MMM Anomalous Transport Model (Version 7.1)

Lixiang Luo (`lixiang.luo@lehigh.edu`)
Tariq Rafiq (`tar207@lehigh.edu`)
Arnold Kritz (`kritz@lehigh.edu`)

Lehigh University, Physics Department
16 Memorial Drive East, Bethlehem, PA 18015, USA

February 9, 2012

Abstract

Multi-Mode anomalous transport module (MMM) is a theory-based transport model which has been used to predict temperature, density and momentum rotation profiles for tokamak plasmas. Significantly advancement on the theoretical foundation of MMM has been achieved since its first version, MMM95. The latest version, MMM7.1, includes an improved Weiland model for the ITG, TEM, and MHD modes, the Horton model for short wavelength ETG model and a new model for the drift resistive inertial ballooning modes (DRIBM). The ETG transport threshold in the Horton model is refined by using threshold obtained from toroidal gyrokinetic ETG turbulence. They provide contributions to transport in the different regions of plasma discharge. MMM7.1 is based on an earlier effort called MMM08, which was an attempt to implement similar physical models inside the PTRANSP code. However, the loose implementation of MMM08 has prevented its application outside the PTRANSP code. Steps have been taken to unify the interface and add supplemental features, so that MMM7.1 fully complies with the NTCC standards.

1 Overview

This document contains a brief description of the multi-mode anomalous model (MMM) Fortran 90 software package. The package consists of two parts:

- The MMM module called `modmmm7_1`, which includes two user-callable subroutines: `mmm7_1` and `set_mmm7_1_switches`
- A simple driver program called `testmmm`

The core subroutine `mmm7_1` evaluates the transport diffusivities for anomalous transport. The subroutine calculates the diffusivities based on three internal models:

1. Weiland module by J. Weiland and his group in Göteborg Sweden [1],
2. Drift-resistive-inertial ballooning modes (DRIBM) by [2],
3. Horton model for ETG anomalous transport [3], with the Jenko threshold [4].

Results from internal models are combined linearly, where the weights of these internal models are equal by default. Some model options, such as the switch for turning on the Jenko threshold, are organized as internal parameters. Both the model weights and internal parameters have their default values, but can be specified if necessary. This feature is handled by Fortran 90 optional arguments. A helper subroutine called `set_mmm7_1_switches` is included in the module to assist users to set up the argument arrays for internal parameters.

The MMM code package also include a simple driver program, `testmmm`, along with several test cases, in which sample input and output files are given. The driver program mainly serves two objectives. First, it allows users of MMM to verify the integrity of their compilation of MMM. Second, it can be used as an example or template on how to use MMM.

An important feature of this version of MMM is the extensive use of optional arguments, which requires Fortran 90 explicit interface. Argument association by keywords is strongly recommended. See Section 4 for more details.

All floating-point numbers (both variables and constants) in this software package are defined with a `REAL(R8)` type, where `R8=SELECTED_REAL_KIND(12,100)`, such that this type is equivalent to the predefined type `DOUBLE PRECISION`. Users are strongly encouraged to use consistent data types throughout their own codes. It is not yet possible to use the module with single precision numbers.

The core subroutine `mmm7_1` can be used in parallel programs. In the case that the loop over plasma radial points need to be parallelized, subroutine `mmm7_1` should be called with `npoints=1` with the optional argument `gelong`. This allows the subroutine to do calculation locally (one radial point at a time).

The MMM7.1 module does not generate its own I/O resource. The only file I/O unit, intended for diagnostic output, need to be opened and passed as the `nprout` argument to the core subroutine `mmm7_1` by the main program.

2 Subroutine mmm7_1

The transport equations are assumed to take the transport-diffusion form. For example, the ion temperature (T_H) and hydrogenic ion density (n_H) equations are given by

$$\begin{aligned}\frac{\partial\,(n_\mathrm{H}T_\mathrm{H})}{\partial t} &= \nabla\cdot(D_1 n_\mathrm{H}\nabla T_\mathrm{H}) + \nabla\cdot(v_1\, n_\mathrm{H}T_\mathrm{H}) + S_{T_\mathrm{H}}\,, \\ \frac{\partial n_\mathrm{H}}{\partial t} &= \nabla\cdot(D_2\nabla n_\mathrm{H}) + \nabla\cdot(v_2\, n_\mathrm{H}) + S_{n_\mathrm{H}}\,,\end{aligned}$$

where S_{T_H} and S_{n_H} are the source terms. Other transport equations are assumed to take a similar form. The task of `mmm7_1` is to calculate the diffusivities, such as D_i ($i = 1,2$) and convective fluxes v_i in the equations above. The input argument variables are geometry parameters characterizing equilibrium flux surface shapes and plasma profiles and the corresponding gradients of temperature, density, magnetic q, toroidal and poloidal angular velocity. Output from `mmm7_1` includes thermal diffusivities of electron, thermal ions and impurity ions, density diffusivities of electron and hydrogenic ions, transport coefficients of toroidal, poloidal and parallel momentum, thermal and density fluxes, thermal and momentum pinches. Both overall diffusivities and their contributing components from internal models can be obtained, while the latter is optional.

2.1 Input Arguments

The majority of input arguments are plasma state profiles, listed in Table 1. All the 1-D arrays listed therein are assumed to be defined on flux surfaces called radial points or zone boundaries where the transport fluxes are to be computed. The number of radial points is given by another input argument `npoints`. Note that these dummy arguments are defined as assumed-shape arrays. This allows actual arguments whose sizes are larger than `npoints` to be passed to the subroutine safely, although only the first `npoints` elements are involved in calculations. `gelong` and `rmaj0` are optional arguments, which should only be used by those codes which use `mmm7_1` locally, where `npoints` is always 1. If `gelong` is not provided and `npoints` is less then 3, an error will be generated (`nerr=MMM_ERR_ETG_NOT_ENOUGH_ZONE`).

The remaining input arguments are used for fine control of the subroutine's behaviors. `lprint` controls the verbose level of diagnostic output and `nprout` specifies the I/O unit number for diagnostic output. Because diagnostic output is only used at very few places in the current version, these two arguments mainly serve as place holders for code developers who may need to debug the code.

`cmodel` specifies the linear weights for internal models. This is an optional argument and only the first three elements are used.

- If specified: `cmodel(1)~cmodel(3)` are assigned as the weights for Weiland20, DRIBM and ETG, respectively.

- If not specified: equivalent to `cmodel=(/1.0,1.0,1.0/).`

`cswitch` specifies internal parameters of `REAL(R8)` type. This is an optional argument. The second dimension is corresponding to the index of an internal model (same definition as in `cmodel`), and the first dimension to the index of the adjustable parameter for that particular model. For example, the first real adjustable parameter for the Weiland model should be stored in `cswitch(1,1).`

- If specified: the real internal parameters are assigned the given values of the actual argument.

- If not specified: all real internal parameters default to internally set values.

An up-to-date list of the real internal parameters is given in Table 2, along with their default values.

`lswitch` specifies internal parameters of `INTEGER` type. This is an optional argument. The second dimension is corresponding to the index of an internal model (in the same way as in `cmodel`), and the first dimension to the index of an integral adjustable parameter for that particular model. For example, the first integral adjustable parameter for the DRIBM model should be stored in `lswitch(1,2).`

- If specified: the integral internal parameters are assigned to the given values of the actual argument.

- If not specified: all integral internal parameters default to internally set values.

An up-to-date list of integral adjustable internal parameters is given in Table 3, along with the default values. For ON/OFF type integral switches, 0 stands for OFF and a positive integer stands for ON.

Instead of specifying all the `lswitch` and `cswitch` elements manually, users of MMM7.1 can use the supplemental subroutine `set_mmm7_1_switches`. This can also greatly reduce the chance of human errors. Please refer to Section 3 for more details.

Table 1: Input arguments: plasma variables

Name	**Sym.**	**Unit**	**Meaning**
`rmin`	r	m	Half-width of the flux surface
`rmaj`	R	m	Major radius to geometric center of the flux surface
`rmaj0`	R_0	m	Major radius at plasma axis (scalar)
`elong`	κ		Local elongation of flux surface
`ne`	n_e	m^{-3}	Electron density
`nh`	n_h	m^{-3}	Hydrogenic thermal particle density
`nz`	n_z	m^{-3}	Impurity ion density
`nf`	n_f	m^{-3}	Density from fast (non-thermal) ions
`zeff`	Z_eff		Mean charge $\sum_i n_i Z_i^2 / \sum_i n_i Z_i$
`te`	T_e	keV	Electron temperature
`ti`	T_i	keV	Temperature of thermal ions
`q`	q		Magnetic q-value
`btor`	B_T	Tesla	Toroidal magnetic field $(RB_\mathrm{tor})/r_\mathrm{maj}$
`zimp`	Z_imp		Mean charge of impurities $\sum_\mathrm{imp} n_\mathrm{imp} Z_\mathrm{imp} / \sum_\mathrm{imp} n_\mathrm{imp}$
`aimp`	M_imp		Mean atomic mass of impurities $\sum_\mathrm{imp} n_\mathrm{imp} M_\mathrm{imp} / \sum_\mathrm{imp} n_\mathrm{imp}$
`ahyd`	M_h		Mean atomic mass of hydrogen ions $\sum_\mathrm{h} n_\mathrm{h} M_\mathrm{h} / \sum_\mathrm{h} n_\mathrm{h}$
`aimass`	M_i		Mean atomic mass of thermal ions $\sum_\mathrm{i} n_\mathrm{i} M_\mathrm{i} / \sum_\mathrm{i} n_\mathrm{i}$
`wexbs`	$\omega_{\mathrm{E}\times\mathrm{B}}$	rad/s	$E \times B$ shearing rate [5]
`gne`	g_{n_e}		Normalized n_e gradient $-R\left(\mathrm{d}n_\mathrm{e}/\mathrm{d}r\right)/n_\mathrm{e}$
`gni`	g_{n_i}		Normalized n_i gradient $-R\left(\mathrm{d}n_\mathrm{i}/\mathrm{d}r\right)/n_\mathrm{i}$
`gnh`	g_{n_H}		Normalized n_H gradient $-R\left(\mathrm{d}n_\mathrm{h}/\mathrm{d}r\right)/n_\mathrm{h}$
`gnz`	g_{n_Z}		Normalized n_Z gradient $-R\left(\mathrm{d}Zn_\mathrm{Z}/\mathrm{d}r\right)/\left(Zn_\mathrm{Z}\right)$
`gte`	g_{T_e}		Normalized T_e gradient $-R\left(\mathrm{d}T_\mathrm{e}/\mathrm{d}r\right)/T_\mathrm{e}$
`gti`	g_{T_i}		Normalized T_i gradient $-R\left(\mathrm{d}T_\mathrm{i}/\mathrm{d}r\right)/T_\mathrm{i}$
`gq`	g_q		Normalized q gradient $R\left(\mathrm{d}q/\mathrm{d}r\right)/q$
`vtor`	v_tor	m/s	Toroidal velocity
`gvtor`	g_{v_tor}		Normalized toroidal velocity gradient $R\left(\mathrm{d}v_\mathrm{tor}/\mathrm{d}r\right)/v_\mathrm{tor}$
`vpol`	v_pol	m/s	Poloidal velocity
`gvpol`	g_{v_pol}		Normalized poloidal velocity gradient $R\left(\mathrm{d}v_\mathrm{pol}/\mathrm{d}r\right)/v_\mathrm{pol}$
`vpar`	v_par	m/s	Parallel velocity
`gvpar`	g_{v_par}		Normalized poloidal velocity gradient $R\left(\mathrm{d}v_\mathrm{par}/\mathrm{d}r\right)/v_\mathrm{par}$
`gelong`	κ'		Elongation gradient w.r.t. aspect ratio $\mathrm{d}\kappa/\mathrm{d}\rho$, where $\rho = r/R$

Table 2: Real internal parameters

Model	**#**	**Default**	**Meaning**	**Keyword**
Weiland	1	1.0	$E \times B$ shear multiplier	KW20_C_EXB
	2	1.0	Momentum pinch scaling factor	KW20_C_MOM_PINCH_SCALE
	3	10^{-4}	Lower bound of electron thermal diffusivity	KW20_C_XTE_MIN
	4	100.0	Upper bound of electron thermal diffusivity	KW20_C_XTE_MAX
	5	10^{-4}	Lower bound of ion thermal diffusivity	KW20_C_XTI_MIN
	6	100.0	Upper bound of ion thermal diffusivity	KW20_C_XTI_MAX
ETG	1	0.06	Scaling factor for electrostatic regime	KETG_C_CEES_SCALE
	2	0.06	Scaling factor for electromagnetic regime	KETG_C_CEEM_SCALE
DRIBM	1	0.0	$E \times B$ shear multiplier	KDBM_C_EXB

Table 3: Integer internal parameters

Model	**#**	**Default**	**Meaning**	**Keyword**
ETG	1	1	Threshold selection 0 - ES: Horton, EM: No threshold 1 - ES: Jenko, EM: No threshold 2 - ES: Jenko, EM: Jenko	KETG_L_NLTHR

Table 4: Total diffusivities

Name	Unit	Meaning
`xti`	m^2/s	Effective ion thermal diffusivity
`xdi`	m^2/s	Effective hydrogenic ion diffusivity
`xte`	m^2/s	Effective electron thermal diffusivity
`xdz`	m^2/s	Impurity ion diffusivity from the Weiland model
`xvt`	m^2/s	Toroidal momentum transport from the Weiland model
`xvp`	m^2/s	Poloidal momentum transport from the Weiland model

Table 5: Component diffusivities

Name	Unit	Meaning
`xtiW20`	m^2/s	Ion thermal diffusivity from the Weiland model
`xdiW20`	m^2/s	Particle diffusivity from the Weiland model
`xteW20`	m^2/s	Electron thermal diffusivity from the Weiland model
`xtiDBM`	m^2/s	Ion thermal diffusivity from the DRIBM model
`xdiDBM`	m^2/s	Hydrogenic ion diffusivity from the DRIBM model
`xteDBM`	m^2/s	Electron thermal diffusivity from the DRIBM model
`xteETG`	m^2/s	Electron thermal diffusivity from the Horton ETG model

2.2 Output Arguments

Most of the output arguments are result profile arrays. The only non-array output argument is `nerr`, which stores the error code (a negative integer), if any error is encountered, or zero, if the execution of the subroutine is successful. The dummy arguments for profile output are defined as Fortran 90 assumed-shaped arrays. The actual arguments must be allocated in advance with enough space (`npoints` is the minimal dimension) to store all return values. The diffusivities D_i are given in Table 4. They are weighted sums of contributions from internal models, whose weights can be individually adjusted (see `cmodel` in Section 2.1).

Table 5 lists the component output arrays, which give the individual contribution from internal models. Generally, these arrays are used for diagnostic output only. Because they are optional, users are not required to associate them with actual arguments. Not specifying the actual arguments for them does not affect the outputs listed in Table 4. When they are specified, the actual argument arrays must be allocated in advance with enough space to store the output data (at least `npoints` elements). Note that the momentum transport is only provided by the Weiland model (Table 4).

The most unstable Weiland modes are given by `gammaW20` and `omegaW20`, which contain the the growth rate and its frequency, respectively. Note that there are four growth rates given by the Weiland model. Similarly, `gammaDBM` and `omegaDBM` give the growth rate and the corresponding frequency for the most unstable DRIBM mode. The exact meaning of these arguments are given in Table 6.

`vflux` contains the return values of the total fluxes. `vconv` contains convective velocities and momentum pinches. The content of these two argument is explained in Table 7. They are all optional, whose behavior is similar to the component diffusivities.

Table 6: Growth rates and frequencies of the most unstable Weiland and DRIBM modes

Name	**Unit**	**Meaning**
`gammaDBM`	s^{-1}	Growth rate of the most unstable DRIBM mode
`omegaDBM`	rad/s	Frequency of the most unstable DRIBM mode
`gammaW20(1,:)`	s^{-1}	Growth rate of the most unstable ion mode in Weiland positive-frequency direction
`omegaW20(1,:)`	rad/s	Frequency of the most unstable ion mode in Weiland positive-frequency direction
`gammaW20(2,:)`	s^{-1}	Growth rate of the most unstable electron mode in Weiland positive-frequency direction
`omegaW20(2,:)`	rad/s	Frequency of the most unstable electron mode in Weiland positive-frequency direction
`gammaW20(3,:)`	s^{-1}	Growth rate of the most unstable ion mode in Weiland negative-frequency direction
`omegaW20(3,:)`	rad/s	Frequency of the most unstable ion mode in Weiland negative-frequency direction
`gammaW20(4,:)`	s^{-1}	Growth rate of the most unstable electron mode in Weiland negative-frequency direction
`omegaW20(4,:)`	rad/s	Frequency of the most unstable electron mode in Weiland negative-frequency direction

Table 7: Fluxes and pinches

Name	**Unit**	**Meaning**
`vflux(1,:)`	W/m^2	Total ion thermal flux (Weiland + DRIBM)
`vflux(2,:)`	$m^{-2}s^{-1}$	Total hydrogenic ion flux (Weiland + DRIBM)
`vflux(3,:)`	W/m^2	Total electron thermal flux (Weiland + DRIBM)
`vflux(4,:)`	$m^{-2}s^{-1}$	Total impurity ion flux (Weiland)
`vconv(1,:)`	m/s	Ion thermal convective velocity (Weiland)
`vconv(2,:)`	m/s	Hydrogenic ion particle convective velocity (Weiland)
`vconv(3,:)`	m/s	Electron thermal convective velocity (Weiland)
`vconv(4,:)`	m/s	Impurity ion particle convective velocity (Weiland)
`vconv(5,:)`	m/s	Toroidal momentum pinch (Weiland)
`vconv(6,:)`	m/s	Poloidal momentum pinch (Weiland)

3 Subroutine set_mmm7_1_switches

This is a subroutine to assist users setting up internal parameters using a "`<keyword> = <value>`" approach, instead of manually setting the values of `lswitch` and `cswitch` arrays for subroutine `mmm7_1`. Users do not need to know the index numbers of specific parameters. Also, only the parameters of interest need to be specified, while all other parameters will be assigned the default values automatically. Currently MMM7.1 does not have a large number of internal parameters. However, as more features are added to the future versions of MMM, the index numbers of the internal parameters may be subject to change. Because the keywords remain the same even the index numbers are changed, users of MMM7.1 do not need to update their codes if they are already using `set_mmm7_1_switches` to set internal parameters.

The general syntax of using this subroutine is as follows:

```
CALL set_mmm7_1_switches( &
   cmmm = <cswitch>, lmmm = <lswitch>, &
   <keyword 1> = <value 1>, &
   <keyword 2> = <value 2>, &
   ... )
```

where `<cswitch>` and `<lswitch>` are the array variables which will be passed to `mmm7_1` subroutine. Keywords are listed in Table 2 and 3. Only those parameters that need to be change should be listed, the subroutine will fill the remaining parameters using their corresponding default values. Note that if only the integer parameters are involved, the argument for real-type is not required, and vice versa. For example, if the user only want to turn on $E \times B$ shear effects in DRIBM model and leave everything else by default, they can use

```
CALL set_mmm7_1_switches( cmmm = cmmm7, KDBM_C_EXB = 1.0 )
```

and then pass `cmmm7` as the actual argument for `cswitch` to `mmm7_1`:

```
CALL mmm7_1( ... , cswitch = cmmm7 )
```

where all other elements of `cmmm7` are already given the default values by `set_mmm7_1_switches`. All the integer-type internal parameters will take the default values because no actual argument is specified for `lswitch`.

4 Using the module

To use MMM in your own program, the following issues need to be taken care of:

1. Compile the module and generate the static-link library file `libmmm7_1.a`
2. The `USE` statement at the beginning of your Fortran program
3. A proper `CALL` statement of the `mmm7_1` subroutine
4. Linking of `libmmm7_1.a` against other binary object files

MMM7.1 module does not have any dependence on external codes. A successful build of the MMM7.1 module will generate a number of binary files, among them are the two most important, `libmmm7_1.a` and `modmmm7_1.mod` (`modmmm7_1.MOD` if PathScale compilers are used) in the `libmmm7_1` subdirectory. `libmmm7_1.a` is the static-link library and `modmmm7_1.mod` is the Fortran module file.

The compilation of any source file that use the `modmmm7_1` module requires the compiler to correctly locate the module file (`modmmm7_1.mod`). Most compiler search `*.mod` files in directories listed after the -I option. Module files are generally incompatible among different compilers. You will not be able to compile the driver program with compiler B if the MMM module is compiled using compiler A.

Linking of `libmmm7_1.a` against other binary object files should only involve putting it in the object file list, as long as the file can be located by the compiler. Note that only static linking is supported in this version. Please follow the instructions of your Fortran compiler to set appropriate command line arguments.

No initialization is needed before the `CALL` statement of the `mmm7_1` subroutine. However, one may want to set up the the internal parameter arrays using the `set_mmm7_1_switches` subroutine. Note that MMM7.1 uses optional arguments extensively, thus requiring the use of explicit interface. If any optional argument is omitted, argument association by keywords must be used. Even in the case where no optional argument is omitted, the use of argument keywords is still strongly recommended, considering the large amount of arguments involved. An example of this style of subroutine call is given in the source file of the driver program `testmmm.f90`. One clear advantage of argument keywords is that the compiler can always determine the correct argument association, regardless of the order and the selection of actual arguments. This also minimize the need to update the user's codes if the argument list of MMM is to be changed in the future.

All array dummy arguments are defined as assumed-shape arrays, in contrast to earlier versions of MMM, which used deferred-shape arrays. Several arguments are optional. Please refer to Section 2 for more details on the selection of optional arguments.

A complete argument keyword association of `mmm7_1` can be found in the driver program:

```
CALL mmm7_1( &
   rmin   = rmin,   rmaj   = rmaj,   rmaj0  = rmaj(1),   &
   elong  = elong,  ne     = ne,     nh     = nh,        &
   nz     = nz,     nf     = nf,     zeff   = zeff,      &
   te     = te,     ti     = ti,     q      = q,         &
   btor   = btor,   zimp   = zimp,   aimp   = aimp,      &
   ahyd   = ahyd,   aimass = aimass, wexbs  = wexbs,     &
   gne    = gne,    gni    = gni,    gnh    = gnh,       &
   gnz    = gnz,    gte    = gte,    gti    = gti,       &
   gq     = gq,                                          &
   gvtor  = gvtor,  vtor   = vtor,   gvpol  = gvpol,     &
   vpol   = vpol,   gvpar  = gvpar,  vpar   = vpar,      &
   xti    = xti,    xdi    = xdi,    xte    = xte,       &
   xdz    = xdz,    xvt    = xvt,    xvp    = xvp,       &
   xtiW20 = xtiW20, xdiW20 = xdiW20, xteW20 = xteW20,    &
   xtiDBM = xtiDBM, xdiDBM = xdiDBM, xteDBM = xteDBM,    &
```

```
    xteETG = xteETG,                                        &
    gammaW20 = gammaW20, omegaW20 = omegaW20,               &
    gammaDBM = gammaDBM, omegaDBM = omegaDBM,               &
    npoints = npoints,                                      &
    lprint  = lprint, nprout  = hfDebug, nerr    = nerr,    &
    vconv   = vconv,  vflux   = vflux ,                     &
    cmodel  = cmodel, cswitch = cswitch, lswitch = lswitch)
```

where the association of actual arguments and dummy arguments are explicitly indicated by a "`<dummy argument> = <actual argument>`" form. In fact, the order of arguments has no effect on argument association, eliminating the frequent and hard-to-debug error which happens when some arguments are left out. Users can take advantage of the optional arguments, without worrying about the order of arguments. Consider a much simplified case:

- All internal models are turn on, with default weights.
- Default internal parameters are used.
- Only the thermal diffusivities are needed.
- No diagnostic output is needed.

In this case, the statement can be shortened to

```
CALL mmm7_1( &
   rmin   = rmin,   rmaj   = rmaj,   rmaj0  = rmaj(1),    &
   elong  = elong,  ne     = ne,     nh     = nh,         &
   nz     = nz,     nf     = nf,     zeff   = zeff,       &
   te     = te,     ti     = ti,     q      = q,          &
   btor   = btor,   zimp   = zimp,   aimp   = aimp,       &
   ahyd   = ahyd,   aimass = aimass, wexbs  = wexbs,      &
   gne    = gne,    gni    = gni,    gnh    = gnh,        &
   gnz    = gnz,    gte    = gte,    gti    = gti,        &
   gq     = gq,                                           &
   gvtor  = gvtor,  vtor   = vtor,   gvpol  = gvpol,      &
   vpol   = vpol,   gvpar  = gvpar,  vpar   = vpar,       &
   xti    = xti,    xdi    = xdi,    xte    = xte,        &
   xdz    = xdz,    xvt    = xvt,    xvp    = xvp,        &
   npoints = npoints, lprint = 0, nprout = 0, nerr = nerr,&
   vconv   = vconv,  vflux   = vflux )
```

As we can see, the unused arguements do not need to be specified at all. This subroutine call can also be conveniently expanded. For example, if we want to turn off Jenko's threshold for the Horton ETG model, we can simply write

```
CALL set_mmm7_1_switches( lmmm = lmmm7, KETG_L_NLTHR = 0 )
CALL mmm7_1( &
   rmin    = rmin,    rmaj    = rmaj,    rmaj0   = rmaj(1),    &
   elong   = elong,   ne      = ne,      nh      = nh,         &
   nz      = nz,      nf      = nf,      zeff    = zeff,       &
   te      = te,      ti      = ti,      q       = q,          &
   btor    = btor,    zimp    = zimp,    aimp    = aimp,       &
   ahyd    = ahyd,    aimass  = aimass,  wexbs   = wexbs,      &
   gne     = gne,     gni     = gni,     gnh     = gnh,        &
   gnz     = gnz,     gte     = gte,     gti     = gti,        &
   gq      = gq,                                               &
   gvtor   = gvtor,   vtor    = vtor,    gvpol   = gvpol,      &
   vpol    = vpol,    gvpar   = gvpar,   vpar    = vpar,       &
   xti     = xti,     xdi     = xdi,     xte     = xte,        &
   xdz     = xdz,     xvt     = xvt,     xvp     = xvp,        &
   npoints = npoints, lprint = 0, nprout = 0, nerr = nerr,&
   vconv    = vconv,  vflux    = vflux,  lswitch = lmmm7   )
```

The change involves only one variable `lmmm7` and a subroutine call to `set_mmm7_1_switches` (see Section 3 for more details).

5 Driver program `testmmm`

The driver program looks for a file called "`input`" in the current directory and invokes `mmm7_1` with the supplied input data. Both the input data and results are then written into a file called "`output`" as tables. The input file is written in the Fortran NAMELIST format, such that the variables can be arranged in any order. Two kinds of input data can be accepted. In the first kind the contents of the plasma state arrays are given in numbers. The size of arrays must be specified by the `npoints` variable. With the second kind of input data, the user needs to specify a series of polynomial parameters for constructing the plasma state profiles.These are mostly parabolic profiles (or an exponentiation with a specified exponent). Note that this type can only generate trivial (zero) profiles for momentum profiles (and their gradients). The NAMELIST header of the data file need to be changed according to the choice of the input type. Use

```
&testmmm_input_1stkind
```

if the first kind is given, or use

```
&testmmm_input_2ndkind
```

if the second kind is given. The variables listed in Table 8 are expected from the input file, regardless of the choice of data type.

If first kind is used, all the plasma profiles must be specified using arrays of double precision numbers. In this case, `testmmm` expect the input file to provide the arrays listed in Table 1. Note that `rmaj0` is not needed because its value is determined by the first element of array `rmaj`. The definition of the corresponding dummy arguments can be found in Table 1. `gte`, `gti`, `gne`, `gnh`, `gnz` and `gni` can be calculated using their corresponding profile variables by the driver program

Table 8: Input variables used by both input file types

Input variable	`mmm7_1` dummy argument association
`npoints`	`npoints`
`cmodel`	`cmodel`
`lprint`	`lprint`
`cW20`	`cswitch(1:,1)`
`cDBM`	`cswitch(1:,2)`
`cETG`	`cswitch(1:,3)`
`lW20`	`lswitch(1:,1)`
`lDBM`	`lswitch(1:,2)`
`lETG`	`lswitch(1:,3)`

instead of being provided by the user. To use this feature, simply assign a value smaller than -100 to the gradient variable. An example of this feature can be found in the sample input file of `case-hmode`.

If the second kind is used, the actual arguments for `mmm7_1` are constructed by a number of parameters. The number of radial points is given by input variable `npoints`. The minor radius of plasma is given by input variable `k_rminor`. The half width of a magnetic surface r is proportional to the radial index. The major radius for all radial points is given the value of `k_rmajor`. Plasma profiles arguments for subroutine `mmm7_1` are constructed using polynomials in the following form

$$f(r) = f_{\text{edge}} + \left(f_{\text{axis}} - f_{\text{edge}}\right)(1-\rho)^p ,$$

where $\rho = r/r_{\text{edge}}$ is the normalized radius; f_{axis} and f_{edge} are the variable values at the center and the edge of the plasma; exponent p can be used to control the shape of the profile. The first nine entries in Table 9 are the plasma profiles directly generated by polynomials, whose gradients are calculated using the derivative of $f(r)$ by `testmmm`. The variables with a "`axis`" and a "`edge`" suffix indicate the values at the plasma center and edge, respectively. Their units are given in the first column of the table. The variables with a "`exp`" suffix indicate the exponent p, which is dimensionless.

$\omega_{\text{E}\times\text{B}}$ is calculated differently, using the following polynomial:

$$\omega_{\text{E}\times\text{B}} = 16\omega_0 \frac{(\rho-a)^2(\rho-b)^2}{(a-b)^4} ,$$

where ω_0 is the maximum flow shearing rate; a is the inner cutoff and b is the outer cutoff. ω_0, a and b take their values from variables `wexbmax`, `xwexbinn` and `xwexbout`, respectively. `vtor`, `gvtor`, `vpol`, `gvpol`, `vpar` and `gvpar` are always set to zero inside the driver program. The final four entries in Table 9 lists are the scalar input variables which are used to generate flat profiles as `mmm7_1` arguments.

Effective charge (`mmm7_1` argument `zeff`) is given by this definition:

$$Z_{\text{eff}} = \frac{n_{\text{h}} + Z_{\text{imp}}^2 n_{\text{imp}} + Z_{\text{fi}}^2 n_{\text{fi}}}{n_{\text{e}}} ,$$

where subscript "imp" indicates the impurities and "fi" indicates the fast ions. Electron density (`mmm7_1` argument `ne`) and fast electron density are calculated according to the pseudo-neutrality

Table 9: Variables used in the second kind input. The "Argument" column indicates the corresponding `mmm7_1` dummy arguments.

Meaning	Input variables	Argument
Ion density [m^{-3}]	`denhaxis`, `denhedge`, `denhexp`	`ni`
Average impurity density [m^{-3}]	`denzaxis`, `denzedge`, `denzexp`	`nz`
Fast ion density [m^{-3}]	`denfaxis`, `denfedge`, `denfexp`	
Average charge of impurities	`chrzaxis`, `chrzedge`, `chrzexp`	`zimp`
Average charge of super-thermal ions	`chrfaxis`, `chrfedge`, `chrfexp`	
Electron temperature [KeV]	`teaxis`, `teedge`, `teexp`	`te`
Ion temperature [KeV]	`tiaxis`, `tiedge`, `tiexp`	`ti`
Magnetic q	`qaxis`, `qedge`, `qexp`	`q`
Flow shearing rate $\omega_{\mathrm{E}\times\mathrm{B}}$ [rad/s]	`wexbmax`, `xwexbinn`, `xwexbout`	`wexbs`
Major radius of plasma axis [m]	`k_rmajor`	
Minor radius [m]	`k_rminor`	
Local elongation	`k_elong`	`elong`
Toroidal magnetic field [Tesla]	`k_btor`	`btor`
Mean atomic mass of impurities	`k_amassz`	`aimp`
Mean atomic mass of hydrogenic ions	`k_amassh`	`ahyd`
Minimal electron density [m^{-3}]	`k_denmin`	
Minimal electron temperature [KeV]	`k_temin`	

condition:

$$\begin{aligned} n_{\mathrm{e}} &= n_{\mathrm{h}} + Z_{\mathrm{imp}} n_{\mathrm{imp}} + Z_{\mathrm{fi}} n_{\mathrm{fi}} \,, \\ n_{\mathrm{fe}} &= Z_{\mathrm{fi}} n_{\mathrm{fi}} \,, \end{aligned}$$

where the "fe" subscript indicates fast electrons. The electron density gradient (`mmm7_1` argument `gne`) is then calculated accordingly. Average ion atomic mass (`mmm7_1` argument `aimass`) is given by

$$M_{\mathrm{i}} = \frac{n_{\mathrm{h}} M_{\mathrm{h}} + n_{\mathrm{imp}} M_{\mathrm{imp}}}{n_{\mathrm{h}} + n_{\mathrm{imp}}} \,,$$

where M represents atomic mass.

The output file is a plain text spreadsheet with clearly defined headers and column numbers. The first line is a section header, followed by a line of a column index. The third and the fourth line includes the units and the names of the input profile arrays, respectively. The profiles are organized in columns, whose names are self-explanatory. The output section also starts with a line of header, followed by the output profile units and names. All the output arguments of `mmm7_1` are listed in this section. Most of them are self-explanatory as they use the exact same name of the corresponding dummy argument of subroutine `mmm7_1`. The growth rates and frequencies of the most unstable modes are associated in a way given in Table 10. The definition of the dummy arguments therein can be found in Table 6.

As an example, the output of `case-lmode` case can be visualized using gnuplot (a freely distributed plotting tool) by this command run in the `case-lmode` subdirectory:

Table 10: Growth rates and frequencies of the most unstable Weiland and DRIBM modes

Name	`mmm7_1` dummy argument association
`gmaW20ii`	`gammaW20(1,:)`
`omgW20ii`	`omegaW20(1,:)`
`gmaW20ie`	`gammaW20(2,:)`
`omgW20ii`	`omegaW20(2,:)`
`gmaW20ei`	`gammaW20(3,:)`
`omgW20ei`	`omegaW20(3,:)`
`gmaW20ee`	`gammaW20(4,:)`
`omgW20ee`	`omegaW20(4,:)`
`gmaDBM`	`gammaDBM`
`omgDBM`	`omegaDBM`

```
gnuplot> plot '< tail -n 51 output' u 1:4 w l
```

Note that `case-lmode` has 51 radial points. This gnuplot command extracts the last 51 lines of the output file and generates an X-Y curve using a solid line, with the the magnetic surface half-width (column 1) on X axis and the electron thermal diffusivity (column 4) on the Y axis.

6 PTRANSP settings

MMM7.1 can be used with the predictive mode of PTRANSP. It has been installed and tested to be working correctly in PTRANSP running on PPPL clusters. As for 2011 the predictive model of MMM7.1 is capable of temperature prediction while density prediction is being developed. MMM7.1 has been included in the PTRANSP source repository maintained by PPPL. To use MMM7.1 as the anomalous transport model in PTRANSP, `NKEMOD` and `NKIMODA` should be set to 19. The TRDAT variables used by PTRANSP for controlling MMM7.1 is given in Table 11. For switch type parameters, .F. or 0 means to disable and .TRUE. or 1 means to enable.

References

[1] J. Weiland, *Collective modes in inhomogeneous plasma: kinetic and advanced fluid theory*, ser. Plasma Physics. Institute of Physics Publishing, 2000.

[2] T. Rafiq, G. Bateman, A. H. Kritz, and A. Y. Pankin, "Development of drift-resistive-inertial ballooning transport model for tokamak edge plasmas," *Physics of Plasmas*, vol. 17, no. 8, p. 082511, 2010.

[3] W. Horton, P. Zhu, G. T. Hoang, T. Aniel, M. Ottaviani, and X. Garbet, "Electron transport in Tore Supra with fast wave electron heating," *Physics of Plasmas*, vol. 7, no. 5, pp. 1494–1510, 2000.

[4] F. Jenko, W. Dorland, and G. W. Hammett, "Critical gradient formula for toroidal electron temperature gradient modes," *Physics of Plasmas*, vol. 8, no. 9, pp. 4096–4104, 2001.

Table 11: MMM7.1 related PTRANSP parameters

Name	**Type**	**Default**	**Purpose**
XIMINMMM	Real	0.0	Inner boundary for predictions
XIMAXMMM	Real	1.0	Outer boundary for predictions
CMMM07(1)	Real	1.0	Multiplier for pinches in Weiland model
FACEXB	Real	1.0	$E \times B$ shear multiplier in Weiland model
L_DRBM	Integer	1	Switch for DRIBM model
LMMM07(4)	Integer	1	Switch for disabling $E \times B$ shear effects in DRIBM model (0: $E \times B$ shear effects enabled, 1:$E \times B$ shear effects disabled)
CMMM07(3)	Real	1.0	$E \times B$ shear multiplier in DRIBM model
NLETG	Logical	.T.	Switch for Horton ETG model
NLETGJTHR	Logical	.T.	F: Horton threshold, T: Jenko threshold
LMMM07(3)	Integer	1	Jenko threshold selection (only used for NLETGJTHR=.T.) 1: Enabled for electrostatic regime, disabled for electromagnetic regime 2: Enabled for both electrostatic and electromagnetic regimes

[5] K. H. Burrell, “Effects of $E \times B$ velocity shear and magnetic shear on turbulence and transport in magnetic confinement devices,” *Physics of Plasmas*, vol. 4, no. 5, pp. 1499–1518, 1997.