GITM User Manual

Version 2.0

A.J. Ridley

January 1, 2012

Contents

Chapter 1

Introduction to GITM

The Global Ionosphere Thermosphere Model (GITM) is a 3D model of the upper atmosphere. It runs for Earth, Mars and Titan. A version is being worked on for Jupiter. GITM solves for the coupled continuity, momentum and energy equations of the neutrals and ions. For the ions, the time rate of change of the velocity is ignored, so the steady-state ion flow velocity is solved for. The ion temperature is a mixture of the electron and neutral temperature.

The neutrals are solved for using the Navier Stokes Equations. The continuity equation is solved for for each major species. One of the problems with GITM that needs to be rectified is that there are no real tracer species, so a species is either solved for completely or is not at all. These species can still be included in the chemistry calculation. There is only one horizontal velocity that is computed, while there are vertical velocities for each of the major species. A bulk vertical velocity is calculated as a mass weighted average. The temperature is a bulk temperature.

1.1 Source Terms

Chemistry is the only real source term for the continuity equation. Typically, diffusion is added in the continuity equation to allow for eddy diffusion, but this is not the case in GITM. What happens is that the vertical velocities are solved for, then a friction term is applied to that the velocities stay very close together in the eddy diffusion part of the code. This way, the velocities can't differ too much from each other. Diffusion is not needed, then.

For the horizontal momentum equation, there are the following sources: (1) ion drag; (2) viscosity; and (3) gravity wave acceleration. For the vertical velocity, the source terms are ion drag and friction between the different neutral species.

For the neutral temperature, the following source terms are included: (1) radiative cooling; (2) EUV heating; (3) auroral heating; (4) Joule heating; (5) conduction; and (6) chemical heating. The biggest pain for the temperature equation is the use of a normalized temperature. This means that the `temperature` variable in GITM does not contain the actual temperature, it contains the temperature multiplies by Boltzmann's Constant divided by the mean mass. This turns out to be a factor that is very similar to the specific heat, or roughly or order 1000. In order to get the actual temperature, the variable has to be multiplied by `temp_unit`.

1.2 Ghost Cells

GITM is a parallel code. It uses a 2D domain decomposition, with the altitude domain being the only thing that is not broken up. Blocks of latitude and longitude are used. These blocks are then distributed among different processors. In order to communicate between the processors, ghostcells are used. These are cells that essentially overlap with the neighboring block. MPI (message passing interface) is then used to move information from one block to another, filling in the ghostcells. The code then loops from 1-N, where the flux is calculated at the boundaries from the 0-1 boundary to the N-N+1 boundary. A second order scheme is used to calculate the fluxes, along with a flux limiter. Therefore, two ghost cells are needed.

In the vertical direction, ghost cells are also used to set boundary conditions. The values in these cells are used to calculate the fluxes, just as described above. Different types of boundary conditions (constant values, constant fluxes, constant gradients, floating, zero fluxes, etc) can be set by carefully choosing the right values in the ghost cells.

1.3 Code Outline

This is an outline of GITM. To produce this file, go into the `src` directory, and type:

```
cd ../src
./calling_sequence.pl > outline.tex
mv outline.tex ../srcDoc
cd ../srcDoc
```

`main program` in **main.f90**

- `init_mpi` in file **init_mpi.f90**
  - `MPI_INIT`
  - `MPI_COMM_RANK`
  - `MPI_COMM_SIZE`
- `start_timing` in file **timing.f90**
- `delete_stop` in file **stop_file.f90**
  - `report` in file library.f90
- `init_planet` in file **ModEarth.f90**
  - `time_int_to_real` in file time_routines.f90
- `set_defaults` in file **ModInputs.f90**
  - `set_strings` in file ModInputs.f90
  - `time_int_to_real` in file time_routines.f90
  - `set_planet_defaults` in file Earth.f90
- `read_inputs` in file **read_inputs.f90**
  - `report` in file library.f90
  - `stop_gitm` in file library.f90
  - `stop_gitm` in file library.f90
  - `stop_gitm` in file library.f90
  - `MPI_Bcast`
  - `stop_gitm` in file library.f90
  - `MPI_Bcast`
  - `stop_gitm` in file library.f90
- `set_inputs` in file **set_inputs.f90**
  - `report` in file library.f90
  - `read_in_int` in file set_inputs.f90

- `time_int_to_real` in file time_routines.f90
- `time_real_to_int` in file time_routines.f90
- `fix_vernal_time` in file set_inputs.f90
- `read_in_int` in file set_inputs.f90
- `time_int_to_real` in file time_routines.f90
- `read_in_time` in file set_inputs.f90
- `read_in_int` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `time_real_to_int` in file time_routines.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `IO_set_f107_single`
- `IO_set_f107a_single`
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `IO_set_hpi_single`
- `read_in_real` in file set_inputs.f90
- `IO_set_kp_single`
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `IO_set_imf_by_single`

- `IO_set_imf_bz_single`
- `IO_set_sw_v_single`
- `read_in_string` in file set_inputs.f90
- `IO_set_inputs`
- `read_MHDIMF_Indices`
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_string` in file set_inputs.f90
- `read_in_string` in file set_inputs.f90
- `read_in_string` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_int` in file set_inputs.f90
- `read_in_int` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90

- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_int` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_string` in file set_inputs.f90
- `read_in_string` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90
- `read_in_int` in file set_inputs.f90
- `read_in_int` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_real` in file set_inputs.f90
- `read_in_logical` in file set_inputs.f90

  - `read_in_logical` in file set_inputs.f90
  - `read_in_time` in file set_inputs.f90
  - `read_in_time` in file set_inputs.f90
  - `read_in_real` in file set_inputs.f90
  - `read_in_logical` in file set_inputs.f90
  - `read_in_real` in file set_inputs.f90
  - `read_in_int` in file set_inputs.f90
  - `read_in_string` in file set_inputs.f90
  - `read_in_real` in file set_inputs.f90
  - `read_in_int` in file set_inputs.f90
  - `read_in_string` in file set_inputs.f90
  - `read_in_real` in file set_inputs.f90
  - `read_satellites` in file satellites.f90
  - `read_in_real` in file set_inputs.f90
  - `read_in_real` in file set_inputs.f90
  - `read_in_real` in file set_inputs.f90
  - `read_in_logical` in file set_inputs.f90
  - `read_in_string` in file set_inputs.f90
  - `read_in_string` in file set_inputs.f90
  - `read_in_real` in file set_inputs.f90
  - `read_in_logical` in file set_inputs.f90
  - `read_in_string` in file set_inputs.f90
  - `Set_Euv` in file calc_euv.f90
  - `read_in_logical` in file set_inputs.f90
  - `read_in_real` in file set_inputs.f90
  - `read_in_string` in file set_inputs.f90
  - `IO_set_inputs`
  - `read_NGDC_Indices`
  - `read_in_string` in file set_inputs.f90
  - `read_in_string` in file set_inputs.f90
  - `IO_set_inputs`
  - `read_SWPC_Indices`
  - `read_in_string` in file set_inputs.f90
  - `IO_set_inputs`
  - `read_NOAAHPI_Indices`
  - `stop_gitm` in file library.f90
  - `time_int_to_real` in file time_routines.f90

- `initialize_gitm` in file **initialize.f90**
  - `init_radcooling` in file ModEarth.f90

- `start_timing` in file timing.f90
- `report` in file library.f90
- `time_real_to_int` in file time_routines.f90
- `fix_vernal_time` in file set_inputs.f90
- `get_f107`
- `get_f107a`
- `init_grid` in file init_grid.f90
- `read_inputs` in file read_inputs.f90
- `set_inputs` in file set_inputs.f90
- `read_restart` in file restart.f90
- `init_msis` in file init_msis.Earth.f90
- `set_RrTempInd` in file ModRates.Earth.f90
- `init_euv` in file calc_euv.f90
- `init_altitude` in file init_altitude.f90
- `UAM_gradient`
- `init_heating_efficiency` in file Earth.f90
- `init_magheat` in file ModEarth.f90
- `init_isochem` in file Mars.f90
- `init_aerosol` in file ModEarth.f90
- `init_isochem` in file Mars.f90
- `init_msis` in file init_msis.Earth.f90
- `get_temperature` in file init_altitude.f90
- `init_iri` in file init_iri.Earth.f90
- `read_waccm_tides` in file tides.f90
- `update_waccm_tides` in file tides.f90
- `read_tides` in file tides.f90
- `update_tides` in file tides.f90
- `init_b0` in file init_b0.f90
- `init_energy_deposition` in file init_energy_deposition.f90
- `report` in file library.f90
- `SUBSOLR`
- `exchange_messages_sphere` in file exchange_messages_sphere.f90
- `calc_pressure` in file calc_pressure.f90
- `UA_calc_electrodynamics` in file calc_electrodynamics.f90
- `calc_eddy_diffusion_coefficient` in file Earth.f90
- `calc_rates` in file calc_rates.Earth.f90
- `calc_viscosity` in file calc_rates.Earth.f90
- `calc_rates` in file calc_rates.Earth.f90
- `end_timing` in file timing.f90

- `write_output` in file **write_output.f90**
  - `output` in file output_common.f90
  - `move_satellites` in file satellites.f90
  - `write_restart` in file restart.f90
  - `logfile` in file logfile.f90
- `report` in file **library.f90**
- Loop Start
  - `calc_pressure` in file **calc_pressure.f90**
    - `report` in file library.f90
  - `calc_timestep_vertical` in file **calc_timestep.f90**
    - `report` in file library.f90
    - `MPI_AllREDUCE`
    - `stop_gitm` in file library.f90
  - `calc_timestep_horizontal` in file **calc_timestep.f90**
    - `report` in file library.f90
    - `MPI_AllREDUCE`
  - `advance` in file **advance.f90**
    - `report` in file library.f90
    - `start_timing` in file timing.f90
    - `update_tides` in file tides.f90
    - `update_waccm_tides` in file tides.f90
    - `advance_vertical_all` in file advance.f90
    - `add_sources` in file add_sources.f90
    - `advance_horizontal_all` in file advance.f90
    - `time_real_to_int` in file time_routines.f90
    - `get_f107`
    - `stop_gitm` in file library.f90
    - `get_f107a`
    - `stop_gitm` in file library.f90
    - `init_msis` in file init_msis.Earth.f90
    - `init_iri` in file init_iri.Earth.f90
    - `init_b0` in file init_b0.f90
    - `end_timing` in file timing.f90
    - `report` in file library.f90
    - `start_timing` in file timing.f90
    - `calc_rates` in file calc_rates.Earth.f90
    - `calc_viscosity` in file calc_rates.Earth.f90
    - `advance_vertical` in file advance_vertical.f90
    - `end_timing` in file timing.f90
    - `report` in file library.f90
    - `start_timing` in file timing.f90

        - `exchange_messages_sphere` in file exchange_messages_sphere.f90
        - `calc_rates` in file calc_rates.Earth.f90
        - `calc_physics` in file calc_physics.f90
        - `advance_horizontal` in file advance_horizontal.f90
        - `calc_physics` in file calc_physics.f90
        - `calc_rates` in file calc_rates.Earth.f90
        - `advance_horizontal` in file advance_horizontal.f90
        - `exchange_messages_sphere` in file exchange_messages_sphere.f90
        - `end_timing` in file timing.f90
    - `check_stop` in file **stop_file.f90**
        - `report` in file library.f90
        - `start_timing` in file timing.f90
        - `MPI_AllREDUCE`
        - `check_start` in file stop_file.f90
        - `end_timing` in file timing.f90
    - `write_output` in file **write_output.f90**
        - `output` in file output_common.f90
        - `move_satellites` in file satellites.f90
        - `write_restart` in file restart.f90
        - `logfile` in file logfile.f90
- Loop End
- `finalize_gitm` in file **finalize.f90**
    - `UA_calc_electrodynamics` in file calc_electrodynamics.f90
    - `output` in file output_common.f90
    - `write_restart` in file restart.f90
    - `end_timing` in file timing.f90
    - `report_timing` in file timing.f90
    - `UAM_XFER_destroy` in file ModSphereInterface.f90
    - `UAM_write_error` in file ModSphereInterface.f90
    - `stop_gitm` in file library.f90
    - `MPI_FINALIZE`
- `stop_gitm` in file **library.f90**
    - `CON_stop` in file main.f90
    - `MPI_abort`

Chapter 2

Running GITM

2.1 Quick Start

2.1.1 Extracting the code from a tar file

Create a new and empty directory, and open the tar file you received, e.g.:

```
mkdir Gitm
cd Gitm
mv ../gitm.tgz .
tar -xvzf gitm.tgz
```

2.1.2 Checking out the code with CVS

If CVS (Concurrent Versions System) is available on your computer and you have an account on the CVS server machine herot.engin.umich.edu, you can use CVS to install the current or a particular version of the code. First of all have the following environment variables:

```
setenv CVSROOT UserName@herot.engin.umich.edu:/CVS/FRAMEWORK
setenv CVS_RSH ssh
```

where UserName is your user name on herot. Here it is assumed that you use csh or tcsh. Also put these settings into your .cshrc file so it is automatically executed at login. Alternatively, use

```
CVSROOT=UserName@herot.engin.umich.edu:/CVS/FRAMEWORK
export CVSROOT
CVS_RSH=ssh
export CVS_RSH
```

under sh, ksh and bash shells, and also put these commands into your .bashrc or .profile file so it is automatically executed at login.

Once the CVS environment variables are set, you can download the current (HEAD) version of the GITM distribution with

```
cvs checkout GITM2
```

If you want a particular version, use

```
cvs checkout -r v2_0 GITM2
```

where v2_0 is the it tag associated with the version. To download bug fixes or new features, the

```
cvs update
```

command can be used. See `man cvs` for more information.

A lot of times, you don't really want the `GITM2` directory to stay that name, since you might download a couple different version (maybe one for development and one for runs). Therefore, typically you will:

```
mv GITM2 GITM2.Development
```

2.1.3 Configuring and Making GITM

In order to compile GITM, you have to configure it first. The configure script is inhereted from the Space Weather Modeling Framework. There are two primary reasons you need to do the configure: (1) put the right `Makefile` in the right place, specifying the compiler and the version of MPI that you will use to link the code; (2) put the right MPI header in the right place. It also does some things like hard-codes the path of the source code into the `Makefile`. Some examples are below.

Installing on Nyx:

```
./Config.pl -install -compiler=ifortmpif90 -earth
```

Installing on a Mac with gfortran and openmpi:

```
./Config.pl -install -compiler=gfortran -earth
```

Installing on a computer with gfortran and not using MPI:

```
./Config.pl -install -compiler=gfortran -earth -nompi
```

Sometimes people have a hard time with the ModUtilities.F90 file. If you have errors with this file, try (for example):

```
./Config.pl -uninstall
./Config.pl -install -compiler=gfortran -earth -noflush
```

2.1.4 Running the Code

GITM requires a bunch of files to be in the right place in order to run. Therefore, it is best to use the makefile to create a run directory:

```
make rundir
mv run myrun
```

where `myrun` can be whatever you want. I will use `myrun` as an example. You can actually put this directory where ever you want. On many system, there is a `nobackup` scratch disk that you are supposed to use for runs, instead of your home directory. Therefore, a lot of times, I do the following:

```
make rundir
mv run /nobackup/myaccount/gitm/myrun
ln -s /nobackup/myaccount/gitm/myrun .
```

Then you can treat the run directory just like it is in your home directory, but it isn't! Once you have created the run directory, you can run the default simulation, by:

```
cd myrun
mpirun -np 4 GITM.exe
```

This, hopefully should run GITM for Earth for 5 minutes. If it doesn't work, then you might have mpi set up incorrectly. The default is to allow you to run 4 blocks per processor, and the default `UAM.in` file is set up for 4 blocks, so you could try just running GITM without mpi, just to see if it works at all:

```
./GITM.exe
```

If that doesn't work, then it probably didn't compile correctly. Hopefully, it just worked!

2.1.5 Post Processing

GITM, by default, produces one file per block per output. If you are outputting often and you are running with many blocks, you can produce a huge number of files. To post process all of these files, simply:

```
cd UA
./pGITM
```

This merges all of the files for one time period, for one file type into the same file. You can actually running this while the code is running, since GITM doesn't use old files. Well, this isn't actually true. Since I just got the APPENDFILE thing working, if you are using satellites, I won't run this during a run just yet. I need to test this feature. If you are NOT using satellites and NOT using APPENDFILE, then you are free and clear to use this as often as you want during a run. I have run into a case in which GITM was in the middle of writing a file, and I ran this and it deleted the file that GITM was writing. Oops! This has only happened to me once, though. I typically run this in a script in which there is a five minute pause between running it, just to be on the safe side. I also tend to have a script running that does an rsync to my computer. So, the script runs `pGITM`, then rsyncs the directory (excluding all of the files that are not processed), and then removing the processed and rsynced files. A very simple, but very useful script!

2.1.6 The Code Won't Compile!!

Im sorry. I tried to make this work on many different platforms, but sometimes machines are very specific, and it just doesn't work out of the box. Here are some ideas on how to quickly get this thing compiling.

Can't find the right Makefile.whatever

If make does not work, then there is probably a problem with not finding the FORTRAN 90 compiler. The platform and machine specific Makefiles are in srcMake. If you type:

```
uname
ls srcMake
```

If you don't see a file named something like Makefile.uname (where uname is the output of the uname command), then you will have to build a proper general Makefile.

You will need a little a little information about your computer, like what the mpif90 compiler is called and where it is located. Take a look at srcMake/Makefile.Linux, and try to figure out what all of the flags are for your system. Then create a srcMake/Makefile.uname with the correct information in it.

The compiler doesn't recognize flag -x

You have an operating system that is recognized, but probably a different compiler. In the srcMake/Makefile.uname file (where uname is the output of the uname command), there is a line:

OSFLAGS = -w -dusty

You need to change this line to something more appropriate for your compiler. Try deleting the flags and compile. If that doesn't work, you will have to check the man pages of your compiler.

2.2 Setting the Grid

Setting the grid resolution in GITM is not very complicated, but it does involve some thought. There are a few variables that control this. In `ModSize.f90`, the following variables are defined:

```
integer, parameter :: nLons = 9
integer, parameter :: nLats = 9
integer, parameter :: nAlts = 50

integer, parameter :: nBlocksMax = 4
```

The first three variables (nLons, nLats and nAlts) define the size of a single block. In the example above, there are 9 cells in latitude, 9 cells in longitude and 50 cells in altitude. The final variable (nBlocksMax) defines the maximum number of blocks you can have on a single processor. Typically, most people run with one single block per processor, so setting this to "1" is almost always fine. In theory, this can save memory.

2.2.1 Running 3D Over the Whole Globe

Once the number of cells is defined, then the number of blocks in latitude and longitude need to be defined. This is done in the `UAM.in` file. For example, this sets the number of blocks to 8 in the latitude direction and 8 in the longitude direction:

```
#GRID
8           lons
8           lats
-90.0       minimum latitude to model
90.0        maximum latitude to model
0.0         minimum longitude to model
0.0         maximum longitude to model
```

The number of cells in the simulation domain will then be 72 in longitude, 72 in latitude, and 50 in altitude. Given that there are 360° in longitude and 180° in latitude, the resolution would be $360^\circ/72 = 5.0^\circ$ and $180^\circ/72 = 2.5^\circ$ in latitude. If one block were put on each processor, 64 processors would be required. This problem fits quite nicely on either four- or eight-core processors. On 12-core processors, this would not work very well at all. We have started to run simulations of 1° in latitude and 5° in longitude, which can fit nicely on a 12-core processor machine. For example, in `ModSize.f90`:

```
integer, parameter :: nLons = 9
integer, parameter :: nLats = 15
```

and in `UAM.in`:

```
#GRID
8 lons
12 lats
```

This can then run on 96 cores, which is nicely divisible by 12. Essentially, an infinite combination of cells per block and number of blocks can be utilized. Typically, the number of blocks in latitude and longitude are even numbers.

2.2.2 Running 3D Over the Part of the Globe

GITM can be run over part of the globe - both in latitude and in longitude. It can be run over a single polar region (by setting either the minimum or maximum latitude to be greater (or less) than $\pm 90^\circ$). If this is selected, message passing over the poles is implemented. If the pole is not selected, then boundary conditions have to be set in `set_horizontal_bcs.f90`. By default, a continuous gradient boundary condition is used on the densities and temperatures, while a continuous value is used on the velocity. This is true in both latitude and longitude. In longitude, message passing is implemented all of the time, but the values are over-written by the boundary conditions if the maximum and minimum longitude are not equal to each other.

The longitudinal resolution ($\Delta\phi$) is set by:

$$\Delta\phi = \frac{\phi_{end} - \phi_{start}}{nBlocksLon \times nCellsLon} \tag{2.1}$$

while, the latitudinal resolution ($\Delta\theta$) is set by:

$$\Delta\theta = \frac{\theta_{end} - \theta_{start}}{nBlocksLat \times nCellsLat} \tag{2.2}$$

2.2.3 Running in 1D

GITM can run in 1D mode, in which the call to advance_horizontal is not completed. This means that GITM runs exactly the same way, but ignoring all of the horizontal advection terms. You have to do two things to make GITM run in 1D. First, in `ModSize.f90`:

```
integer, parameter :: nLons = 1
integer, parameter :: nLats = 1
integer, parameter :: nAlts = 50

integer, parameter :: nBlocksMax = 1
```

This tells the code that you only want one single latitude and longitude location. To specify the exact location, in `UAM.in`:

```
#GRID
1           lons
1           lats
41.75       minimum latitude to model
41.75       maximum latitude to model
275.0       minimum longitude to model
275.0       maximum longitude to model
```

This is pretty close to some place in Michigan. GITM will model this exact point for as long as you specify. One thing to keep in mind with running in 1D is that the Earth still rotates, so the spot will have a strong day to night variation in temperature. In 3D, the winds decrease some of the variability between day and night, but in 1D, this doesn't happen. So, the results are going to be perfect. But, 1D is great for debugging.

2.2.4 Stretching the Grid

You can stretch the grid in GITM in the latitudinal direction. It takes some practice to get the stretching just the way that you might like. Here is an example that we typically use for stretching near the equator for an equatorial electrodynamics run:

```
#STRETCH
0.0            ! Equator
0.7            ! Amount of stretch
0.8            ! more control
```

Chapter 3

Inputs

This sets the starting time of the simulation. Even when you restart, the starttime should be to the real start time, not the restart time.

```
#STARTTIME
iYear    (integer)
iMonth   (integer)
iDay     (integer)
iHour    (integer)
iMinute  (integer)
iSecond  (integer)
```

This sets the ending time of the simulation.

```
#ENDTIME
iYear    (integer)
iMonth   (integer)
iDay     (integer)
iHour    (integer)
iMinute  (integer)
iSecond  (integer)
```

This will set a time for the code to just pause. Really, this should never be used.

```
#PAUSETIME
iYear iMonth iDay iHour iMinute iSecond
```

This is typically only specified in a restart header. If you specify it in a start UAM.in it will start the counter to whatever you specify.

```
#ISTEP
iStep     (integer)
```

This sets the maximum CPU time that the code should run before it starts to write a restart file and end the simulation. It is very useful on systems that have a queueing system and has limited time runs. Typically, set it for a couple of minutes short of the max wall clock, since it needs some time to write the restart files.

```
#CPUTIMEMAX
CPUTimeMax    (real)
```

If you just want GITM to run MSIS and IRI over and over again, use this. Then GITM is never actually run. You just get MSIS and IRI.

```
#STATISTICALMODELSONLY
UseStatisticalModelsOnly    (logical)
DtStatisticalModels         (real)
```

This is typically only specified in a restart header. It sets the offset from the starttime to the currenttime. Should really only be used with caution.

```
#TSIMULATION
tsimulation    (real)
```

Sets the F10.7 and 81 day average F10.7. This is used to set the initial altitude grid, and drive the lower boundary conditions.

```
#F107
f107  (real)
f107a (real - 81 day average of f107)
```

This specifies the initial conditions and the lower boundary conditions. For Earth, we typically just use MSIS and IRI for initial conditions. For other planets, the vertical BCs can be set here.

```
#INITIAL
UseMSIS        (logical)
UseIRI         (logical)
If UseMSIS is .false. then :
TempMin        (real, bottom temperature)
TempMax        (real, top initial temperature)
TempHeight     (real, Height of the middle of temp gradient)
TempWidth      (real, Width of the temperature gradient)
```

This says how to use tides. The first one is using MSIS with no tides. The second is using MSIS with full up tides. The third is using GSWM tides, while the forth is for experimentation with using WACCM tides.

```
#TIDES
UseMSISOnly        (logical)
UseMSISTides       (logical)
UseGSWMTides       (logical)
UseWACCMTides      (logical)
```

If you selected to use GSWM tides above, you can specify which components to use.

```
#GSWMCOMP
GSWMdiurnal(1)        (logical)
GSWMdiurnal(2)        (logical)
GSWMsemidiurnal(1)    (logical)
GSWMsemidiurnal(2)    (logical)
```

This is probably for damping vertical wind oscillations that can occur in the lower atmosphere.

```
#DAMPING
UseDamping        (logical)
```

I dont know what this is for...

```
#GRAVITYWAVE
UseGravityWave        (logical)
```

This sets the hemispheric power of the aurora. Typical it ranges from 1-1000, although 20 is a nominal, quiet time value.

```
#HPI
HemisphericPower  (real)
```

I dont think that GITM actually uses this unless the Foster electric field model is used.

```
#KP
kp  (real)
```

The CFL condition sets how close to the maximum time step that GITM will take. 1.0 is the maximum value. A value of about 0.75 is typical. If instabilities form, a lower value is probably needed.

```
#CFL
cfl  (real)
```

This sets the driving conditions for the high-latitude electric field models. This is static for the whole run, though. It is better to use the MHD_INDICES command to have dynamic driving conditions.

```
#SOLARWIND
bx  (real)
by  (real)
bz  (real)
vx  (real)
```

Use this for dynamic IMF and solar wind conditions. The exact format of the file is discussed further in the manual.

```
#MHD_INDICES
filename  (string)
```

This is for using Pat Newells aurora (Ovation).

```
#NEWELLAURORA
UseNewellAurora   (logical)
UseNewellAveraged (logical)
UseNewellMono (logical)
UseNewellWave (logical)
UseNewellRemoveSpikes (logical)
UseNewellAverage      (logical)
```

```
#AMIEFILES
cAMIEFileNorth  (string)
cAMIEFileSouth  (string)
```

The limiter is quite important. It is a value between 1.0 and 2.0, with 1.0 being more diffuse and robust, and 2.0 being less diffuse, but less robust.

```
#LIMITER
TypeLimiter  (string)
```

This will set how much information the code screams at you - set to 0 to get minimal, set to 10 to get EVERYTHING. You can also change which processor is shouting the information - PE 0 is the first one. If you set the iDebugLevel to 0, you can set the dt of the reporting. If you set it to a big value, you wont get very many outputs. If you set it to a tiny value, you will get a LOT of outputs. UseBarriers will force the code to sync up a LOT.

```
#DEBUG
iDebugLevel (integer)
iDebugProc  (integer)
DtReport    (real)
UseBarriers (logical)
```

```
#THERMO
UseSolarHeating   (logical)
UseJouleHeating   (logical)
UseAuroralHeating (logical)
UseNOCooling      (logical)
UseOCooling       (logical)
UseConduction     (logical)
UseTurbulentCond  (logical)
UseUpdatedTurbulentCond  (logical)
EddyScaling  (real)
```

```
#THERMALDIFFUSION
KappaTemp0    (thermal conductivity, real)
```

```
#VERTICALSOURCES
UseEddyInSolver              (logical)
UseNeutralFrictionInSolver   (logical)
MaximumVerticalVelocity      (real)
```

```
#EDDYVELOCITY
UseBoquehoAndBlelly              (logical)
UseEddyCorrection   (logical)
```

```
#WAVEDRAG
UseStressHeating             (logical)
```

If you use eddy diffusion, you must specify two pressure levels - under the first, the eddy diffusion is constant. Between the first and the second, there is a linear drop-off. Therefore The first pressure must be larger than the second!

```
#DIFFUSION
UseDiffusion (logical)
EddyDiffusionCoef (real)
EddyDiffusionPressure0 (real)
EddyDiffusionPressure1 (real)
```

```
#FORCING
UsePressureGradient (logical)
UseIonDrag          (logical)
UseNeutralFriction  (logical)
UseViscosity        (logical)
UseCoriolis         (logical)
UseGravity          (logical)
```

```
#DYNAMO
UseDynamo              (logical)
DynamoHighLatBoundary  (real)
nItersMax              (integer)
MaxResidual            (V,real)
```

```
#IONFORCING
UseExB                 (logical)
UseIonPressureGradient (logical)
UseIonGravity          (logical)
UseNeutralDrag         (logical)
UseDynamo              (logical)
```

```
#DIPOLE
MagneticPoleRotation   (real)
MagneticPoleTilt       (real)
xDipoleCenter          (real)
yDipoleCenter          (real)
zDipoleCenter          (real)
```

```
#APEX
UseApex (logical)
        Sets whether to use a realistic magnetic
        field (T) or a dipole (F)
```

```
#ALTITUDE
AltMin                (real, km)
AltMax                (real, km)
UseStretchedAltitude  (logical)
```

If LatStart and LatEnd are set to ¡ -90 and ¿ 90, respectively, then GITM does a whole sphere. If not, it models between the two. If you want to do 1-D, set nLons=1, nLats=1 in ModSizeGitm.f90, then recompile, then set LatStart and LonStart to the point on the Globe you want to model.

```
#GRID
nBlocksLon   (integer)
nBlocksLat   (integer)
LatStart     (real)
LatEnd       (real)
LonStart     (real)
LonEnd       (real)
```

```
#STRETCH
ConcentrationLatitude (real, degrees)
StretchingPercentage  (real, 0-1)
StretchingFactor      (real)
Example (no stretching):
#STRETCH
65.0 ! location of minimum grid spacing
0.0  ! Amount of stretch 0 (none) to 1 (lots)
1.0
```

```
#TOPOGRAPHY
UseTopography (logical)
```

```
#RESTART
DoRestart (logical)
```

This will allow you to change the output cadence of the files for a limited time. If you have an event then you can output much more often during that event.

```
#PLOTTIMECHANGE
yyyy mm dd hh mm ss ms (start)
yyyy mm dd hh mm ss ms (end)
```

For satellite files, you can have one single file per satellite, instead of one for every output. This makes GITM output significantly less files. It only works for satellite files now.

```
#APPENDFILES
DoAppendFiles    (logical)
```

This sets the output files. The most common type is 3DALL, which outputs all primary state variables. Types include : 3DALL, 3DNEU, 3DION, 3DTHM, 3DCHM, 3DUSR, 3DGLO, 2DGEL, 2DMEL, 2DUSR, 1DALL, 1DGLO, 1DTHM, 1DNEW, 1DCHM, 1DCMS, 1DUSR.

```
#SAVEPLOT
DtRestart (real, seconds)
nOutputTypes  (integer)
Outputtype (string, 3D, 2D, ION, NEUTRAL, ...)
DtPlot    (real, seconds)
```

```
#SATELLITES
nSats     (integer - max = ',nMaxSats,')
SatFile1  (string)
DtPlot1   (real, seconds)
etc...
```

Sets the time for updating the high-latitude (and low-latitude) electrodynamic drivers, such as the potential and the aurora.

```
#ELECTRODYNAMICS
DtPotential (real, seconds)
DtAurora    (real, seconds)
```

```
#LTERadiation
DtLTERadiation (real)
```

This is really highly specific. You dont want this.

```
#IONPRECIPITATION
UseIonPrecipitation     (logical)
IonIonizationFilename   (string)
IonHeatingRateFilename  (string)
```

You really want a log file. They are very important. It is output in UA/data/log*.dat. You can output the log file at whatever frequency you would like, but if you set dt to some very small value, you will get an output every iteration, which is probably a good thing.

```
#LOGFILE
DtLogFile   (real, seconds)
```

This is for a FISM or some other solar spectrum file.

```
#EUV_DATA
UseEUVData            (logical)
cEUVFile              (string)
```

3.1 Auxiliary Input Files

3.1.1 IMF and Solar Wind

This file controls the high-latitude electric field and aurora when using models that depend on the solar wind and interplanetary magnetic field (IMF). It allows for dynamically controlling these quantities. You can create realistical IMF files or hypothetical ones. For realistic IMF files, we typically use CDF files downloaded from the CDAWEB ftp site, and IDL code that merges the solar wind and IMF files to create one single file. The IDL code also propagates the solar wind and IMF from L1 to 32 Re upstream of the Earth. You can use the `DELAY` statement to shift the time more (e.g.,in the example below, it shifts by an additional 15 minutes). The IDL code to process the CDF files is called `cdf_to_mhd.pro`. It requires both a solar wind file and an IMF file. For example, the IMF file would be `ac_h0_mfi_20011231_v04.cdf` and the solar wind file would be `ac_h0_swe_20011231_v06.cdf`. The code assumes that the data starts at `#START`, and ends when it encounters an error. This can mean that if there is an error in the data somewhere, the code will only read up to that point. To validate that the solar wind and IMF is what you think it is, it is recommended that you use the IDL code `imf_plot.pro`.

Here is an example file:

```
This file was created by Aaron Ridley to do some
wicked cool science thing.

The format is:
 Year MM DD HH Mi SS mS   Bx  By   Bz     Vx   Vy   Vz    N        T

Year=year
MM = Month
DD = Day
HH = Hour
Mi = Minute
SS = Second
mS = Millisecond
Bx = IMF Bx GSM Component (nT)
By = IMF By GSM Component (nT)
Bz = IMF Bz GSM Component (nT)
Vx = Solar Wind Vx (km/s)
Vy = Solar Wind Vy (km/s)
Vz = Solar Wind Vz (km/s)
N  = Solar Wind Density (/cm3)
T  = Solar Wind Temperature (K)

#DELAY
900.0

#START
 2000  3 20  2 53  0  0  0.0 0.0  2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  2 54  0  0  0.0 0.0  2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  2 55  0  0  0.0 0.0  2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  2 56  0  0  0.0 0.0  2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  2 57  0  0  0.0 0.0  2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  2 58  0  0  0.0 0.0  2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  2 59  0  0  0.0 0.0  2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  3  0  0  0  0.0 0.0 -2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  3  1  0  0  0.0 0.0 -2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  3  2  0  0  0.0 0.0 -2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  3  3  0  0  0.0 0.0 -2.0 -400.0  0.0  0.0  5.0  50000.0
 2000  3 20  3  4  0  0  0.0 0.0 -2.0 -400.0  0.0  0.0  5.0  50000.0
```

To actually read in this file, in `UAM.in`, use the command:

```
#MHD_INDICES
filename
```

3.1.2 Hemispheric Power

These files describe the dynamic variation of the auroral power going into each hemisphere. Models such as **?** use the Hemispheric Power to determine which level of the model it should use. The Hemispheric Power is converted to a Hemispheric Power Index using the formula (check):

$$HPI = 2.09log(HP)^{1.0475} \tag{3.1}$$

Example File 1:

```
# Prepared by the U.S. Dept. of Commerce, NOAA, Space Environment Center.
# Please send comments and suggestions to sec@sec.noaa.gov
#
# Source: NOAA POES (Whatever is aloft)
# Units: gigawatts

# Format:

# The first line of data contains the four-digit year of the data.
# Each following line is formatted as in this example:

# NOAA-12(S)  10031     9.0  4    .914

# Please note that if the first line of data in the file has a
# day-of-year of 365 (or 366) and a HHMM of greater than 2300,
# that polar pass started at the end of the previous year and
# ended on day-of-year 001 of the current year.

# A7    NOAA POES Satellite number
# A3    (S) or (N) - hemisphere
# I3    Day of year
# I4    UT hour and minute
# F8.1  Estimated Hemispheric Power in gigawatts
# I3    Hemispheric Power Index (activity level)
# F8.3  Normalizing factor

2000
NOAA-15(N)  10023    35.5  7    1.085
NOAA-14(S)  10044    25.3  7     .843
NOAA-15(S)  10114    29.0  7     .676
NOAA-14(N)  10135   108.7 10    1.682
NOAA-15(N)  10204    36.4  7    1.311
```

```
.
.
.
```

The second example shows a file format that is much better. This format started in 2007, while all of the files before this time are of the first example type.

```
:Data_list: power_2010.txt
:Created: Sun Jan  2 10:12:58 UTC 2011


# Prepared by the U.S. Dept. of Commerce, NOAA, Space Environment Center.
# Please send comments and suggestions to sec@sec.noaa.gov
#
# Source: NOAA POES (Whatever is aloft)
# Units: gigawatts

# Format:

# Each line is formatted as in this example:

# 2006-09-05 00:54:25 NOAA-16 (S)  7  29.67   0.82

# A19   Date and UT at the center of the polar pass as YYYY-MM-DD hh:mm:ss
# 1X    (Space)
# A7    NOAA POES Satellite number
# 1X    (Space)
# A3    (S) or (N) - hemisphere
# I3    Hemispheric Power Index (activity level)
# F7.2  Estimated Hemispheric Power in gigawatts
# F7.2  Normalizing factor

2010-01-01 00:14:37 NOAA-17 (N)  1   1.45   1.16
2010-01-01 00:44:33 NOAA-19 (N)  1   1.45   1.17
.
.
.
```

3.1.3 Solar Irradiance

More to come here.

3.1.4 Satellites

```
#SATELLITES
2                    nSats
guvi.2002041623.in
15.0                 SatDtPlot
stfd.fpi.in
60.0                 SatDtPlot
```

Here is a sample satellite input file:

```
year mm dd hh mm ss msec long lat alt
#START
2002 4 16 23 34 25 0 299.16 -2.21 0.00
2002 4 16 23 34 25 0 293.63 -1.21 0.00
2002 4 16 23 34 25 0 291.28 -0.75 0.00
2002 4 16 23 34 25 0 289.83 -0.45 0.00
2002 4 16 23 34 25 0 288.79 -0.21 0.00
2002 4 16 23 34 25 0 287.98 -0.01 0.00
2002 4 16 23 34 25 0 287.32  0.16 0.00
2002 4 16 23 34 25 0 286.76  0.31 0.00
2002 4 16 23 34 25 0 286.26  0.46 0.00
2002 4 16 23 34 25 0 285.81  0.60 0.00
2002 4 16 23 34 25 0 285.39  0.74 0.00
```

At this time, GITM ignores the altitude, and just outputs the entire column.

Chapter 4

Outputs

Once you have the output from GITM in a bunch of output files, there are a few IDL programs that you can use to visualize them:

1. **thermo_plotsat:** This is plotting 1D results. It is the most commonly used program for plotting 1D. It can be used for plotting satelite files and for plotting 1D simulations. It is relatively straight forward to use, but experimentation can be help. This is an actual program, so you have to `.run` it.

2. **thermo_gui:** This is a graphical user interface code for plotting 3D results that is somewhat simplistic. The filename has to be entered manually in the upper left. You then have to press the button for loading the file. Variables appear on the left side, and you can select which one you want to plot. You then select which plane you would like to look at (lat/lon, lat/alt, lon/alt). You can scroll through which altitude/latitude/longitude you want to look at. You can also add wind vectors on, plot in polar coordinats, and plot the log of the variable.

3. **thermo_batch_new:** This code will let you look at at 3D files exactly the same way as thermo_gui, but is all scripted. There are a few features that this has that thermo_batch doesn't have: (1) you can use wildcards for the file name, so it can read in a list of files. The postscript file name will be appended with numbers. (2) When plotting a lat/alt plane, you can do a zonal average. (3) You can do a global average.

4. **thermo_plotter:** All of the above plotting codes will only plot one plot per page. This code will plot many more than one plot per page. You can plot multiple variables on the same page, or multiple files with the same variable, or both.