_y\phi) - (\nabla^2\phi)\mathbf{e}_y \\
&= \nabla_h(\partial_y\phi) + \partial_y^2\phi\mathbf{e}_y - (\nabla^2\phi)\mathbf{e}_y = \nabla_h(\partial_y\phi) - (\nabla_h^2\phi)\mathbf{e}_y
\end{aligned}$$

where $\nabla_h^2 \equiv \partial_x^2 + \partial_z^2$ and $\nabla_h \equiv \mathbf{e}_x\partial_x + \mathbf{e}_z\partial_z$.

$$\begin{aligned}
u &= -\partial_z\psi(x,y,z) + \partial_{xy}\phi(x,y,z) + f(y) \\
v &= -\nabla_h^2\phi(x,y,z) \\
w &= \partial_x\psi(x,y,z) + \partial_{yz}\phi(x,y,z) + g(y)
\end{aligned}$$

Note that ∂_x and ∂_z cancel any (x,z)-independent components in ψ and ϕ. For this reason, the inclusion of mean fields f and g is necessary to represent such components. See, for example, F. Marquès, Phys. Fluids A **2**, 729 (1990).

We use manipulations similar to the above to obtain

$$\begin{aligned}
\mathbf{e}_y \cdot \nabla \times \mathbf{u} &= \mathbf{e}_y \cdot \nabla \times \nabla \times \psi\mathbf{e}_y = -\nabla_h^2\psi \\
\mathbf{e}_y \cdot \nabla \times \nabla^2\mathbf{u} &= \mathbf{e}_y \cdot \nabla \times \nabla^2\nabla \times \psi\mathbf{e}_y = -\nabla^2\nabla_h^2\psi \\
\mathbf{e}_y \cdot \nabla \times \nabla \times \mathbf{u} &= \mathbf{e}_y \cdot \nabla \times \nabla \times \nabla \times \nabla \times \phi\mathbf{e}_y = \nabla^2\nabla_h^2\phi \\
\mathbf{e}_y \cdot \nabla \times \nabla \times \nabla^2\mathbf{u} &= \mathbf{e}_y \cdot \nabla \times \nabla \times \nabla^2\nabla \times \nabla \times \phi\mathbf{e}_y = \nabla^2\nabla^2\nabla_h^2\phi
\end{aligned}$$

Starting from the Navier-Stokes equations

$$\partial_t\mathbf{u} = -\nabla p + \mathbf{NL} + \tfrac{1}{Re}\nabla^2\mathbf{u}$$

we apply $\mathbf{e}_y \cdot \nabla\times$ and $\mathbf{e}_y \cdot \nabla \times \nabla\times$ to obtain evolution equations for $\nabla_h^2\psi$ and $\nabla^2\nabla_h^2\phi$:

$$\begin{aligned}
\left(\partial_t - \tfrac{1}{Re}\nabla^2\right)\nabla_h^2\psi &= \mathbf{e}_y \cdot \nabla \times \mathbf{NL} \\
\left(\partial_t - \tfrac{1}{Re}\nabla^2\right)\nabla^2\nabla_h^2\phi &= -\mathbf{e}_y \cdot \nabla \times \nabla \times \mathbf{NL}
\end{aligned}$$

Evolution equations for f and g are obtained by taking the mean $\langle\rangle_{x,z}$:

$$\partial_t(f\mathbf{e}_x + g\mathbf{e}_z) = \langle\mathbf{NL}\rangle_{x,z} + \tfrac{1}{Re}\left(\partial_y^2 f\mathbf{e}_x + \partial_y^2 g\mathbf{e}_z\right)$$

Normally, boundary conditions would be required in the non-periodic direction y: two for ψ and four for ϕ, the degree in y of their governing equations. These would be supplied by the six boundary conditions on u, v, w. The free-slip conditions $\partial_y u = \partial_y w = v = 0$ at $y = \pm 1$ required by Waleffe flow are satisfied by the following trionometric expansion, where $\beta \equiv \pi/2$.

$$\begin{aligned}
u(x,y,z) &= u_0(x,z) + u_1(x,z)\sin(\beta y) + u_2(x,z)\cos(2\beta y) + u_3(x,z)\sin(3\beta y), \\
v(x,y,z) &= v_1(x,z)\cos(\beta y) + v_2(x,z)\sin(2\beta y) + v_3(x,z)\cos(3\beta y), \\
w(x,y,z) &= w_0(x,z) + w_1(x,z)\sin(\beta y) + w_2(x,z)\cos(2\beta y) + w_3(x,z)\sin(3\beta y),
\end{aligned}$$

The functions ψ, f, g match the y-formulation of u, while v matches that of v.

Waleffe flow, has an imposed force:

$$\mathbf{F} \equiv V/Re(pi/2H)^2 \sin(\pi y/2H)\mathbf{e}_{\rm strm}$$

The laminar solution is:

$$\mathbf{u}_{\rm lam} \equiv V \sin(\pi y/2H)\mathbf{e}_{\rm strm}$$

Thus, the viscous term

$$\frac{1}{Re}\nabla^2\mathbf{u}_{\rm lam} = -\frac{V}{Re}\left(\frac{\pi}{2H}\right)^2 \sin(\pi y/2H)\mathbf{e}_{\rm strm}$$

counterbalances the imposed force and neither have to be included in the equations.

The laminar solution $\mathbf{u}_{\rm lam}$ has the same functional form in y as the u_1 and w_1 terms and so is added to their (0,0) Fourier modes before calculating the nonlinear term:

$$(\mathbf{u} + \mathbf{u}_{\rm lam}) \cdot \nabla(\mathbf{u} + \mathbf{u}_{\rm lam})$$

Data Types MPT, Spec, Phys

Type: MPT – Mean-Poloidal-Toroidal (complex)

$$\psi(x,y,z) = \sum_{k=0}^{K_0-1} \sum_{m=-MM+2}^{MM-1} \sum_{n=0}^{NN-1} \mathbf{MPT}[k+1, m \bmod (2*(MM-1)), n]\, h_{k,0}(y)\, e^{i(m\alpha x + n\gamma z)}$$

$$\phi(x,y,z) = \sum_{k=1}^{K_0-1} \sum_{m=-MM+2}^{MM-1} \sum_{n=0}^{NN-1} \mathbf{MPT}[K_0+k, m \bmod (2*(MM-1)), n]\, h_{k,1}(y)\, e^{i(m\alpha x + n\gamma z)}$$

$$f(y) = \sum_{k=0}^{K_0-1} \mathbf{MPT}[k+1, 0, 0)\, h_{k,0}(y) \qquad g(y) = \sum_{k=0}^{K_0-1} \mathbf{MPT}[K_0+k, 0, 0)\, h_{k,1}(y)$$

where $h_{k,j}(y) = \begin{cases} \cos(k\beta y) \text{ if } k+j \text{ is even} \\ \sin(k\beta y) \text{ if } k+j \text{ is odd} \end{cases}$

according to the expansions on the previous page. Note the different k values of **MPT** for the different components: the total number of k values is $K_0 + K_0 - 1 = 2K_0 - 1 \equiv K$.
Example: with $K_0 = 4$ (modes 0, 1, 2, 3 for ψ), we have $K_0 - 1 = 3$ (modes 1, 2, 3, for ϕ) and $K \equiv 2K_0 - 1 = 7$. Hence $K_0 + K_0 - 1 = 2K_0 - 1 = 7$.

Since only x and z derivatives of ψ and ϕ are used, modes $(m, n) = (0, 0)$ are irrelevant and so can be used for storing the mean fields f and g.

The wavenumbers are multiples m and n of $\alpha \equiv 2\pi/L_x$ and $\gamma \equiv 2\pi/L_z$. Since $0 \le x < L_x$ and $0 \le z < L_z$, we have $0 \le \alpha x < 2\pi$ and $0 \le \gamma z < 2\pi$.

The x Fourier coefficients stored are $-MM + 2 \le m \le MM - 1$. Coefficients $m = 0, 1, \ldots MM - 1, -MM + 2, \ldots -1$ are stored in locations $0 \ldots M1 \equiv M - 1 \equiv 2*(MM-1) - 1$ of **MPT**. Wavenumber m is stored in $m \bmod (2*(MM-1))$, i.e. m for $m \ge 0$ and $2*(MM-1) - m$ if $m < 0$.
Example: $MM = 8$ so $M1 = 13$. Wavenumbers $0, 1, \ldots 7, -6, \ldots -1$ are stored in locations $0, 1, ...7, 8, \ldots 13$.

The z Fourier coefficients stored are $0 \le n \le NN - 1$, stored in the same locations of **MPT**. Coefficients $(m, -n)$ are the complex conjugates of coefficients $(-m, n)$, so only $n \ge 0$ is stored.

Type Spec – Velocity (complex)

$$u(x,y,z) = \sum_{k=0}^{K_0-1} \sum_{m=-MM+2}^{MM-1} \sum_{n=0}^{NN-1} \mathbf{Spec}[k+1, m \bmod (2*(MM-1)), n]\, h_{k,0}(y)\, e^{i(m\alpha x + n\gamma z)}$$

$$v(x,y,z) = \sum_{k=1}^{K_0-1} \sum_{m=-MM+2}^{MM-1} \sum_{n=0}^{NN-1} \mathbf{Spec}[K_0+k, m \bmod (2*(MM-1)), n]\, h_{k,1}(y)\, ^{i(m\alpha x + n\gamma z)}$$

$$w(x,y,z) = \sum_{k=0}^{K_0-1} \sum_{m=-MM+2}^{MM-1} \sum_{n=0}^{NN-1} \mathbf{Spec}[2*K_0+k, m \bmod (2*(MM-1)), n]\, h_{k,0}(y)\, e^{i(m\alpha x + n\gamma z)}$$

Note the different subscripts on **Spec** for the different components u, v, and w: the number of k elements in **Spec** is $K_0 + K_0 - 1 + K_0 = 3K_0 - 1 \equiv KK$.

Type Phys – Velocity (real)

$$u(x_m,y,z_n) = \sum_{k=0}^{K_0-1} \mathbf{Phys}[k+1, m, n]\, h_{k,0}(y)$$

$$v(x_m,y,z_n) = \sum_{k=1}^{K_0-1} \mathbf{Phys}[K_0+k, m, n]\, h_{k,1}(y)$$

$$w(x_m,y,z_n) = \sum_{k=0}^{K_0-1} \mathbf{Phys}[2*K_0+k, m, n]\, h_{k,0}(y)$$

Grid: $x_m = L_x \dfrac{m}{3*MM}, \quad m = 0, \ldots 3*MM-1 \qquad z_n = L_z \dfrac{n}{3*NN}, \quad n = 0, \ldots 3*NN-1$

The y direction is still in spectral space, containing coefficients of trigonometric functions.
Example: If $MM = 8$, then there are $3M \equiv 3*MM = 24$ gridpoints in x, compared to the $M \equiv 2*(MM-1) = 14$ spectral coefficients.

The basis for this is the standard 3/2 de-aliasing procedure.
Multiplication of two Fourier series $|m| \le MM$ will yield a Fourier series with $|m| \le 2*MM$.
Coeffs $|m| \le MM$ will be correct if the original functions are:
–padded with zeros to a series with $|m| \le 3*MM/2$,
–transformed to a physical representation with $3*MM$ gridpoints,
–multiplied together,
–transformed back to a Fourier series with $|m| \le 3*MM/2$,
–truncated to a series with the original length $|m| \le MM$.
(We have not worried about MM vs $MM-1$ vs $MM-2$ in this explanation.)

Standalone utility programs in directory util

• **doubleX**

Doubles the domain in the x direction by padding in Fourier space. Adds noise.

• **interpstate**

Linearly interpolates between two **MPT** fields, with scalar d controlling the amount of each.

• **quadX**

Quadruples the domain in the x direction by padding in Fourier space. Adds noise and saves field and spectrum.

• **seriesvtk**

Wrapper for **io_writevtk_xz**. Inputs a sequence of **MPT** files **statexxxx.cdf.dat** and outputs VTK files **XZxxxx.vtk** to be visualized using Paraview. If **mpi_sze == 1**, specify interactively initial and final state number and sampling rate in x and z. Otherwise, read these from file called **series.in**.

• **turbfrac**

Wrapper for **io_writevtk_txz**. For a series of saved files **statecnum.cdf.dat**, makes corresponding files of turbulent kinetic energy averaged over y. Interactively specify initial and final state number, the sampling rate in x and z and the threshold for turbulence.

• **wholevtk**

Wrapper for **io_writevtk**. Inputs a **MPT** files **statexxxx.cdf.dat** and outputs VTK files **XYZxxxx.vtk** to be visualized using Paraview. Interactively specify state number and sampling rate in x and z.

Note: here turbulent kinetic energy means the kinetic energy of the deviation from laminar flow, not the deviation from the temporal mean.

io.f90

• **subroutine io_precompute()**

Initializes to zero **io_save1**, which labels the next field to be saved. Initializes to zero **io_save2**, which labels the next energy, history, and turbulence files to be saved. **io_KE** specifies Fortran unit into which the total kinetic energy is saved (every **i_save_rate2** timesteps). If **io_KE**=0, do not save. **io_hi** specifies Fortran unit into which the the history lines are saved (every **i_save_rate2** timesteps). If **io_hi**=0, do not save. Logical variable **s_HIS** specifies whether to save thresholded turbulence files.

• **subroutine io_openfiles()**

Opens files **vel_energy.dat** and **vel_history.dat** if energy and history are to be saved.

• **subroutine io_closefiles()**

Closes files **vel_energy.dat** and **vel_history.dat**.

• **subroutine io_clk_time(t)**

For timing.

• **subroutine io_write2files()**

Writes the state and the spectrum if **tim_step** is a multiple of **i_save_rate1** and updates **io_save1**. Writes the energy, the history and the turbulence files if **tim_step** is a multiple of **i_save_rate2** and updates **io_save2**. Close and open files if energy, history and turbulence files have already been written to 50 times. (Why?)

• **subroutine io_load_state()**

Use **netcdf** command **nf90_open** to open **io_statefile**. If errors occurred (file does not exist), then **nf90_open** does not return the value **nf90_noerr** and program stops with error message. Read and print the values of time **tim_t**, Reynolds number **Re**, x-wavenumber α such that $L_x = 2\pi/\alpha$, z-wavenumber γ such that $L_z = 2\pi/\gamma$. Also print the values specified in file **parameters.f90**, which are those which will actually be used in the calculation except if **d_time** ¡ 0, in which case **tim_t in** Load **MPT** field by calling subroutine **io_load_mpt**. Initial condition is then in **vel_c**.

• **subroutine io_load_mpt(f,nm, a)**

Check presence of field **nm** in **netcdf** file **f**. Check dimensions. Fill **MPT** type field **a**.

• **subroutine io_save_state()**

Create file called **state.io_save1.cdf.dat** (uses value of **io_save1** Write time, Reynolds number, α, γ, NN, M, K, 2, where $M = 2 * (MM - 1)$ is the total number of x wavenumbers (positive, zero, and negative, NN is the number of z wavenumbers, K is the total number of y-functions in an **MPT** field containing ψ and ϕ, and 2 corresponds to the fact that the field elements have both real and imaginary parts. Calls **io_define_mpt** to set up file of type **MPT**. Calls **io_save_mpt** write the data to the file.

• **subroutine io_define_mpt(f,nm,dims, id)**

Sets up **MPT** fields for the **netcdf** dataset **f** whose type is **nm**='mpt'. **dims** is a four-vector containing (K, M, NN, 2) which are the dimensions of a **MPT** field. id is output

• **subroutine io_save_mpt(f,id,a)**

Saves real and imaginary parts of **MPT** field **a** to **netcdf** file identified by **f** and **id**.

• **subroutine io_save_Uspec()**

Convert mean-poloidal-toroidal **MPT** field **vel_c** to u,v,w representation in (x, z)-Fourier space by calling **var_mpt2spec**. Maximize spectral components over n, k (for each m) and over m, k (for each n). Open file **vel_spec io_save1.dat** and print m and n spectral list. This information can be used to check the resolution or the general aspect of the spectrum of the field.

• **subroutine io_save_spectrum()**

Seems the same as **io_save_Uspec**.

• **subroutine io_write_energy(sp)**

Calls **vel_energy(sp,E)** to calculate energy **E** of field **sp** and writes line in KE file.

• **subroutine io_write_history(sp)**

Calls **vel_history(sp,0,H)** to calculate history points **H(4,i_H)** and writes lines in history file.

• **Subroutine io_writeVTK_xz(V,y,cnum,xs,zs)**

Write out field at specified y value in VTK format for Paraview visualization. **V** is the field in **Phys** format: physical (x, z) and trig-y. The output file is named **XZcnum.vtk**. Data is output at every **xs, zs** values, x and z, where there are $3M \equiv 3 * MM$ and $3N \equiv 3 * NN$ points.

• **Subroutine io_writeVTK_txz(V,cnum,xs,zs,ct)**

Uses function **epos** to calculate turbulent kinetic energy integrated over y at every **xs, zs** values in x and z, where there are $3M \equiv 3 * MM$ and $3N \equiv 3 * NN$ points. The output file is named **TXZcnum.vtk**. Repeat the $ix = 0$ and $iz = 0$ rows for periodicity. Count up the (x, z) values for which turbulent kinetic energy exceeds threshold value **ct**, divide by total number of values and print out.

• **Subroutine io_writeVTK(V,cnum,xs,zs)**

Write entire 3D field with N_y (specified in file **parameters.f90**) equally spaced points in x, as well as $3 * MM$ points in x and $3 * NN$ points in z. The output file is named **XYZcnum.vtk**.

velocity.f90

• **subroutine vel_TS ()**

One timestep. Call **vel_nonlinear** to create nonlinear term **vel_nl**. If first timestep (i.e. **nst** true) copy into **vel_onl** and set **nst** to false. Carry out one CNAB (Crank-Nicolson-Adams-Bashforth) time step:

$$vel_{c2} = lhs^{-1}\left[rhs * vel_c + (dt/2)\left(3 * vel_{nl} - vel_{onl}\right)\right]$$

Copy **vel_c2** into **vel_c** and **vel_nl** into **vel_onl**.

• **subroutine vel_precompute()**

• Set up the elliptic operators **rhs** and **lhs** via subroutine **var_LHSRHS**. Set **vel_c** and **vel_nl** to zero via subroutine **var_mpt_init**, set **nst** to true.

• **subroutine vel_clk_time(t)**

For timing.

• **subroutine vel_imposesym()**

Acts on **vel_c**. Imposes any of three types of reflection symmetry, depending on values of logical variables **s_reflect, s_uvreflect, s_wreflect**.

• **subroutine vel_addlam(u)**

Add constant $\cos(\theta)$ to $\mathbf{Spec}(2,0,0)$, which is the coefficient of $h_{1,0} = \sin(\beta y)$ of u.
Add constant $\sin(\theta)$ to $\mathbf{Spec}(2 * K_0 - 1, 0, 0)$, which is the coefficient of $h_{1,0} = \sin(\beta y)$ of w.

• **subroutine vel_mpt2phys(u,p)**

Input u in **MPT** space $\rightarrow$ output p in **Phys** space by calling **var_mpt2spec** and **tra_spec2phys**.

• **subroutine vel_nonlinear()**

Transform **vel_c**=$(\psi, \phi) \Longrightarrow u = (u, v, w)$ via **var_mpt2spec**. Add laminar flow with **vel_addlam** Construct ∂_x and ∂_z of each of (u, v, w) using **var_spec_grad**. Transform (u, v, w), $\partial_x(u, v, w)$ and $\partial_z(u, v, w)$ into physical space using **tra_spec2phys**. Calculate nonlinear term in (x, z)-physical and y-trig space via function `udotgradu`. Transform back from **Phys** to **MPT** space. using **tra_phys2spec** and **var_spec2mpt**.

• **subroutine vel_energy(a,e)** Calculates turbulent kinetic energy of **Phys** field **a** by calling function **epos** for each (x, z) value, finally dividing by 3M*3N.

• **subroutine vel_history(V,y,ans)**

Input **Phys** field **V**, number **i_H** of history points, y coordinate. For each (x, z) physical point, uses functions **velU, velV, velW** to evaluate the trigonometric form at the specified y value. Returns **ans(4,i_H)** containing (z, u, v, w) at $x = 0$, input y, and z equally spaced points $ji = 1, 1 + i_{3N}/i_H, 1 + 2i_{3N}/i_H, \ldots, 1 + (i_H - 1)i_{3N}/i_H$.
Example: $i_H = 8$, $i_{3N} = 96$ leads to $ji = 1, 13, 25, 37, 49, 61, 73, 85$.

• **function epos(u)**

Use functions **nluw, nlvv, nluw** to take inner product of (u, v, w) with itself, where (u, v, w) is a vector of y-trig coefficients.

• **function eposC**

Use functions **nluw, nlvv, nluw** to take inner product of (u, v, w) with itself, where (u, v, w) is a vector of y-trig coefficients, outputing the three contributing components u^2, v^2, w^2 separately.

variables.f90

• **subroutine var_uvreflect(c)**

Imposes symmetry $(u, v, w)(-x, -y, z) = (-u, -v, w)(x, y, z)$ (?)

• **subroutine var_wreflect(c)**

Imposes symmetry $(u, v, w)(x, y, -z) = (u, v, -w)(x, y, z)$ (?)

• **subroutine var_reflect(c)**

• **subroutine var_maskmpt(c)**

Multiply elements (k, m, n) of ψ, ϕ, f, g by $0.005 \times 10^{-\sqrt{m^2+n^2}/3}$

• **subroutine var_mpt2spec(mp,s)**

Transforms a **MPT** field (ψ, ϕ, f, g) to **Spec** field (u, v, w) via

$$
\begin{aligned}
u_{k,m,n} &= -in\gamma\ \psi_{k,m,n} + ((-1)^k k\beta)(im\alpha)\ \phi_{k,m,n} + f_k \delta_{m,0}\delta_{n,0} \\
v_{k,m,n} &= ((\alpha m)^2 + (\gamma n)^2)\ \phi_{k,m,n} \\
w_{k,m,n} &= im\alpha\ \psi_{k,m,n} + ((-1)^k k\beta)(in\beta)\ \phi_{k,m,n} + g_k \delta_{m,0}\delta_{n,0}
\end{aligned}
$$

• **subroutine var_spec2mpt(s,mp)**

Transforms nonlin term **s** calculated in **Spec** format in **subroutine vel_nonlinear()** to **MPT** field **mp** involving $\nabla_h^2\psi$ and $\nabla_h^2\phi$ needed in subroutine **vel_TS**.

• **subroutine var_precompute()**

Distribute modes across processors. Set up the differentiation matrices

$$
\begin{array}{lll}
ad_{m1}(m) = \alpha m & ad_{n1}(n) = \gamma n & ad_{k1}(k) = (-1)^k k\beta \\
ad_{m2}(m) = -(\alpha m)^2 & ad_{n1}(n) = -(\gamma n)^2 & ad_{k2}(k) = -(k\beta)^2
\end{array}
$$

where we recall that wavenumber m is stored in location m for $m \geq 0$ and location $M - m$ if $m < 0$.

• **subroutine var_LHSRHS(l,r)**

Constructs the differential operators ∇^2, ∇_h^2, $(I - dt\nabla^2/Re)$ and products of these, storing them in $\mathbf{l}, \mathbf{r}$. Can include hypoviscosity and drag if requested.

• **subroutine var_spec_grad(p, ux,uz)**

From field **p**=(u, v, w) in **Spec** format, compute $\partial_x(u, v, w)$ and $\partial_z(u, v, w)$.

• **subroutine var_printmpt(c)**	For debugging of **MPT** field type?
• **subroutine var_printmpt_all(c)**	More debugging of **MPT** field type?
• **subroutine var_printphys(c)**	For debugging of **Phys** field type?
• **subroutine var_printtran(c)**	For debugging of **Tran** field type?
• **subroutine var_printspec(c)**	For debugging of **Spec** field type.
• **subroutine var_randspec(c)**	Fills **Spec** array with random numbers.
• **subroutine var_randmpt(c)**	Fills **MPT** field with random numbers.
• **subroutine var_simplempt(c)**	Fills some elements of **MPT** field with ± 1
• **subroutine var_mpt_init(a)**	Sets **MPT** field to zero.
• **subroutine var_spec_init(a)**	Sets **Spec** array `a` to zero.
• **subroutine var_phys_init(a)**	Sets **Phys** array `a` to zero.
• **subroutine var_mpt_copy(in, out)**	Copies **MPT** field `in` to **MPT** field `out`.
• **subroutine var_mpt_nonzero(in,in2)**	Print if $k = 1, m = 0$ elements of **in** and `in2` differ.
• **subroutine var_mpt_add(ac, a)**	Adds **MPT** field `ac` to **MPT** field `a`.
• **subroutine var_mpt_sub(ac, a)**	Subtracts **MPT** field `ac` from **MPT** field `a`.
• **subroutine var_spec_copy(in, out)**	Copies **Spec** array `in` to **Spec** array `out`.
• **subroutine var_spec_add(ac, a)**	Adds **Spec** array `ac` to **Spec** array `a`.
• **subroutine var_spec_sub(ac, a)**	Subtracts **Spec** array `ac` from **Spec** array `a`
• **subroutine var_null()**	Null function.

modes.M4.f90 and modes.M6.f90: treat $K_0 = 4$ and $K_0 = 6$ trigonometric expansions

• **function udotgradu**

Takes $KK = 3K_0 - 1$ y-trigonometric coefficients of (u, v, w), $\partial_x(u, v, w)$, $\partial_z(u, v, w)$ and produces y-trigonometric representation of

$$\mathbf{NL} \equiv \begin{cases} u\partial_x u + v\partial_y u + w\partial_z u \\ u\partial_x v + v\partial_y v + w\partial_z v \\ u\partial_x w + v\partial_y w + w\partial_z w \end{cases}$$

• **function udotgradu2**

I think that this is the same, but uses intermediate functions **nluw, nluyv, nluv, nlvvy, nlvv**.

• **function nluw**

Input two sets of y-trig coefficients $u(0 : i\_K1)$ and $w(0 : i\_K1)$ at physical (x, z) point, and form y-trig coefficients of the product $u * w$ at that (x, z) point. Can also be used for $u * u$ or $w * w$.

• **function nluyv**

Input two sets of y-trig coefficients $u(0 : i\_K1)$ and $v10 : i\_K1)$ at physical (x, z) point, and form y-trig coefficients of the product $u_y v$ at that (x, z) point. Can also be used for $w_y v$.

• **function nluv**

Input two sets of y-trig coefficients $u(0 : i\_K1)$ and $v(1 : i\_K1)$ at physical (x, z) point, and form y-trig coefficients of the product $u * v$ at that (x, z) point. Can also be used for $v * w$.

• **function nlvvy**

Input one set of y-trig coefficients $v(1 : i\_K1)$ at physical (x, z) point, and form y-trig coefficients of the product $y_y v$ at that (x, z) point.

• **function nlvv** Input set of y-trig coefficients $v(1 : i\_K1)$ at physical (x, z) point, and form y-trig coefficients of the product $v * v$ at that (x, z) point.

• **function velU**

Given a value of y and $V(i\_KK)$, a set of y-trig coefficients for (u, v, w) at a single value of (x, z), evaluates trigonometric expression for u at y.

• **function velV**

Given a value of y and $V(i\_KK)$, a set of y-trig coefficients for (u, v, w) at a single value of (x, z), evaluates trigonometric expression for v at y.

• **function velW**

Given a value of y and $V(i\_KK)$, a set of y-trig coefficients for (u, v, w) at a single value of (x, z), evaluates trigonometric expression for w at y.

• **subroutine tra_precompute**

Setup plans for 2d transforms.
plan_inp2mid is plan for transforming in x/m direction from Fourier to physical.
plan_mid2inp is plan for transforming in x/m direction from physical to Fourier.
Size of these transforms is $3 * MM$, number of transforms is KK.

plan_mid2phy for transforming in z/n direction from Fourier to physical, complex to real
plan_phy2mid for transforming in z/n direction from physical to Fourier, real to complex
Size of these transforms is $3 * NN$, number of transforms is KK.

• **subroutine tra_spec2phys(s, p)**

Convert spectral to physical space
Transform in x/m. For each n, fill larger array in m, transform over m by calling **dfftw_execute(plan_inp2mid)**. Call **tra_T2Ts** to place x on right. For each m, transform over n by calling **dfftw_execute(plan_inp2mid)**.

• **subroutine tra_spectest(s)**

For testing

• **subroutine tra_phys2spec(p, s)**

Convert physical to spectral space
For each m, transform over n, by calling **dfftw_execute(plan_phy2mid)**. Fill **Ts** with field truncated to $NN1$. Call subroutine **tra_Ts2T** to place z on right. For each n, transform over m by calling **dfftw_execute(plan_mid2inp)**. Fill array **s** with field truncated in m.

• **subroutine tra_Ts2T()**

Prior to Fourier transform in z/n, transpose x and z directions so z/n direction is on the right and $(y, x)/(k, m)$ directions are on the left:

$$T(:, m, n) = Ts(:, n, m)$$

• **subroutine tra_T2Ts()**

Prior to Fourier transform in x/m, transpose x and z directions so that x/m direction is on the right and $(y, z)/(k, n)$ directions are on the left:

$$Ts(:, n, m) = T(:, m, n)$$

Dimension parameters:

$$
\begin{aligned}
MM &= 4096 \\
NN &= 1024 \\
K_0 &= 4
\end{aligned}
$$

$$
\begin{aligned}
N &= 2 * (NN - 1) \\
K &= 2 * K_0 - 1 \\
KK &= 3 * K_0 - 1 \\
M &= 2 * (MM - 1) \\
Ny &= 15
\end{aligned}
$$

$$
\begin{aligned}
3M &= 3 * MM \\
3N &= 3 * NN \\
SM &= 3M/DM \\
SN &= 3N/DN
\end{aligned}
$$

$$
\begin{aligned}
Mp &= (\_Np + 3M - 1)/\_Np \\
Np &= (\_Np + NN - 1)/\_Np \\
SMp &= (\_Np + SM - 1)/\_Np
\end{aligned}
$$

$$
\begin{aligned}
NN1 &= NN - 1 \\
N1 &= N - 1 \\
M1 &= M - 1 \\
MM1 &= MM - 1 \\
KK1 &= KK - 1 \\
K1 &= K_0 - 1
\end{aligned}
$$