TIEGCM Documentation

Release 1.94

Ben Foster

January 06, 2012

CONTENTS

Note: This document is up to date for TIEGCM version 1.94 (Diagnostics section is updated to version 1.94.1)

Contents:

CHAPTER

ONE

INTRODUCTION

The NCAR/HAO Thermosphere Ionosphere Electrodynamics General Circulation Model, or TIEGCM, is a three-dimensional, time-dependent, numeric simulation model of the Earth's upper atmosphere, including the upper Stratosphere, Mesosphere and Thermosphere.

This document is intended to assist members of the upper atmosphere research community in building, executing, and visualizing the TIEGCM as an integral part of their research. We also welcome development efforts on the model itself, subject to the Academic Research License Agreement `tiegcmlicense.txt`.

For more information about the TIEGCM and other related models, please see:

- Main TGCM website
- TIEGCM Release Documentation for version 1.94.
- TIEGCM Model Description
- tgcmgroup email list

To contact a live body at NCAR/HAO, send email to Ben Foster at foster@ucar.edu (see also *Contact Information*)

CHAPTER

TWO

QUICKSTART PROCEDURE

This document is intended to provide a tutorial-like procedure for downloading source code and start-up data files, building the model, and executing a short default run on a 64-bit Linux system.

Note: This document is up to date for version 1.94 of the TIEGCM

2.1 Downloading the model source code and required data files

The model source code and related input data files may be downloaded from the TIEGCM download page of the main TGCM website:

http://www.hao.ucar.edu/modeling/tgcm/download.php

You will need to provide an email address (login and password are NOT required). Documentation and Postprocessor codes are also available on the download site, but all you need for now is the source code, and corresponding data files. Both are provided as gzipped tar files.

After downloading the two gzipped tar files to an empty working directory (*workdir*) on a large disk system (for example, `/mydisk/tiegcm`), uncompress and extract the source code and related scripts and documentation:

```
$ gunzip tiegcmx.xx.tar.gz
$ tar xvf tiegcmx.xx.tar
```

where `x.xx` is the model version downloaded.

Next, make a directory to hold the data files (for example, `/mydisk/tiegcm/data`), and uncompress and extract the data tar file into that directory. Then set environment variable `$TGCMDATA`, e.g., for the c-shell, add this line to your .cshrc file:

```
setenv TGCMDATA /mydisk/tiegcm/data
```

At this point, you should have something like this in your working directory `/mydisk/tiegcm`:

```
total 2376
-rw-rw-r-- 1 user tgcm    4928 Jun  1  2010 README.download
-rw-r--r-- 1 user tgcm    2886 Jun  1  2010 Release_Notes
drwxrwxr-x 2 user tgcm    4096 Apr 22 08:52 data/
-rwxrwxr-x 1 user tgcm   10137 May 31  2010 tiegcm-ibm.job*
-rwxrwxr-x 1 user tgcm   10671 May 31  2010 tiegcm-linux.job*
drwxrwxr-x 5 user tgcm    4096 Jun  1  2010 tiegcmx.xx/
-rw-r--r-- 1 user tgcm    6116 May 31  2010 tiegcmlicense.txt
```

These files and directories contain the following:

README.download
`README.download` Instructions for building and making a short default run.

Release_Notes
Release notes for this version of the model.

data/
Directory containing the downloaded data files (this is `$TGCMDATA`)

tiegcm-ibm.job
Job script for building and executing under IBM/AIX systems. (`default tiegcm-ibm.job`)

Read more about *Running the model on IBM/AIX systems*.

tiegcm-linux.job
Job script for building and executing under Linux (64-bit) systems. (`default tiegcm-linux.job`)

tiegcmx.xx/
Model root directory, containing source code, supporting scripts, and documentation.

You are now prepared to build the model and make a short default run using the job script.

2.2 Making a default run on a 64-bit Linux system

Take a look at the Linux job script `tiegcm-linux.job`. Near the top are several shell variables, with their default settings, which configure the job script (variables and values may vary somewhat between model versions):

```
set modeldir = tiegcmx.xx
set execdir  = tiegcm-linux
 set make     = Make.intel_hao64
#set make     = Make.pgi_hao64
#set input    = tiegcm.inp
set output   = tiegcm.out
set mpi      = TRUE
set nproc    = 4
set modelres = 5.0
set debug    = FALSE
set exec     = TRUE
set utildir  = $modeldir/scripts
```

Following are brief explanations of the job script shell variables:

Note: Absolute or relative paths are acceptable when specifying directories. Relative paths should be relative to the *working directory* (*workdir*).

---

modeldir
The model root directory (*modeldir* from the source code download). This will contain subdirectories `src/`, `scripts/`, `doc/`, etc.

tiegcm-linux
This is the execution directory (*execdir*), in which the model will be built and executed. It will be created if it does not already exist. This directory will also contain the model output netCDF history files.

make
Make file containing platform-specific compiler flags, library locations, etc. If not otherwise specified with a path, the job script will look for this file in `modeldir/scripts`. This file is included in the main Makefile (`scripts/Makefile`). The user can either make necessary adjustments to an existing `make` file, or write their own for a different platform/compiler system.

Here is an example `make` file for 64-bit HAO Linux systems using the ifort Intel compiler: `Make.intel_hao64`

input
The namelist input file. When this is commented (as above), the job script will make a default namelist file `tiegcm_default.inp`, and use it for the default run. Later, you can edit this file for your own runs, rename it, and reset and uncomment the `input` shell variable in the job script.

output
Name of the file to receive stdout output from the model. If this pre-exists, it will be overwritten when the model is executed.

Here is an example stdout file from a single-processor default run: `tiegcm_default.out`

mpi
Logical flag indicating whether or not to link the MPI library for a multi-processor parallel run. If FALSE, the MPI library is not linked, and it is assumed the model will be run in serial (single-processor) mode.

nproc
Number of processors to use in a parallel execution. This is ignored if `mpi` is FALSE.

modelres

Model resolution. Two resolutions are supported:

- modelres = 5.0 sets 5-degree lat x lon horizontal, and dz=0.50 vertical
- modelres = 2.5 sets 2.5-degree lat x lon horizontal, and dz=0.25 vertical

If the resolution is changed, the model should be recompiled before re-executing the job script (type "*gmake clean*" in the execdir).

For more information, see *Grid Structure and Resolution*.

debug
If `debug` = TRUE, the job script will compile the build with debug flags set. Debug flags specific to the compiler are set in the `make` file. If `debug` is changed, the code should be recompiled (type "gmake clean" in the `execdir` before re-executing the job script).

exec
If `exec` = TRUE, the job script will execute the model after compilation, otherwise, the job script will stop after compilation without execution.

utildir
The utility directory containing supporting scripts. This is normally the `scripts/` subdirectory in the model root directory `modeldir`.

You are now ready to build and execute a default run. To do this, simply execute the job script as follows:

```
$ tiegcm-linux.job &
```

The compilation output will be displayed. If the build is successful (and exec=TRUE), the model will be executed, and stdout will go to the specified `output` file. If the job is successful, you can edit and rename the namelist input file, reset `input` in the job script, and re-execute the job script. If there has been no change to the source code, it will not need to recompile, and will use the pre-existing executable.

2.3 Running the model on IBM/AIX Platforms

Note: This section contains some information that is specific to user's of the NCAR IBM system "bluefire". User's of other IBM systems may need to make adjustments for their particular environment. For more information about the NCAR bluefire system, see http://www2.cisl.ucar.edu/docs/bluefire-user-guide

---

The model can be built and executed on IBM platforms running AIX with the xlf90 (mpxlf_r) compiler. You can use the same procedure described in the previous section, except that you use the IBM job script `tiegcm-ibm.job` instead of the Linux job script tiegcm-linux.job.

The IBM job script has the same user-settable shell variables as the Linux job script, but the default settings are slightly different:

```
set modeldir = tiegcm_trunk
set execdir  = tiegcm_trunk-aix
#set input    = tiegcm.inp
set output   = tiegcm.out
set make     = Make.bluefire
set mpi      = TRUE
set modelres = 5.0
set debug    = FALSE
set exec     = TRUE
set utildir  = $modeldir/scripts
```

Note the `execdir` name, and the `make` file `Make.bluefire`

Also note the special "`#BSUB`" directives at the top of the IBM job script (descriptions in the right-hand column are for this document only, and are not in the script itself):

```
#BSUB -J tiegcm_trunk               # Job name
#BSUB -P 24100004                   # NCAR project number
##BSUB -q regular                   # regular queue (commented here)
##BSUB -n 32                        # number of processors (commented here)
#BSUB -q debug                      # debug queue
#BSUB -n 8                          # number of processors (MPI tasks)
#BSUB -o tiegcm_trunk.%J.out        # stdout file
#BSUB -e tiegcm_trunk.%J.out        # stderr file
#BSUB -N
#BSUB -u $LOGNAME@ucar.edu          # email notification address
#BSUB -W 1:00                       # wallclock limit (6-hr max at NCAR)
```

These are resource settings for the Load Sharing Facility (LSF), the batch queuing system sold by Platform Computing. The LSF is used for scheduling jobs on the `bluefire` IBM system at NCAR. This job will be submitted to the debug queue, requesting 8 processors, with a wallclock limit of 1 hour. Note the double pound-sign "##" indicates a commented field.

To submit the IBM job script to the LSF batch system, type:

```
$ bsub < tiegcm-ibm.job
```

Watch the progress of your LSF job with the command:

```
$ bjobs
```

You can kill a LSF job with this command:

```
$ bkill job_ID
```

Where `job_ID` is the job identifier given in the `bjobs` command.

For more information about the LSF, see the Wikipedia site:

http://en.wikipedia.org/wiki/Platform_LSF

or the Platform Computing site:

http://www.platform.com/workload-management/high-performance-computing/lp

CHAPTER

THREE

DIRECTORY STRUCTURE

This section describes a typical directory structure the user will be working with when running the TIEGCM model. The working directory (*workdir*) is the "root" directory of the user's project.

The model directory (*modeldir*) is typically a subdirectory under the working directory, and contains the model source code, supporting scripts, documentation, and scripts for running tests.

The data directory (*datadir*) may also be a subdirectory under the working directory, or it may be on a large temporary disk that is accessible from the working directory. The data directory contains start-up and input data files for running the model.

3.1 Working Directory (*workdir*)

The user's working directory will typically look something like this (the *datadir* can be on a large separate disk system):

```
                    workdir
                       |
-----------------------------------------------
   |           |             |            |
*.inp       execdir/     modeldir/     datadir/
*.job          |
*.out          *.o
               *.mod
               *.nc
               Make*
               exec
```

Here, `\*.inp` are *namelist input* files, `\*.job` are *job script* s, and `\*.out` are are stdout files from model runs. The *execdir* is the build/execution directory (created by the first build), with object code (`\*.o`, `\*.mod`), model *output history files* (`\*.nc`), make files (`Make\*`), and an executable file (`exec`). Various other files may also be in the execdir. The modeldir and datadir directories are described below.

The job script in your working directory contains a shell variable specifying the *modeldir*, so it knows where to find the source code and supporting scripts for the build process. The namelist input file also refers to the *datadir* for start-up and other data input files (e.g., *SOURCE*, *GPI_NCFILE*, *IMF_NCFILE*, etc). These namelist parameters can use the environment variable *TGCMDATA* to specify the *datadir* (see section on *namelist input files*).

3.2 Model Directory (*modeldir*)

The model root directory is what you get when you *download* the model source code tar file. The model directory contains subdirectories with the model source code, supporting scripts, documentation, and test scripts:

```
                              modeldir
                                 |
-------------------------------------------------------------------
   |               |                |                |
  src/          scripts/           doc/            tests/
   |               |                |                |
                 Make.*          userguide/        control/
  *.F           linux.job       description/     climatology/
  *.h            ibm.job          release/         dec2006
               default.inp       diags.table       nov2003/
              tgcm_contents    README.download     whi2008/
               tgcm_ncdump
                  etc
```

src/ directory contents:

- Fortran source code `*.F`, `*.h`. The source code is f90 compliant, and most source files are in fixed-format fortran. There is a single header file, `defs.h`, which contains grid definitions and dimensions.

scripts/ directory contents:

- **Make.***: Makefiles containing platform-dependent compiler flags, Make variables, and library locations. For example, `Make.intel_hao64`. These file can be copied, renamed, and customized for the user's platform/machine environment.
- **tiegcm-linux.job**: Default model build/execute script for Linux systems.
- **tiegcm-ibm.job**: Default model build/execute script for IBM/AIX systems.
- **tiegcm_default.inp**: Default namelist input file.
- **tgcm_contents**: Utility script to print "contents" of netCDF output history files.
- **tgcm_ncdump**: Utility script to print an "ncdump" of history files, including data for scalars and 1-d vectors.

doc/ directory contents:

- **userguide/**: Directory containing source files for the User's Guide (this document)
- **description/**: Directory containing source files for the TIEGCM Model Description
- **release/**: Directory containing source files for the Release Documentation
- **diags.table**: `Table of diagnostic fields` that can be saved on secondary history files.
- **README.download**: `Instructions` for how to make a quick-start default build and execution of the model after downloading the source code and data.

tests/ directory contents:

- **README.tests**: Summary of benchmark test runs made with the current version of the model
- **Several directories** (climatology, control, dec2006, nov2003, etc) containing namelist input files and job scripts that can be used to reproduce the benchmark runs for validation and testing, comparing results from code changes, etc.
- For more information on benchmark runs made for the current release, please see Release Documentation

3.3 Data Directory (*datadir*)

The public TIEGCM data directory is what you get when you *download* the data tar file. This directory is typically referred to with the environment variable *TGCMDATA*. Subsequently, after the data download, you may obtain additional needed data files from the *NCAR Community Data Portal*. Here is a partial schematic of the datadir (where "tiegcmx.xx" is the desired model version):

```
                    datadir
                       |
-------------------------------------------
           |                        |
       gpi*.nc                 tiegcmx.xx/
       gswm*.nc                    |
       imf *.nc           TGCM.tiegcmx.xx.p*.nc
         etc                  pcntr*smin*.nc
                              pcntr*smax*.nc
                                  etc
```

Files listed on the left side refer to data input files that may be needed when running the model in different modes. These are netCDF files, specifically prepared for import into the TIEGCM model (they are **not** model start-up SOURCE files). These files are version-independent (can be used by (almost) any version of the model). They are usually provided to the model as namelist input parameters:

- **gpi*.nc** GeoPhysical Indices data files (3-hourly Kp and F10.7 cm solar flux). Namelist Input parameter: *GPI_NCFILE*
- **gswm*.nc** Global Scale Wave Model data files, used to specify tidal perturbations for the lower boundary of the TIEGCM. There are 4 separate files for diurnal, semi-diurnal, migrating and non-migrating tides. For the namelist input parameters, please see *GSWM*.
- **imf*.nc** Interplanetary Magnetic Field OMNI data files. Namelist read parameter is *IMF_NCFILE*. These files contain data for the BX,BY,BZ components of the IMF, solar wind velocity and solar wind density.

The "tiegcmx.xx/" subdirectory refers to the version of the model that was downloaded. This subdirectory contains start-up *SOURCE* files from *benchmark runs* executed by that version of the model (see Section on *Benchmark Test Runs*). These files can be used to remake the benchmark runs for testing and validation. Here is an example of start-up files provided for benchmark runs made by TIEGCM version 1.94:

```
TGCM.tiegcm1.94.p_dec2006_heelis_gpi_001.nc
TGCM.tiegcm1.94.p_dec2006_weimer_imf_001.nc
TGCM.tiegcm1.94.p_nov2003_heelis_gpi_001.nc
TGCM.tiegcm1.94.p_nov2003_weimer_imf_001.nc
TGCM.tiegcm1.94.p_whi2008_heelis_gpi_001.nc
TGCM.tiegcm1.94.p_whi2008_weimer_imf_001.nc
TGCM.tiegcm1.94.pclim_heelis_001.nc
pcntr_decsol_smax.nc
pcntr_decsol_smin.nc
pcntr_junsol_smax.nc
pcntr_junsol_smin.nc
pcntr_mareqx_smax.nc
pcntr_mareqx_smin.nc
pcntr_sepeqx_smax.nc
pcntr_sepeqx_smin.nc
```

CHAPTER

FOUR

RUN CONTROL PARAMETERS: THE NAMELIST INPUT FILE

The *namelist input* file specifies user-provided run control parameters for a model run. These parameters include the model startup file, start and stop times, solar inputs, lower boundary files, and several other flags and data files as necessary. This document describes each valid namelist parameter, their valid combinations, and provides several example input files for running the model in different modes and resolutions.

4.1 Example Namelist Input Files

Please refer to the following examples of namelist input files:

Note: Any part of a line in the namelist file following a semi-colon will be treated as a comment (see example files).

Example Namelist Input Files:

- The default input file: `default.inp`
- A continuation of the default run: `continuation.inp`
- Saving diagnostics on secondary history files: `diags.inp`
- Full year climatology with constant forcing: `climatology.inp`
- Seasonal control runs with constant forcing: `mareqx.inp`
- Heelis potential model with constant solar forcing: `heelis_smax.inp`
- Heelis potential model with GPI (Kp) data file: `heelis_gpi.inp`
- Weimer potential model with constant IMF forcing: `weimer_con.inp`
- Weimer potential model with IMF data file: `weimer_imf.inp`
- Weimer potential model with IMF+GPI data files: `weimer_imf+gpi.inp`
- Default double-resolution run:
- Example "storm runs":
  - December, 2006 "AGU" storm run: `dec2006.inp`

4.2 Explanation of Valid Namelist Parameters

Following is a table of valid TIEGCM 1.94 namelist parameters, and their descriptions. Follow the parameter name links to explanations below.

Parameter Name	Data Type and Default	Description
AURORA	integer: 1	0/1 flag for auroral parameterization
BXIMF or BXIMF_TIME	real or real array	X-component of the IMF
BYIMF or BYIMF_TIME	real or real array	Y-component of the IMF
BZIMF or BZIMF_TIME	real or real array	Z-component of the IMF
CALENDAR_ADVANCE	real: 1	0/1 switch to advance calendar time
COLFAC	real: 1.5	O-O+ collision factor
CTPOTEN	real:	Cross-Tail Potential
CTPOTEN_TIME	real: [none]	Time-dependent Cross-Tail Potential
F107 or F107_TIME	real or real array	Daily F10.7 cm solar flux
F107A or F107A_TIME	real or real array	81-day average F10.7 cm solar flux
GPI_NCFILE	string: [none]	Geophysical Indices (Kp) data file
GSWM data files	string: [none]	GSWM Model tidal lbc data files
HIST	integer(3)	Primary history write frequency
IMF_NCFILE	string: [none]	IMF OMNI data files
KP or KP_TIME	real or real array	Kp for calc of hpower and ctpoten
LABEL	string:	Arbitrary string identifying the run
MXHIST_PRIM	integer: 10	Max histories on primary file
MXHIST_SECH	integer: 24	Max histories on secondary file
OUTPUT	string array	Primary history output file(s)
POTENTIAL_MODEL	string: [HEELIS]	High-latitude Potential Model
POWER or POWER_TIME	real or real array	Hemispheric Power (GW)
SECSTART	integer(3)	Secondary history start time (day,hour,minute)
SECSTOP	integer(3)	Secondary history stop time (day,hour,minute)
SECHIST	integer(3)	Secondary history write frequency (day,hour,minute)
SECFLDS	string array	Fields to be stored on secondary histories
SECOUT	string array	Secondary history output file(s)
SOURCE	string: [none]	Primary SOURCE (start-up) file
SOURCE_START	integer(3)	Model time to start on SOURCE file
START	integer(3)	Model start time (day,hour,minute)
START_YEAR	integer: 2002	Starting year
START_DAY	integer: 80	Starting day of year
STEP	integer: [none]	Model time step (seconds)
STOP	integer(3)	Model stop time (day,hour,minute)
SWDEN or SWDEN_TIME	real or real array	Solar Wind Density
SWVEL or SWVEL_TIME	real or real array	Solar Wind Velocity
TIDE	real(10)	Amplitudes and phases of semi-diurnal tide (rarely used)
TIDE2	real(2)	Amplitudes and phases of diurnal tide (rarely used)

AURORA

If AURORA > 0 then the auroral parameterization (aurora.F) is called by dynamics (dynamics.F), otherwise it is not called.

Data type: scalar integer

Default: 1

Back to top

BXIMF or BXIMF_TIME

X-component of the IMF. Can be specified as either a constant (BXIMF), or series of time-dependent values (BXIMF_TIME). If IMF_NCFILE is set and BXIMF is not provided, then BXIMF will be taken from the IMF data file.

Data type: real or real array

Examples:

- BXIMF = 0. ; constant for entire run
- BXIMF_TIME = 80,0,0,40., 80,1,0,30., 80,5,0,20. ; time series

See also:

- *BYIMF or BYIMF_TIME*
- *BZIMF or BZIMF_TIME*
- *IMF_NCFILE*

Back to top

BYIMF or BYIMF_TIME

Y-component of the IMF. Can be specified as either a constant (BYIMF), or series of time-dependent values (BYIMF_TIME). If IMF_NCFILE is set and BYIMF is not provided, then BYIMF will be taken from the IMF data file.

Data type: real or real array

Examples:

- BYIMF = 0. ; constant for entire run
- BYIMF_TIME = 80,0,0,40., 80,1,0,30., 80,5,0,20. ; time series

See also:

- *BXIMF or BYIMF_TIME*
- *BZIMF or BZIMF_TIME*
- *IMF_NCFILE*

Back to top

BZIMF or BZIMF_TIME

Z-component of the IMF. Can be specified as either a constant (BZIMF), or series of time-dependent values (BZIMF_TIME). If IMF_NCFILE is set and BZIMF is not provided, then BZIMF will be taken from the IMF data file.

Data type: real or real array

Examples:

- BZIMF = 0. ; constant for entire run
- BZIMF_TIME = 80,0,0,40., 80,1,0,30., 80,5,0,20. ; time series

See also:

- *BXIMF or BXIMF_TIME*
- *BYIMF or BYIMF_TIME*
- *IMF_NCFILE*

Back to top

CALENDAR_ADVANCE

Set CALENDAR_ADVANCE=1 to advance calendar time from START_DAY, otherwise calendar time is not advanced. If advancing calendar time, iday (init_module) is incremented every 24 hours, and the sun's declination and longitude is recalculated (see sub advance_day in advance.F and sub sunloc in magfield.F), thereby allowing seasonal change to take place. The earth's orbital eccentricity "sfeps" is also updated as a 6% variation in solar output over a year.

A run with CALENDAR_ADVANCE=0 is referred to as a "steady-state" run. This is often used to advance the model to a "steady-state" for a given date, prior to a seasonal run with CALENDAR_ADVANCE=1.

Back to top

COLFAC

O-O+ Collision Frequency, alias the "Burnside Factor". Default is 1.5, but there have been recommendations for 1.3. COLFAC is used in lamdas.F and oplus.F.

Data type: real
Default: 1.5

Back to top

CTPOTEN or CTPOTEN_TIME

Cross-tail (or cross-cap) potential. This is used in the auroral precipitation parameterization. It can be provided either as a single constant (CTPOTEN), or several time-dependent values (CTPOTEN_TIME). If GPI_NCFILE is set and CTPOTEN is not provided, it will be calculated from 3-hourly Kp data read from GPI_NCFILE.

The time-dependent example below specifies increasing CTPOTEN from model times 80,0,0 to 80,1,0, and 80,5,0. Interpolated values will be used between these specified model times.

Note that if POTENTIAL_MODEL='WEIMER' or 'WEIMER05', then the user is not allowed to provide CTPOTEN because it will be calculated from the Weimer electric potential.

Data type: real or real array

Examples:

- CTPOTEN = 60.
- CTPOTEN_TIME = 80,0,0,60., 80,1,0,65., 80,5,0,100.

See also:

- *POWER or POWER_TIME*
- *KP or KP_TIME*
- *GPI_NCFILE*

Back to top

F107 or F107_TIME

Daily F10.7 cm solar flux. This can be provided either as a single constant (F107), or several time-dependent values (F107_TIME). If GPI_NCFILE is set and F107 is not set, then F107 will be set from the data. The below example of F107_TIME increases the f10.7 flux from 120 to 150 in the first hour of model time, then to 200 by the fifth hour. Values are linearly interpolated at each time-step.

Data type: real or real array

Examples:

- F107 = 120.
- F107_TIME = 80,0,0,120., 80,1,0,150., 80,5,0,200.

See also:

- *F107A*
- *POTENTIAL_MODEL*
- *GPI_NCFILE*
- *IMF_NCFILE*

Back to top

F107A or F107A_TIME

81-day average F10.7 cm solar flux. This can be provided either as a single constant (F107A), or several time-dependent values (F107A_TIME). If GPI_NCFILE is set and F107A is not set, then F107A will be set from the data. The below example of F107A_TIME increases the f10.7a flux from 120 to 130 in 12 hours of model time.

Data type: real or real array

Examples:

- F107A = 120.
- F107A_TIME = 80,0,0,120., 80,6,0,125., 80,12,0,130.

See also:

- *F107*
- *POTENTIAL_MODEL*
- *GPI_NCFILE*
- *IMF_NCFILE*

Back to top

GPI_NCFILE

Specifies a netCDF data file containing 3-hourly Kp and daily F10.7 data to drive high-latitude convection and the auroral precipitation oval. If GPI_NCFILE is specified, and POTENTIAL_MODEL='HEELIS', then at least one of CTPOTEN,POWER,F107,F107A must **not** be specified. If CTPOTEN or POWER are not specified, they are calculated from the Kp data using empirical relationships (see source file gpi.F). If F107 or F107A are not specified, the data will be used.

If GPI_NCFILE is specified when POTENTIAL_MODEL='WEIMER' and IMF_NCFILE is specified, then the Weimer model and aurora will be driven by the IMF data, and only F107 and F107A will be read from the GPI data file (F107 is not available on IMF data files).

If the current model time is not available on the GPI data file, the model will print an error message to stdout, and stop.

Data Source: Ascii data is obtained from NOAA/NGDC, and an equivalent netCDF data file is written for import to the TGCM models (see code in hao:$TGCMROOT/mkgpi).

Datatype: string

Example:

- GPI_NCFILE = '$TGCMDATA/gpi_2000001-2009031.nc'

See also:

- *CTPOTEN or CTPOTEN_TIME*
- *POWER or POWER_TIME*
- *F107 or F107_TIME*
- *IMF_NCFILE*

Back to top

GSWM model data files for lbc

Paths to netCDF data files containing tidal perturbations from the Global Scale Wave Model. If provided, the files will be read and the perturbations will be added to the lower boundary conditions of T,U,V, and Z. If provided, then TIDE and TIDE2 must be zeroed out.

Warning: As of version 1.94, the model is not tuned to use the non-migrating GSWM tidal components. The default namelist input file specifies migrating diurnal and semi-diurnal tides, but not the non-migrating components. In later releases, non-migrating tides may be supported at the 2.5-deg resolution.

GSWM files must contain data compatable with the lower boundary of the model (99 km), and the horizontal resolution of the model being run (either 5 or 2.5 degrees). See examples below.

Datatype: string

Examples:

- GSWM files for the 5-degree TIEGCM:

```
GSWM_MI_DI_NCFILE   = '$TGCMDATA/gswm_diurn_5.0d_99km.nc'
GSWM_MI_SDI_NCFILE  = '$TGCMDATA/gswm_semi_5.0d_99km.nc'
GSWM_NMI_DI_NCFILE  = '$TGCMDATA/gswm_nonmig_diurn_5.0d_99km.nc'
GSWM_NMI_SDI_NCFILE = '$TGCMDATA/gswm_nonmig_semi_5.0d_99km.nc'
```

- GSWM files for 2.5-degree TIEGCM:

```
GSWM_MI_DI_NCFILE   = '$TGCMDATA/gswm_diurn_2.5d_99km.nc'
GSWM_MI_SDI_NCFILE  = '$TGCMDATA/gswm_semi_2.5d_99km.nc'
GSWM_NMI_DI_NCFILE  = '$TGCMDATA/gswm_nonmig_diurn_2.5d_99km.nc'
GSWM_NMI_SDI_NCFILE = '$TGCMDATA/gswm_nonmig_semi_2.5d_99km.nc'
```

See also:

- *TIDE*
- *TIDE2*

Back to top

HIST

Primary history write frequency, specified as a model time (day,hour,minute). HIST time must divide evenly into STOP minus START times.

Examples:

- HIST = 1,0,0 ;request daily histories
- HIST = 0,1,0 ;request hourly histories
- HIST = 0,0,12 ;request 12-minute histories

See also:

- *SECHIST*

Back to top

POWER or POWER_TIME

Hemispheric Power (GW). This is used in the auroral precipitation parameterization. It can be provided either as a single constant (POWER), or several time-dependent values (POWER_TIME). If GPI_NCFILE is set and POWER is not provided, it will be calculated from 3-hourly Kp data read from GPI_NCFILE.

The time-dependent example below specifies increasing POWER from model times 80,0,0 to 80,1,0, and 80,5,0. Interpolated values will be used between these specified model times.

Data type: real or real array

Examples:

- POWER = 16.
- POWER_TIME = 80,0,0,16., 80,1,0,20., 80,5,0,70.

See also:

- *CTPOTEN or CTPOTEN_TIME*
- *KP or KP_TIME*
- *GPI_NCFILE*

Back to top

KP or KP_TIME

Geomagnetic Activity index. If KP is specified and POWER and/or CTPOTEN are commented, then the given KP will be used with empirical formulas to calculate POWER and/or CTPOTEN, which are used in the Auroral parameterization.

KP can be provided as a scalar constant (KP), or as a series of time-dependent values (KP_TIME), as in the below examples. KP cannot be set if GPI_NCFILE data file is specified.

Empirical formula used to calculate POWER from KP (see function hp_from_kp in util.F):

```
if (kp <=7.) hp_from_kp = 16.82*exp(0.32*kp)-4.86
if (kp > 7.) hp_from_kp = 153.13 + (kp-7.)/(9.-7.)*(300.-153.13)
```

Empirical formula used to calculate CTPOTEN from KP (see function ctpoten_from_kp in util.F):

```
ctpoten_from_kp = 15.+15.*kp + 0.8*kp**2
```

Examples:

- KP = 4.0
- KP_TIME = 80,0,0,4., 80,6,0,4.5, 80,12,0,5.0

See also:

- *CTPOTEN*
- *POWER*
- *GPI_NCFILE*

Back to top

IMF_NCFILE

Specifies a netCDF data file containing hourly IMF parameters BX,BY,BZ,SWVEL, and SWDEN. This can be set only when POTENTIAL_MODEL='WEIMER'. The data will be used to drive the Weimer 2005 potential model. When set, the user must **not** provide at least one of the above IMF parameters. Data will be used for

IMF parameters not provided by the user. Values (scalar or time-dependent) that are provided by the user will take precedence over the data file.

If the current model time is not available on the IMF data file, the model will print an error message to stdout and stop.

Notes on creation of IMF OMNI data files:

- IMF data is derived from 1-minute OMNI satellite data available on CDAweb CDAweb. Our derivation is a multi-step process:
- Data gaps in the raw 1-minute OMNI data are linearly interpolated. If a gap happens to occur at the beginning or end of the time interval, it is set to the next known good data point.
- Gap-filled data is used to compute a 15 minute trailing average lagged by 5 minutes.
- Time averaged data is sampled at 5 minutes
- A data quality flag is calculated for every 5-minute sample point. The data quality flag is a boolean value set to "1" for all sample points derived from valid (not gap-filled) data. The data quality flag is set to "0" for any sample point that is derived from gap-filled data anywhere in the 15 minute trailing average lagged by 5 minutes.
- The data quality flag is stored in the NetCDF-formatted IMF input file. For any variable (ie. "swvel" - solar wind velocity), there exists a mask (ie. "velMask"). Find a complete list of IMF variables with command "ncdump -h [imf-file.nc]".
- Note: You should verify the IMF data quality before doing storm simulations. Known periods of invalid IMF data include approximately days 301 to 304 of 2003 (during the "Halloween Storm").

Example:

- IMF_NCFILE = '$TGCMDATA/imf_OMNI_2002001-2002365.nc'

Back to top

LABEL

LABEL may be any string up to 80 characters long, used to identify a run. The LABEL is written to output history files as a global file attribute. This parameter is purely a user convenience, and does not effect the model run in any way.

Data type: string
Default: 'tiegcm res=5'

Back to top

MXHIST_PRIM

Maximum number of histories to be written to primary OUTPUT files. When this many histories have been written to the current OUTPUT file, the next OUTPUT file is created and it receives subsequent histories. This parameter can be adjusted to control the size of primary OUTPUT files.

Data type: integer
Default: 10

Examples:

- MXHIST_PRIM = 15 ; allow maximum of 15 histories per primary output file

See also:

- *OUTPUT*

Back to top

MXHIST_SECH

Maximum number of histories to be written to secondary output files (SECOUT). When this many histories have been written to the current SECOUT file, the next SECOUT file is created and it receives subsequent histories. This parameter can be adjusted to control the size of secondary OUTPUT files.

Data type: integer
Default: 24

Examples:

- MXHIST_SECH = 24 ; allow 1 day of hourly histories per file
- MXHIST_SECH = 48 ; allow 2 days of hourly histories per file

See also:

- *SECOUT*

Back to top

OUTPUT

List of primary history output files. Each file may be an absolute path, or relative to the execution directory. If an initial run (SOURCE is specified), then pre-existing OUTPUT files will be overwritten. If a continuation run (SOURCE is *not* specified), then the first OUTPUT file should contain the source history at START time. In this case, subsequent output histories will be appended to the first OUTPUT file until it is full. As each OUTPUT file is filled (see MXHIST_PRIM), the next OUTPUT file is created and histories are written until it is full, and so on.

OUTPUT files are usually specified with increasing integers imbedded in the names. See examples below. As a convenience, large sequences of files may be specified in a "short-form", see example 3 below specifying 20 files. By convention, primary history output files may use the letter "p" to indicate primary file series (see all 3 examples below, and contrast with SECOUT).

Examples:

```
OUTPUT = 'p_myoutput_001.nc'
OUTPUT = 'myoutput.p001.nc','myoutput.p002.nc','myoutput.p003.nc'
OUTPUT = 'myoutput_p001.nc','to','myoutput_p020.nc','by','1'
```

See also:

- *SECOUT*
- *SOURCE*
- *MXHIST_PRIM*

Back to top

POTENTIAL_MODEL

The high-latitude potential model used to calculate electric potential above a specified latitude. This string can have one of two values:

POTENTIAL_MODEL = 'HEELIS'
POTENTIAL_MODEL = 'WEIMER'

'HEELIS' is the Rod Heelis model (heelis.F). 'WEIMER' is the Dan Weimer 2005 model (wei05sc.F).

Note: The Weimer model of high-latitude potential is the intellectual property of Daniel Weimer and may not be extracted, distributed, or used for any purpose other than as implemented in the TIE-GCM. For further information concerning this model, please contact Dan Weimer (dweimer@vt.edu).

For a brief discussion of the use of the Weimer 2005 model in TIEGCM, please see *Notes on Weimer05 in TIEGCM*.

Data type: string
Default: 'HEELIS'

Back to top

SECFLDS

List of fields to be saved to secondary histories. These may be either fields that are also saved on primary histories (so-called "prognostic" fields), fields that have been requested via addfld calls in the source code, or fields available via the diagnostics module (see example below).

Note the final size of secondary output files is affected by the number of fields specified as well as the number of histories on the file. The file size can be controlled by setting the number of histories allowed on a secondary file *MXHIST_SECH*.

Data type: one or more character strings

Examples:

```
;
; Example for tiegcm1.9: all fields are "prognostic" except EEX,EEY,EEZ,
; which are saved by addfld calls in sub ionvel (ionvel.F).
;
 SECFLDS = 'TN','UN','VN','O1','NO','N4S','NE','TE','TI',
           'O2','O2P','OMEGA','POTEN','EEX','EEY','EEZ'
;
; This example lists all diagnostic fields available via the diags module
; (it is not necessary to call addfld in the code to obtain these fields)
;
 SECFLDS = 'CO2_COOL','NO_COOL','DEN','HEATING','QJOULE',
           'SIGMA_PED','SIGMA_HAL','TEC','UI_ExB','VI_ExB','WI_ExB',
           'LAMDA_PED','LAMDA_HAL','HMF2','NMF2','SCHT','MU_M'
```

See also: *MXHIST_SECH*

Back to top

SECSTART

Secondary history start time, specified as a model time (day,hour,minute).

Data type: 3 integers (day,hour,minute)
Valid range: 0-365 for day, 0-23 for hour, 0-59 for minute.

SECSTART time must follow these rules:

- It must be a multiple of timestep STEP and less than SECSTOP time.
- It must be greater than or equal to START time, and less than or equal to STOP time.
- In the case of an initial run (SOURCE history provided), SECSTART must not be equal to START time. This is to avoid zero valued secondary history fields.

Examples:

- SECSTART = 80,1,0 ; Start saving secondary histories at model time 80,1,0

See also:

- *SECSTOP*
- *SECHIST*

Back to top

SECSTOP

Secondary history stop time, specified as a model time (day,hour,minute).

Data type: 3 integers (day,hour,minute)
Valid range: 0-365 for day, 0-23 for hour, 0-59 for minute.

SECSTOP time must follow these rules:

- It must be a multiple of timestep STEP and greater than SECSTART time.
- It must be greater than or equal to START time, and less than or equal to STOP time.

Examples:

- SECSTOP = 81,0,0 ; Start saving secondary histories at model time 80,1,0

See also:

- *SECSTART*
- *SECHIST*

Back to top

SECHIST

Secondary history write frequency, specified as a model time (day,hour,minute). SECHIST time must divide evenly into SECSTOP minus SECSTART times.

Data type: 3 integers (day,hour,minute)
Valid range: 0-365 for day, 0-23 for hour, 0-59 for minute.

Examples:

- SECHIST = 0,1,0 ;request hourly histories
- SECHIST = 0,0,12 ;request 12-minute histories

See also:

- *HIST*

Back to top

SECOUT

List of secondary history output files. Secondary histories store diagnostic fields, usually at a higher temporal resolution than primary files. Each file may be an absolute path, or relative to the execution directory. Beware that SECOUT will overwrite any pre-existing files with the same names. As each SECOUT file is filled (see MXHIST_SECH), the next SECOUT file is created and histories are written until it is full, and so on.

SECOUT files are usually specified with increasing integers imbedded in the names. See examples below. As a convenience, large sequences of files may be specified in a "short-form", see example 3 below specifying 20 files. By convention, secondary history output files may use the letter "s" to indicate secondary file series (see all 3 examples below).

Examples:

```
SECOUT = 's_myoutput_001.nc'
SECOUT = 'myoutput.s001.nc','myoutput.s002.nc','myoutput.s003.nc'
SECOUT = 'myoutput_s001.nc','to','myoutput_s020.nc','by','1'
```

See also:

- *OUTPUT*
- *SOURCE*
- *MXHIST_SECH*

Back to top

SOURCE

SOURCE is the start-up history file for an initial run. SOURCE may be a full path or relative to the execution directory. It must be a TIEGCM history with the same grid resolution as the model being run. It does not need to be from the same model version as that being run.

If SOURCE is specified, then SOURCE_START, the model time of the history to read on the SOURCE file, must also be specified. The code will search the SOURCE file for the SOURCE_START history. If SOURCE is *not* specified, then the run is a continuation run, and the source history is provided in the first OUTPUT file at START time.

The SOURCE file must be on the local disk. The model will not look for the SOURCE history on any archive file system.

Examples:

- SOURCE = '$TGCMDATA/TGCM.tiegcm1.94.pcntr_eqnx_smed.nc'

See also:

- *SOURCE_START*

Back to top

SOURCE_START

This is the model time of the history to read from the SOURCE file. Model time is specified as a 3-integer triplet: day,hour,minute. If SOURCE is specified, then SOURCE_START must also be specified. If the SOURCE_START history is not found on the SOURCE file, the model will stop and print an appropriate error message to stdout.

Data type: 3 integers

Valid range: 0-365 for day, 0-23 for hour, 0-59 for minute.

Example:

- SOURCE_START = 80,0,0

See also:

- *SOURCE*

Back to top

START

Model time for start of the run. Model time is a 3-integer triplet: day,hour, minute. If CALENDAR_ADVANCE=0, then START day can be any number between 0 and 365. If CALENDAR_ADVANCE=1, then START day must be the same as START_DAY. If an initial run, START time does not have to be the same as SOURCE_START.

Data type: 3 integers Valid range: 0-365 for day, 0-23 for hour, 0-59 for minute.

Examples:

- START = 80,0,0

See also:

- *SOURCE_START*

Back to top

START_DAY

Calendar starting day.

Data type: integer
Default: 80
Valid range: 1 to 365

Back to top

START_YEAR

Starting year for the run.

Data type: integer
Default: 2002

Back to top

STEP

Model time-step in seconds. Default value is 120, although during periods of quiet solar activity, the model will run fine at 180. During periods of intense solar activity (e.g., F10.7 > 200, or high magnitude BZ southward), the model may become numerically unstable. In this case, reducing the timestep to as low as 60 seconds may help the model get through the rough period.

Data type: integer
Default: Usually 120 or 180

Back to top

STOP

Model stop time for the run. Model time is specified as a 3-integer triplet: day,hour,minute.

Data type: 3 integers

Valid range: 0-365 for day, 0-23 for hour, 0-59 for minute.

Example:

- STOP = 81,0,0

Back to top

SWDEN or SWDEN_TIME

Solar Wind Density. Can be specified as either a constant (SWDEN), or series of time-dependent values (SWDEN_TIME). If IMF_NCFILE is set and SWDEN is not provided, then it SWDEN will be taken from the IMF data file.

Data type: real or real array

Examples:

- SWDEN = 4.0 ; constant for entire run
- SWDEN_TIME = 80,0,0,2., 80,1,0,3., 80,2,0,4. ; time series

See also:

- *IMF_NCFILE*

Back to top

SWVEL or SWVEL_TIME

Solar Wind Velocity. Can be specified as either a constant (SWVEL), or series of time-dependent values (SWVEL_TIME). If IMF_NCFILE is set and SWVEL is not provided, then it SWVEL will be taken from the IMF data file.

Data type: real or real array

Examples:

- SWVEL = 400. ; constant for entire run
- SWVEL_TIME = 80,0,0,100., 80,1,0,200., 80,2,0,300. ; time series

See also:

- *IMF_NCFILE*

Back to top

TIDE

Hough mode amplitudes and phases of the semi-diurnal tide. If GSWM tidal perturbations are specified, TIDE should be set to 0.

Note: TIDE and TIDE2 should be specified only for experiments where amplitude and phases of the tides must be used. Normally, GSWM tides are specified instead of TIDE,TIDE2.

Data type: 10 reals

Example:

```
TIDE= 1.9300E+04, 1.5000E+04, 2.3100E+04, 0.7700E+04, 0.1660E+04,
      -2.600E+00,  0.000E+00, -3.300E+00, 4.2000E+00, 5.0000E+00
```

See also:

- *GSWM_MI_SDI_NCFILE*
- *GSWM_NM_SDI_NCFILE*

Back to top

TIDE2

Hough mode amplitudes and phases of the diurnal tide. If GSWM tidal perturbations are specified, TIDE2 should be set to 0.

Note: TIDE and TIDE2 should be specified only for experiments where amplitude and phases of the tides must be used. Normally, GSWM tides are specified instead of TIDE,TIDE2.

Data type: 2 floats

Example:

```
TIDE2 = 4.1E+4, -3.7
```

See also:

- *GSWM_MI_DI_NCFILE*
- *GSWM_NM_DI_NCFILE*

Back to top

CHAPTER

FIVE

STRUCTURE OF OUTPUT HISTORY FILES

5.1 NetCDF Output Files (*netCDF*)

TIEGCM history files are output in netCDF, a self-describing platform-independent data format written and maintained by the UCAR Unidata program.

Each netCDF file contains one or more *histories*, i.e., the state of the model output fields at a discrete instant in *model time*. Here is an example of the metadata content of a sample primary history file: `primary.ncd`. This example file contains six daily histories (days 355 to 360 of 2002). This metadata is obtained via the "ncdump" command in the netCDF utilities. This example ncdump file contains data values for scalar and singly-dimensioned vectors only.

TIEGCM history files are "CF compliant", i.e., they conform to the NetCDF Climate and Forecast (CF) Metadata Convention.

5.2 Primary and Secondary History Files

"Primary" history files contain the "prognostic" fields necessary to restart the model. They can be specified in namelist input as the *SOURCE* file for starting the model in an *initial run* (a *continuation run* does not specify a SOURCE file, and is continued from the START time found on the first OUTPUT file). Typically, daily histories are stored on primary history files.

Fields on primary histories necessary for start-up of the TIEGCM are as follows: TN, UN, VN, O2, O1, N4S, NO, O+, N2D, TI, TE, NE, O2P, OMEGA, Z, POTEN

"Secondary" history files contain diagnostic fields and/or primary history fields. Fields to be saved on the secondary history files are listed by the namelist input parameter *SECFLDS*. Diagnostics can be saved by calling subroutine addfld in the code (addfld.F), or by including one or more of the "standard" diagnostic fields available via the *diagnostics* module (diags.F). Secondary histories are often saved at a higher temporal resolution than primary histories, typically hourly. `Here` is an ncdump of an example secondary history file.

CHAPTER

SIX

SAVING DIAGNOSTIC FIELDS

The diagnostics module (`diags.F`) in the TIEGCM will calculate and save *diagnostic fields* to the secondary histories. The user can add any subset of these fields to the *SECFLDS* parameter list in the namelist input file. See the `diagnostics namelist example`.

6.1 Table of Available Diagnostics

Following is a table of diagnostic fields that can be saved on secondary histories by including the short names in the *SECFLDS* namelist input parameter. Click on the short name to obtain detailed information about the calculation and saving of a diagnostic field.

On the history files, "Short Name" will be the variable name, and "Long Name" and "Units" will be attributes of the variable. "Grid" refers to the number of dimensions (2d lat-lon, or 3d lat-lon-level), and whether the field is on the geographic or geomagnetic grid.

A `text version of the table` is also available, and is printed to stdout by a model run (ordering of the fields in the text table may be different than in the below table).

Short Name	Long Name	Units	Grid
CO2_COOL	CO2 Cooling	erg/g/s	3d geo
NO_COOL	NO Cooling	erg/g/s	3d geo
DEN	Total Neutral Density	g/cm3	3d geo
HEATING	Total Heating	erg/g/s	3d geo
SCHT	Pressure Scale Height	km	3d geo
SIGMA_HAL	Hall Conductivity	S/m	3d geo
SIGMA_PED	Pedersen Conductivity	S/m	3d geo
LAMDA_HAL	Hall Ion Drag Coefficient	1/s	3d geo
LAMDA_PED	Pedersen Ion Drag Coefficient	1/s	3d geo
UI_ExB	Zonal Ion Drift	cm/s	3d geo
VI_ExB	Meridional Ion Drift	cm/s	3d geo
WI_ExB	Vertical Ion Drift	cm/s	3d geo
MU_M	Molecular Viscosity Coefficient	g/cm/s	3d geo
WN	Neutral Vertical Wind	cm/s	3d geo
O_N2	O/N2 Ratio	[none]	3d geo
QJOULE	Joule Heating	erg/g/s	3d geo
QJOULE_INTEG	Height-integrated Joule Heating	erg/cm2/s	2d geo
HMF2	Height of the F2 Layer	km	2d geo
NMF2	Peak Density of the F2 Layer	1/cm3	2d geo
TEC	Total Electron Content	1/cm2	2d geo
JE13D	Eastward current density (3d)	A/m2	3d mag
JE23D	Downward current density (3d)	A/m2	3d mag
JQR	Upward current density (2d)	A/m2	2d mag
KQLAM	Height-integ current density (+north)	A/m	2d mag
KQPHI	Height-integ current density (+east)	A/m	2d mag

6.2 Saving Fields/Arrays from the Source Code

In addition to the "sanctioned" diagnostics, arbitrary 2d and 3d arrays can be saved from the model to secondary histories by inserting a call to subroutine *addfld* (`addfld.F`) in the source code. (See the chapter on *Modifying Source Code* in this document for information about modifying the source code.) There are many examples of this in the source code, just grep on "call addfld". For more information about how to make calls to addfld, please see comments in the addfld.F source file.

Here are a couple of examples of addfld calls from near the end of subroutine qrj (qrj.F). These calls are inside a latitude loop, where the loop variable index is "lat". Normally, in parallel code, subdomains of the field are passed, e.g., lon0:lon1 and lat0:lat1:

```
 call addfld('QO2P'  ,' ',' ',  qo2p(lev0:lev1,lon0:lon1,lat),
|  'lev',lev0,lev1,'lon',lon0,lon1,lat)
 call addfld('QN2P'  ,' ',' ',  qn2p(lev0:lev1,lon0:lon1,lat),
|  'lev',lev0,lev1,'lon',lon0,lon1,lat)
 call addfld('QNP'   ,' ',' ',   qnp(lev0:lev1,lon0:lon1,lat),
|  'lev',lev0,lev1,'lon',lon0,lon1,lat)
```

The calling sequence for subroutine addfld is explained in comments at the top of source file `addfld.F`.

6.3 Details of Diagnostic Field Calculations

CO2_COOL

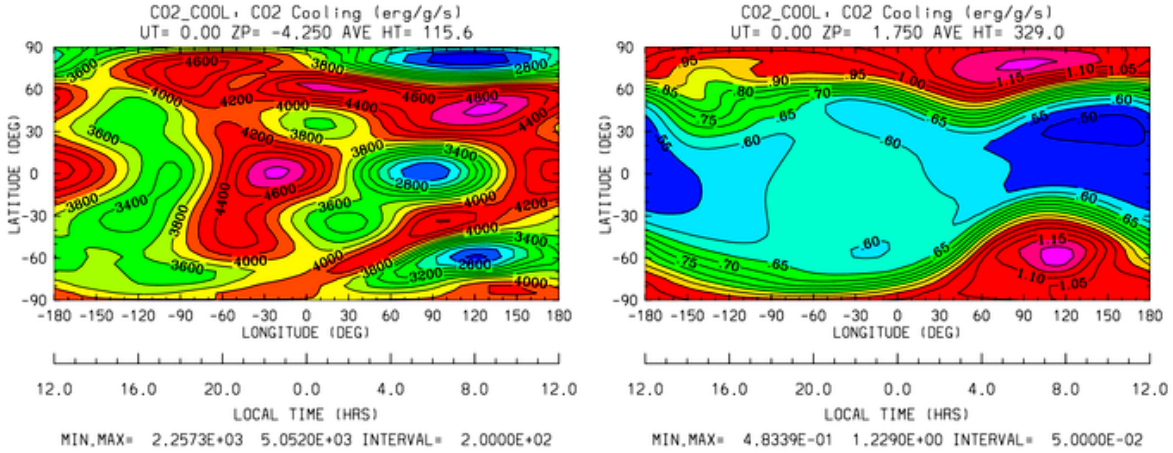
Diagnostic field: CO2 Cooling (erg/g/s):

```
diags(n)%short_name = 'CO2_COOL'
diags(n)%long_name  = 'CO2 Cooling'
diags(n)%units      = 'erg/g/s'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'newton.F'
```

This field is calculated in `newton.F` and passed to mkdiag_CO2COOL (`diags.F`), where it is saved to the secondary history. The calculation of CO2 cooling in newton.F is as follows:

```
      co2_cool(k,i) = 2.65e-13*nco2(k,i)*exp(-960./tn(k,i))*
|       avo*((o2(k,i)*rmassinv_o2+(1.-o2(k,i)-o1(k,i))*rmassinv_n2)*
|       aco2(k,i)+o1(k,i)*rmassinv_o1*bco2(k,i))
```

Sample images: CO2_COOL Global maps at Zp -4, +2:



Back to diagnostics table

NO_COOL

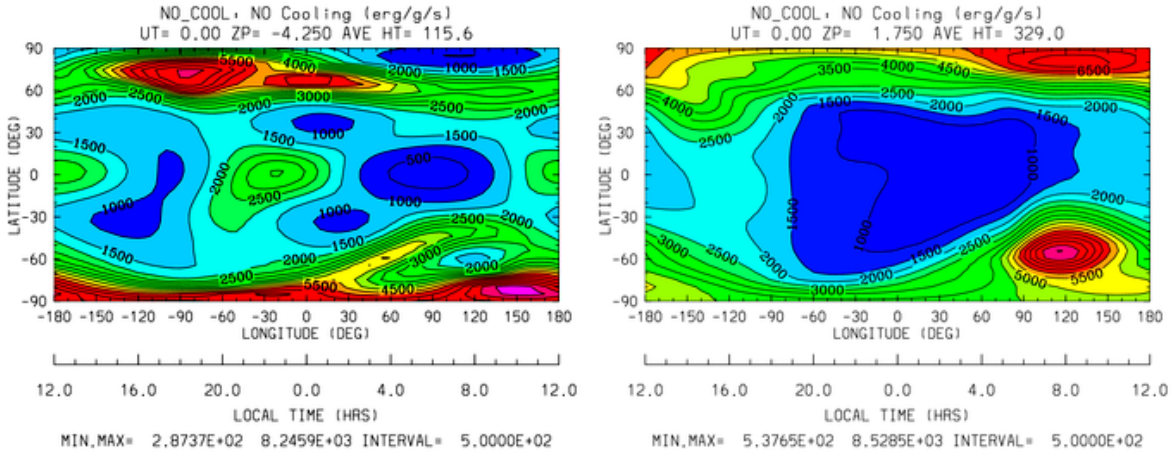
Diagnostic field: NO Cooling (erg/g/s):

```
diags(n)%short_name = 'NO_COOL'
diags(n)%long_name  = 'NO Cooling'
diags(n)%units      = 'erg/g/s'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'newton.F'
```

This field is calculated in `newton.F` and passed to mkdiag_NOCOOL (`diags.F`), where it is saved to the secondary history. The calculation of NO cooling in newton.F is as follows:

```
      no_cool(k,i) = 4.956e-12*(avo*no(k,i)*rmassinv_no)*
|       (ano(k,i)/(ano(k,i)+13.3))*exp(-2700./tn(k,i))
```

Sample images: NO_COOL Global maps at Zp -4, +2:



Back to diagnostics table

DEN

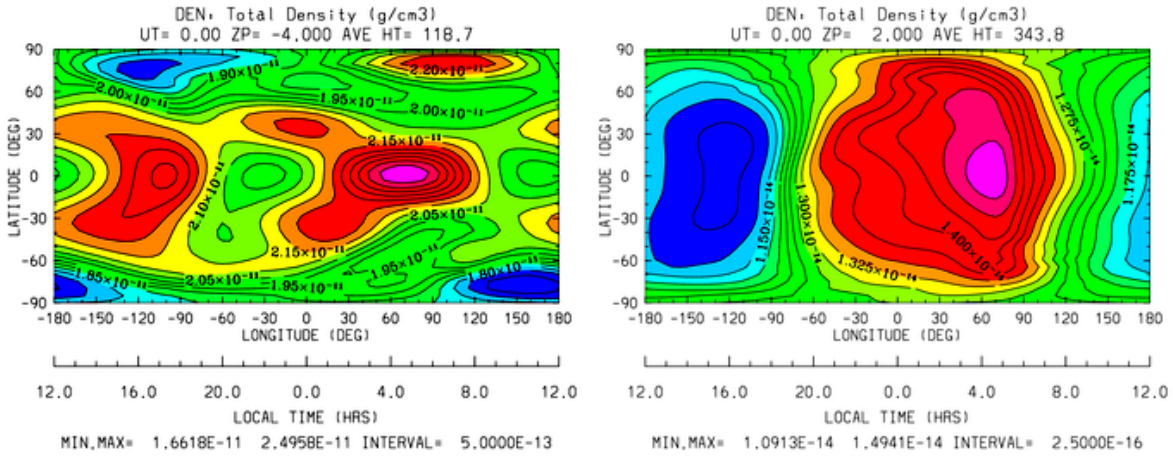
Diagnostic field: Total Density (g/cm3):

```
diags(n)%short_name = 'DEN'
diags(n)%long_name  = 'Total Density'
diags(n)%units      = 'g/cm3'
diags(n)%levels     = 'ilev'
diags(n)%caller     = 'dt.F'
```

This field is calculated in `dt.F` and passed to mkdiag_DEN (`diags.F`), where it is saved to the secondary history. The calculation of DEN (rho) in dt.F is as follows:

```
do i=lon0,lon1
  do k=lev0+1,lev1-1
    tni(k,i) = .5*(tn(k-1,i,lat)+tn(k,i,lat))
    h(k,i) = gask*tni(k,i)/barm(k,i,lat)
    rho(k,i) = p0*expzmid_inv*expz(k)/h(k,i)
  enddo ! k=lev0+1,lev1-1
  rho(lev0,i) = p0*expzmid_inv*expz(lev0)/h(lev0,i)
  rho(lev1,i) = p0*expzmid*expz(lev1-1)/h(lev1,i)
enddo ! i=lon0,lon1
```

Sample images: DEN Global maps at Zp -4, +2:



Back to diagnostics table

HEATING

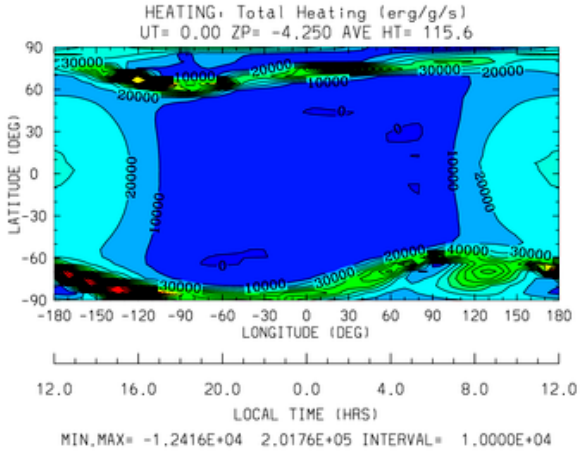
Diagnostic field: Total Heating (erg/g/s):

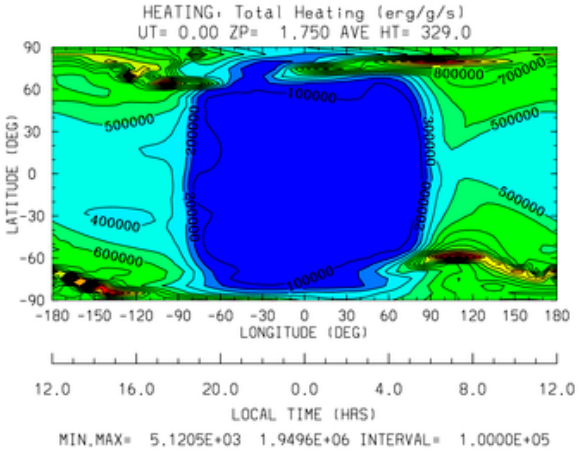
```
diags(n)%short_name = 'HEATING'
diags(n)%long_name  = 'Total Heating'
diags(n)%units      = 'erg/g/s'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'dt.F'
```

This field is calculated in `dt.F` and passed to mkdiag_HEAT (`diags.F`), where it is saved to the secondary history. The calculation of HEATING (rho) in dt.F sums the following heat sources:

- Total solar heating (see *qtotal* in `qrj.F`)
- Heating from 4th order horizontal diffusion
- Heating due to atomic oxygen recombination
- Ion Joule heating
- Heating due to molecular diffusion

Sample images: HEATING Global maps at Zp -4, +2:





Back to diagnostics table

HMF2

Diagnostic field (2d lat x lon): Height of the F2 Layer (km):

```
diags(n)%short_name = 'HMF2'
diags(n)%long_name  = 'Height of the F2 Layer'
diags(n)%units      = 'km'
diags(n)%levels     = 'none' ! hmf2 is 2d lon x lat
diags(n)%caller     = 'elden.F'
```

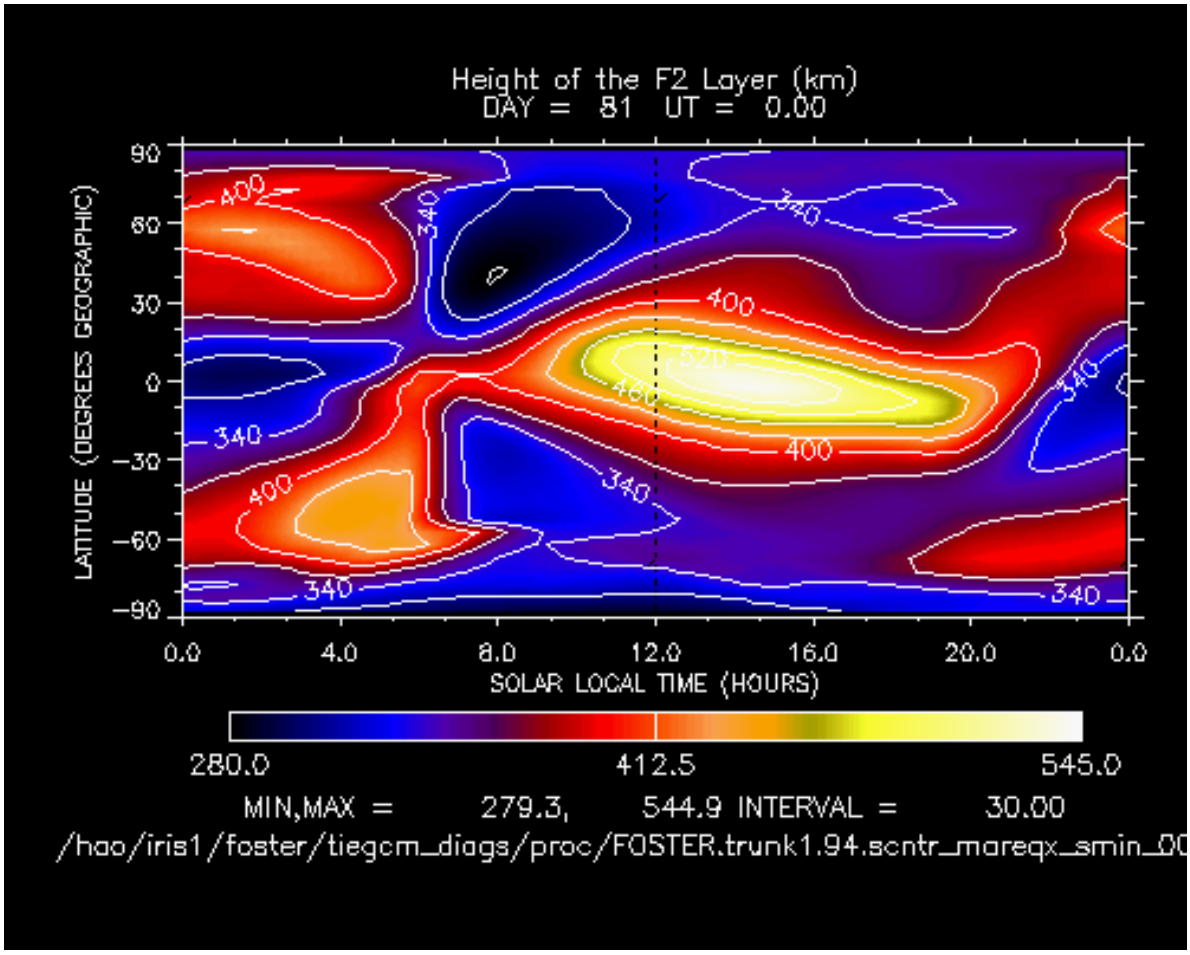
The height of the F2 layer is calculated and saved by subroutines *mkdiag_HNMF2* and *hnmf2* in source file `diags.F`.

Sub *mkdiag_HNMF2* is called by subroutine *elden* in source file `elden.F`, as follows:

call mkdiag_HNMF2('HMF2',z,electrons,lev0,lev1,lon0,lon1,lat)

Note: Occaisionally this algorithm will return the peak electron density in the E-region, instead of the F-region, in small areas of the global domain, usually at high latitide. This can result in pockets of anonymously low values for HMF2, e.g., around 125 km.

Sample images: HMF2 Global map:



Back to diagnostics table

NMF2

Diagnostic field (2d lat x lon): Peak Density of the F2 Layer (1/cm3):

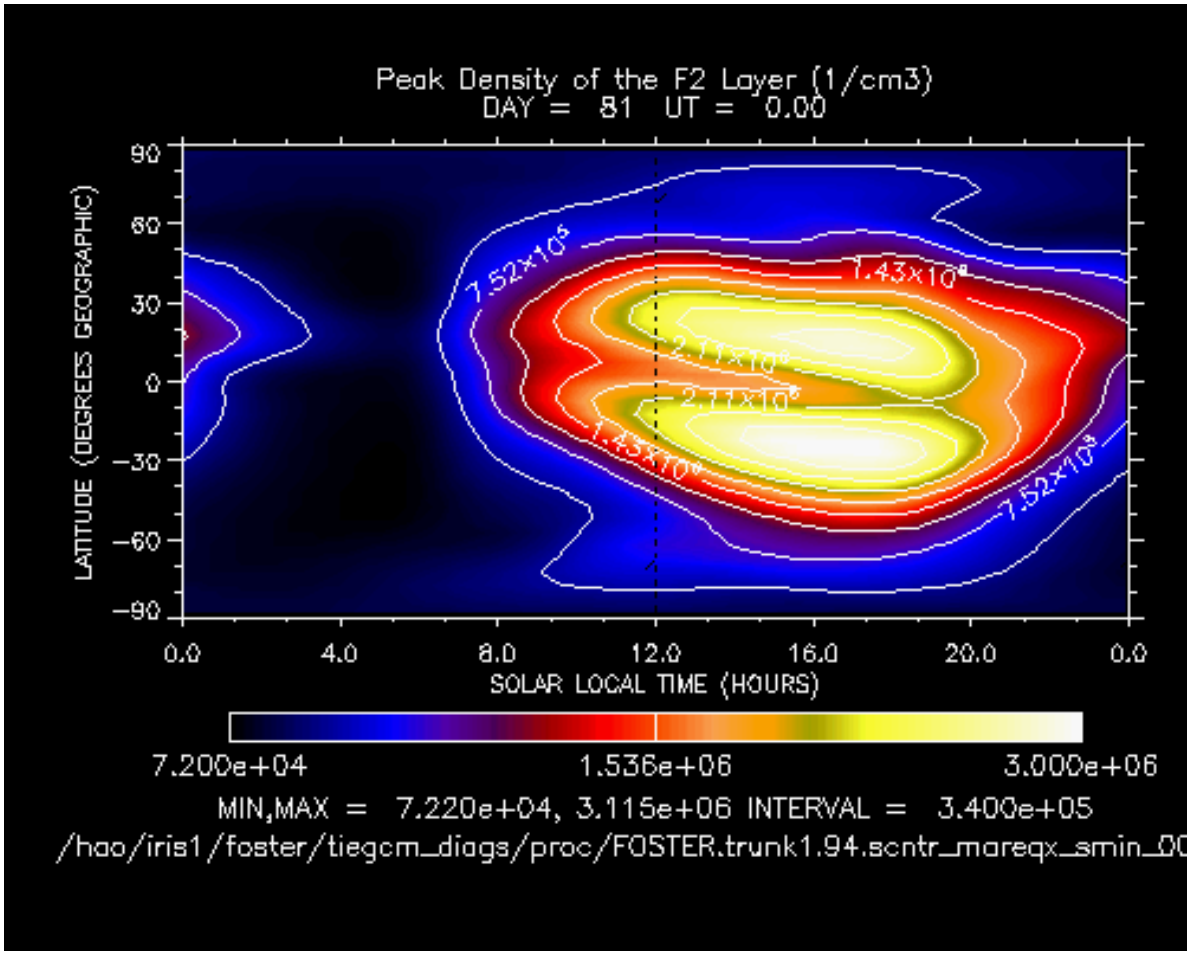
```
diags(n)%short_name = 'NMF2'
diags(n)%long_name  = 'Peak Density of the F2 Layer'
diags(n)%units      = '1/cm3'
diags(n)%levels     = 'none' ! nmf2 is 2d lon x lat
diags(n)%caller     = 'elden.F'
```

The peak density of the the F2 layer is calculated and saved by subroutines *mkdiag_HNMF2* and *hnmf2* in source file `diags.F`.

Sub *mkdiag_HNMF2* is called by subroutine *elden* in source file `elden.F`, as follows:

```
call mkdiag_HNMF2('NMF2',z,electrons,lev0,lev1,lon0,lon1,lat)
```

Sample images: NMF2 Global map:



Back to diagnostics table

TEC

Diagnostic field (2d lat x lon): Total Electron Content (1/cm2):

```
diags(n)%short_name = 'TEC'
diags(n)%long_name  = 'Total Electron Content'
diags(n)%units      = '1/cm2'
diags(n)%levels     = 'none' ! 2d lon x lat
diags(n)%caller     = 'elden.F'
```

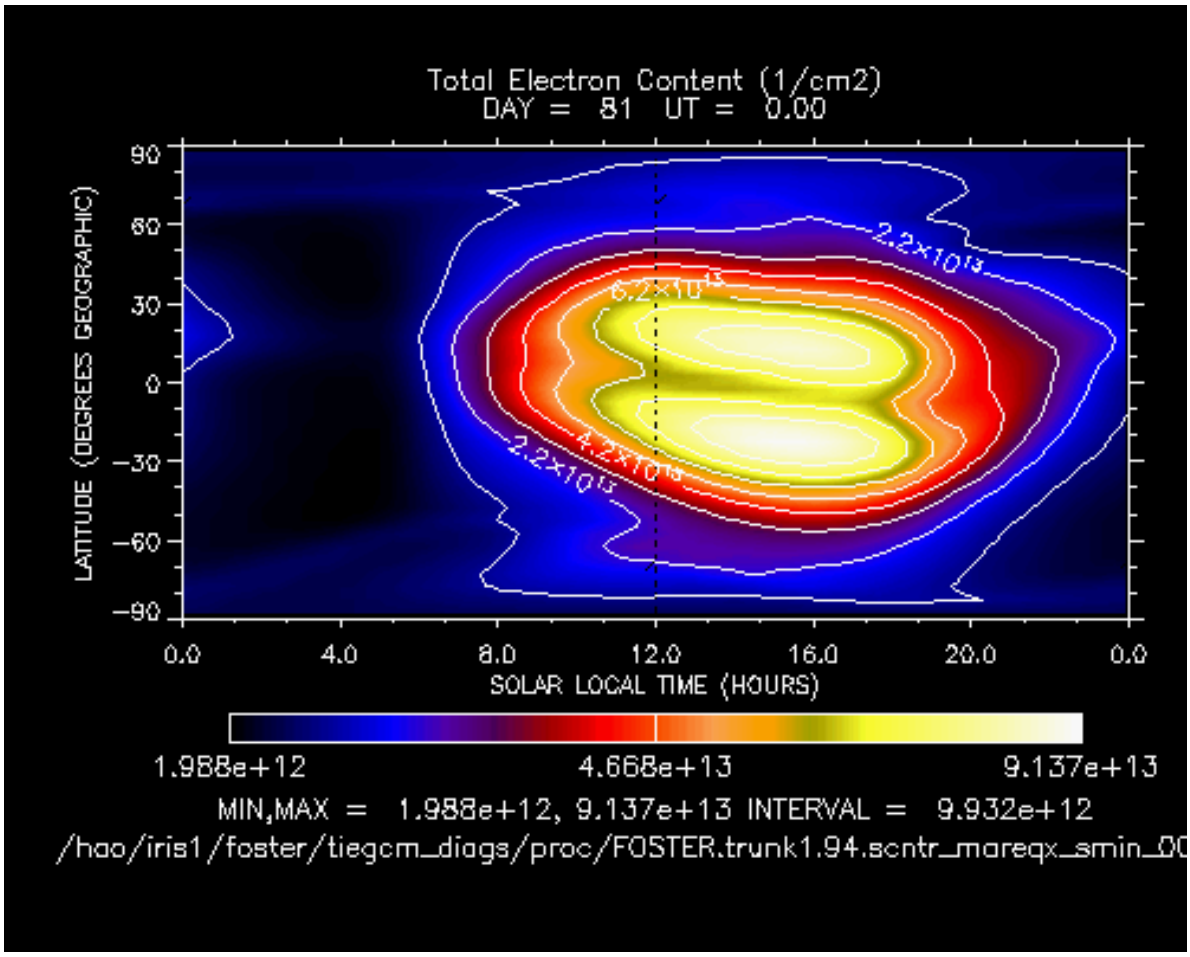
Total Electron Content is calculated by subroutine *mkdiag_TEC* in source file `diags.F`, as follows:

```
!
! Integrate electron content in height at current latitude:
      tec(:) = 0.
      do i=lon0,lon1
        do k=lev0,lev1-1
          tec(i) = tec(i)+(z(k+1,i)-z(k,i))*electrons(k,i)
        enddo
      enddo
```

Subroutine *mkdiags_TEC* is called by subroutine *elden* in source file `elden.F` as follows:

```
call mkdiag_TEC('TEC',tec,z,electrons,lev0,lev1,lon0,lon1,lat)
```

Sample images: TEC Global map



Back to diagnostics table

SCHT

Diagnostic field: Pressure Scale Height (km):

```
diags(n)%short_name = 'SCHT'
diags(n)%long_name  = 'Pressure Scale Height'
diags(n)%units      = 'km'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'addiag.F'
```

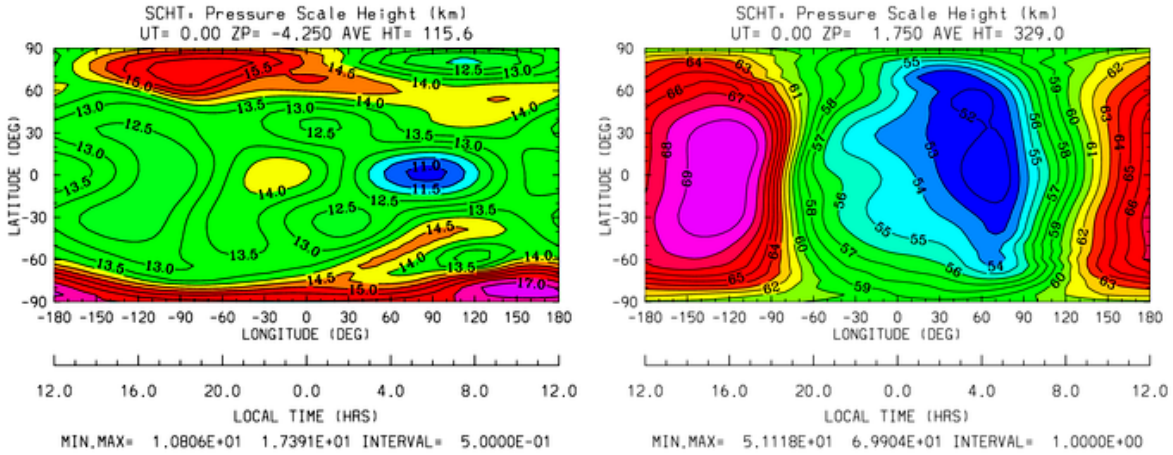
The Pressure Scale Height is calculated from the geopotential and saved by subroutine *mkdiag_SCHT* in source file `diags.F`. This code summarizes the calculation:

```
!
! Take delta Z:
      do j=lat0,lat1
        do i=lon0,lon1
          do k=lev0,lev1-1
            pzps(k,i) = zcm(k+1,i,j)-zcm(k,i,j)
          enddo
```

```
          pzps(lev1,i) = pzps(lev1-1,i)
!
! Generic for dlev 0.5 or 0.25 resolution:
          pzps(:,i) = pzps(:,i)/dlev
        enddo ! i=lon0,lon1
        pzps = pzps*1.e-5 ! cm to km
      enddo ! j=lat0,lat1
```

Subroutine mkdiag_SCHT is called from subroutine *addiag* (source file `addiag.F`).

Sample images: SCHT Global maps at Zp -4, +2:



Back to diagnostics table

SIGMA_HAL

Diagnostic field: Hall Conductivity (S/m):

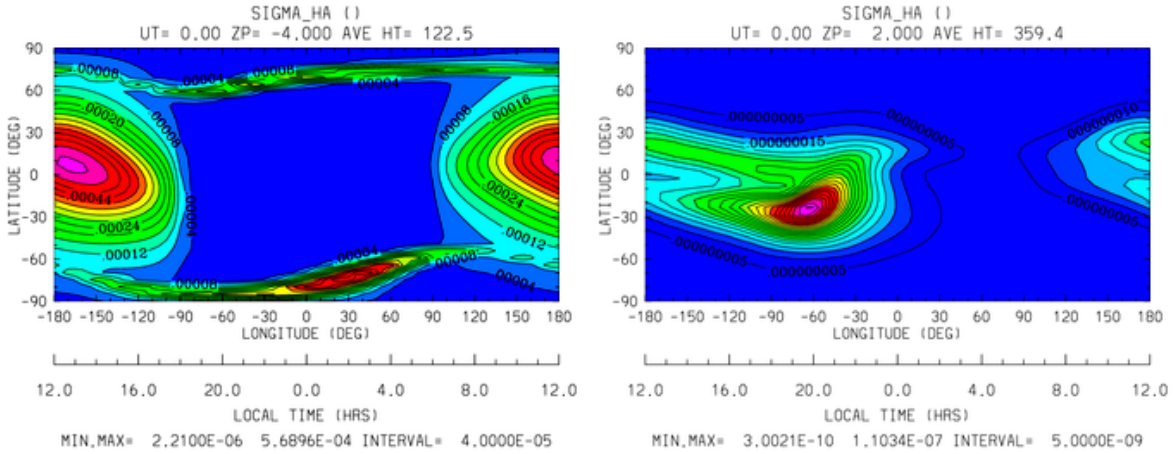
```
diags(n)%short_name = 'SIGMA_HAL'
diags(n)%long_name  = 'Hall Conductivity'
diags(n)%units      = 'S/m'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'lamdas.F'
```

The Hall Conductivity is calculated by subroutine *lamdas* (source file `lamdas.F`), and passed to sub *mkdiag_SIGMAHAL* (`diags.F`), where it is saved to secondary histories. The calculation in lamdas.F is summarized as follows:

```
! Pedersen and Hall conductivities (siemens/m):
! Qe_fac includes conversion from CGS to SI units
!  -> e/B [C/T 10^6 m^3/cm^3], see above.
! number densities [1/cm^3]
!
      do i=lon0,lon1
        do k=lev0,lev1-1
!
! ne = electron density assuming charge equilibrium [1/cm3]:
          ne(k,i) = op(k,i)+o2p(k,i)+nop(k,i)
!
! Hall conductivity [S/m] (half level):
          sigma_hall(k,i) = qe_fac(i)*
|           (ne (k,i)/(1.+rnu_ne (k,i)**2)-
```

```
|             op (k,i)/(1.+rnu_op (k,i)**2)-
|             o2p(k,i)/(1.+rnu_o2p(k,i)**2)-
|             nop(k,i)/(1.+rnu_nop(k,i)**2))
        enddo ! k=lev0,lev1-1
      enddo ! i=lon0,lon1
```

Sample images: SIGMA_HAL Global maps at Zp -4, +2:



Back to diagnostics table

SIGMA_PED

Diagnostic field: Pedersen Conductivity (S/m):

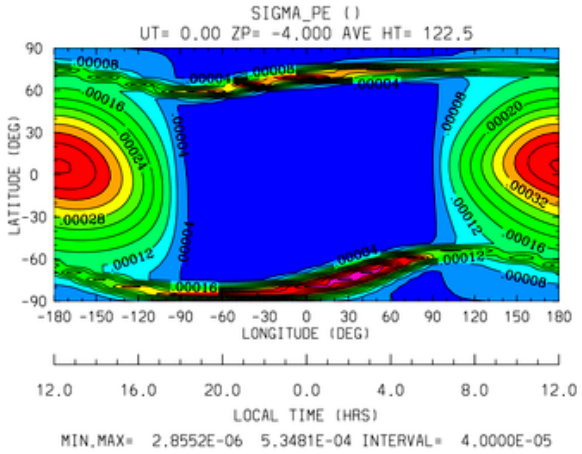
```
diags(n)%short_name = 'SIGMA_PED'
diags(n)%long_name  = 'Pedersen Conductivity'
diags(n)%units      = 'S/m'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'lamdas.F'
```

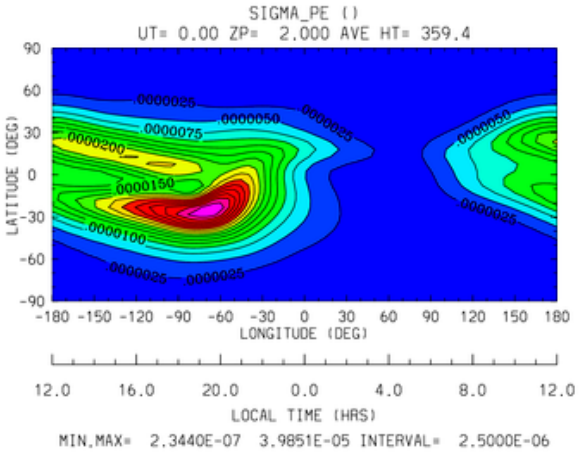
The Pedersen Conductivity is calculated by subroutine *lamdas* (source file `lamdas.F`), and passed to sub *mkdiag_SIGMAPED* (`diags.F`), where it is saved to secondary histories. The calculation in lamdas.F is summarized as follows:

```
! Pedersen and Hall conductivities (siemens/m):
! Qe_fac includes conversion from CGS to SI units
!  -> e/B [C/T 10^6 m^3/cm^3], see above.
! number densities [1/cm^3]
!
      do i=lon0,lon1
        do k=lev0,lev1-1
!
! ne = electron density assuming charge equilibrium [1/cm3]:
          ne(k,i) = op(k,i)+o2p(k,i)+nop(k,i)
!
! Pedersen conductivity [S/m] (half level):
          sigma_ped(k,i) = qe_fac(i)*
     |       ((op (k,i)*rnu_op (k,i)/(1.+rnu_op (k,i)**2))+
     |        (o2p(k,i)*rnu_o2p(k,i)/(1.+rnu_o2p(k,i)**2))+
     |        (nop(k,i)*rnu_nop(k,i)/(1.+rnu_nop(k,i)**2))+
     |        (ne (k,i)*rnu_ne (k,i)/(1.+rnu_ne (k,i)**2)))
```

```
      enddo ! k=lev0,lev1-1
    enddo ! i=lon0,lon1
```

Sample images: SIGMA_PED Global maps at Zp -4, +2:





Back to diagnostics table

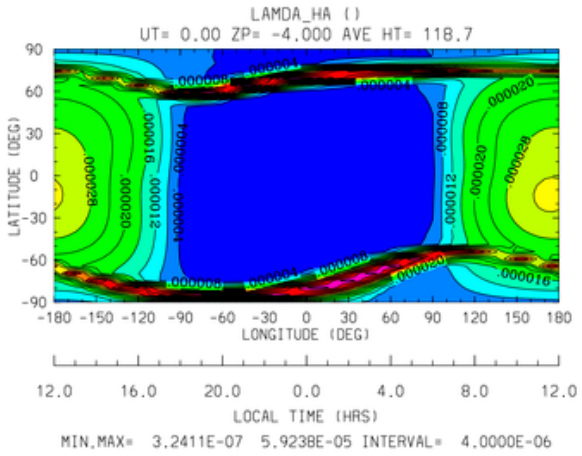
LAMDA_HAL

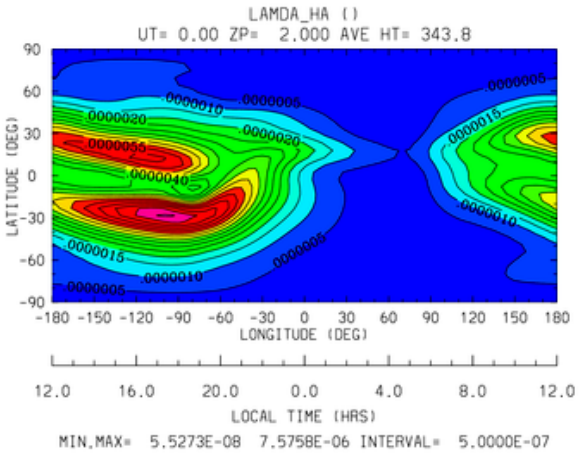
Diagnostic field: Hall Ion Drag Coefficient (1/s):

```
diags(n)%short_name = 'LAMDA_HAL'
diags(n)%long_name  = 'Hall Ion Drag Coefficient'
diags(n)%units      = '1/s'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'lamdas.F'
```

The Hall Ion Drag Coefficient is calculated in subroutine *lamdas* (source file `lamdas.F`), and saved to seconday histories by subroutine *mkdiag_LAMDAHAL* (`diags.F`).

Sample images: LAMDA_HAL Global maps at Zp -4, +2:





Back to diagnostics table

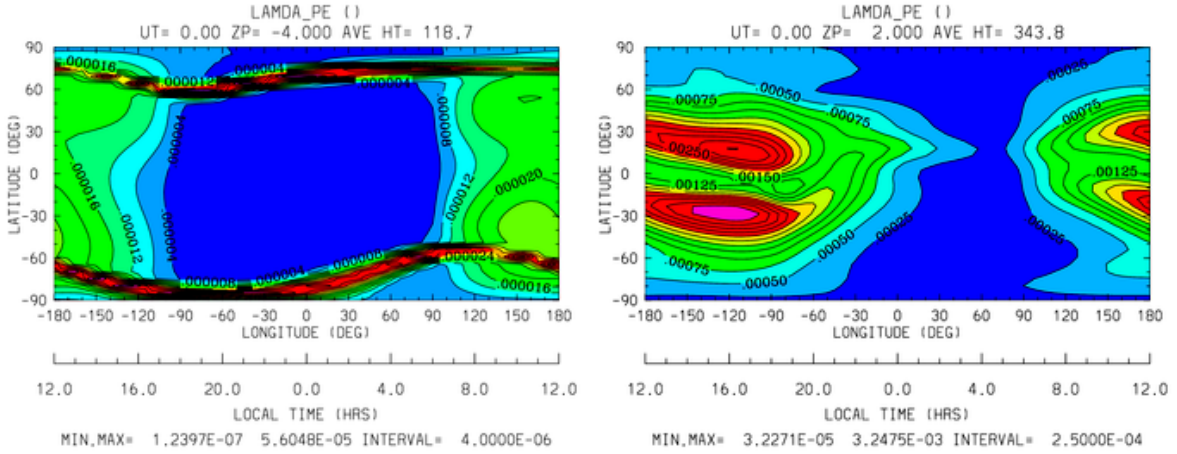
LAMDA_PED

Diagnostic field: Hall Ion Drag Coefficient (1/s):

```
diags(n)%short_name = 'LAMDA_PED'
diags(n)%long_name  = 'Pedersen Ion Drag Coefficient'
diags(n)%units      = '1/s'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'lamdas.F'
```

The Pedersen Ion Drag Coefficient is calculated in subroutine *lamdas* (source file `lamdas.F`), and saved to secondary histories by subroutine *mkdiag_LAMDAPED* (`diags.F`).

Sample images: LAMDA_PED Global maps at Zp -4, +2:



Back to diagnostics table

UI_ExB

Diagnostic field: Zonal Ion Drift (ExB) (cm/s):

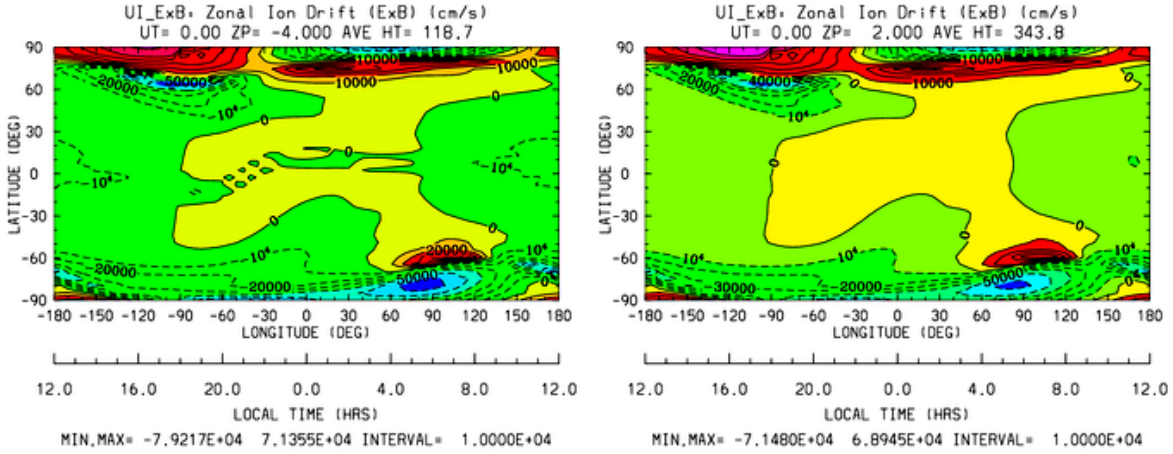
```
diags(n)%short_name = 'UI_ExB'
diags(n)%long_name  = 'Zonal Ion Drift (ExB)'
diags(n)%units      = 'cm/s'
diags(n)%levels     = 'ilev'
diags(n)%caller     = 'ionvel.F'
```

Calculated by subroutine *ionvel* (`ionvel.F`):

```
!
! ion velocities = (e x b/b**2)
! ui = zonal, vi = meridional, wi = vertical
        do k=lev0,lev1
          do i=lonbeg,lonend
            ui(k,i,lat) = -(eey(k,i)*zb(i-2,lat)+eez(k,i)*xb(i-2,lat))*
|             1.e6/bmod(i-2,lat)**2
            vi(k,i,lat) =  (eez(k,i)*yb(i-2,lat)+eex(k,i)*zb(i-2,lat))*
|             1.e6/bmod(i-2,lat)**2
            wi(k,i,lat) =  (eex(k,i)*xb(i-2,lat)-eey(k,i)*yb(i-2,lat))*
|             1.e6/bmod(i-2,lat)**2
          enddo ! i=lon0,lon1
        enddo ! k=lev0,lev1
```

Subroutine ionvel calls subroutine *mkdiag_UI* (`diags.F`) to save the field to secondary histories. The field is converted from m/s to cm/s in ionvel before the call to mkdiag_UI.

Sample images: UI_ExB Global maps at Zp -4, +2:



Back to diagnostics table

VI_ExB

Diagnostic field: Meridional Ion Drift (ExB) (cm/s):

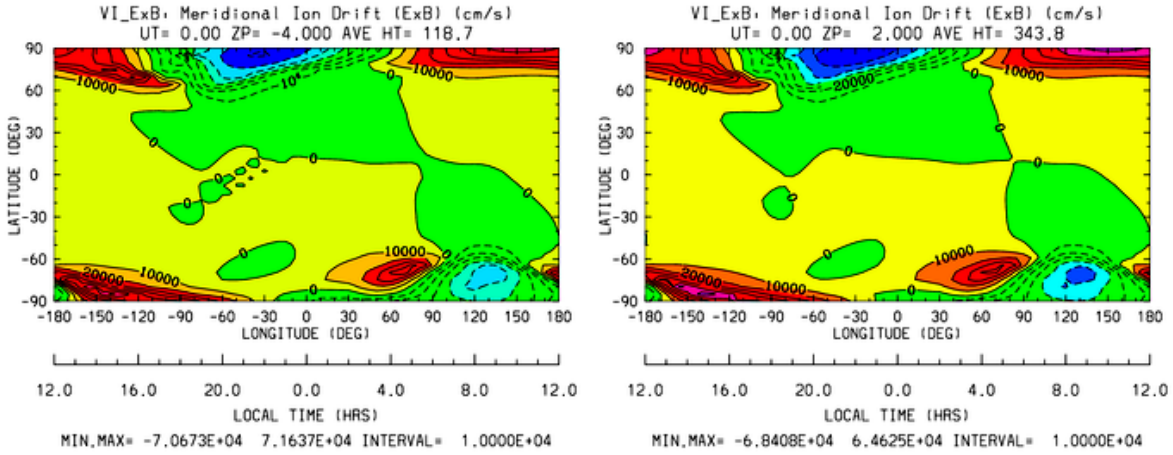
```
diags(n)%short_name = 'VI_ExB'
diags(n)%long_name  = 'Meridional Ion Drift (ExB)'
diags(n)%units      = 'cm/s'
diags(n)%levels     = 'ilev'
diags(n)%caller     = 'ionvel.F'
```

Calculated by subroutine *ionvel* (`ionvel.F`):

```
!
! ion velocities = (e x b/b**2)
! ui = zonal, vi = meridional, wi = vertical
        do k=lev0,lev1
          do i=lonbeg,lonend
            ui(k,i,lat) = -(eey(k,i)*zb(i-2,lat)+eez(k,i)*xb(i-2,lat))*
|             1.e6/bmod(i-2,lat)**2
            vi(k,i,lat) =  (eez(k,i)*yb(i-2,lat)+eex(k,i)*zb(i-2,lat))*
|             1.e6/bmod(i-2,lat)**2
            wi(k,i,lat) =  (eex(k,i)*xb(i-2,lat)-eey(k,i)*yb(i-2,lat))*
|             1.e6/bmod(i-2,lat)**2
          enddo ! i=lon0,lon1
        enddo ! k=lev0,lev1
```

Subroutine ionvel calls subroutine *mkdiag_VI* (`diags.F`) to save the field to secondary histories. The field is converted from m/s to cm/s in ionvel before the call to mkdiag_VI.

Sample images: VI_ExB Global maps at Zp -4, +2:



Back to diagnostics table

WI_ExB

Diagnostic field: Vertical Ion Drift (ExB) (cm/s):

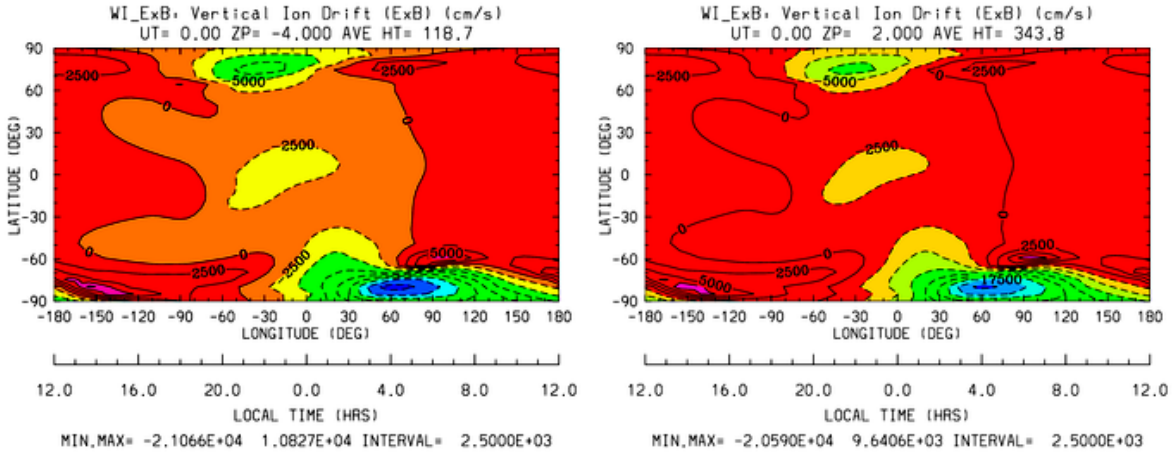
```
diags(n)%short_name = 'WI_ExB'
diags(n)%long_name  = 'Vertical Ion Drift (ExB)'
diags(n)%units      = 'cm/s'
diags(n)%levels     = 'ilev'
diags(n)%caller     = 'ionvel.F'
```

Calculated by subroutine *ionvel* (`ionvel.F`):

```
!
! ion velocities = (e x b/b**2)
! ui = zonal, vi = meridional, wi = vertical
        do k=lev0,lev1
          do i=lonbeg,lonend
            ui(k,i,lat) = -(eey(k,i)*zb(i-2,lat)+eez(k,i)*xb(i-2,lat))*
|             1.e6/bmod(i-2,lat)**2
            vi(k,i,lat) =  (eez(k,i)*yb(i-2,lat)+eex(k,i)*zb(i-2,lat))*
|             1.e6/bmod(i-2,lat)**2
            wi(k,i,lat) =  (eex(k,i)*xb(i-2,lat)-eey(k,i)*yb(i-2,lat))*
|             1.e6/bmod(i-2,lat)**2
          enddo ! i=lon0,lon1
        enddo ! k=lev0,lev1
```

Subroutine ionvel calls subroutine *mkdiag_UI* (`diags.F`) to save the field to secondary histories. The field is converted from m/s to cm/s in ionvel before the call to mkdiag_WI.

Sample images: WI_ExB Global maps at Zp -4, +2:



Back to diagnostics table

MU_M

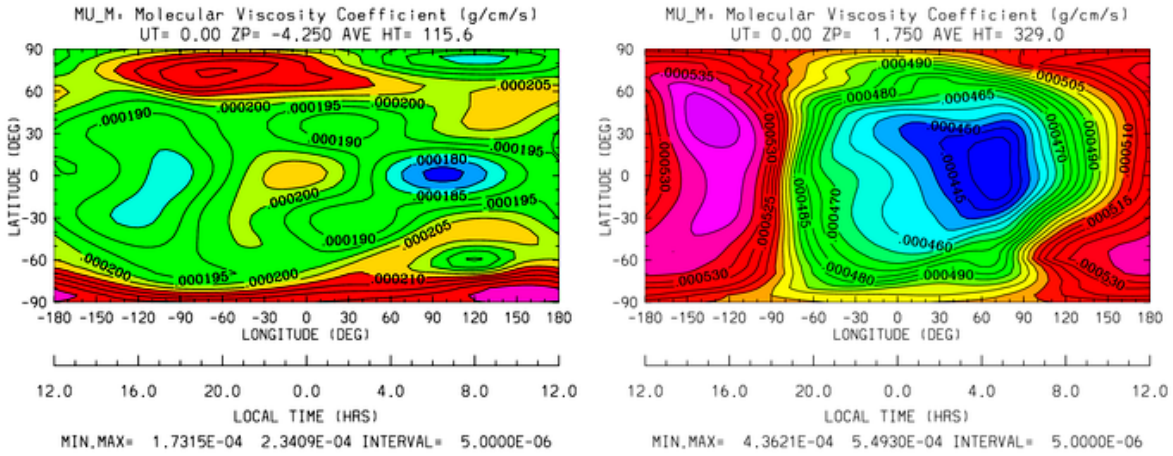
Diagnostic field: Molecular Viscosity Coefficient (g/cm/s):

```
diags(n)%short_name = 'MU_M'
diags(n)%long_name  = 'Molecular Viscosity Coefficient'
diags(n)%units      = 'g/cm/s'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'cpktkm.F'
```

The Molecular Viscosity Coefficient is calculated by subroutine *cpktkm* (source file `cpktkm.F`), and saved to secondary histories by subroutine *mkdiag_MU_M* (`diags.F`). The calculation in cpktkm is summarized as follows:

```
fkm(k,i) = po2(k,i)*4.03 + pn2(k,i)*3.42 + po1(k,i)*3.9
```

Sample images: MU_M Global maps at Zp -4, +2:



Back to diagnostics table

WN

Diagnostic field: Neutral Vertical Wind (cm/s):

```
diags(n)%short_name = 'WN'
diags(n)%long_name  = 'NEUTRAL VERTICAL WIND (plus up)'
diags(n)%units      = 'cm/s'
diags(n)%levels     = 'ilev'
diags(n)%caller     = 'swdot.F'
```

Note: This 3d field is calculated on fixed pressure surfaces ln(p0/p), i.e., there is no interpolation to height.

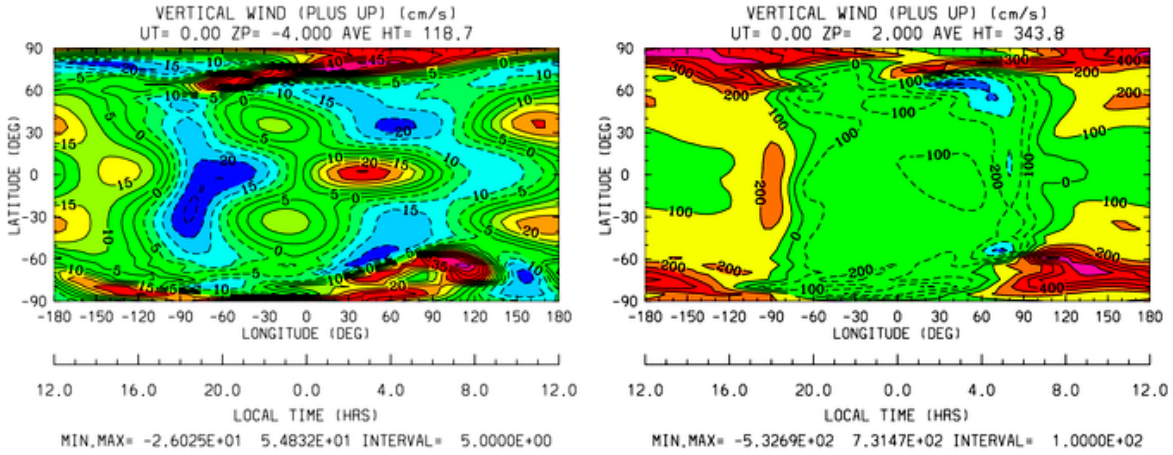
Calculated from OMEGA (vertical motion) and pressure scale height by subroutine *mkdiag_WN* in source file `diags.F`:

```
!-----------------------------------------------------------------------
      subroutine mkdiag_WN(name,omega,zcm,lev0,lev1,lon0,lon1,lat)
!
! Neutral Vertical Wind, from vertical motion OMEGA and scale height.
! Scale height pzps is calculated from input geopotential z (cm).
!
! Args:
      character(len=*),intent(in) :: name
      integer,intent(in) :: lev0,lev1,lon0,lon1,lat
      real,intent(in),dimension(lev0:lev1,lon0:lon1) :: omega,zcm
!
! Local:
      integer :: i,k,ix
      real,dimension(lev0:lev1,lon0:lon1) :: wn
      real,dimension(lev0:lev1) :: pzps,omega1
!
! Check that field name is a diagnostic, and was requested:
      ix = checkf(name) ; if (ix==0) return
!
! Calculate scale height pzps:
      do i=lon0,lon1
        do k=lev0+1,lev1-1
          pzps(k) = (zcm(k+1,i)-zcm(k-1,i))/(2.*dlev)
        enddo
        pzps(lev0) = (zcm(lev0+1,i)-zcm(lev0,i))/dlev
        pzps(lev1) = pzps(lev1-1)
!
        omega1(:) = omega(:,i)
        omega1(lev1) = omega1(lev1-1)
!
! Output vertical wind (cm):
        wn(:,i) = omega1(:)*pzps(:)
      enddo ! i=lon0,lon1

      call addfld(diags(ix)%short_name,diags(ix)%long_name,
     |  diags(ix)%units,wn,'lev',lev0,lev1,'lon',lon0,lon1,lat)

      end subroutine mkdiag_WN
!-----------------------------------------------------------------------
```

Called by: subroutine *swdot* in source file `swdot.F` as follows:

```
do lat=lat0,lat1
  call mkdiag_WN('WN',w(:,lon0:lon1,lat),z(:,lon0:lon1,lat),lev0,lev1,lon0,lon1,lat)
enddo
```

Sample images: WN Global maps at Zp -4, +2:



Back to diagnostics table

O_N2

Diagnostic field: O/N2 RATIO:

```
diags(n)%short_name = 'O_N2'
diags(n)%long_name  = 'O/N2 RATIO'
diags(n)%units      = ' '
diags(n)%levels     = 'lev'
diags(n)%caller     = 'comp.F'
```

Note: Please note that this field is calculated at constant pressure surfaces (ln(p0/p)), and is very sensitive to fluctuations in the height of the pressure surfaces. If this field is interpolated to constant height surfaces, it will look very different than when plotted on pressure surfaces.

Note: Also note that O/N2 is a 3d field (not integrated in the vertical coordinate), and is the quotient of the mixing ratios of the species (i.e., there is no units conversion from MMR).

O/N2 is calculated and saved by subroutine *mkdiag_O_N2* in source file `diags.F`:

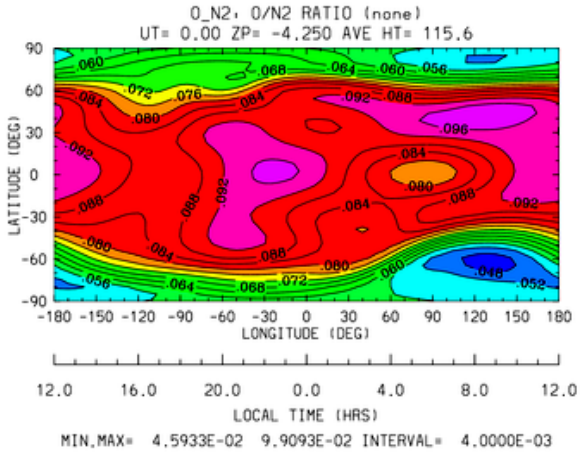
```
!-----------------------------------------------------------------------
      subroutine mkdiag_O_N2(name,o1,o2,lev0,lev1,lon0,lon1,lat)
!
! Calculate O/N2 ratio from o2 and o (mmr).
! In mass mixing ratio, this is simply o/(1-o2-o)
!
! Args:
      character(len=*),intent(in) :: name
      integer,intent(in) :: lev0,lev1,lon0,lon1,lat
      real,intent(in),dimension(lev0:lev1,lon0:lon1) :: o1,o2
!
! Local:
      integer :: ix
      real,dimension(lev0:lev1,lon0:lon1) :: n2, o_n2
!
! Check that field name is a diagnostic, and was requested:
      ix = checkf(name) ; if (ix==0) return
!
```

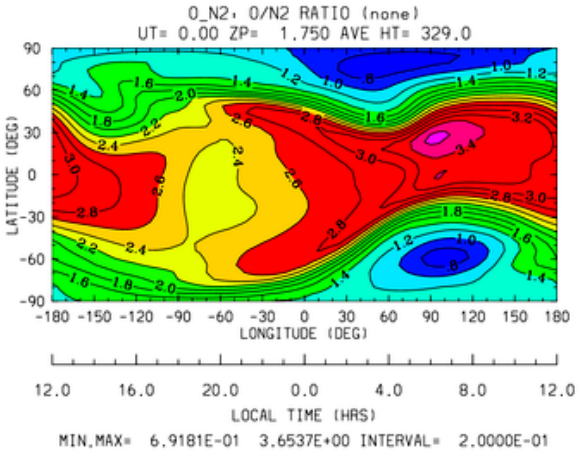
```
! N2 mmr:
      n2 = 1.-o2-o1
!
! O/N2 ratio:
      o_n2 = o1/n2

      call addfld(diags(ix)%short_name,diags(ix)%long_name,
     |  diags(ix)%units,o_n2,'lev',lev0,lev1,'lon',lon0,lon1,lat)

      end subroutine mkdiag_O_N2
!-----------------------------------------------------------------------
```

Called by: subroutine *comp* in source file `comp.F` as follows:

```
   call mkdiag_O_N2('O_N2',o1_upd(:,lon0:lon1,lat),
|    o2_upd(:,lon0:lon1,lat),lev0,lev1,lon0,lon1,lat)
```

Sample images: O_N2 Global maps at Zp -4, +2:





Back to diagnostics table

QJOULE

Diagnostic field: Joule Heating (erg/g/s):

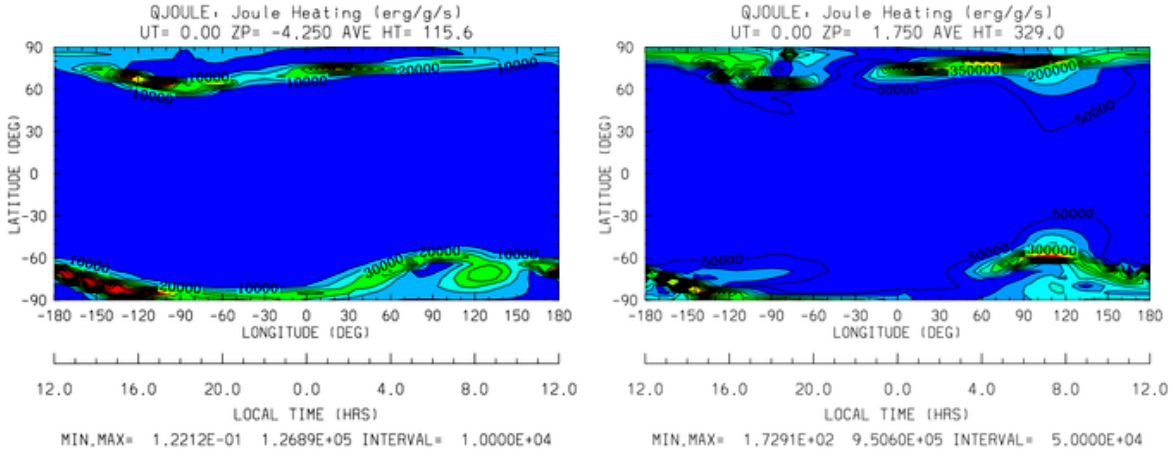
```
diags(n)%short_name = 'QJOULE'
diags(n)%long_name  = 'Joule Heating'
diags(n)%units      = 'erg/g/s'
diags(n)%levels     = 'lev'
diags(n)%caller     = 'qjoule.F'
```

Total Joule Heating is calculated in source file `qjoule.F` as *qji_tn*, and is passed to subroutine *mkdiag_QJOULE* (`diags.F`), where it is saved to secondary histories. The following code summarizes the calculation in qjoule.F:

```
 do i=lon0,lon1
   do k=lev0,lev1-1
     scheight(k,i) = gask*tn(k,i)/
|         (.5*(barm(k,i)+barm(k+1,i))*grav)
     vel_zonal(k,i) = .5*(ui(k,i)+ui(k+1,i))-un(k,i)   ! s2
     vel_merid(k,i) = .5*(vi(k,i)+vi(k+1,i))-vn(k,i)   ! s3
     vel_vert(k,i)  = .5*(wi(k,i)+wi(k+1,i)-scheight(k,i)*
```

```
|          ( w(k,i)-w(k+1,i)) )
   enddo ! k=lev0,lev1-1
 enddo ! i=lon0,lon1
 do i=lon0,lon1
   do k=lev0,lev1-1
     qji_tn(k,i) = .5*(lam1(k,i)+lam1(k+1,i))*
|      (vel_zonal(k,i)**2 + vel_merid(k,i)**2 +
|       vel_vert(k,i)**2)
   enddo ! k=lev0,lev1-1
 enddo ! i=lon0,lon1

 call mkdiag_QJOULE('QJOULE',qji_tn,lev0,lev1,lon0,lon1,lat)
```

Sample images: QJOULE Global maps at Zp -4, +2:



Back to diagnostics table

QJOULE_INTEG

Diagnostic field: Height-integrated Joule Heating (W/m^2):

```
diags(n)%short_name = 'QJOULE_INTEG'
diags(n)%long_name  = 'Height-integrated Joule Heating'
diags(n)%units      = 'erg/cm2/s'
diags(n)%levels     = 'none'
diags(n)%caller     = 'qjoule.F'
```

Note: This field is integrated on pressure surfaces (not height), so is a 2d field. Also note it is first calculated in W/m^2, then converted to erg/g/cm2, for consistency with the model. See comment below if you would like the field to be returned in W/m^2.

Calculated and saved by subroutine *mkdiag_QJOULE_INTEG* in source file `diags.F`:

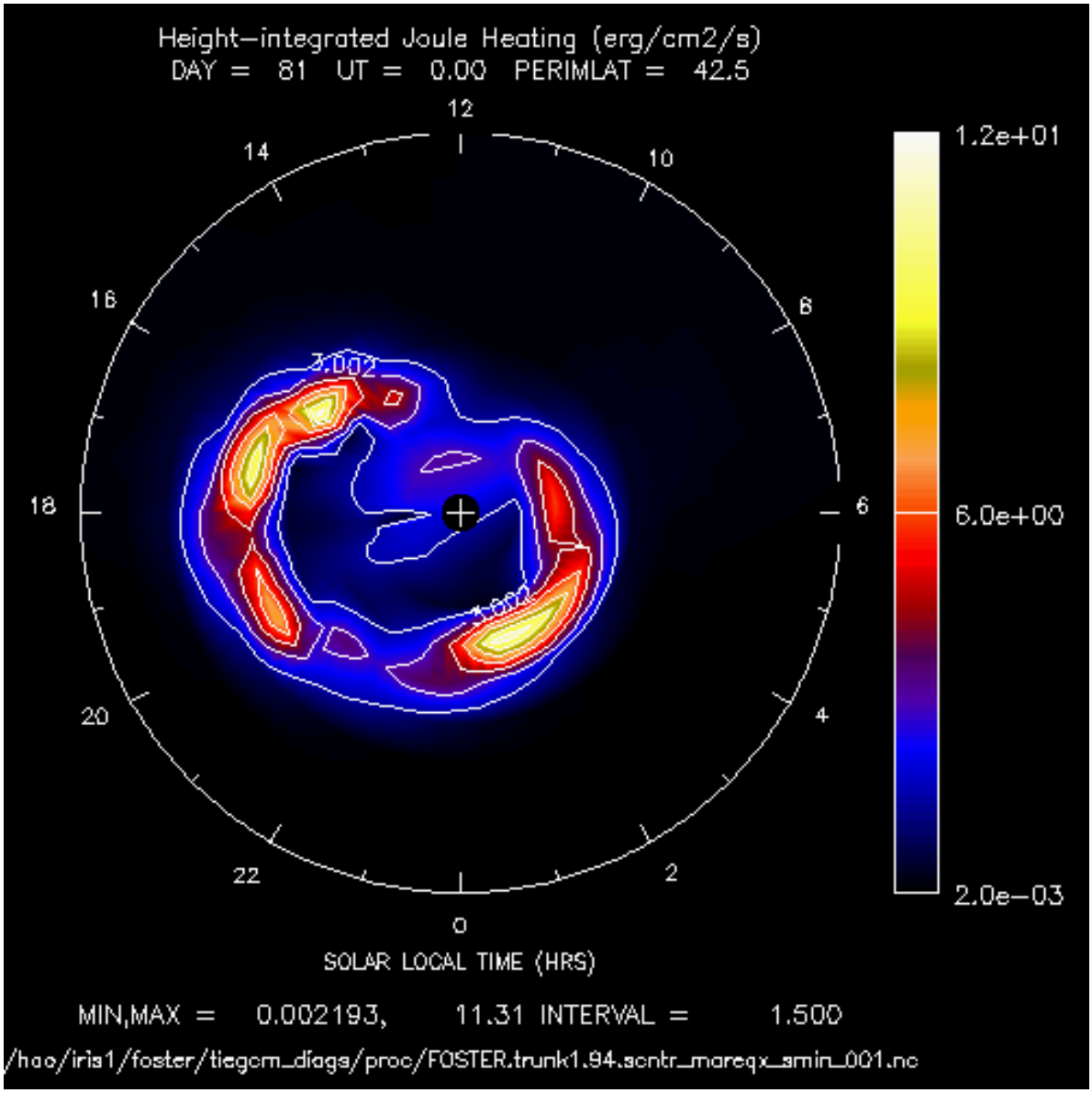
```
!-----------------------------------------------------------------------
      subroutine mkdiag_QJOULE_INTEG(name,qji_tn,lev0,lev1,lon0,lon1,
     |  lat)
      use cons_module,only: p0,grav
      use init_module,only: zpint
!
! Calculate height-integrated Joule heating (called from qjoule.F)
```

```
! This method is adapted from ncl code provided by Astrid (7/20/11)
!
! Args
      character(len=*),intent(in) :: name
      integer,intent(in) :: lev0,lev1,lon0,lon1,lat
      real,intent(in),dimension(lev0:lev1,lon0:lon1) :: qji_tn
!
! Local:
      integer :: ix,k,i
      real,dimension(lon0:lon1) :: qji_integ
      real,dimension(lev0:lev1,lon0:lon1) :: qj
      real :: myp0,mygrav
!
! Check that field name is a diagnostic, and was requested:
      ix = checkf(name) ; if (ix==0) return
!
! First integrate to get MKS units W/m^2:
! (If you want these units, comment out the below conversion to CGS)
!
      mygrav = grav*.01     ! cm/s^2 to m/s^2
      myp0 = p0*1.e-3*100.  ! to Pa
      qj = qji_tn*.0001     ! ergs/g/s to W/kg 10^(-7)*10^3

      qji_integ = 0.
      do i=lon0,lon1
        do k=lev0,lev1-1
          qji_integ(i) = qji_integ(i) + myp0/mygrav*exp(-zpint(k))*
     |      qj(k,i)*dlev
        enddo
      enddo
!
! Output in CGS units, to be consistent w/ the model:
! (note that 1 erg/cm^2/s == 1 mW/m^2)
      qji_integ = qji_integ*1000. ! W/m^2 to erg/cm^2/s
!
! Save 2d field on secondary history:
      call addfld(diags(ix)%short_name,diags(ix)%long_name,
     |  diags(ix)%units,qji_integ,'lon',lon0,lon1,'lat',lat,lat,0)

      end subroutine mkdiag_QJOULE_INTEG
!-----------------------------------------------------------------------
```

Called by: subroutine *qjoule_tn* in source file `qjoule.F` as follows:

```
 call mkdiag_QJOULE_INTEG('QJOULE_INTEG',qji_tn(:,lon0:lon1),
|  lev0,lev1,lon0,lon1,lat)
```

Sample images: QJOULE_INTEG North polar projection



Back to diagnostics table

JE13D

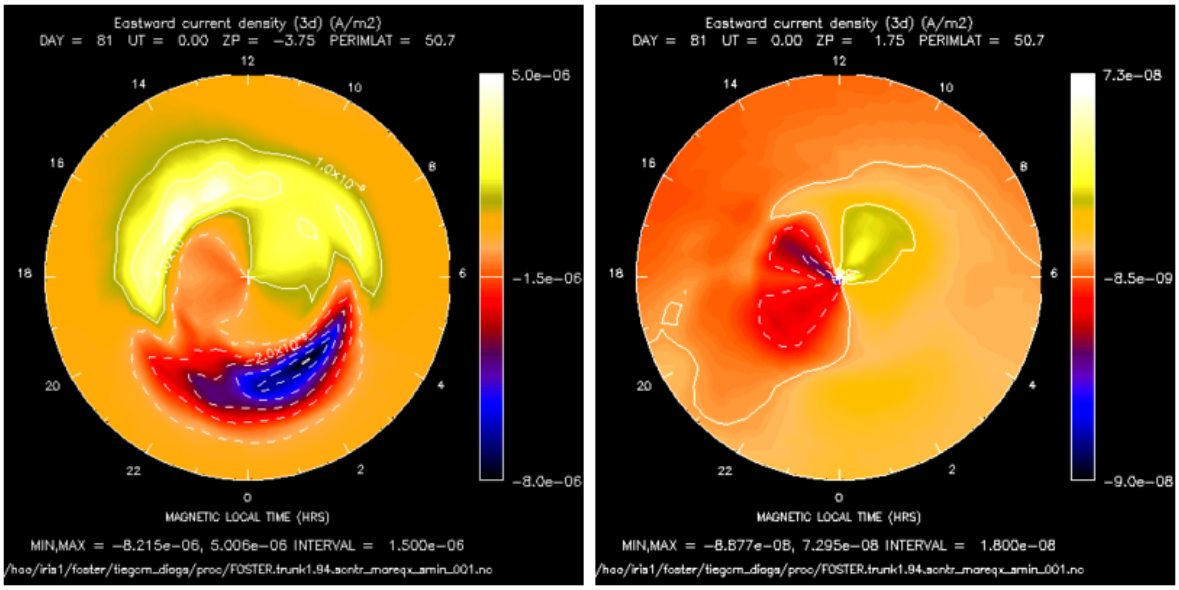
Diagnostic field: Eastward current density (A/m2) (3d on geomagnetic grid):

```
diags(n)%short_name = 'JE13D'
diags(n)%long_name  = 'Eastward current density (3d)'
diags(n)%units      = 'A/m2'
diags(n)%levels     = 'mlev'
diags(n)%caller     = 'current.F'
```

Je1/D is calculated in subroutine *nosocrdens* in source file `current.F`, and saved to secondary histories by subroutine *mkdiag_JE13D* (`diags.F`)

Note: JE13D is calculated and saved ONLY if the integer parameter icalkqlam is set to 1 in source file `dynamo.F` (the default is icalkqlam=0).

Sample images: JE13D North polar projection at Zp -4, +2



Back to diagnostics table

JE23D

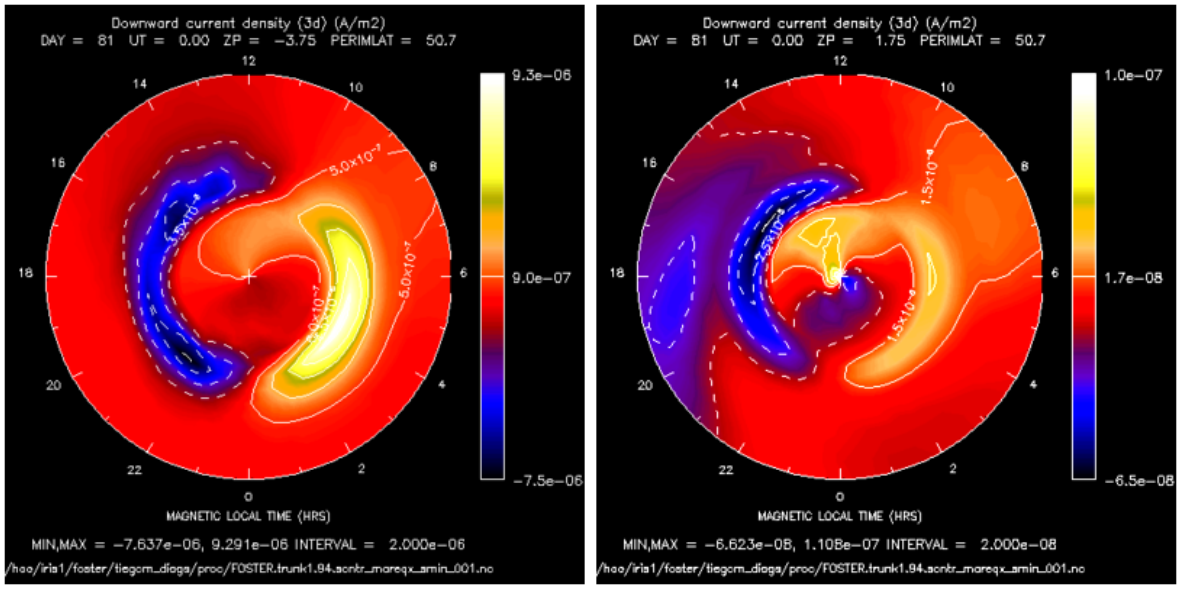
Diagnostic field: Downward current density (A/m2) (3d on geomagnetic grid):

```
diags(n)%short_name = 'JE23D'
diags(n)%long_name  = 'Downward current density (3d)'
diags(n)%units      = 'A/m2'
diags(n)%levels     = 'mlev'
diags(n)%caller     = 'current.F'
```

Je2/D is calculated in subroutine *nosocrdens* in source file `current.F`, and saved to secondary histories by subroutine *mkdiag_JE23D* (`diags.F`)

Note: JE23D is calculated and saved ONLY if the integer parameter icalkqlam is set to 1 in source file `dynamo.F` (the default is icalkqlam=0).

Sample images: JE23D North polar projection at Zp -4, +2



Back to diagnostics table

JQR

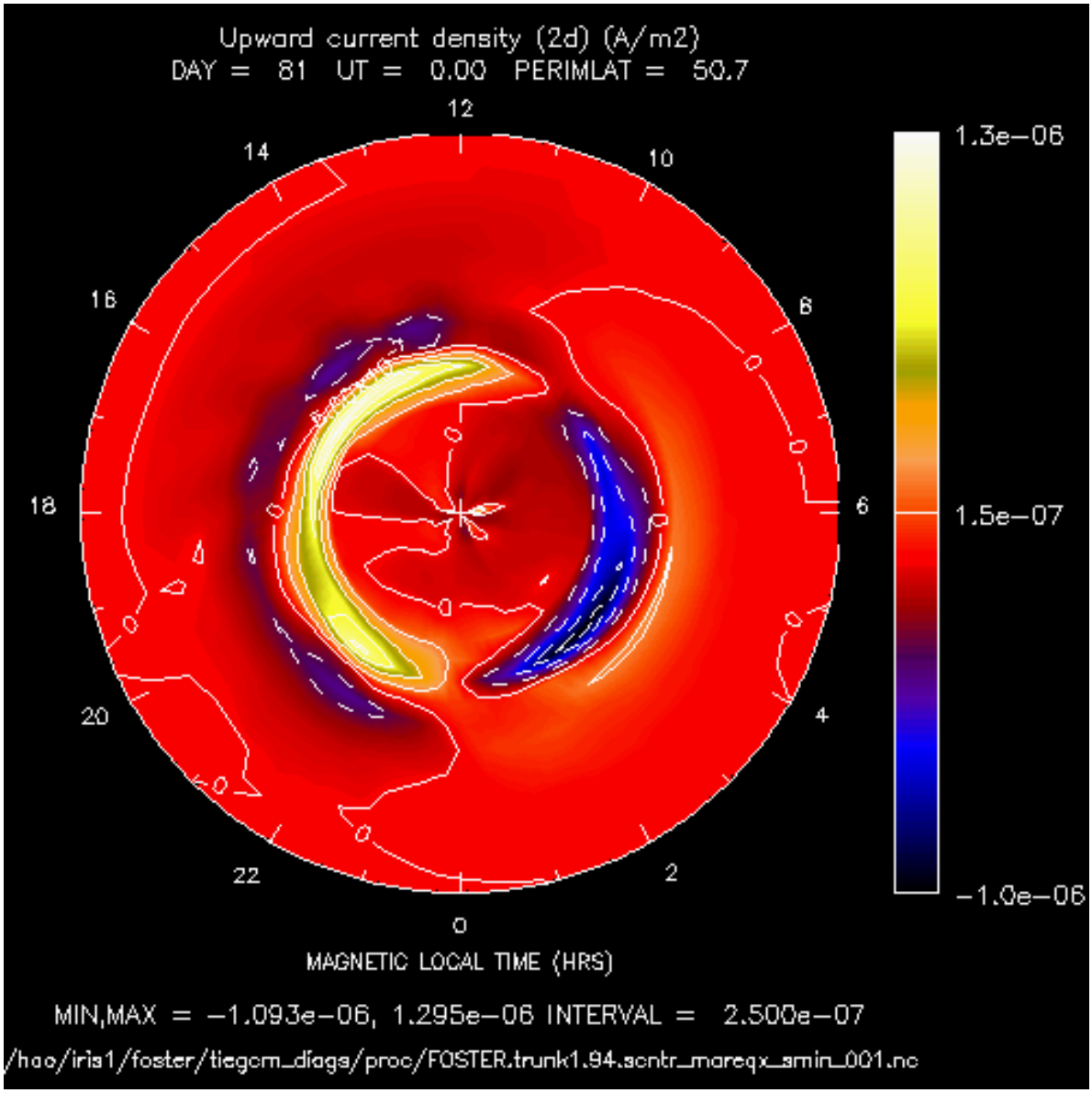
Diagnostic field: Upward current density (A/m2) (2d mlat-mlon on geomagnetic grid):

```
diags(n)%short_name = 'JQR'
diags(n)%long_name  = 'Upward current density (2d)'
diags(n)%units      = 'A/m2'
diags(n)%levels     = 'none'
diags(n)%caller     = 'current.F'
```

Jqr is calculated in subroutine *nosocrrt* in source file `current.F`, and saved to secondary histories by subroutine *mkdiag_JQR* (`diags.F`)

Note: Jqr is calculated and saved ONLY if the integer parameter icalkqlam is set to 1 in source file `dynamo.F` (the default is icalkqlam=0).

Sample images: JQR North polar projection



Back to diagnostics table

KQLAM

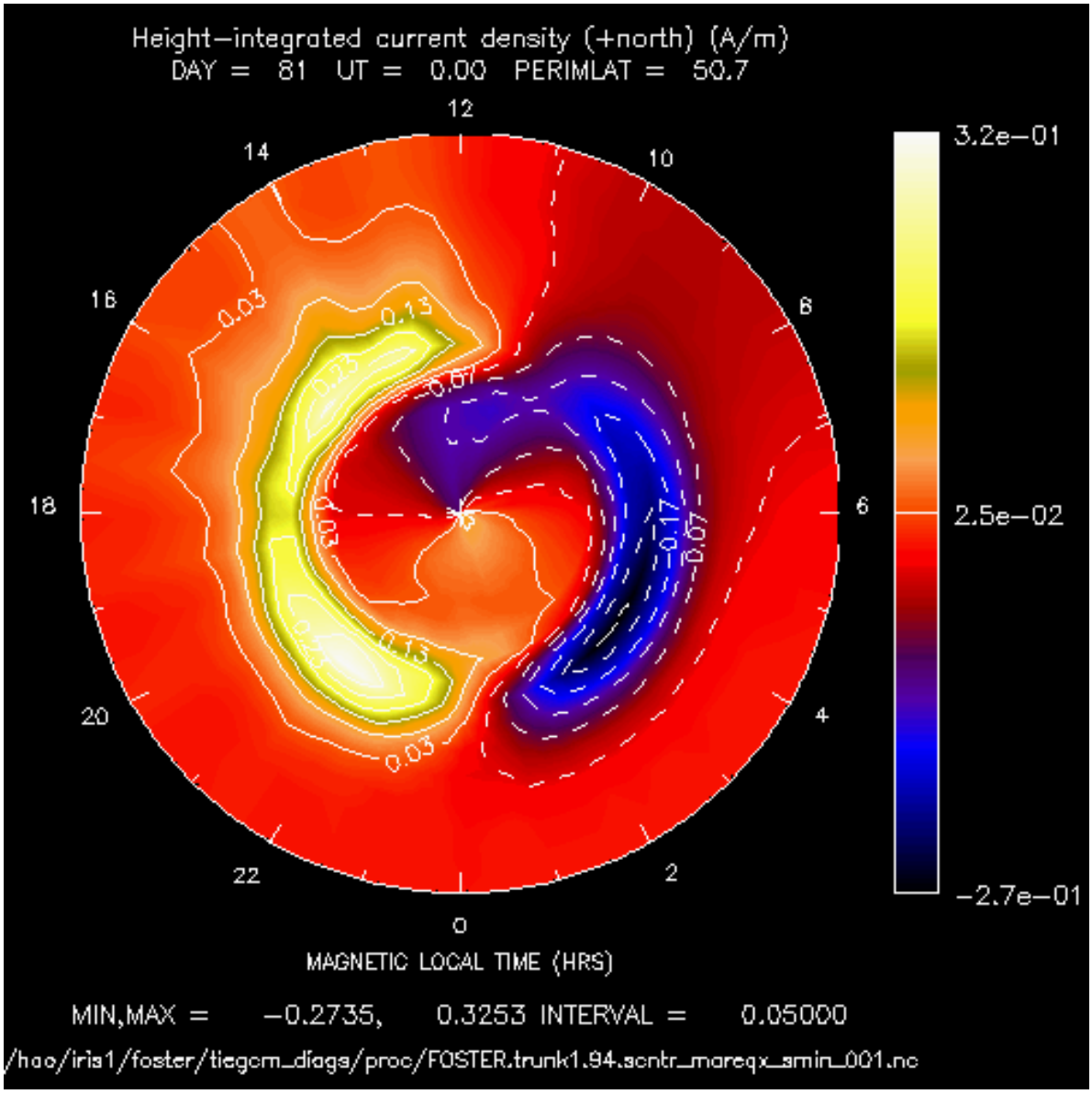
Diagnostic field: Height-integrated current density (+north) (A/m2) (2d mlat-mlon on geomagnetic grid):

```
diags(n)%short_name = 'KQLAM'
diags(n)%long_name  = 'Height-integrated current density (+north)'
diags(n)%units      = 'A/m'
diags(n)%levels     = 'none'
diags(n)%caller     = 'current.F'
```

Kqlam is calculated in subroutine *nosocrdens* in source file `current.F`, and saved to secondary histories by subroutine *mkdiag_KQLAM* (`diags.F`)

Note: Kqlam is calculated and saved ONLY if the integer parameter icalkqlam is set to 1 in source file `dynamo.F` (the default is icalkqlam=0).

Sample images: KQLAM North polar projection



Back to diagnostics table

KQPHI

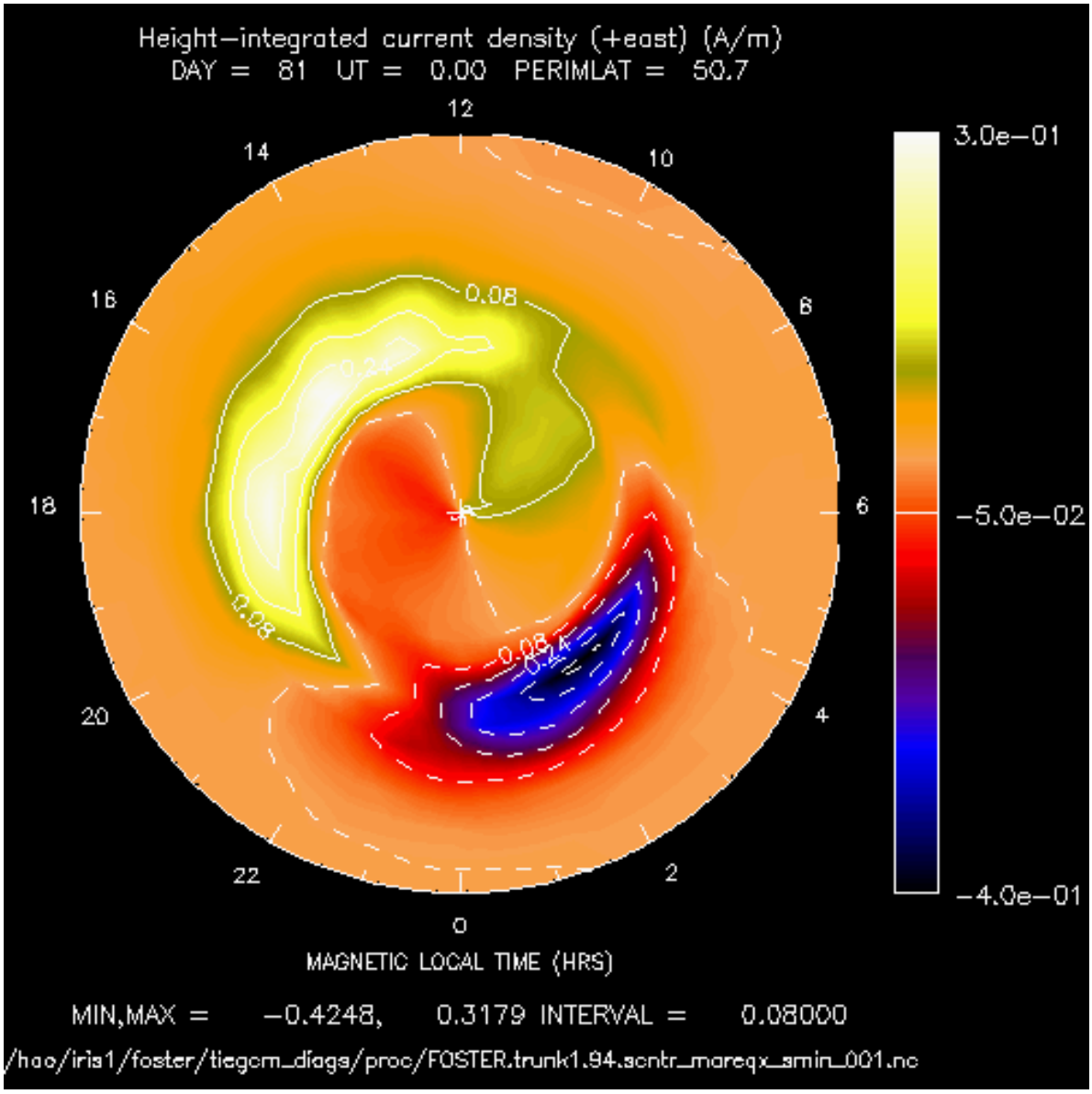
Diagnostic field: Height-integrated current density (A/m2) (2d mlat-mlon on geomagnetic grid):

```
diags(n)%short_name = 'KQPHI'
diags(n)%long_name  = 'Height-integrated current density (+east)'
diags(n)%units      = 'A/m'
diags(n)%levels     = 'none'
diags(n)%caller     = 'current.F'
```

Kqphi is calculated in subroutine *nosocrdens* in source file `current.F`, and saved to secondary histories by subroutine *mkdiag_KQLAM* (`diags.F`)

Note: Kqphi is calculated and saved ONLY if the integer parameter icalkqlam is set to 1 in source file `dynamo.F` (the default is icalkqlam=0).

Sample images: KQPHI North polar projection



Back to diagnostics table

CHAPTER

SEVEN

MODEL SOURCE CODE

The source code is in the *src/* subdirectory of the model root directory (*modeldir*), which is provided in the model *download* file.

7.1 The Academic License Agreement

The TIEGCM `Open Source Academic Research License Agreement` specifies the terms and restrictions under which the NCAR/UCAR grants permission to use the model, including the source code, for research, academic, and non-profit purposes.

7.2 Source Code Flow Diagram

A detailed flow diagram and calling tree of the source code structure is available in single and multi-page pdf files:

Warning: Some details of these flow charts may be out of date with respect to TIEGCM version 1.94

- TIEGCM Code Structure (multi-page pdf)
- TIEGCM Code Structure (single-page pdf)

7.3 Grid Structure and Resolution

The TIEGCM can be configured for two spatial/temporal resolutions (use the *modelres* shell variable in the *job script* to set the model resolution):

- 5 degrees lat x lon, 2 grid points per scale height
  (default time step = 120 secs)
- 2.5 degrees lat x lon, 4 grid points per scale height
  (default time step = 60 secs)
- The vertical coordinate `lev`, or `Zp`, is a log-pressure scale `ln(p0/p)`, where p is pressure and p0 is a reference pressure. Fields are calculated at either "interface" levels (`ilev`), or at "midpoint" levels (`lev`) (see lev and ilev coordinate definitions below).
  - At 5.0 degree horizontal, Zp at interfaces = -7 to +5 by 0.5
  - At 2.5 degree horizontal, Zp at interfaces = -7 to +5 by 0.25

Note: To interpolate model fields to constant height surfaces, you should use geometric height, which is available on the 3d model grid as "ZG" on secondary histories.

The spatial coordinates at the 5-degree resolution are defined as follows:

```
double lon(lon) ;
  lon:long_name = "geographic longitude (-west, +east)" ;
  lon:units = "degrees_east" ;

lon = -180, -175, -170, -165, -160, -155, -150, -145, -140, -135, -130,
  -125, -120, -115, -110, -105, -100, -95, -90, -85, -80, -75, -70, -65,
  -60, -55, -50, -45, -40, -35, -30, -25, -20, -15, -10, -5, 0, 5, 10, 15,
  20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105,
  110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175 ;


double lat(lat) ;
  lat:long_name = "geographic latitude (-south, +north)" ;
  lat:units = "degrees_north" ;

lat = -87.5, -82.5, -77.5, -72.5, -67.5, -62.5, -57.5, -52.5, -47.5, -42.5,
  -37.5, -32.5, -27.5, -22.5, -17.5, -12.5, -7.5, -2.5, 2.5, 7.5, 12.5,
  17.5, 22.5, 27.5, 32.5, 37.5, 42.5, 47.5, 52.5, 57.5, 62.5, 67.5, 72.5,
  77.5, 82.5, 87.5 ;

double lev(lev) ;
  lev:long_name = "midpoint levels" ;
  lev:short_name = "ln(p0/p)" ;
  lev:units = "" ;
  lev:positive = "up" ;
  lev:standard_name = "atmosphere_ln_pressure_coordinate" ;
  lev:formula_terms = "p0: p0 lev: lev" ;
  lev:formula = "p(k) = p0 * exp(-lev(k))" ;

lev = -6.75, -6.25, -5.75, -5.25, -4.75, -4.25, -3.75, -3.25, -2.75, -2.25,
  -1.75, -1.25, -0.75, -0.25, 0.25, 0.75, 1.25, 1.75, 2.25, 2.75, 3.25,
  3.75, 4.25, 4.75, 5.25, 5.75, 6.25, 6.75, 7.25 ;

double ilev(ilev) ;
  ilev:long_name = "interface levels" ;
  ilev:short_name = "ln(p0/p)" ;
  ilev:units = "" ;
  ilev:positive = "up" ;
  ilev:standard_name = "atmosphere_ln_pressure_coordinate" ;
  ilev:formula_terms = "p0: p0 lev: ilev" ;
  ilev:formula = "p(k) = p0 * exp(-ilev(k))" ;

ilev = -7, -6.5, -6, -5.5, -5, -4.5, -4, -3.5, -3, -2.5, -2, -1.5, -1, -0.5,
  0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7 ;
```

Note: The 2.5 degree configuration ("double-resolution"), is not fully tuned and validated in TIEGCM version 1.94.

7.4 Modifying the Source Code

As a community user, student, research scientist or developer, you may need to modify the model source code. It is best to do this after building and at least making a default execution of the model (see the *QuickStart* Section). To change one or more source files, simply go to the *src/* subdirectory in the model root directory *modeldir*, and edit the files as necessary. Then return to the working directory *workdir* and re-execute the job script. It will recompile the modified files, and any other source files that depend on the modified files, and re-execute the model. Alternatively, you can enter the execution directory *execdir*, and recompile the code by typing "gmake" on the command line, then return to the working directory and re-execute the job script.

CHAPTER

EIGHT

THE MAKE/BUILD PROCESS: COMPILE AND LINK

The TIEGCM model is formally supported on two platform systems: 64-bit Linux, and IBM/AIX. However, the model has been built and executed on several other platforms. The source code is f90 standard compliant, and is mostly fixed-format fortran.

Compilers used on Linux systems include Intel's ifort 11.1 (with OpenMPI), and PGI's fortran compiler pgf90. The compiler used on the NCAR IBM/AIX bluefire system is xlf90.

Library dependencies consist mainly of netCDF and MPI. The MPI library is often bundled in with the compiler. Locations of these libraries are specified in "Make.machine" files, which set platform-specific compile flags and other parameters for the build process. The following Make.machine files are provided in the *scripts/* directory:

- `Make.bluefire` (NCAR IBM/AIX machine)
- `Make.intel_hao64` (ifort compiler on HAO 64-bit Linux desktops)
- `Make.pgi_hao32` (pgf90 compiler on HAO 32-bit Linux desktops)
- `Make.pgi_hao64` (pgf90 compiler on HAO 64-bit Linux desktops)

One of these files, or the user's own, is specified by the csh variable "make" in the *job script*. The specified file is included in the main `Makefile`. User's outside NCAR are encouraged to copy and rename one of these files, and customize it for your own operating system and compiler.

CHAPTER

NINE

BENCHMARK TEST RUNS OF V1.94

A series of benchmark runs are made for each release of the TIEGCM (since v1.93). Job scripts and namelist input files for these runs are available in subdirectories in the *tests/* directory.

Netcdf files with the first history of each benchmark run are available in the *data download file*. These files can be used as start-up *SOURCE* files to reproduce the runs. The namelist input files and job scripts used to make the runs are provided in the *tests/* directory of the release.

Following is a summary of benchmark runs made by TIEGCM version 1.94. For full history file output, steady-state start-up files, and "sanity check" plots of the benchmark runs made by version 1.94, please see Release Documentation

9.1 Control

control: 5-day control runs, started from steady-state histories at both equinoxes and both solstices, and at solar minimum and maximum conditions:

```
 mareqx: March Equinox (day 80)
 junsol: June Solstice (day 172)
 sepeqx: September Equinox (day 264)
 decsol: December Solstice (day 355)

* Solar Minimum::

   POWER   = 18.
   CTPOTEN = 30.
   F107    = 70.
   F107A   = 70.

* Solar Maximum::

   POWER   = 39.
   CTPOTEN = 60.
   F107    = 200.
   F107A   = 200.
```

9.2 Climatology

climatology: Full-year Climatology with constant solar forcing:

- Heelis potential model with constant solar forcing:

```
POWER   = 18.
CTPOTEN = 30.
F107    = 100.
F107A   = 100.
```

9.3 December, 2006 "AGU" Storm Case

dec2006: December, 2006 "AGU" storm case:

- Heelis potential model with GPI (Kp) data
- Weimer potential model with IMF data (F10.7 from GPI)

9.4 November, 2003 Storm Case

nov2003: November 19-24 (days 323-328), 2003 storm case:

- Heelis potential model with GPI (Kp) data
- Weimer potential model with IMF data (F10.7 from GPI)

9.5 Whole Heliosphere Interval

whi2008: Whole Heliosphere interval (WHI) (March 21 to April 16, 2008)

- Heelis potential model with GPI (Kp) data
- Weimer potential model with IMF data (F10.7 from GPI)

Note: For more detailed information and access to history file output, and preliminary post-processing of these runs, see Release Documentation

---

CHAPTER

TEN

POST-PROCESSING AND VISUALIZATION

TIEGCM netCDF history files can be read by any application with access to the netCDF library, including many freely available software packages developed for mainpulating or displaying netCDF data (see http://www.unidata.ucar.edu/software/netcdf/software.html). At HAO, we often use the netCDF Operators NCO for file manipulation (subset extracting, concatenation, hyperslabbing, metadata editing, etc). However for visualization, we typically use one of three post-processors developed at HAO:

10.1 tgcmproc_f90

- *tgcmproc_f90* is a batch-style processor written in fortran 90. This program reads a user namelist input file via stdin, and outputs multi-frame plot files (cgm and/or ps), and output data files (e.g., ascii, netCDF).
- Uses the freely available NCAR Graphics libraries for basic contouring, making maps at various projections, vector plots, etc.
- Plots 2d horizontal and vertical slices, and time-dependent plots on the model grid.
- Calculates a large number of diagnostics from fields on the histories.
- Custom contouring (setting cmin,cmax,cint)
- Can interpolate to constant height surfaces.
- Can be downloaded from the TGCM website, but the f90 code must be compiled, and NCAR Graphics libraries must be linked.

10.2 tgcmproc_idl

- *tgcmproc_idl* is an IDL application for browsing and plotting TIEGCM output, with an easy to use Graphical User Interface (GUI).
- 2d contouring of horizontal and vertical slices, including maps at various projections.
- Can save images and plots to a variety of image formats.
- Custom contouring (setting cmin,cmax,cint).
- Can interpolate to constant height surfaces.
- Can plot fields on the magnetic grid (tgcmproc_f90 does not do this).

- Can make and save png movie animations.
- Should run fine for anybody w/ IDL, but IDL is licensed, and can be expensive.

10.3 utproc

- *utproc* is an IDL/GUI application that makes time-series contours and images including ut vs zp pressure at selected grid lat x lon locations, and ut vs latitude at selected zp pressure surfaces.

These applications are available at the TGCM download page. Tgcmproc_f90 is best for generating large numbers of plots in a "batch-style" environment, whereas tgcmproc_idl is best for browsing history files in a GUI interface, and saving plots or images as desired. The utproc processor is a hybrid in the sense that a series of plots can be setup using the GUI, and then created when requested.

At HAO, we also use the NCAR Command Language (NCL) for plotting, analysis, and converting to/from various file formats (GRIB, HDF, etc). NCL scripts can be used to generate customized plots and images, as well as providing a variety of analysis and file-manipulation tools.

CHAPTER

ELEVEN

CONTACT INFORMATION

The TIEGCM and related Thermosphere-Ionosphere models have been developed by the "Atmosphere Ionosphere Magnetosphere" (AIM) Section of the High Altitude Observatory (HAO) at NCAR (see http://www.hao.ucar.edu/modeling/tgcm).

For information and assistance regarding the software (source code and supporting script infrastructure, including building and executing the models and using the post-processors, you may contact Ben Foster at foster@ucar.edu.

For questions and information regarding the physics and chemistry implementation of the model, numerical algorithms, and analysis of model results, you may contact any of the following HAO scientists:

- Art Richmond (richmond@ucar.edu): upper atmosphere dynamics and electrodynamics
- Stan Solomon (stans@ucar.edu): airglow, ionosphere chemistry
- Barbara Emery (emery@ucar.edu): auroral processes and parameterization
- Astrid Maute (maute@ucar.edu): tidal dynamics and electro-dynamo code
- Wenbin Wang (wbwang@ucar.edu): mesoscale processes, model coupling
- Liying Qian (lqian@ucar.edu): neutral density, solar irradiance
- Alan Burns (aburns@ucar.edu): thermosphere dynamics and composition

CHAPTER

TWELVE

GLOSSARY

benchmark runs A series of *validation/test runs* made with each release of the model. Namelist files and job scripts used to make these benchmark runs are available in subdirectories under the *tests/* directory of the current release (e.g., namelist input files for the "control" test run are `modeldir/tests/control/*.inp`).

continuation run A continuation run continues from the last output history of the previous run. That history (*START* time) must be on the first *OUTPUT* file provided by the namelist input file. A continuation run must not specify a *SOURCE* file or *SOURCE_START* time.

datadir The directory containing start-up and other input data files required for running the model. A minimal set of datadir files are available via *download*. The datadir is sometimes referred to by the *TGCMDATA* environment variable. Additional data files are available via the *NCAR Community Data Portal*.

diagnostic fields A list of diagnostic fields are available to be saved on secondary history files. See section *Saving Diagnostic Fields*.

doc/ Subdirectory under the *modeldir* containing documentation, e.g., the User's Guide, Model Description, Release Notes, etc.

execdir The model execution directory. This is the directory where the model is built and executed. It is typically, but not necessarilly, a subdirectory of your working directory *workdir*. When a job script is executed from a working directory, the execdir is created if it does not already exist. During a model run, output history files are written to the execdir.

history A model history records the state of the model at a discrete instant in *model time*. One or more histories are stored in netCDF history files.

initial run An initial run is started from a history on a *SOURCE* file (see also *SOURCE_START*). Subsequent *continuation runs* do not provide SOURCE or SOURCE_START, but rather search for the *START* time on the first *OUTPUT* history file provided in the namelist input, and continue the run from there.

job script A csh script in the scripts/ directory which, when executed, will build and execute the model. The user defines a few shell variables in the job script, such as the *modeldir*, and the *namelist input*. For more details, please see *job scripts*.

model time TIEGCM model time is represented by an integer triplet: day,hour,minute, where day is the julian day of the year, and hour is the ut. The variable for model time on history files is mtime(3,ntimes). For example, a history file may contain 24 hourly histories for day 80: mtime = 80,1,0, 80,2,0, ... 81,0,0.

modeldir The model root directory. This directory typically contains subdirectories *src/* (model source code), *scripts/* (utility scripts), *doc/* (documentation), and *tests/* (test runs). The modeldir is available via *download*, and is typically a subdirectory of the model working directory (*workdir*).

namelist input The model reads user specified parameters from the *namelist input file* via f90 standard namelist read. Keyword/Value pairs are read from unit 5, and are validated by the input module (input.F).

NCAR Community Data Portal The NCAR Community Data Portal is a public data repository with datasets from NCAR, UCAR, UOP, and participating organizations. To browse TIEGCM-related files (mostly netCDF history files for model start-up, or results of *benchmark runs*), click on the "Models" link, then to the "Thermospheric General Circulation Models" link, and finally to the desired model version. NetCDF Metadata is available without actually downloading files.

netCDF TIEGCM output history files are written in netCDF, a self-describing platform-independent data format written and maintained by the UCAR Unidata program.

resolution The TIEGCM can be run in one of two resolutions:

- 5 x 5 deg lat x lon, 2 grid levels per scale height (dz = 0.50)
- 2.5 x 2.5 deg lat x lon, 4 grid levels per scale height (dz = 0.25)

The resolution is set by the "modelres" shell variable in the TIEGCM *job script*. See also the section on *Grid Structure and Resolution*.

Note: The 2.5-degree resolution model is available in version 1.94, but it is not fully validated or supported by the public release.

---

scripts/ Subdirectory under the *modeldir* containing supporting and utility scripts, including job scripts, the default namelist input file, several Make files, etc.

src/ Subdirectory under the *modeldir* containing the model source code (*.F, *.h files).

tests/ Subdirectory under the *modeldir*. The tests directory contains subdirectories for *benchmark runs* that were made for the current release. The subdirectories contain job scripts and namelist input files that can be used to reproduce benchmark runs for testing and validation purposes. For more information, see the section on *Benchmark Test Runs*.

TGCMDATA A unix environment variable that refers to the *datadir*. This environment variable may be used when referring to data files in the namelist read file, e.g., "GPI_NCFILE = *$TGCMDATA/gpi_xxxxx.nc*". See *namelist read files*.

tgcmproc_f90 Post-processor and visualizer for TIEGCM netCDF history files. Written in f90, and available at the TGCM download site. See *tgcmproc_f90*.

tgcmproc_idl Post-processor and visualizer for TIEGCM netCDF history files. This processor is Written in IDL with a GUI, and is available at the TGCM download site. See *tgcmproc_idl*.

utproc Post-processor and visualizer for TIEGCM netCDF history files. This processor reads time-series history files and makes ut vs pressure and ut vs latitude contours. It is written in IDL with a GUI, and is available at the TGCM download site. See *utproc*.

workdir Your local working directory. This will typically contain the model root directory *modeldir*, the execution directory *execdir*, and related namelist input files, job scripts, stdout files, etc. It may also contain a data subdirectory *datadir*.

CHAPTER

THIRTEEN

INDICES AND TABLES

- *genindex*
- *search*

Note: This document was last updated on January 06, 2012

INDEX

K

L

M

N

O

P

Q

R

S

T

U

V

W