User Manual for the LMD Generic Climate Model

E. Millour, F. Forget, J. Leconte, R. Wordsworth.

October 11, 2017

Contents

Chapter 1

Introduction

This document is a user manual for the Generic Climate Model developed by the Laboratoire de Météorologie Dynamique of the CNRS in Paris. It corresponds to the version of the model available since January 2011, that includes the new dynamic code lmdz3.3 and input and output data in NetCDF format. The physical part includes generalized correlated-k radiative transfer, generalized tracer transport, and a water cycle that includes water vapour and ice transport, radiative and thermodynamic effects, and simple hydrology.

Chapter 2 of this document, to be read before any of the others, describes the main features of the model. The model is divided into two relatively independent parts: (1) The hydrodynamic code, which integrates the fluid mechanical *primitive equations* in time over the globe, and (2) the physical parameterizations, which include the radiative transfer, tracer transport / evolution, and surface-atmosphere interaction. It is followed by a list of references for anyone requiring a detailed description of the physics and the numerical formulation of the parameterizations (Chapter 4).

For your **first contact with the model**, Chapter 5 guides the user through a practice simulation (choosing the initial states and parameters and visualizing the output files). The document then describes the code used for the model, including a user computer manual for compiling and running it (Chapter 6).

Chapter 7 describes the input/output data of the model. The input files are the files needed to initialize the model (state of the atmosphere at instant $t0$ as well as a dataset of boundary conditions). The output files are "historical files", archives of the atmospheric flow history as simulated by the model, the "diagfi files", the "stats files", the daily averages, and so on. Common ways of editing or visualizing these files (editor "ncdump" and the graphics software "grads") are also explained. Chapter 8 explains how to run a simulation that includes the water cycle. Finally, Chapter 9 will help you to use a 1-dimensional version of the model, which may be a simpler tool for some analysis work.

Chapter 2

Main features of the model

2.1 Basic principles

The General Circulation Model (GCM) calculates the temporal evolution of the different **variables** (listed below) that control or describe the planetary meteorology and climate at different points of a **3D "grid"** (see below) that covers the entire atmosphere.

From an initial state, the model calculates the evolution of these variables, timestep by timestep:

- At instant t, we know variable X_t (temperature for example) at one point in the atmosphere.
- We calculate the evolution (the **tendencies**) $(\frac{\partial X}{\partial t})_1$, $(\frac{\partial X}{\partial t})_2$, etc. arising from each physical phenomenon, calculated by a **parameterization** of each of these phenomenon (for example, heating due to absorption of solar radiation).
- At the next time step $t + \delta t$, we can calculate $X_{t+\delta t}$ from X_t and $(\frac{\partial X}{\partial t})$. This is the **"integration"** of the variables in time. (For example, $X_{t+\delta t} = X_t + \delta t(\frac{\partial X}{\partial t})_1 + \delta t(\frac{\partial X}{\partial t})_2 + ...$)

The main task of the model is to calculate these tendencies $(\frac{\partial X}{\partial t})$ arising from the different parameterized phenomena.

2.2 Dynamical-Physical separation

In practice, the 3D model operates in two parts:
- a **dynamical part** containing the numerical solution of the general equations for atmospheric circulation. This part (including the programming) is common to all terrestrial-type atmospheres, and applicable in certain cases to the upper atmospheres of gas giant planets.
- a **physical part** that is specific to the planet in question and which calculates the circulation forcing and climatic details at each point.

The calculations for the dynamical part are made on a 3D grid with horizontal exchanges between the grid boxes, whereas the physical part can be seen as a juxtaposition of atmosphere "columns" that do not interact with each other (see diagram 2.1).

The dynamical and physical parts deal with variables of different natures, and operate on grids that are differently constructed. The temporal integration of the variables is based on different numerical schemes (simple, such as the one above for the physical part, and more complicated, the "Matsuno-Leapfrog" scheme for the dynamical part). The timesteps are also different. The physical timestep is `iphysiq` times longer than the dynamical

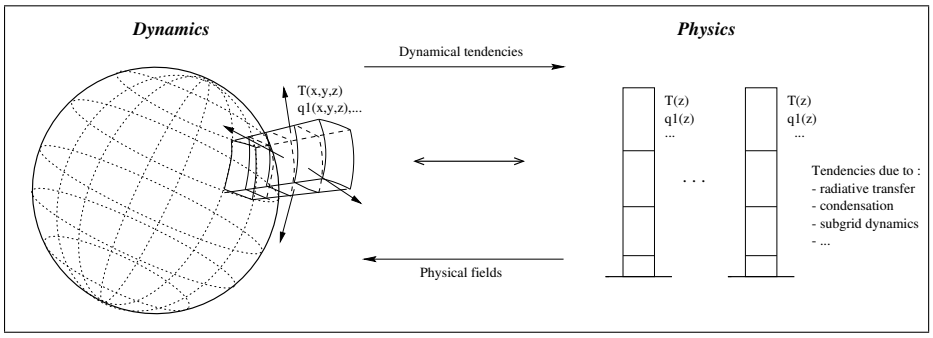

Figure 2.1: Physical/dynamical interface

timestep, as the solution of the dynamic equations requires a shorter timestep than the forced calculation for the physical part.

In practice, the main program that handles the whole model (`gcm.F`) is located in the dynamical part. When the temporal evolution is being calculated, at each timestep the program calls the following:

1. Call to the subroutine that handles the total tendency calculation ($\frac{\partial X}{\partial t}$) arising from the dynamical part (`caldyn.F`)
2. Integration of these dynamical tendencies to calculate the evolution of the variables at the following timesteps (subroutine `integrd.F`)
3. Every `iphysiq` dynamical timestep, a call to the interface subroutine (`calfis.F`) with the physical model (`physiq.F90`), that calculates the evolution of some of the purely physical variables (e.g: surface temperature `tsurf`) and returns the tendencies ($\frac{\partial X}{\partial t}$) arising from the physical part.
4. Integration of the physical variables (subroutine `addfi.F`)
5. Similarly, calculation and integration of tendencies due to the horizontal dissipation and the "sponge layer" is done every `idissip` dynamical time step.

Remark: The physical part can be run separately for a 1-D calculation for a single column using program `rcm1d.F`.

2.3 Grid boxes:

Examples of typical grid values are 64x48x25, 64x48x32 or 32x24x25 in longitudexlatitudexaltitude. Grid box size depends on the planetary radius: for Mars (radius$\sim$3400 km), for example, a 64x48 horizontal grid corresponds to grid boxes of the order of 330x220 kilometers near the equator.

2.3.1 Horizontal grids

Dynamics and physics use different grids. Figure 2.2 shows the correspondence and indexing of the physical and dynamical grids as well as the different locations of variables on these grids. To identify the coordinates of a variable (at one grid point up, down, right or left) we use coordinates `rlonu`, `rlatu`, `rlonv`, `rlatv` (longitudes and latitudes, in **radians**).

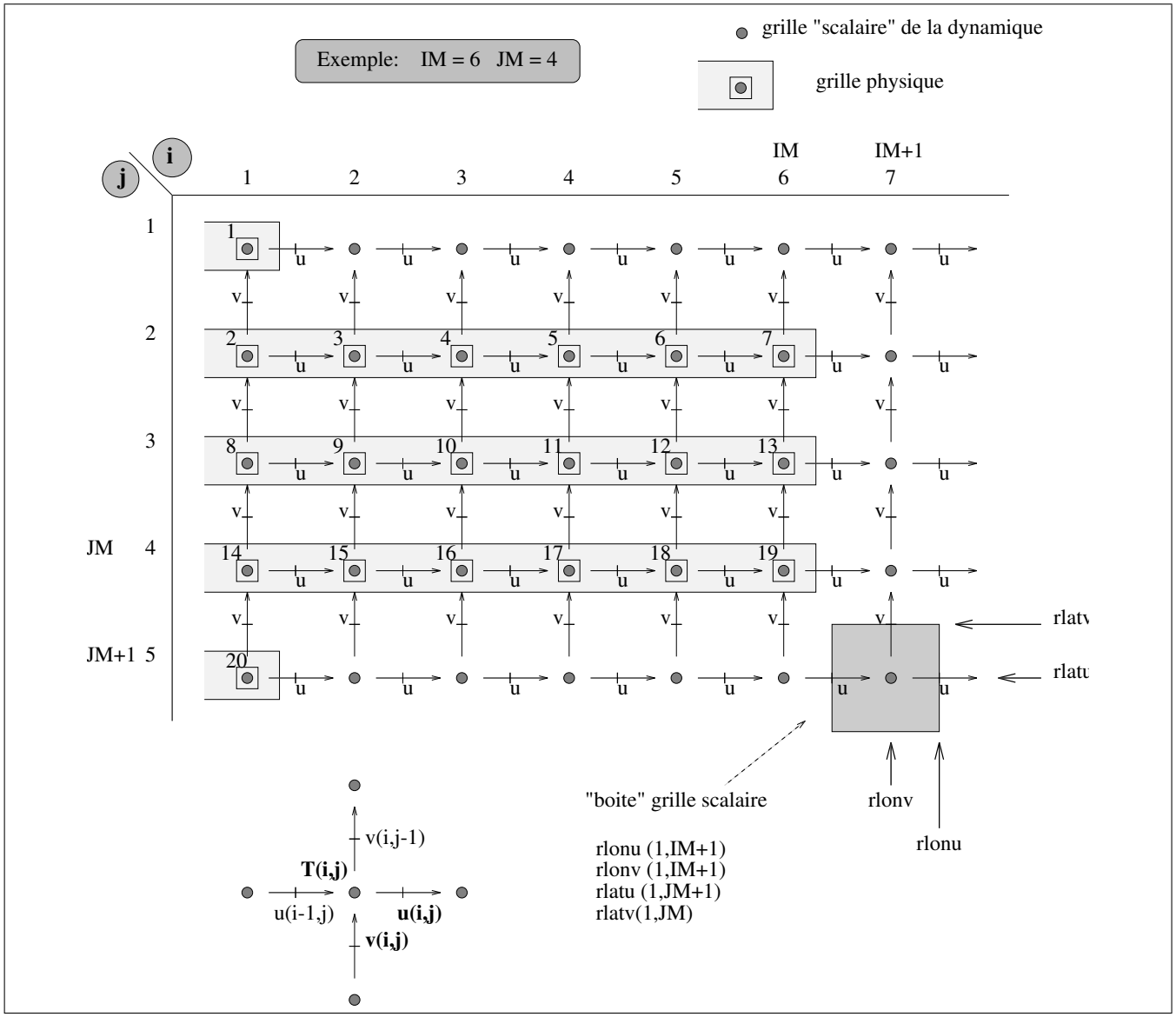

Figure 2.2: Dynamical and physical grids for a 6 × 7 horizontal resolution. In the dynamics (but not in the physics) winds u and v are on specific staggered grids. Other dynamical variables are on the dynamical "scalar" grid. The physics uses the same "scalar" grid for all the variables, except that nodes are indexed in a single vector containing NGRID=2+(JM-1)×IM points when counting from the north pole. N.B.: In the Fortran program, the following variables are used: `iim=IM , iip1=IM+1, jjm=JM , jjp1=JM+1`.

On the dynamical grid, values at `i=1` are the same as at `i=IM+1` as the latter node is a redundant point (due to the periodicity in longitude, these two nodes are actualy located at the same place). Similarly, the extreme `j=1` and `j=JM+1` nodes on the dynamical grid (respectively corresponding to North and South poles) are duplicated IM+1 times.
In contrast, the physical grid does not contain redundant points (only one value for each pole and no extra point along longitudes), as shown in figure 2.2. In practice, computations relative to the physics are made for a series of `ngrid` atmospheric columns, where `NGRID=IM`x(`JM`-1)+2.

2.3.2 Vertical grids

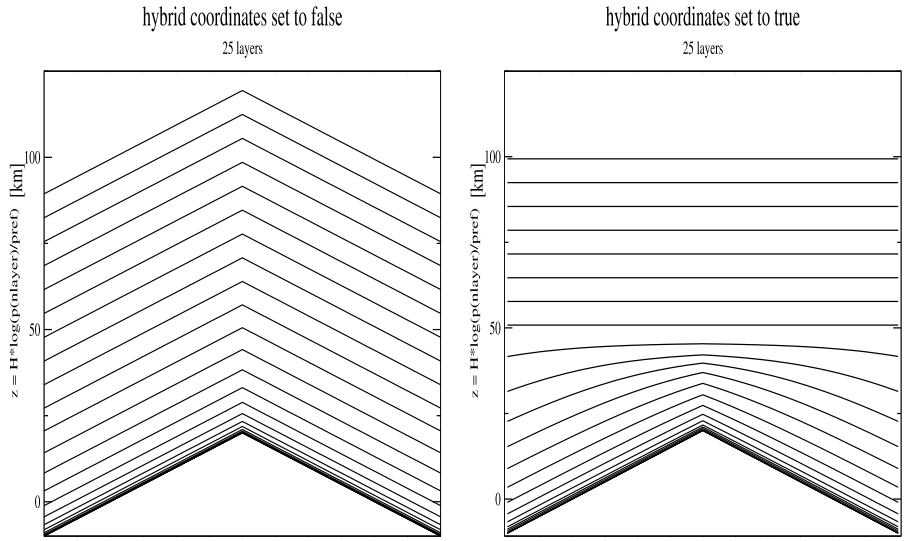

Figure 2.3: Sketch illustrating the difference between hybrid and non-hybrid coordinates

The GCM was initially programmed using sigma coordinates $\sigma = p/ps$ (atmospheric pressure over surface pressure ratio) which had the advantage of using a constant domain ($\sigma = 1$ at the surface and $\sigma = 0$ at the top of the atmosphere) whatever the underlying topography. However, it is obvious that these coordinates significantly disturb the stratospheric dynamical representation as the topography is propagated to the top of the model by the coordinate system. This problem can elegantly be solved by using a hybrid sigma-P (sigma-pressure) hybrid coordinate which is equivalent to using σ coordinates near the surface and gradually shifting to purely pressure p coordinates with increasing altitude. Figure 2.3 illustrates the importance of using these hybrid coordinates compared to simple σ coordinates. The distribution of the vertical layers is irregular, to enable greater precision at ground level. In general we use 25 levels to describe the atmosphere to a height of 80 km, 32 levels for simulations up to 120 km, or 50 levels to rise up to thermosphere. The first layer describes the first few meters above the ground, whereas the upper layers span several kilometers. Figure 2.4 describes the vertical grid representation and associated variables.

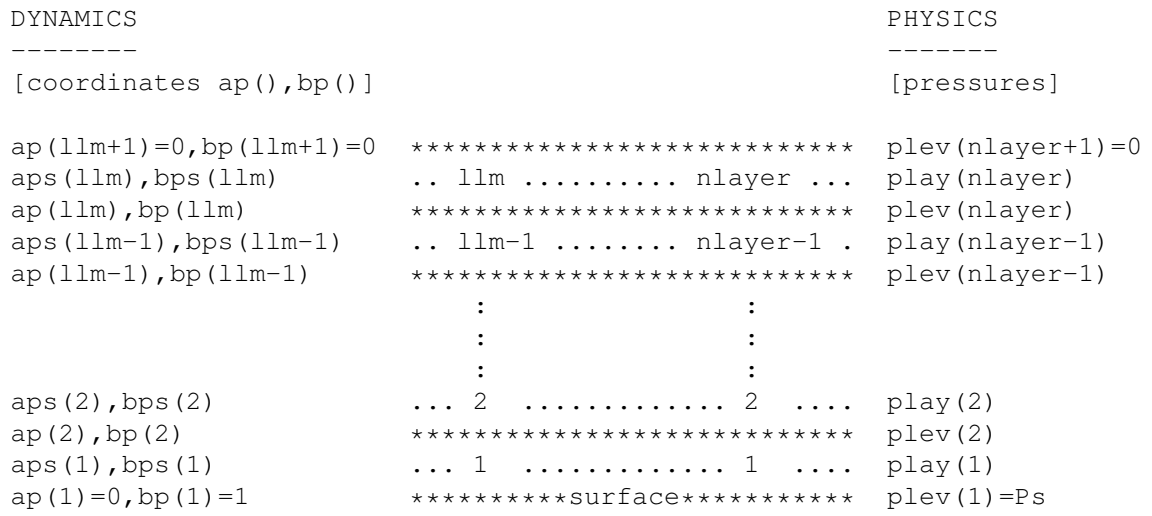

Figure 2.4: Vertical grid description of the `llm (or nlayer)` atmospheric layers in the programming code (`llm` is the variable used in the dynamical part, and `nlayer` is used in the physical part). Variables `ap, bp` and `aps, bps` indicate the hybrid levels at the interlayer levels and at middle of the layers respectively. Pressure at the interlayer is $Plev(l) = ap(l) + bp(l) \times Ps$ and pressure in the middle of the layer is defined by $Play(l) = aps(l) + bps(l) \times Ps$, (where Ps is surface pressure). Sigma coordinates are merely a specific case of hybrid coordinates such that $aps = 0$ and $bps = P/Ps$. Note that for the hybrid coordinates, $bps = 0$ above ~ 50 km, leading to purely pressure levels. The user can choose whether to run the model using hybrid coordinates or not by setting variable `hybrid` in run.def to True or False.

2.4 Variables used in the model

2.4.1 Dynamical variables

The dynamical state variables are the atmospheric temperature, surface pressure, winds and tracer concentrations. In practice, the formulation selected to solve the equations in the dynamics is optimised using the following less "natural" variables:

- **potential temperature** θ (`teta` in the code), linked to temperature **T** by $\theta = T(P/Pref)^{-\kappa}$ with $\kappa = R/C_p$ (note that κ is called `kappa` in the dynamical code, and `rcp` in the physical code). We take $Pref = 610$ Pa on Mars.

- **surface pressure** (`ps` in the code).

- **mass** the atmosphere mass in each grid box (`masse` in the code).

- **the covariant meridional and zonal winds** `ucov` and `vcov`. These variables are linked to the "natural" winds by `ucov = cu * u` and `vcov = cv * v`, where `cu` and `cv` are constants that only depend on the latitude.

- **mixing ratio of tracers** in the atmosphere, typically expressed in kg/kg (array `q` in the code).

`ucov` and `vcov`, "vectorial" variables, are stored on "scalari" grids u and v respectively, in the dynamics (see section 2.2). `teta`, `q`, `ps`, `masse`, "scalar variables", are stored on the "scalar" grid of the dynamics.

2.4.2 Physical variables

In the physics, the state variables of the dynamics are transmitted via an interface that interpolates the winds on the scalar grid (that corresponds to the physical grid) and transforms the dynamical variables into more "natural" variables. Thus we have winds **u** and **v** (m.s^{-1}), temperature **T** (K), pressure at the middle of the layers **play** (Pa) and at interlayers **plev** (Pa), tracers **q**, etc. (kg/kg) on the same grid.

Furthermore, the physics also handle the evolution of the purely physical state variables:

- **tsurf** surface temperature (K)

- **tsoil** temperature at different layers under the surface (K)

- **emis** surface emissivity

- **alb** surface albedo

- **q2** wind variance, or more precisely the square root of the turbulent kinetic energy

- **qsurf** tracer on the surface (kg.m^{-2})

- **rnat** surface type (0 = ocean, 1 = continent)

- **beta** surface wetness ($0 \rightarrow 1$ implies dry $\rightarrow$ saturated)

- [anything else?]

2.4.3 Tracers

The model may include different types of tracers:

- condensed species (e.g., CO_2, H_2O, dust)

- chemically active species (in principle only at the moment)

- radiatively active gases (e.g., water vapor)

In the code, all tracers are stored in one three-dimensional array `q`, the third index of which corresponds to each individual tracer. In input and output files ("start.nc", "startfi.nc", see Section 5) tracers are stored separately using their individual names. Loading specific tracers requires that the approriate tracer names are set in the `traceur.def` file (see Section 7.2.3), and specific computations for given tracers (e.g. computing the water or CO_2 cycles) is controlled by setting the corresponding options in the `callphys.def` file (see Section 7.2.2).

Chapter 3

3D Dynamical Code

3.1 Discretisation of the dynamical equations

Extrait de la note de Robert Sadourny, Phu Le Van et Frédéric Hourdin, Laboratoire de Météorologie Dynamique.

[to be translated when I get the time...]

Le modèle climatique du LMD est bâti, comme tous les modèles de circulation générale atmosphérique, sur la résolution numérique des équations primitives de la météorologie décrites dans de nombreux ouvrages **?**. L'analyse présentée ici a été menée sur la nouvelle version de la dynamique du LMD écrite par Phu Le Van **?** sur une formulation de Robert Sadourny. Cette formulation diffère de l'ancienne essentiellement par deux points: dans la nouvelle formulation, la répartition des points en longitude et en latitude peut être changée arbitrairement. L'autre modification porte sur la répartition des points aux pôles[1].

La coordonnée verticale du modèle est la pression normalisée par sa valeur à la surface: $\sigma = p/p_s$. On utilise en fait σ aux niveaux inter-couches et $s = \sigma^\kappa$ au milieu des couches. On note X et Y les coordonnées horizontales:

X (resp. Y) est une fonction biunivoque de la longitude λ (resp. de la latitude ϕ). Ces deux fonctions peuvent être choisies de façon arbitraire dans le modèle LMDZ ce qui permet d'effectuer un zoom sur une région du globe particulière. Une grille de ce type est montrée sur la Figure 3.1. Les variables scalaires (température potentielle $\theta = c_p T/{p_s}^\kappa$, géopotentiel Φ et pression de surface p_s) sont évaluées aux points correspondant à des couples de valeurs entières $(X, Y) = (i, j)$. Les variables dynamiques sont décalées par rapport aux variables scalaires en utilisant une grille C dans la définition de Arakawa **?**: le vent zonal est calculé aux points $(X, Y) = (i + 1/2, j)$ et le vent méridien aux points $(X, Y) = (i, j + 1/2)$. La disposition des variables sur la grille est illustrée sur la Figure 3.2.

On utilise en fait les composantes covariantes ($\tilde{u}$ et $\tilde{v}$) et contravariantes ($\tilde{\tilde{u}}$ et $\tilde{\tilde{v}}$) du vent définies par

$$\begin{array}{llll} \tilde{u} = c_u u & \text{et} & \tilde{\tilde{u}} = u/c_u & \text{avec} \quad c_u = a\cos\phi\,(d\lambda/dX) \\ \tilde{v} = c_v v & \text{et} & \tilde{\tilde{v}} = v/c_v & \text{avec} \quad c_v = a\,(d\phi/dY) \end{array} \tag{3.1}$$

où u et v sont les composantes physiques du vecteur vent horizontal. On introduit également:

la pression extensive: $\tilde{p}_s$ (pression au sol multipliée par l'aire de la maille).

[1] Aux pôles sont calculés: le vent méridien dans l'ancienne formulation et les variables scalaires dans la nouvelle.

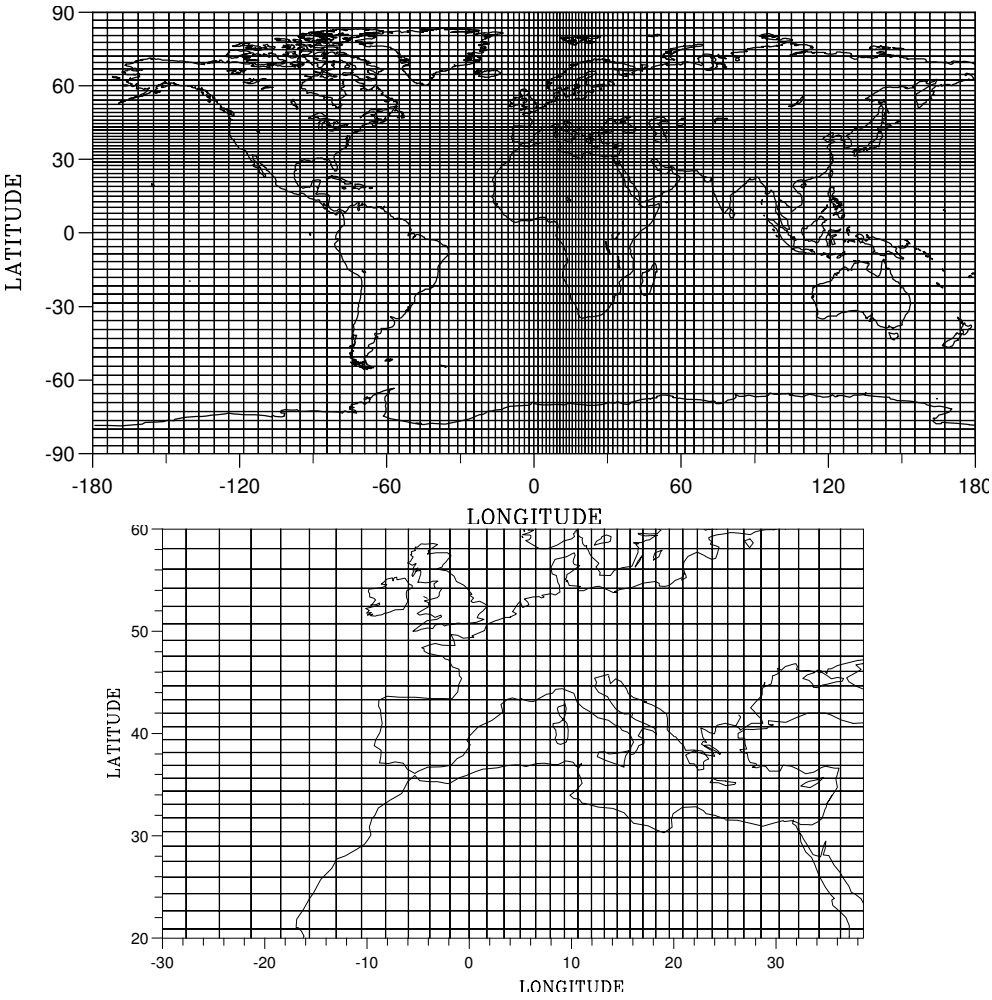

Figure 3.1: Grille obtenue avec 96 points en longitude et 73 en latitude et un zoom d'un facteur 3 centré sur la méditérannée (grille utilisée au laboratoire par Ali Harzallah)

.

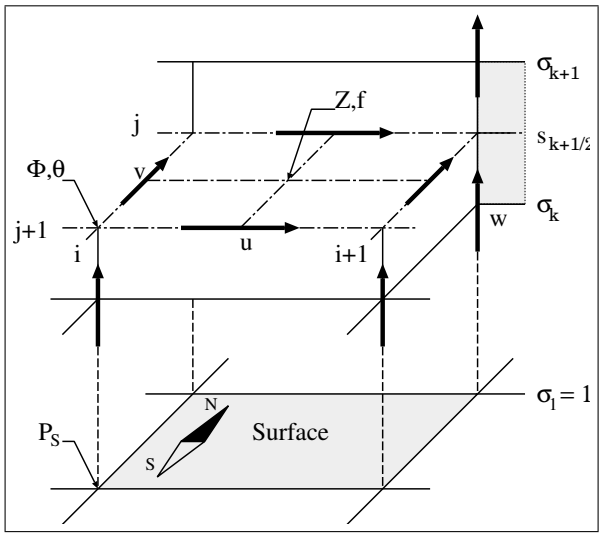

Figure 3.2: Disposition des variables dans la grille du LMD

les trois composantes du flux de masse:

$$U = \overline{\tilde{p}_s}^X \tilde{u},\ V = \overline{\tilde{p}_s}^Y \tilde{v} \text{ et } W = \tilde{p}_s \dot{\sigma} \text{ avec } \dot{\sigma} = \frac{d\sigma}{dt} \tag{3.2}$$

le facteur de Coriolis multiplié par l'aire de la maille: $f = 2\Omega \sin\phi c_u c_v$
où Ω est la vitesse de rotation de la planète.

la vorticité potentielle absolue:

$$Z = \frac{\mathcal{F}\left(\delta_X \tilde{v} - \delta_Y \tilde{u}\right) + f}{\overline{\tilde{p}_s}^{X,Y}} \tag{3.3}$$

l'énergie cinétique

$$K = \frac{1}{2}\left(\overline{\tilde{u}\tilde{u}}^X + \overline{\tilde{v}\tilde{v}}^Y\right) \tag{3.4}$$

La notation δX signifie simplement qu'on effectue la différence entre deux points consécutifs suivant la direction X. La notation $\overline{a}^X$ signifie qu'on prend la moyenne arithmétique de la quantité a suivant la direction X. $\mathcal{F}$ est un filtre longitudinale appliqué dans les régions polaires. Les équations discrétisées sont écrites sous la forme suivante:

équations du mouvement:

$$\frac{\partial \tilde{u}}{\partial t} - \overline{Z}^Y \overline{V}^{X,Y} + \delta_X \mathcal{F}\left(\Phi + K\right) + s\overline{\theta}^X \delta_X \mathcal{F}\left({p_s}^\kappa\right) - \frac{\overline{\tilde{u}_a}^{Y,Y} \delta_Z \overline{W}^X}{\overline{\tilde{p}_s}^X \delta_Z \sigma} + \frac{\delta_Z\left(\overline{W}^X \overline{\tilde{u}_a}^Z\right)}{\overline{\tilde{p}_s}^X \delta_Z \sigma} = S_{\tilde{u}} \tag{3.5}$$

où $\tilde{u}_a$ est la composante zonale covariante du vecteur vent absolu: $\tilde{u}_a = \tilde{u} + c_u a \Omega \cos\phi$ et

$$\frac{\partial \tilde{v}}{\partial t} + \overline{Z}^X \overline{U}^{X,Y} + \delta_Y \mathcal{F}\left(\Phi + K\right) + s\overline{\theta}^Y \delta_Y \mathcal{F}\left({p_s}^\kappa\right) - \frac{\overline{\tilde{v}}^{X,X} \delta_Z \overline{W}^Y}{\overline{\tilde{p}_s}^Y \delta_Z \sigma} + \frac{\delta_Z\left(\overline{W}^Y \overline{\tilde{v}}^Z\right)}{\overline{\tilde{p}_s}^X \delta_Z \sigma} = S_{\tilde{v}} \tag{3.6}$$

équation thermodynamique:

$$\frac{\partial\left(\tilde{p}_s\theta\right)}{\partial t}+\mathcal{F}\left[\delta_X\left(\overline{\theta}^X U\right)+\delta_Y\left(\overline{\theta}^Y V\right)\right]+\frac{\delta_Z\left(\overline{\theta}^Z W\right)}{\delta_Z\sigma}=S_\theta \tag{3.7}$$

équation hydrostatique:

$$\delta_Z\Phi=-p_s^\kappa\overline{\theta}^z\delta_Z s \tag{3.8}$$

équations de continuité:

$$\frac{\partial p_s}{\partial t}=\mathcal{F}\left[\sum_z\delta_Z\sigma\left(\delta_X U+\delta_Y V\right)\right] \tag{3.9}$$

$$\delta_Z W=-\delta_Z\sigma\left[\mathcal{F}\left(\delta_X U+\delta_Y V\right)+\frac{\partial p_s}{\partial t}\right] \tag{3.10}$$

On a noté S les termes sources dans les différentes équations. Dans ces termes sources, on distingue 1) d'une part les paramétrisations physiques mentionnées plus haut et qui font intervenir pour une maille donnée du modèle, tous les points situés sur une même verticale mais ceux-là seulement; 2) les opérateurs de dissipation horizontale, censés rendre compte des échanges entre échelles explicitement représentées dans le modèle et échelles sous-mailles. Ces opérateurs ont la structure de Laplaciens agissant sur des plans horizontaux c'est à dire qu'il font intervenir un voisin de chaque côté dans les deux directions horizontales. Cet opérateur est généralement itéré pour le rendre plus sélectif en échelle (plus on itère un laplacien et plus son effet sur les petites échelles devient important relativement).

3.2 High latitude filters

Extract adapted from Forget et al. [1999]

At high latitude a filter is applied near the singularity in the grid at the pole in order to satisfy the Courant-Friedrichs-Lewy numerical stability criterion without going to an excessively small timestep. In the original version of the dynamical code a classical Fourier filter was used, but we found that because the Martian polar atmosphere appears to be much more dynamically unstable than the Earth's polar atmosphere, a more efficient formulation (based on the grouping of adjacent gridpoints together) was necessary to avoid numerical instability.

In practice the following technique is used in the subroutine called groupeun.F *:*

- *The points are grouped in packets of* 2^{ngroup} *at the poles(e.g.* **ngroup**=*3* $\rightarrow$ *packets of 8), then* $2^{ngroup-1}$*,* $2^{ngroup-2}$*, etc. in the lower latitudes moving away from the pole*
- *The higher* **ngroup** *is, the more efficient the smoothing is, and the more stable the model.*
- *BUT,* **iim** *must be divisible by* 2^{ngroup} *!!!*

3.3 Dissipation

Extract adapted from Forget et al. [1999]

In the LMD grid point model, nonlinear interactions between explicitly resolved scales and subgrid-scale processes are parameterized by applying a scale-selective horizontal dissipation operator based on an n time iterated Laplacian Δ^n. For the grid point model, for instance, this can be written $\partial q/\partial t = ([-1]^n/\tau_{\text{diss}})(\delta x)^{2n}\Delta^n q$ where δx is the smallest horizontal distance represented in the model and τ_{diss} is the dissipation timescale for a st ructure of scale δx. These operators are necessary to ensure the grid point model numerical stability. In practice, the operator is separately applied to (1) potential temperature, (2) the divergence of the flow, and (3) its vorticity. We respectively use $n = 2$, $n = 1$, and $n = 2$ in the grid point model.

Note: In practice, values of n and τ_{diss} are adjustable and prescribed at the beginning of each run, in run definition file "run.def" (cf. 7.2.1)

3.4 Sponge layer

Extract adapted from Forget et al. [1999]

In the upper levels a sponge layer is also used in both models in an attempt to reduce spurious reflections of vertically propagating waves from the model top. Unlike the traditional Rayleigh friction formulation, this operates as a linear drag solely on the eddy components of the vorticity and divergence fields and is not scale-selective. The timescales on which it operates are typically half a day, 1 day, and 2 days at the three uppermost levels, respectively.

Note: the sponge layer "timescale" values and their extensions in altitude are adjustable and prescribed at the beginning of each run, in run definition file "run.def" (cf. 7.2.1)

Chapter 4

Physical parameterizations of the generic model: some references

4.1 General

The Generic Climate Model uses a large number of physical parameterizations based on various scientific theories. Some also use specific numerical methods. A list of these parameterizations is given below, along with the most appropriate references for each one. Most of these documents can be found at
`http://www.lmd.jussieu.fr/mars.html`.

General references: No documents attempt to give a complete scientific description of the current version of the GCM. Here's a reference to a Mars GCM description:

- *Forget et al.* [1999] (article published in the JGR)
- "Updated Detailed Design Document for the Model" (ESA contract, Work Package 6, 1999, available on the web) which is simply a compilation of the preceding article with a few additions that were published separately.

4.2 Radiative transfer

The radiative transfer parameterizations are used to calculate the heating and cooling ratios in the atmosphere and the radiative flux at the surface.

[TO WRITE: IMPORTANT SECTION - REFERENCES HERE ARE FOR MARS ONLY]

4.2.1 Absorption/emission and diffusion by dust:

Dust spatial distribution

(`dustopacity`)

- Vertical distribution and description of "MGS" and "Viking" scenarios in the ESA report *Mars Climate Database V3.0 Detailed Design Document* by Lewis et al. (2001), available on the web.
- For the "MY24" scenario, dust distribution obtained from assimilation of TES data is used (and read via the `readtesassim` routine).

Thermal IR radiation

(`lwmain`)

- Numerical method: *Toon et al.* [1989]
- Optical properties of dust: *Forget* [1998]

Solar radiation

(`swmain`)

- Numerical method: *Fouquart and Bonel* [1980]
- Optical properties of dust: see the discussion in *Forget et al.* [1999], which quotes *Ockert-Bell et al.* [1997] and *Clancy and Lee* [1991].

4.3 Subgrid atmospheric dynamical processes

4.3.1 Turbulent diffusion in the upper layer

(`vdifc`)

- Implicit numerical scheme in the vertical: see the thesis of Laurent Li (LMD, Université Paris 7, 1990), Appendix C2.
- Calculation of the turbulent diffusion coefficients: *Forget et al.* [1999].

4.3.2 Convection

(`convadj`)
See *Hourdin et al.* [1993]

4.4 Surface thermal conduction

(`soil`)
Thesis of Frédéric Hourdin (LMD, Université Paris 7, 1992) : section 3.3 (equations) and Appendix A (Numerical scheme).

4.5 CO_2 Condensation

In *Forget et al.* [1998] (article published in Icarus):
- Numerical method for calculating the condensation and sublimation levels at the surface and in the atmosphere (`newcondens`) explained in the appendix.
- Description of the numerical scheme for calculating the evolution of CO_2 snow emissivity (`co2snow`) explained in section 4.1

4.6 Tracer transport and sources

- "Van-Leer" transport scheme used in the dynamical part (`tracvl` and `vlsplt` in the dynamical part): *Hourdin and Armengaud* [1999]

- Transport by turbulent diffusion (in `vdifc`), convection (in `convadj`), sedimentation (`sedim`), dust lifting by winds (`dustlift`) : see note "Preliminary design of dust lifting and transport in the Model" (ESA contract, Work Package 4, 1998, available on the web).
- **Watercycle**, see *Montmessin et al.* [2004]

Chapter 5

Running the model: a practice simulation

This chapter is meant for first-time users of the LMD model. As the best introduction to the model is surely to run a simulation, here we explain how to go about it. All you will need are files necessary to build the GCM (all are in the `LMDZ.GENERIC` directory) as well as some initial states to initiate simulations (see below).
Once you have followed the example given below, you can then go on to change the control parameters and the initial states as you wish. A more detailed description of the model's organization as well as associated inputs and outputs are given in sections 6 and 7.

5.1 Installing the model from SVN

The first thing is to download the model from our SVN server. If you cannot use SVN, just find an old school way to get a copy of the basic model directory `LMDZ.GENERIC` (and all the other source files needed for visualization) and download it to your account. Then start directly from the fifth point.

- Go to the directory where you want to download the model. Not that only one directory (the root directory) will be added in the current directory.
- If svn is installed on your system, set up the root directory by tipping

  ```
  svn co "http://svn.lmd.jussieu.fr/Planeto/trunk" -N Name_of_root_directory
  cd Name_of_root_directory
  ```

- You can now download one of the LMDZ models (for Generic, Mars, Venus, Titan, ...) by tipping

  ```
  svn update LMDZ.MODEL_YOU_WANT
  ```

  For the Generic model, just tipe

  ```
  svn update LMDZ.GENERIC
  ```

  The contents of the directory that has been created are described in Chapter 6.

- For visualization of the simulations, yo will need some utilities that we might as well download now by doing

```
svn update UTIL
```

- Now we must set up the `makegcm` script that will perform the compilation of the model. Go into the `LMDZ.GENERIC` directory and edit the appropriate `makegcm_mycompiler` (hereafter called `makegcm`), where `mycompiler` is the compiler that you want to use. There are two important environment variables concerning source files that are initialized by `makegcm` and that we need to set properly:

  1. `LMDGCM`, the path to the source files. By default, the line

     ```
     setenv LMDGCM `readlink -f $scriptdir`
     ```

     allows `makegcm` to assume that it is executed in the root source directory so that this should work without any change. If `makegcm` does not find the source, you can enter manually the path by changing the above line by

     ```
     setenv LMDGCM "path/to/source/directory/LMDZ.GENERIC"
     ```

  2. `LIBOGCM`, the path to the compilation directory where all object files will be kept. By default, the line

     ```
     setenv LIBOGCM $LMDGCM/libo
     ```

     specifies that source will be kept in a `libo` directory created in `LMDZ.GENERIC`. You can also change that if needed.

- Install NetCDF http://www.unidata.ucar.edu/packages/netcdf/INSTALL.html and set environment variables `NCDFINC` and `NCDFLIB`:

  The latest version of the NetCDF package is available on the web at the following address: http://www.unidata.ucar.edu/software/netcdf along with instructions for building (or downloading precompiled binaries of) the library.

  Once the NetCDF library has been compiled (or downloaded), you should have access to the library `libnetcdf.a` itself, the various files (`netcdf.inc`, `netcdf.mod`, ...) to include in programs, and basic NetCDF software (*ncdump* and *ncgen*).

  To ensure that during compilation, the model can find the NetCDF library and include files, you must declare environment variables `NCDFLIB` and `NCDFINC`.

  `NCDFLIB` must contain the path to the directory containing the object library `libnetcdf.a` and `NCDFINC` must contain the path to the directory containing the include files (`netcdf.inc`,...)
  As for `LMDGCM` variable, these variables can be declared by changing the right line in `makegcm`

  ```
  setenv NCDFINC /wherever/is/netcdf/include
  setenv NCDFLIB /wherever/is/netcdf/lib
  ```

  For example, if working at LMD and with `ifort`, the path is

  ```
  setenv NCDFINC /donnees/emlmd/netcdf64-4.0.1_ifort/include
  setenv NCDFLIB /donnees/emlmd/netcdf64-4.0.1_ifort/lib
  ```

- Install software for loading and displaying NetCDF files such as GrAdS (http://grads.iges.org/grads/), Ferret (http://ferret.wrc.noaa.gov/Ferret), or Python. Some visualization scripts, especially for Python, can be found in the `UTIL` directory and will be described later.

- Finally, make sure that you have access to all the executables needed for building and using the model and remember to set environment variables to the correct corresponding pathes (note that if you do not want to have to redefine these every session, you should put the definitions in the corresponding `.cshrc` or `.bashrc` files).

  - UNIX function *make*
  - a Fortran compiler
  - ncdump
  - grads (or ferret)

5.2 Installing the model without SVN

Create an alias so that the compilation script **makegcm** is available from anywhere (more convinient than having to type the full path to the script, or copying it over where you want to run it). The `makegcm` script is in the LMDZ.GENERIC directory, which is referenced by the **LMDGCM** variable, so:
If using Csh:

```
alias makegcm $LMDGCM'/makegcm'
```

if using Bash:

```
alias makegcm=$LMDGCM/makegcm
```

5.3 Compiling the LMDZ.GENERIC model (sequential only)

Two options exist to compile the model.

1. Create an alias so that the compilation script `makegcm` is available from anywhere.
   If using Csh:

   ```
   alias makegcm 'path/to/LMDZ.GENERIC/makegcm'
   ```

   if using Bash:

   ```
   alias makegcm=path/to/LMDZ.GENERIC/makegcm
   ```

   Then the compilation is done by tipping

   ```
   makegcm -options gcm
   ```

   This solution can be convenient but is less flexible if you want to compile the model in many different configurations and keep track of it.

2. Create and edit an executable script (that we will call `compile`) in the directory where you will want to run the model. Put the line

   ```
   /path/to/the/model/I/use/makegcm -options gcm
   ```

The advantage of this option is that the `compile` is present in all of the working directories where the model is ran, allowing you to keep track of the options used.

Just remains to choose the options. The basic options are as follows

```
makegcm -d LONxLATxALT -p std -t XX -s YY -b IRxVI gcm
```

where `LONxLATxALT` are the number of grid cells in longitude, latitude and altitude, `XX` is the number of tracers, `YY` is the number of scatterers that will be taken into account in the radiative code and `IRxVI` is the number of spectral bands in the thermal emission and stellar part of the radiative code. The option `-debug` is available with most compilers. The code runs much more slowly but can output more user friendly bug report messages.

- Example 1: Compiling the generic model at grid resolution 64x48x20 for example, type (in compliance with the manual for the makegcm function given in section 6.4)

```
makegcm -d 64x48x20 -p std gcm
```

You can find executable **gcm.e** (the compiled model) in the directory where you ran the makegcm command.

- Example 2: Compiling the generic model with 2 tracers (e.g. water vapour and ice to simulate the water cycle):

```
makegcm -d 32x32x20 -t 2 -p std gcm
```

- Example 3: Compiling the the generic model to check for and trace errors (with ifort compiler - useful for debugging - warning, the model then runs very slowly!):

```
makegcm -d 32x32x20 -p std -O "-g -fpe0 -traceback" gcm
```

5.4 Compiling the LMDZ.COMMON model (sequential or parallel)

1. Prerequisites:
   - Downloaded LMDZ.COMMON and LMDZ.OTHER_MODEL containing the physic you want.
   - Available MPI library and wrapped compiler (mpif90, mpiifort,...)
   - Optional (but recommended) fcm:
     - LMD: /distrib/local/fcm/bin
     - Ciclad: /home/millour/FCM_V1.2/bin
     - Gnome: /san/home/millour/FCM_V1.2/bin
     - Other: fcm is just a collection of perl scripts; can be copied over on any other machine, or simply downloaded using svn:
       svn checkout http://forge.ipsl.jussieu.fr/fcm/svn/PATCHED/FCM_V1.2
2. Then choose the physic you want to couple with the LMDZ.COMMON dynamic core by creating a symbolic link in the LMDZ.COMMON/libf directory.
   If you want to use mars physic:

   ```
   cd LMDZ.COMMON/libf
   ln -s path/to/LMDZ.MARS/libf/phymars .
   ln -s path/to/LMDZ.MARS/libf/aeronomars .
   ```

   Here, we want the LMDZ.GENERIC physic phystd:

```
cd LMDZ.COMMON/libf
ln -s path/to/LMDZ.GENERIC/libf/phystd .
```

3. To compile in LMDZ.COMMON directory:

```
./makelmdz_fcm -s XX -t XX -d LONxLATxALT -b IRxVI -p physicSuffix
-arch archFile [-parallel mpi/mpi_omp] gcm
```

   - **physicSuffix** is `mars` for phymars, `std` for phystd...
   - **archFile** is the name of configuration files from LMDZ.COMMON/arch: use `CICLADifort` the ifort compiler in a CICLAD environment, `X64_ADA` for the ADA architecture...
   - To compile in parallel with mpi, add `-parallel mpi` option. By default it is serial code.
   - For hybrid MPI-OpenMP parallelisation, add `-parallel mpi_omp` option.
   - For faster compilation, the option `-j N` uses N simultaneous tasks.
   - `-full` option forces full (re)-compilation from scratch.
   - Created program is in LMDZ.COMMON/bin directory, with dimensions included in the program name. e.g.: gcm_64x48x29_phymars_para.e

NB: It is possible to compile without fcm by replacing `makelmdz_fcm` by `makelmdz`. Created program is in LMDZ.COMMON directory and named gcm.e.

5.5 Input files (initial states and def files)

- In directory `LMDZ.GENERIC/deftank` you will find some examples of run parameter files (`.def` files) which the model needs at runtime. The four files the model requires (they must be in the same directory as the executable `gcm.e`) are: **run.def** (described in section 7.2) **callphys.def** (see section 7.2.2), **gases.def**, **z2sig.def** and **traceur.def**.

The example `.def` files given in the `deftank` directory are for various configurations (e.g. model resolution, planet type), copy (and eventually rename these files to match the generic names) to the directory where you will run the model.

- Copy initial condition files **start.nc** and startfi.nc (described in section 7.2) to the same directory.
You can extract such files from **start_archive** 'banks of initial states' (i.e. files which contain collections of initial states from stndard scenarios and which can thus be used to check if the model is installed correctly) stored on the LMD website at
`http://www.lmd.jussieu.fr/~forget/datagcm/Starts`. See section 5.10 for a description of how to proceed to extract **start** files from **start_archives**.

[NOTE: WITH THE GENERIC MODEL WE ALMOST ALWAYS START FROM "startplanet" FILES]

5.6 Running the model

IMPORTANT: The following line MUST be in file run.def (or callphys.def):

```
planet_type = mars
```

for using LMDZ.MARS model or

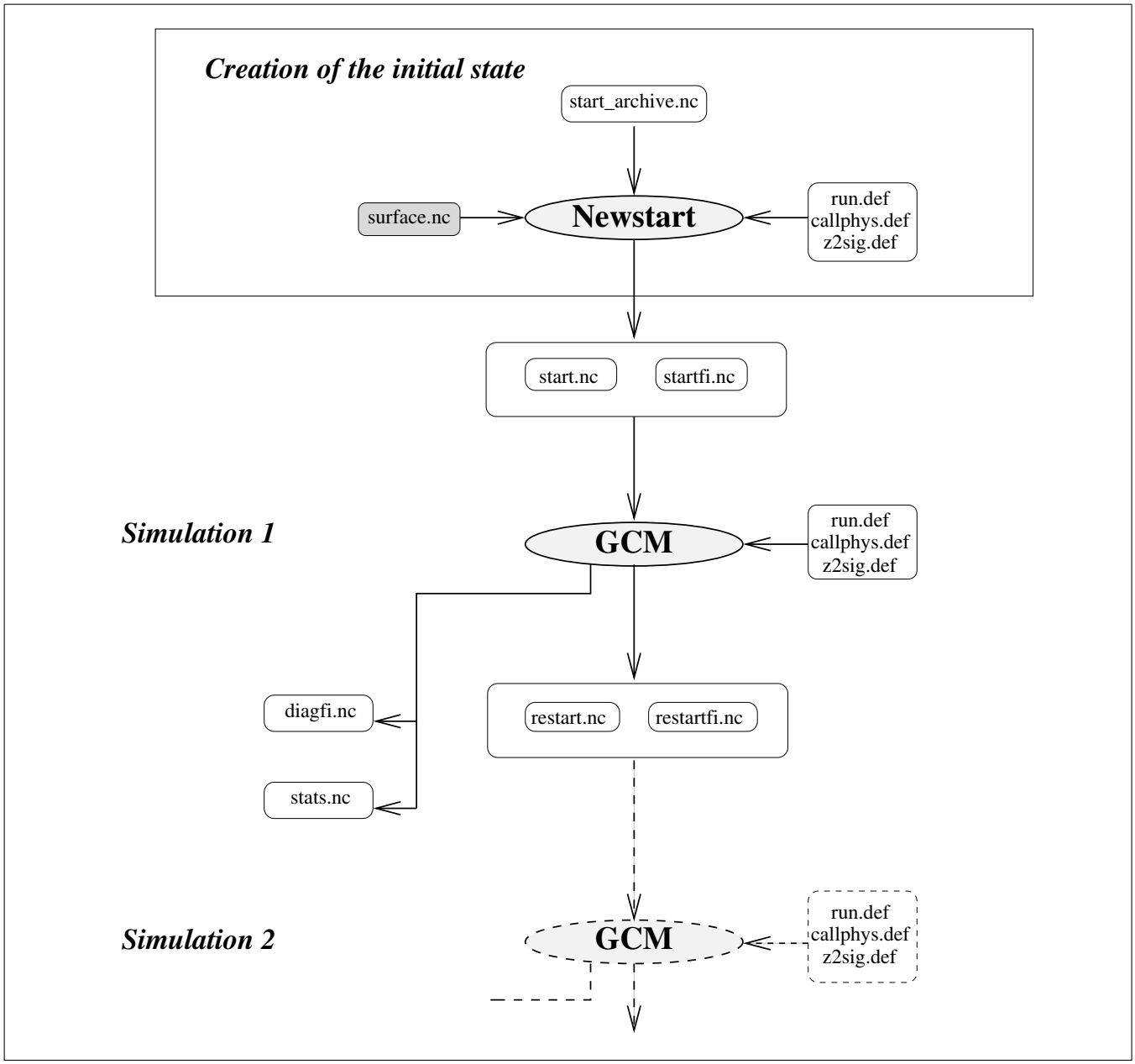

Figure 5.1: Input/output data

`planet_type = generic`

for using LMDZ.GENERIC model.

- To run the serial **gcm.e** interactively:
  Once you have the program **gcm.e**, input files **start.nc startfi.nc**, and parameter files **run.def, callphys.def, gases.def, traceur.def, and z2sig.def** in the same directory, simply execute the program to run a simulation:

  ```
  gcm.e
  ```

  You might need more memory. Use `ulimit -s unlimited` to change user limits.
  You might also want to keep all messages and diagnostics written to standard output (i.e. the screen). You should then redirect the standard output (and error) to some file, e.g. `gcm.out`:
  If using Csh:

  ```
  gcm.e >! gcm.out
  ```

  If using Bash:

  ```
  gcm.e > gcm.out 2>&1
  ```

- To run the MPI-parallel **gcm.e** interactively:

  ```
  mpirun -np N gcm.e > gcm.out 2>&1
  ```

  `-np N` specifies the number of procs to run on.
  IMPORTANT: one MUST use the `mpirun` command corresponding to the `mpif90` compiler specified in the `arch` file.
  Output files (restart.nc, diagfi.nc ,etc.) are just as when running in serial. But standard output messages are written by each process.
  If using chained simulations (run_mcd/run0 scripts), then the command line to run the gcm in `run0` must be adapted for local settings.
  NB: LMDZ.COMMON dynamics set to run in double precision, so keep `NC_DOUBLE` declaration (and real to double precision promotion) in the arch files.

- To run the hybrid parallel **gcm.e** interactively:

  ```
  export OMP_NUM_THREADS=2
  export OMP_STACKSIZE=2500MB
  mpirun -np 2 gcm.e > gcm.out 2>&1
  ```

  In this exemple, each of the 2 process MPI have 2 OpenMP tasks with a 2500MB memory.

- To run the MPI-parallel **gcm.e** with a job scheduler (different on each machine):

  ```
  PBS example (on Ciclad):
  #PBS -S  /bin/bash
  #PBS -N  job_mpi08
  #PBS -q short
  #PBS -j eo
  #PBS -l "nodes=1:ppn=8"
  # go to directory where the job was launched
  cd $PBS_O_WORKDIR
  mpirun gcm_64x48x29_phymars_para.e > gcm.out 2>&1
  ```

```
LoadLeveler example (on Gnome):
# @ job_name = job_mip8
# standard output file
# @ output = job_mpi8.out.$(jobid)
# standard error file
# @ error =  job_mpi8.err.$(jobid)
# job type
# @ job_type = mpich
# @ blocking = unlimited
# time
# @ class = AP
# Number of procs
# @ total_tasks = 8
# @ resources=ConsumableCpus(1) ConsumableMemory(2500 mb)
# @ queue
set -vx
mpirun gcm_32x24x11_phymars_para.e > gcm.out 2>&1

LoadLeveler example (on Ada):
module load intel/2012.0
# @ output  =  output.$(jobid)
# @ error = $(output)
# @ job_type = parallel
## Number of MPI process
# @ total_tasks = 8
## Memory used by each MPI process
# @ as_limit = 2500mb
# @ wall_clock_limit=01:00:00
# @ core_limit = 0
# @ queue
set -x
poe ./gcm.e -labelio yes > LOG 2>&1
```

- To run the hybrid MPI/OpenMP-parallel **gcm.e** with a job scheduler (different on each machine):

```
LoadLeveler example (on Gnome):
# @ job_name = job_mip8
# standard output file
# @ output = job_mpi8.out.$(jobid)
# standard error file
# @ error =  job_mpi8.err.$(jobid)
# job type
# @ job_type = mpich
# @ blocking = unlimited
# time
# @ class = AP
# Number of procs
# @ total_tasks = 8
# @ resources=ConsumableCpus(1) ConsumableMemory(5000 mb)
# @ queue
set -vx
export OMP_NUM_THREADS=2 #sinon par defaut, lance 8 threads OpenMP
export OMP_STACKSIZE=2500MB
```

```
mpirun gcm_32x24x11_phymars_para.e > gcm.out 2>&1
```

IMPORTANT: ConsumableMemory must be equal to OMP_NUM_THREADSxOMP_STACKSIZE. In this case, we are using 8x2 cores.

```
LoadLeveler example (on Ada):
module load intel/2012.0
# @ output  =  output.$(jobid)
# @ error = $(output)
# @ job_type = parallel
## Number of MPI process
# @ total_tasks = 8
## Number of OpenMP tasks attached to each MPI process
# @ parallel_threads = 2
## Memory used by each MPI process
# @ as_limit = 5gb
# @ wall_clock_limit=01:00:00
# @ core_limit = 0
# @ queue
set -x
export OMP_STACKSIZE=2500MB
poe ./gcm.e -labelio yes > LOG 2>&1
```

IMPORTANT: In this case, each core needs 2.5gb and we are using 2 OpenMP tasks for each MPI process so `as_limit` $= 2 \times 2.5$.

5.7 Visualizing the output files

As the model runs it generates output files **diagfi.nc** and **stats.nc** files. The former contains instantaneous values of various fields and the later statistics (over the whole run) of some variables.

5.7.1 Using GrAds to visualize outputs

If you have never used the graphic software **GrAds**, we strongly recommend spending half an hour to familiarize yourself with it by following the demonstration provided for that purpose. The demo is fast and easy to follow and you will learn the basic commands. To do this read file

```
/distrib/local/grads/sample
```

For example, to visualize files `diagfi.nc` and `stats.nc`

NetCDF files `diagfi.nc` and `stats.nc` can be accessed directly using GrAdS thanks to utility program gradsnc, (the user does not need to intervene).

To visualize the temperature in the 5th layer using file `diagfi.nc` for example:

- GrAdS session:

  `grads` *return*

  return (opens a landscape window)

  `ga-> sdfopen diagfi.nc`

  `ga-> query file` (displays info about the open file, including the name of the stored variables. Shortcut: *q file*)

`ga-> set z 5` (fixes the altitude to the 5th layer)

`ga-> set t 1` (fixes the time to the first stored value)

`ga-> query dims` (indicates the fixed values for the 4 dimensions. Shortcut: *q dims*)

`ga-> display temp` (displays the temperature card for the 5th layer and for the first time value stored. Shortcut: *d T*)

`ga-> clear` (clears the display. Shortcut: *c*)

`ga-> set gxout shaded` (not a contour plot, but a shaded one)

`ga-> display temp`

`ga-> set gxout contour` (returns to contour mode to display the levels)

`ga-> display temp` (superimposes the contours if the clear command is not used)

5.8 Resuming a simulation

At the end of a simulation, the model generates **restart** files (files `restart.nc` and `restartfi.nc`) which contain the final state of the model. As shown in figure 5.1, these files (which are of the same format as the start files) can later be used as initial states for a new simulation.

The **restart** files just need to be renamed:

```
mv restart.nc start.nc
mv restartfi.nc startfi.nc
```

and running a simulation with these will in fact resume the simulation from where the previous run ended.

5.9 Chain simulations

In practice, we recommend running a chain of simulations lasting several days or longer (or hundreds of days at low resolution).

To do this, a script named `run0` is available in `LMDZ.GENERIC/deftank` , which should be used as follows:

- Set the length of each simulation in `run.def` (i.e. set the value of `nday`)
- Set the maximum number of simulations at the beginning of the `run0` script (i.e. set the value of `nummax`)
- Copy start files `start.nc startfi.nc` over and rename them `start0.nc startfi0.nc`.
- Run script `run0`

`run0` runs a series of simulations that generate the indexed output files (e.g. `start1, startfi1, diagfi1`, etc.) including files `lrun1, lrun2`, etc. containing the redirection of the display and the information about the run.

NOTE: to restart a series of simulations after a first series (for example, starting from `start5 and startfi5`), just write the index of the initial files (e.g. 5) in the file named `num_run`. If `num_run` exists, the model will start from the index written in `num_run`. If not it will start from, `start0 and startfi0`.

NOTE: A script is available for performing annual runs with 12 seasons at 30^o solar longitude as it is in the database (script `run_mcd`, also found in directory `deftank`). This script functions with script run0. Just set the number of simulations to 1 in run0. Then copy run.def into run.def.ref and set nday to 9999 in this file. To start from startN.c, edit the file run_mcd and comment (with a #) the N months already created and describe N in `num_run`. Then run `run_mcd`.

5.10 Creating and modifying initial states

5.10.1 Using program "newstart"

When working with the generic model, it is common to start with simple initial conditions (e.g., isothermal, motionless atmosphere). For this we create an initial state using **newstart**. In practice, we usually take an old initial state, and simply modify it.

Like the GCM, the program **newstart** must be compiled (using the `makegcm` script) to the required grid resolution. For example:

```
makegcm -d 32x32x20 -p std newstart
```

Then run

```
newstart.e
```

The program then gives you two options:

```
From which kind of files do you want to create newstart and startfi files
    0 - from a file start_archive
    1 - from files start and startfi
```

- - Option "1" allows you to read and modify the information needed to create a new initial state from the files `start.nc, startfi.nc`
- - Option "0" allows you to read and modify the information needed to create a new initial state from file `start_archive.nc` (whatever the `start_archive.nc` grid resolution is).

If you use tracers, make sure that they are taken into account in your start files (either start or start_archive).

Then answer to the various questions in the scroll menu. These questions allow you to modify the initial state for the following parameters.

```
First set of questions:
Change values in tab_cntrl ? :
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
(Current values given above)

(3)          day_ini : Initial day (=0 at Ls=0)
(19)              z0 :  surface roughness (m)
(21)       emin_turb :  minimal energy (PBL)
(20)         lmixmin : mixing length (PBL)
(26)         emissiv : ground emissivity
(24 et 25)   emisice : CO2 ice max emissivity
(22 et 23)  albedice : CO2 ice cap albedos
(31 et 32) iceradius : mean scat radius of CO2 snow
(33 et 34) dtemisice : time scale for snow metamorphism
(27)        tauvis : mean dust vis. reference opacity
```

```
(35)      volcapa : soil volumetric heat capacity
(18)     obliquit : planet obliquity (deg)
(17)     peri_day : periastron date (sols since Ls=0)
(15)     periastr : min. star-planet dist (Mkm)
(16)     apoastr  : max. star-planet (Mkm)
(14)     year_day : length of year (in sols)
(5) rad : radius of the planet (m)
(6) omeg : planet rotation rate (rad/s)
(7) g : gravity (m/s2)
(8) mugaz : molecular mass of the atmosphere (g/mol)
(9) rcp : r/Cp
(10) daysec : length of a sol (s)

Second set of questions :
flat : no topography ("aquaplanet")
bilball : uniform albedo and thermal inertia
coldspole : cold subsurface and high albedo at S.pole
qname : change tracer name
q=0 : ALL tracer =zero
q=x : give a specific uniform value to one tracer
ini_q : tracers initialisation for chemistry, water and ice
ini_q-H2O : tracers initialisation for chemistry and ice
ini_q-iceH2O : tracers initialisation for chemistry only
noglacier : Remove tropical H2O ice if |lat|<45
watercapn : H2O ice on permanent N polar cap
watercaps : H2O ice on permanent S polar cap
oborealis : H2O ice across Vastitas Borealis
iceball   : Thick ice layer all over surface
wetstart  : start with a wet atmosphere
isotherm  : Isothermal Temperatures, wind set to zero
radequi   : Earth-like rad. eq. temperature profile and winds set to zero
co2ice=0 : remove CO2 polar cap
ptot : change total pressure
emis : change surface emissivity
therm_ini_s : Set soil thermal inertia to reference suface values
```

Program **newstart.e** creates files `restart.nc` and `restartfi.nc` that you generally need to rename (for instance rename them in start0.nc and startfi0.nc if you want to use run0 or run_mcd, starting with season 0; rename them `start.nc` and `startfi.nc` if you just want to perform one run with `gcm.e`).

5.10.2 Creating the initial start_archive.nc file

Archive file `start_archive.nc` is created from files `start.nc` and `startfi.nc` by program **start2archive**. Program **start2archive** compiles to the same grid resolution as the `start.nc` and `startfi.nc` grid resolution. For example:

```
makegcm -d 32x32x20 -p std start2archive
```

Then run `start2archive.e`

You now have a `start_archive.nc` file for one season that you can use with newstart. If you want to gather other states obtained at other times of year, rerun `start2archive.e` with the `start.nc` and `startfi.nc` corresponding to these. These additional initial states will automatically be added to the `start_archive.nc` file present in the directory.

5.10.3 Changing the horizontal or vertical grid resolution

To run at a different grid resolution than available initial conditions files, one needs to use tools **newstart** and **start2archive**

For example, to create initial states at grid resolution 32×24×25 from NetCDF files `start` and `startfi` at grid resolution 64×48×32 :

- Create file `start_archive.nc` with **start2archive.e** compiled at grid resolution 64×48×32 using **old file** `z2sig.def` **used previously**
- Create files `newstart.nc` and `newstartfi.nc` with **newstart.e** compiled at grid resolution 32×24×25, using **new file** `z2sig.def`

[NOT RELEVANT??] If you want to create starts files with tracers for 50 layers using a `start_archive.nc` obtained for 32 layers, do not forget to use the `ini_q` option in newstart in order to correctly initialize tracers value for layer 33 to layer 50. You just have to answer yes to the question on thermosphere initialization if you want to initialize the thermosphere part only (l=33 to l=50), and no if you want to initialize tracers for all layers (l=0 to l=50).

Chapter 6

Program organization and compilation script

All the elements of the LMD model are in the **LMDZ.GENERIC** directory (and subdirectories). As explained in Section 5, this directory should be associated with environment variable **LMDGCM**:
If using Csh:

```
setenv  LMDGCM /where/you/put/the/model/LMDZ.GENERIC
```

If using Bash:

```
export  LMDGCM=/where/you/put/the/model/LMDZ.GENERIC
```

Here is a brief description of the **LMDZ.GENERIC** directory contents:

```
libf/  All the model FORTRAN Sources (.F or .F90)
       and  include files (.h) organised in sub-directories
      (physics (phystd), dynamics (dyn3d), filters (filtrez)...)

deftank/   A collection of examples of parameter files required
              to run the GCM (run.def, callphys.def, ...)

makegcm  Script that should be used to compile the GCM as well
          as related utilities (newstart, start2archive, testphys1d)

create_make_gcm   Executable used to create the makefile.
                  This command is run automatically by
                  "makegcm" (see below).
```

6.1 Organization of the model source files

The model source files are stored in various sub directories in directory **libf**. These subdirectories correspond to the different parts of the model:

grid: mainly made up of ”dimensions.h” file, which contains the parameters that define the model grid, i.e. the number of points in longitude (IIM), latitude (JJM) and altitude (LLM), as well as the number of tracers (NQMX).

dyn3d: contains the dynamical subroutines.

bibio: contains some generic subroutines not specifically related to physics or dynamics but used by either or both.

phymars: contains the physics routines.

filtrez: contains the longitudinal filter sources applied in the upper latitudes, where the Courant-Friedrich-Levy stability criterion is violated.

6.2 Programming

The model is written in **Fortran-77** and **Fortran-90**.

- The program sources are written in **"file.F"** or **"file.F90"** files. The extension .F is the standard extension for fixed-form Fortran and the extension .F90 is for free-form Fortran. These files must be preprocessed (by a**C preprocessor** such as (cpp)) before compilation (this behaviour is, for most compilers, implicitly obtained but using a capital F in the extention of the file names).
- Constants are placed in COMMON declarations, located in the common "include" files **"file.h"**
- In general, variables are passed from subroutine to subroutine as arguments (and never as COMMON blocks).
- In some parts of the code, for "historical" reasons, the following rule is sometimes used: in the subroutine, the variables (ex: `name`) passed as an argument by the calling program are given the prefix `p` (ex: `pname`) while the local variables are given the prefix `z` (ex: `zname`). As a result, several variables change their prefix (and thus their name) when passing from a calling subroutine to a called subroutine. We're trying to eliminate this as the code is developed.

6.3 Model organization

Figure 6.1 describes the main subroutines called by physiq.F. OBSOLETE - FOR MARS ONLY!!!

6.4 Compiling the model

Technically, the model is compiled using the Unix utility `make`. The file `makefile`, which describes the code dependencies and requirements, is created automatically by the script

```
create_make_gcm
```

This utility script recreates the `makefile` file when necessary, for example, when a source file has been added or removed since the last compilation.

None of this is visible to the user. To compile the model just run the command

```
makegcm
```

with adequate options (e.g. `makegcm -d 62x48x32 -p mars gcm`), as discussed below and described in section 5.3.

The `makegcm` command compiles the model (**gcm**) and related utilities (**newstart**, **start2archive**, **testphys1d**). A detailed description of how to use it and of the various parameters that can be supplied is given in the help manual below (which will also be given

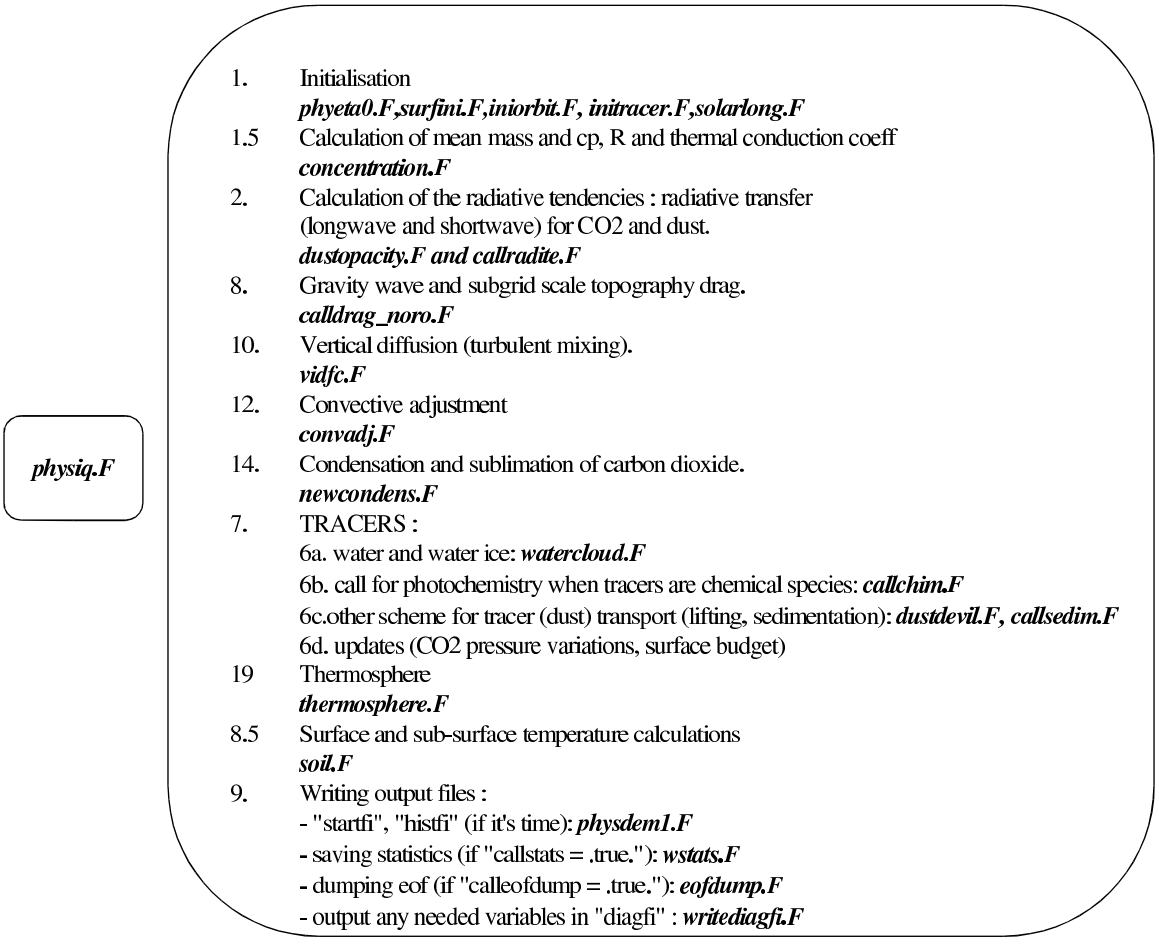

Figure 6.1: Organigram of subroutine function physiq.F90

by the `makegcm -h` command).
Note that before compiling the GCM with `makegcm` you should have set the environment variable **LIBOGCM** to a path where intermediate objects and libraries will be generated.
If using Csh:

```
setenv  LIBOGCM /where/you/want/objects/to/go/libo
```

If using Bash:

```
export  LIBOGCM=/where/you/want/objects/to/go/libo
```

Help manual for the makegcm script

```
makegcm [Options] prog


The makegcm script:
-------------------

1. compiles a series of subroutines located in the $LMDGCM/libf
 sub-directories.
 The objects are then stored in the libraries in $LIBOGCM.

2. then, makegcm compiles program prog.f located by default in
$LMDGCM/libf/dyn3d and makes the link with the libraries.

Environment Variables '$LMDGCM' and '$LIBOGCM'
 must be set as environment variables or directly
 in the makegcm file.

The makegcm command is used to control the different versions of the model
 in parallel, compiled using the compilation options
 and the various dimensions, without having to recompile the whole model.

The FORTRAN libraries are stored in directory $LIBOGCM.


OPTIONS:
--------

The following options can either be defined by default by editing the
makegcm "script", or in interactive mode:

-d imxjmxlm  where im, jm, and lm are the number of longitudes,
             latitudes and vertical layers respectively.

-t ntrac   Selects the number of tracers present in the model

             Options -d and -t overwrite file
             $LMDGCM/libf/grid/dimensions.h
             which contains the 3 dimensions of the
             horizontal grid
             im, jm, lm plus the number of tracers passively advected
             by the dynamics ntrac,
             in 4 PARAMETER FORTRAN format
             with a new file:
             $LMDGCM/libf/grid/dimension/dimensions.im.jm.lm.tntrac
             If the file does not exist already
             it is created by the script
             $LMDGCM/libf/grid/dimension/makdim

-p PHYS    Selects the set of physical parameterizations
           you want to compile the model with.
           The model is then compiled using the physical
           parameterization sources in directory:
            $LMDGCM/libf/phyPHYS
```

```
-g grille  Selects the grid type.
           This option overwrites file
           $LMDGCM/libf/grid/fxyprim.h
           with file
           $LMDGCM/libf/grid/fxy_grille.h
           the grid can take the following values:
           1. reg - the regular grid
           2. sin - to obtain equidistant points in terms of sin(latitude)
           3. new - to zoom into a part of the globe

-O "compilation options" set of fortran compilation options to use

-include path
           Used if the subroutines contain #include files (ccp) that
           are located in directories that are not referenced by default.

-adjnt     Compiles the adjoint model to the dynamical code.

-filtre  filter
           To select the longitudinal filter in the polar regions.
           "filter" corresponds to the name of a directory located in
           $LMDGCM/libf. The standard filter for the model is "filtrez"
           which can be used for a regular grid and for a
           grid with longitudinal zoom.

-link "-Ldir1 -lfile1 -Ldir2 -lfile2 ..."
           Adds a link to FORTRAN libraries
           libfile1.a, libfile2.a ...
           located in directories dir1, dir2 ...respectively
           If dirn is a directory with an automatic path
           (/usr/lib ... for example)
           there is no need to specify  -Ldirn.
```

Chapter 7

Input/Output

7.1 NetCDF format

GCM input/output data are written in **NetCDF** format (Network Common Data Form). NetCDF is an interface used to store and access geophysical data, and a library that provides an implementation of this interface. The NetCDF library also defines a machine-independent format for representing scientific data. Together, the interface, library and format support the creation, access and sharing of scientific data. NetCDF was developed at the Unidata Program Center in Boulder, Colorado. The freely available source can be obtained from the Unidata websitehttp://www.unidata.ucar.edu/software/netcdf.

A data set in NetCDF format is a single file, as it is self-descriptive.

7.1.1 NetCDF file editor: ncdump

The editor is included in the NetCDF library. By default it generates an ASCII representation as standard output from the NetCDF file specified at the input.

Main commands for ncdump

ncdump diagfi.nc

dump contents of NetCDF file `diagfi.nc` to standard output (i.e. the screen).

ncdump -c diagfi.nc

Displays the **coordinate** variable values (variables which are also dimensions), as well as the declarations, variables and attribute values. The values of the non-coordinate variable data are not displayed at the output.

ncdump -h diagfi.nc

Shows only the informative header of the file, which is the declaration of the dimensions, variables and attributes, but not the values of these variables. The output is identical to that in option **-c** except for the fact that the coordinated variable values are not included.

ncdump -v var1,...,varn diagfi.nc

The output includes the specific variable values, as well as all the dimensions, variables and attributes. More that one variable can be specified in the list following this option. The list must be a simple argument for the command, and must not contain any spaces. If no variable is specified, the command displays all the values of the variables in the file by default.

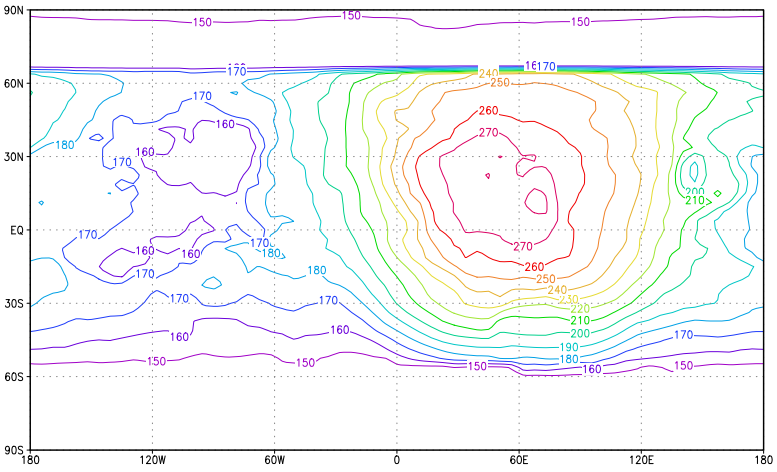

Figure 7.1: Example of temperature data (in this case for present-day Mars) at a given time using GrADS visualization

7.1.2 Graphic visualization of the NetCDF files using GrAds

GrAdS (The Grid Analysis and Display System) is a graphic software developed by Brian Doty at the "Center for Ocean-Land-Atmosphere (COLA)".

One of its functions is to enable data stored in NetCDF format to be visualized directly. In figure 7.1 for example, we can see the GrADS visualization of the temperature data at a given moment. However, unlike NetCDF, GrADS only recognizes files where all the variables are stored on the same horizontal grid. These variables can be in 1, 2, 3 or 4 dimensions (X,Y,Z and t).

GrADS can also be obtained on the WWWhttp://grads.iges.org/grads/.

7.2 Input and parameter files

The (3D version of the) GCM requires the input of two initialization files (in NetCDF format):
-**start.nc** contains the initial states of the dynamical variables.
-**startfi.nc** contains the initial states of the physical variables.
Note that collections of initial states can be retreived at:
`http://www.lmd.jussieu.fr/~forget/datagcm/Starts`
Extracting `start.nc` and `startfi.nc` from these archived requires using program `newstart`, as described in section 5.10.

To run, the GCM also requires the four following parameter files (ascii text files):
-**run.def** the parameters of the dynamical part of the program, and the temporal integration of the model.
-**callphys.def** the parameters for calling the physical part.
-**traceur.def** the names of the tracer to use.
-**z2sig.def** the vertical distribution of the atmospheric layers.
Examples of these parameter files can be found in the `LMDZ.MARS/deftank` directory.

7.2.1 run.def

A typical `run.def` file is given as an example below. The choice of variables to be set is simple (e.g. `nday` number of modeled days to run), while the others do not need to be changed for normal use.
The format of the `run.def` file is quite straightforward (and flexible): values given to parameters must be given as:

```
parameter = value
```

Any blank line or line beginning with symbol **#** is a comment, and instruction lines may be written in any order. Moreover, not specifying a parameter/value set (e.g. deleting it or commenting it out) means you want the GCM to use a default built-in value. Additionally, one may use a specific keyword **INCLUDEDEF** to specify another (text) file in which to also read values of parameters; e.g.:

```
INCLUDEDEF=callphys.def
```

Here are some details about some of the parameters which may be set in `run.def`:

- **day_step**, the number of dynamical steps per day to use for the time integration. This needs to be large enough for the model to remain stable (this is related to the CFL stability criterion which essentially depends on the horizontal resolution of the model). On Mars, in theory, the GCM can run with `day_step`=480 using the 64×48 grid, but model stability improves when this number is higher: `day_step`=960 is recommended when using the 64×48 grid. According to the CFL criterion, `day_step` should vary in proportion with the resolution: for example `day_step`=480 using the 32×24 horizontal resolution. Note that `day_step` must also be divisible by `iperiod`. For other planets... [FINISH]

- **tetagdiv, tetagrot, tetatemp** control the dissipation intensity. It is better to limit the dissipation intensity (tetagdiv, tetagrot, tetatemp should not be too low). However the model diverges if tetagdiv, tetagrot, tetatemp are too high, especially if there is a lot of dust in the atmosphere.
  Example used with nitergdiv=1 and nitergrot=niterh=2 :
  - using the 32×24 grid tetagdiv=6000 s ; tetagrot=tetatemp=30000 s
  - using the 64×48 grid: tetagdiv=3000 s ; tetagrot=tetatemp=9000 s

- **idissip** is the time step used for the dissipation: dissipation is computed and added every `idissip` dynamical time step. If `idissip` is too short, the model waste time in these calculations. But if idissip is too long, the dissipation will not be parametrized correctly and the model will be more likely to diverge. A check must be made, so that: `idissip` < `tetagdiv`×`daystep`/86400 (same rule for `tetagrot` and `tetatemp`). This is tested automatically during the run.

- **iphysiq** is the time step used for the physics: physical tendencies are computed every `iphysiq` dynamical time step. In practice, we usually set the physical time step to be of the order of half an hour. We thus generally set `iphysiq`= `day_step`/48

Example of run.def file:

```
#------------------------------
# Parametres de controle du run
#------------------------------

# Nombre de jours d'integration
     nday=669

# nombre de pas par jour (multiple de iperiod) ( ici pour  dt = 1 min )
```

```
 day_step = 960

# periode pour le pas Matsuno (en pas)
  iperiod=5

# periode de sortie des variables de controle (en pas)
  iconser=120

# periode d'ecriture du fichier histoire (en jour)
    iecri=100

# periode de stockage fichier histmoy (en jour)
 periodav=60.

# periode de la dissipation (en pas)
  idissip=5

# choix de l'operateur de dissipation (star ou  non star )
 lstardis=.true.

# avec ou sans coordonnee hybrides
 hybrid=.true.

# nombre d'iterations de l'operateur de dissipation   gradiv
nitergdiv=1

# nombre d'iterations de l'operateur de dissipation  nxgradrot
nitergrot=2

# nombre d'iterations de l'operateur de dissipation  divgrad
   niterh=2

# temps de dissipation des plus petites long.d ondes pour u,v (gradiv)
 tetagdiv=10000.

# temps de dissipation des plus petites long.d ondes pour u,v(nxgradrot)
 tetagrot=10000.

# temps de dissipation des plus petites long.d ondes pour  h ( divgrad)
 tetatemp=10000.

# coefficient pour gamdissip
  coefdis=0.

# choix du shema d'integration temporelle (Matsuno ou Matsuno-leapfrog)
  purmats=.false.

# avec ou sans physique
   physic=.true.

# periode de la physique (en pas)
  iphysiq=20

# choix d'une grille reguliere
  grireg=.true.

# frequence (en pas) de l'ecriture du fichier diagfi
 ecritphy=1920

# longitude en degres du centre du zoom
   clon=63.

# latitude en degres du centre du zoom
   clat=0.

# facteur de grossissement du zoom,selon longitude
  grossismx=1.
```

```
# facteur de grossissement du zoom ,selon latitude
 grossismy=1.

#  Fonction  f(y)  hyperbolique  si = .true.  , sinon  sinusoidale
  fxyhypb=.false.

# extension en longitude  de la zone du zoom  ( fraction de la zone totale)
   dzoomx= 0.

# extension en latitude de la zone  du zoom  ( fraction de la zone totale)
   dzoomy=0.

#  raideur du zoom en  X
    taux=2.

#  raideur du zoom en  Y
    tauy=2.

#  Fonction  f(y) avec y = Sin(latit.) si = .TRUE. ,  Sinon  y = latit.
  ysinus= .false.

# Avec sponge layer
  callsponge  = .true.

# Sponge:  mode0(u=v=0), mode1(u=umoy,v=0), mode2(u=umoy,v=vmoy)
  mode_sponge= 2

# Sponge:  hauteur de sponge (km)
  hsponge= 90

# Sponge:  tetasponge (secondes)
  tetasponge = 50000

# some definitions for the physics, in file 'callphys.def'
INCLUDEDEF=callphys.def
```

7.2.2 callphys.def

The `callphys.def` file (along the same format as the `run.def` file) contains parameter/value sets for the physics.

Example of callphys.def file:

```
## Orbit / general options
## ~~~~~~~~~~~~~~~~~~~~~~
# Run with or without tracer transport ?
tracer    = .true.
# Diurnal cycle ?  if diurnal=false, diurnally averaged solar heating
diurnal   = .true.
# Seasonal cycle ? if season=false, Ls stays constant, to value set in "start"
season    = .true.
# Tidally resonant orbit ? must have diurnal=false, correct rotation rate in newstart
tlocked   = .false.
# Tidal resonance ratio ? ratio T_orbit to T_rotation
nres      = 10
# Write some more output on the screen ?
lwrite    = .false.
# Save statistics in file "stats.nc" ?
callstats = .true.
# Test energy conservation of model physics ?
enertest  = .true.
```

```
## Radiative transfer options
## ~~~~~~~~~~~~~~~~~~~~~~~~~~
# call radiative transfer?
callrad    = .true.
# the rad. transfer is computed every "iradia" physical timestep
iradia     = 4
# call multilayer correlated-k radiative transfer ?
corrk      = .true.
# folder in which correlated-k data is stored ?
corrkdir   = CO2_H2Ovar
# call visible gaseous absorption in radiative transfer ?
callgasvis = .true.
# Include Rayleigh scattering in the visible ?
rayleigh   = .true.
# Characteristic planetary equilibrium (black body) temperature
# This is used only in the aerosol radiative transfer setup. (see aerave.F)
tplanet    = 215.
# Output spectral OLR in 1D/3D?
specOLR    = .false.
# Output global radiative balance in file 'rad_bal.out' - slow for 1D!!
meanOLR    = .true.
# Variable gas species: Radiatively active ?
varactive  = .true.
# Variable gas species: Fixed vertical distribution ?
varfixed   = .false.
# Variable gas species: Saturation percentage value at ground ?
satval     = 0.0

## Star type
## ~~~~~~~~~
startype = 1
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# The choices are:
#
# startype = 1 Sol        (G2V-class main sequence)
# startype = 2 Ad Leo     (M-class, synthetic)
#   startype = 3        GJ644
#   startype = 4        HD128167
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Stellar flux at 1 AU. Examples:
# 1366.0 W m-2 Sol today
# 1024.5 W m-2 Sol today x 0.75 = weak early Sun
# 18.462 W m-2 The feeble Gl581
# 19.960 W m-2 Gl581 with e=0.38 orbital average
Fat1AU = 1024.5

## Tracer and aerosol options
## ~~~~~~~~~~~~~~~~~~~~~~~~~~
# Gravitational sedimentation of tracers (KEEP FALSE FOR NOW) ?
sedimentation = .false.

## Other physics options
## ~~~~~~~~~~~~~~~~~~~~~
# call turbulent vertical diffusion ?
calldifv = .true.
# call convective adjustment ?
calladj  = .true.
# call thermal conduction in the soil ?
callsoil = .true.

########################################
## extra specific options for Early Mars
########################################

## Tracer and aerosol options
## ~~~~~~~~~~~~~~~~~~~~~~~~~~
# Fixed aerosol distributions?
```

```
aerofixed     = .false.
# Varying H2O cloud fraction?
CLFvarying    = .false.
# H2O cloud fraction?
CLFfixval     = 0.5
# number mixing ratio of CO2 ice particles
Nmix_co2      = 100000.
# number mixing ratio of water ice particles
Nmix_h2o      = 100000.

## Water options
## ~~~~~~~~~~~~~
# Model water cycle
water         = .true.
# Model water cloud formation
watercond     = .true.
# Model water precipitation (including coagulation etc.)
waterrain     = .true.
# WATER: Precipitation threshold (simple scheme only) ?
rainthreshold = 0.0011
# Include hydrology ?
hydrology     = .true.
# H2O snow (and ice) albedo ?
albedosnow    = 0.5
# Maximum sea ice thickness ?
maxicethick   = 0.05
# Freezing point of seawater (degrees C) ?
Tsaldiff      = 0.0

## CO2 options
## ~~~~~~~~~~~
# gas is non-ideal CO2 ?
nonideal      = .false.
# call CO2 condensation ?
co2cond       = .true.
# Set initial temperature profile to 1 K above CO2 condensation everywhere?
nearco2cond   = .false.
```

7.2.3 traceur.def

Tracers in input (`start.nc` and `startfi.nc`) and output files (`restart.nc` and `restartfi.nc`) are stored using individual tracer names (e.g. `co2` for CO2 gas, `h2o_vap` for water vapour, `h2o_ice` for water ice, ...).
The first line of the `traceur.def` file (an ASCII file) must contain the number of tracers to load and use (this number should be the same as given to the `-t` option of the `makegcm` script when the GCM was compiled), followed by the tracer names (one per line). Note that if the corresponding tracers are not found in input files `start.nc` and `startfi.nc`, then the tracer is initialized to zero.

Example of a traceur.def file: (with water vapour and ice tracers)

```
2
h2o_ice
h2o_vap
```

7.2.4 z2sig.def

The `z2sig.def` file contains the pseudo-altitudes (in km) at which the user wants to set the vertical levels.
Note that levels should be unevenly spread, with a higher resolution near the surface in order to capture the rapid variations of variables there. It is recommended to use the altitude

levels as set in the `z2sig.def` file provided in the `deftank` directory.

Example of z2sig.def file

```
10.00000     H: atmospheric scale height (km) (used as a reference only)
0.0040       Typical pseudo-altitude (m) for 1st layer (z=H*log(sigma))
0.018       ,, ,,  ,, ,, ,, ,,  ,, ,, ,,  2nd layer, etc...
0.0400
0.1000
0.228200
0.460400
0.907000
1.73630
3.19040
5.54010
8.97780
13.5138
18.9666
25.0626
31.5527
38.4369
45.4369
52.4369
```

7.2.5 Initialization files: start and startfi

Files `start.nc` and `startfi.nc`, like all the NetCDF files of the GCM, are constructed on the same model (see NetCDF file composition, figure 7.2). They contain:
- a header with a "control" variable followed by a series of variables defining the (physical and dynamical) grids
- a series of non temporal variables that give information about surface conditions on the planet.
- a "time" variable giving the values of the different instants at which the temporal variables are stored (a single time value (t=0) for start, as it describes the dynamical initial states, and no time values for startfi, as it describes only a physical state).

To visualize the contents of a `start.nc` file using the `ncdump` command:

ncdump -h start.nc

```
netcdf start {
dimensions:
        index = 100 ;
        rlonu = 33 ;
        latitude = 25 ;
        longitude = 33 ;
        rlatv = 24 ;
        altitude = 18 ;
        interlayer = 19 ;
        Time = UNLIMITED ; // (1 currently)
variables:
        float controle(index) ;
                controle:title = "Parametres de controle" ;
        float rlonu(rlonu) ;
                rlonu:title = "Longitudes des points U" ;
        float rlatu(latitude) ;
                rlatu:title = "Latitudes des points U" ;
        float rlonv(longitude) ;
```

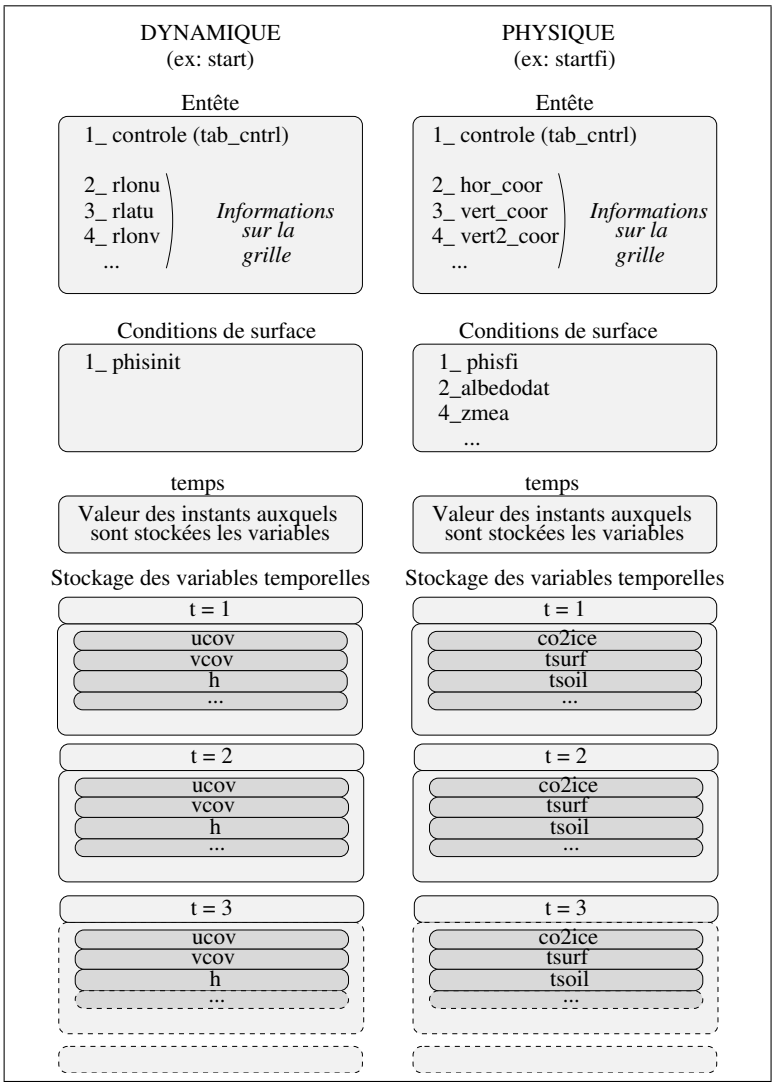

Figure 7.2: Organization of NetCDF files

```
        rlonv:title = "Longitudes des points V" ;
float rlatv(rlatv) ;
        rlatv:title = "Latitudes des points V" ;
float ap(interlayer) ;
        ap:title = "Coef A: hybrid pressure levels" ;
float bp(interlayer) ;
        bp:title = "Coef B: hybrid sigma levels" ;
float aps(altitude) ;
        aps:title = "Coef AS: hybrid pressure at midlayers" ;
float bps(altitude) ;
        bps:title = "Coef BS: hybrid sigma at midlayers" ;
float presnivs(altitude) ;
float latitude(latitude) ;
        latitude:units = "degrees_north" ;
        latitude:long_name = "North latitude" ;
float longitude(longitude) ;
        longitude:long_name = "East longitude" ;
        longitude:units = "degrees_east" ;
float altitude(altitude) ;
        altitude:long_name = "pseudo-alt" ;
```

```
                altitude:units = "km" ;
                altitude:positive = "up" ;
        float cu(latitude, rlonu) ;
                cu:title = "Coefficient de passage pour U" ;
        float cv(rlatv, longitude) ;
                cv:title = "Coefficient de passage pour V" ;
        float aire(latitude, longitude) ;
                aire:title = "Aires de chaque maille" ;
        float phisinit(latitude, longitude) ;
                phisinit:title = "Geopotentiel au sol" ;
        float Time(Time) ;
                Time:title = "Temps de simulation" ;
                Time:units = "days since    1-01-01 00:00:00" ;
        float ucov(Time, altitude, latitude, rlonu) ;
                ucov:title = "Vitesse U" ;
        float vcov(Time, altitude, rlatv, longitude) ;
                vcov:title = "Vitesse V" ;
        float teta(Time, altitude, latitude, longitude) ;
                teta:title = "Temperature" ;
        float h2o_ice(Time, altitude, latitude, longitude) ;
                h2o_ice:title = "Traceur h2o_ice" ;
        float h2o_vap(Time, altitude, latitude, longitude) ;
                h2o_vap:title = "Traceur h2o_vap" ;
        float masse(Time, altitude, latitude, longitude) ;
                masse:title = "C est quoi ?" ;
        float ps(Time, latitude, longitude) ;
                ps:title = "Pression au sol" ;

// global attributes:
                :title = "Dynamic start file" ;
}
```

List of contents of a `startfi.nc` file:

ncdump -h startfi.nc

```
netcdf startfi {
dimensions:
        index = 100 ;
        physical_points = 738 ;
        subsurface_layers = 18 ;
        nlayer_plus_1 = 19 ;
        number_of_advected_fields = 3 ;
variables:
        float controle(index) ;
                controle:title = "Control parameters" ;
        float soildepth(subsurface_layers) ;
                soildepth:title = "Soil mid-layer depth" ;
        float longitude(physical_points) ;
                longitude:title = "Longitudes of physics grid" ;
        float latitude(physical_points) ;
                latitude:title = "Latitudes of physics grid" ;
        float area(physical_points) ;
                area:title = "Mesh area" ;
        float phisfi(physical_points) ;
                phisfi:title = "Geopotential at the surface" ;
        float albedodat(physical_points) ;
                albedodat:title = "Albedo of bare ground" ;
        float ZMEA(physical_points) ;
                ZMEA:title = "Relief: mean relief" ;
        float ZSTD(physical_points) ;
                ZSTD:title = "Relief: standard deviation" ;
        float ZSIG(physical_points) ;
                ZSIG:title = "Relief: sigma parameter" ;
        float ZGAM(physical_points) ;
                ZGAM:title = "Relief: gamma parameter" ;
        float ZTHE(physical_points) ;
```

```
                ZTHE:title = "Relief: theta parameter" ;
        float co2ice(physical_points) ;
                co2_ice:title = "CO2 ice cover" ;
        float inertiedat(subsurface_layers, physical_points) ;
                inertiedat:title = "Soil thermal inertia" ;
        float tsurf(physical_points) ;
                tsurf:title = "Surface temperature" ;
        float tsoil(subsurface_layers, physical_points) ;
                tsoil:title = "Soil temperature" ;
        float emis(physical_points) ;
                emis:title = "Surface emissivity" ;
        float q2(nlayer_plus_1, physical_points) ;
                q2:title = "pbl wind variance" ;
        float h2o_ice(physical_points) ;
                h2o_ice:title = "tracer on surface" ;

// global attributes:
                :title = "Physical start file" ;
}
```

Physical and dynamical headers There are two types of headers: one for the physical headers, and one for the dynamical headers. The headers always begin with a "control' variable (described below), that is allocated differently in the physical and dynamical parts. The other variables in the header concern the (physical and dynamical) grids. They are the following:

the horizontal coordinates
- **rlonu**, **rlatu**, **rlonv**, **rlatv** for the dynamical part,
- **lati**, **long** for the physical part,

the coefficients for passing from the physical grid to the dynamical grid
- **cu**,**cv** only in the dynamical header

and finally, the grid box areas
- **aire** for the dynamical part,
- **area** for the physical part.

Surface conditions The surface conditions are mostly given in the physical NetCDF files by variables:
- **phisfi** for the initial state of surface geopotential,
- **albedodat** for the bare ground albedo,
- **inertiedat** for the surface thermal inertia,
- **zmea**, **zstd**, **zsig**, **zgam** and **zthe** for the subgrid scale topography.

For the dynamics:
- **physinit** for the initial state of surface geopotential

Remark: variables **phisfi** and **physinit** contain the same information (surface geopotential), but **phisfi** gives the geopotential values on the physical grid, while **physinit** give the values on the dynamical grid.

Physical and dynamical state variables To save disk space, the initialization files store the variables used by the model, rather than the "natural" variables.

For the dynamics:

- **ucov** and **vcov** the covariant winds
  These variables are linked to the "natural" winds by
  `ucov = cu * u` and `vcov = cv * v`

- **teta** the potential temperature,
  or more precisely, the potential enthalpy linked to temperature **T** by $\theta = T\left(\frac{P}{Pref}\right)^{-K}$

- the tracers,

- **ps** surface pressure.

- **masse** the atmosphere mass in each grid box.

"Vectorial" variables **ucov** and **vcov** are stored on "staggered" grids u and v respectively (in the dynamics) (see section 2.2).
Scalar variables **h**, **q** (tracers), **ps**, **masse** are stored on the "scalar" grid of the dynamical part.

For the physics:

- **co2ice** surface dry ice,

- **tsurf** surface temperature,

- **tsoil** temperatures at different layers under the surface,

- **emis** surface emissivity,

- **q2** wind variance,
  or more precisely, the square root of the turbulent kinetic energy.

- the surface "tracer" budget (kg.m^{-2}),

All these variables are stored on the "physical" grid (see section 2.2).

The "control" array Both physical and dynamical headers of the GCM NetCDF files start with a **controle** variable. This variable is an array of 100 reals (the vector called `tab_cntrl` in the program), which contains the program control parameters. Parameters differ between the physical and dynamical sections, and examples of both are listed below. The contents of table `tab_cntrl` can also be checked with the command `ncdump -ff -v controle`.

The "control" array in the header of a dynamical NetCDF file: start

```
tab_cntrl(1)  = FLOAT(iim) ! number of nodes along longitude
tab_cntrl(2)  = FLOAT(jjm) ! number of nodes along latitude
tab_cntrl(3)  = FLOAT(llm) ! number of atmospheric layers
tab_cntrl(4)  = FLOAT(idayref) ! initial day
tab_cntrl(5)  = rad   ! radius of the planet
tab_cntrl(6)  = omeg  ! rotation of the planet (rad/s)
tab_cntrl(7)  = g     ! gravity (m/s2) ~3.72 for Mars
tab_cntrl(8)  = cpp
tab_cntrl(9) = kappa   ! = r/cp
tab_cntrl(10) = daysec ! lenght of a sol (s) ~88775
tab_cntrl(11) = dtvr   ! dynamical time step (s)
tab_cntrl(12) = etot0  ! total energy
tab_cntrl(13) = ptot0  ! total pressure
tab_cntrl(14) = ztot0  ! total enstrophy
tab_cntrl(15) = stot0  ! total enthalpy
```

```
       tab_cntrl(16) = ang0   ! total angular momentum
       tab_cntrl(17) = pa
       tab_cntrl(18) = preff  ! reference pressure (Pa)
       tab_cntrl(19)  = clon  ! longitude of center of zoom
       tab_cntrl(20)  = clat  ! latitude of center of zoom
       tab_cntrl(21)  = grossismx ! zooming factor, along longitude
       tab_cntrl(22)  = grossismy ! zooming factor, along latitude

       tab_cntrl(24) = dzoomx ! extention (in longitude) of zoom
       tab_cntrl(25) = dzoomy ! extention (in latitude) of zoom

       tab_cntrl(27) = taux ! stiffness factor of zoom in longitude
       tab_cntrl(28) = tauy ! stiffness factor of zoom in latitude
```

The "controle" array in the header of a physical NetCDF file: startfi.nc

```
c Informations on the physics grid
      tab_cntrl(1) = float(ngridmx)  ! number of nodes on physics grid
      tab_cntrl(2) = float(nlayermx) ! number of atmospheric layers
      tab_cntrl(3) = day_ini + int(time)         ! initial day
      tab_cntrl(4) = time -int(time)            ! initiale time of day

c Informations about Mars, used by dynamics and physics
      tab_cntrl(5) = rad      ! radius of Mars (m) ~3397200
      tab_cntrl(6) = omeg     ! rotation rate (rad.s-1)
      tab_cntrl(7) = g        ! gravity (m.s-2) ~3.72
      tab_cntrl(8) = mugaz    ! Molar mass of the atmosphere (g.mol-1) ~43.49
      tab_cntrl(9) = rcp      !  = r/cp  ~0.256793 (=kappa dans dynamique)
      tab_cntrl(10) = daysec  ! length of a sol (s)  ~88775

      tab_cntrl(11) = phystep  ! time step in the physics
      tab_cntrl(12) = 0.
      tab_cntrl(13) = 0.

c Informations about Mars, only for physics
      tab_cntrl(14) = year_day  ! length of year (sols) ~668.6
      tab_cntrl(15) = periheli  ! min. Sun-Mars distance (Mkm) ~206.66
      tab_cntrl(16) = aphelie   ! max. SUn-Mars distance (Mkm) ~249.22
      tab_cntrl(17) = peri_day  ! date of perihelion (sols since N. spring)
      tab_cntrl(18) = obliquit  ! Obliquity of the planet (deg) ~23.98

c Boundary layer and turbulence
      tab_cntrl(19) = z0        ! surface roughness (m) ~0.01
      tab_cntrl(20) = lmixmin   ! mixing length ~100
      tab_cntrl(21) = emin_turb ! minimal energy ~1.e-8

c Optical properties of polar caps and ground emissivity
      tab_cntrl(22) = albedice(1)  ! Albedo of northern cap ~0.5
      tab_cntrl(23) = albedice(2)  ! Albedo of southern cap ~0.5
      tab_cntrl(24) = emisice(1)   ! Emissivity of northern cap ~0.95
      tab_cntrl(25) = emisice(2)   ! Emissivity of southern cap ~0.95
      tab_cntrl(26) = emissiv      ! Emissivity of martian soil ~.95
      tab_cntrl(31) = iceradius(1) ! mean scat radius of CO2 snow (north)
      tab_cntrl(32) = iceradius(2) ! mean scat radius of CO2 snow (south)
      tab_cntrl(33) = dtemisice(1) ! time scale for snow metamorphism (north)
      tab_cntrl(34) = dtemisice(2) ! time scale for snow metamorphism (south)

c dust aerosol properties
      tab_cntrl(27) = tauvis      ! mean visible optical depth

      tab_cntrl(28) = 0.
      tab_cntrl(29) = 0.
      tab_cntrl(30) = 0.

! Soil properties:
      tab_cntrl(35) = volcapa ! soil volumetric heat capacity
```

7.3 Output files

7.3.1 NetCDF restart files - restart.nc and restartfi.nc

These files are of the exact same format as `start.nc` and `startfi.nc`

7.3.2 NetCDF file - diagfi.nc

NetCDF file `diagfi.nc` stores the instantaneous physical variables throughout the simulation at regular intervals (set by the value of parameter `ecritphy` in parameter file `run.def`; note that `ecritphy` should be a multiple of `iphysiq` as well as a divisor of `day_step`).
Any variable from any sub-routine of the physics can be stored by calling subroutine `writediagfi`
Illustrative example of the contents of a `diagfi.nc` file (using ncdump):
ncdump -h diagfi.nc

```
netcdf diagfi {
dimensions:
        Time = UNLIMITED ; // (12 currently)
        index = 100 ;
        rlonu = 65 ;
        latitude = 49 ;
        longitude = 65 ;
        rlatv = 48 ;
        interlayer = 26 ;
        altitude = 25 ;
        subsurface_layers = 18 ;
variables:
        float Time(Time) ;
                Time:long_name = "Time" ;
                Time:units = "days since 0000-00-0 00:00:00" ;
        float controle(index) ;
                controle:title = "Control parameters" ;
        float rlonu(rlonu) ;
                rlonu:title = "Longitudes at u nodes" ;
        float latitude(latitude) ;
                latitude:units = "degrees_north" ;
                latitude:long_name = "North latitude" ;
        float longitude(longitude) ;
                longitude:long_name = "East longitude" ;
                longitude:units = "degrees_east" ;
        float altitude(altitude) ;
                altitude:long_name = "pseudo-alt" ;
                altitude:units = "km" ;
                altitude:positive = "up" ;
        float rlatv(rlatv) ;
                rlatv:title = "Latitudes at v nodes" ;
        float aps(altitude) ;
                aps:title = "hybrid pressure at midlayers" ;
                aps:units = "Pa" ;
        float bps(altitude) ;
                bps:title = "hybrid sigma at midlayers" ;
                bps:units = "" ;
        float ap(interlayer) ;
                ap:title = "hybrid pressure at interlayers" ;
                ap:units = "Pa" ;
        float bp(interlayer) ;
                bp:title = "hybrid sigma at interlayers" ;
                bp:units = "" ;
        float soildepth(subsurface_layers) ;
                soildepth:long_name = "Soil mid-layer depth" ;
                soildepth:units = "m" ;
                soildepth:positive = "down" ;
```

```
        float cu(latitude, rlonu) ;
                cu:title = "Conversion coefficients cov <--> natural" ;
        float cv(rlatv, longitude) ;
                cv:title = "Conversion coefficients cov <--> natural" ;
        float aire(latitude, longitude) ;
                aire:title = "Mesh area" ;
        float phisinit(latitude, longitude) ;
                phisinit:title = "Geopotential at the surface" ;
        float emis(Time, latitude, longitude) ;
                emis:title = "Surface emissivity" ;
                emis:units = "w.m-1" ;
        float tsurf(Time, latitude, longitude) ;
                tsurf:title = "Surface temperature" ;
                tsurf:units = "K" ;
        float ps(Time, latitude, longitude) ;
                ps:title = "surface pressure" ;
                ps:units = "Pa" ;
        float co2ice(Time, latitude, longitude) ;
                co2ice:title = "co2 ice thickness" ;
                co2ice:units = "kg.m-2" ;
        float mtot(Time, latitude, longitude) ;
                mtot:title = "total mass of water vapor" ;
                mtot:units = "kg/m2" ;
        float icetot(Time, latitude, longitude) ;
                icetot:title = "total mass of water ice" ;
                icetot:units = "kg/m2" ;
        float tauTES(Time, latitude, longitude) ;
                tauTES:title = "tau abs 825 cm-1" ;
                tauTES:units = "" ;
        float h2o_ice_s(Time, latitude, longitude) ;
                h2o_ice_s:title = "surface h2o_ice" ;
                h2o_ice_s:units = "kg.m-2" ;
}
```

The structure of the file is thus as follows:

- the dimensions
- variable “time” containing the time of the timestep stored in the file (in Martian days since the beginning of the run)
- variable “control” containing many parameters, as described above.
- from “ rhonu” to ’phisinit”: a list of data describing the geometrical coordinates of the data file, plus the surface topography
- finally, all the 2D or 3D data stored in the run.

7.3.3 Stats files

As an option (`stats` must be set to `.true.` in `callphys.def`), the model can accumulate any variable from any subroutine of the physics by calling subroutine `wstat`

This save is performed at regular intervals 12 times a day. An average of the daily evolutions over the whole run is calculated (for example, for a 10 day run, the averages of the variable values at 0hTU, 2hTU, 4hTU,...24hTU are calculated), along with RMS standard deviations of the variables. This ouput is given in file `stats.nc`.

Illustrative example of the contents of a `stats.nc` file (using ncdump):
ncdump -h stats.nc

```
netcdf stats {
dimensions:
```

```
        latitude = 49 ;
        longitude = 65 ;
        altitude = 25 ;
        llmp1 = 26 ;
        Time = UNLIMITED ; // (12 currently)
variables:
        float Time(Time) ;
                Time:title = "Time" ;
                Time:units = "days since 0000-00-0 00:00:00" ;
        float latitude(latitude) ;
                latitude:title = "latitude" ;
                latitude:units = "degrees_north" ;
        float longitude(longitude) ;
                longitude:title = "East longitude" ;
                longitude:units = "degrees_east" ;
        float altitude(altitude) ;
                altitude:long_name = "altitude" ;
                altitude:units = "km" ;
                altitude:positive = "up" ;
        float aps(altitude) ;
                aps:title = "hybrid pressure at midlayers" ;
                aps:units = "" ;
        float bps(altitude) ;
                bps:title = "hybrid sigma at midlayers" ;
                bps:units = "" ;
        float ps(Time, latitude, longitude) ;
                ps:title = "Surface pressure" ;
                ps:units = "Pa" ;
        float ps_sd(Time, latitude, longitude) ;
                ps_sd:title = "Surface pressure total standard deviation over th
e season" ;
                ps_sd:units = "Pa" ;
        float tsurf(Time, latitude, longitude) ;
                tsurf:title = "Surface temperature" ;
                tsurf:units = "K" ;
        float tsurf_sd(Time, latitude, longitude) ;
                tsurf_sd:title = "Surface temperature total standard deviation o
ver the season" ;
                tsurf_sd:units = "K" ;
        float co2ice(Time, latitude, longitude) ;
                co2ice:title = "CO2 ice cover" ;
                co2ice:units = "kg.m-2" ;
        float co2ice_sd(Time, latitude, longitude) ;
                co2ice_sd:title = "CO2 ice cover total standard deviation over t
he season" ;
                co2ice_sd:units = "kg.m-2" ;
        float fluxsurf_lw(Time, latitude, longitude) ;
                fluxsurf_lw:title = "Thermal IR radiative flux to surface" ;
                fluxsurf_lw:units = "W.m-2" ;
        float fluxsurf_lw_sd(Time, latitude, longitude) ;
                fluxsurf_lw_sd:title = "Thermal IR radiative flux to surface tot
al standard deviation over the season" ;
                fluxsurf_lw_sd:units = "W.m-2" ;
        float fluxsurf_sw(Time, latitude, longitude) ;
                fluxsurf_sw:title = "Solar radiative flux to surface" ;
                fluxsurf_sw:units = "W.m-2" ;
        float fluxsurf_sw_sd(Time, latitude, longitude) ;
                fluxsurf_sw_sd:title = "Solar radiative flux to surface total st
andard deviation over the season" ;
                fluxsurf_sw_sd:units = "W.m-2" ;
        float fluxtop_lw(Time, latitude, longitude) ;
                fluxtop_lw:title = "Thermal IR radiative flux to space" ;
                fluxtop_lw:units = "W.m-2" ;
        float fluxtop_lw_sd(Time, latitude, longitude) ;
                fluxtop_lw_sd:title = "Thermal IR radiative flux to space total
standard deviation over the season" ;
                fluxtop_lw_sd:units = "W.m-2" ;
```

```
        float fluxtop_sw(Time, latitude, longitude) ;
                fluxtop_sw:title = "Solar radiative flux to space" ;
                fluxtop_sw:units = "W.m-2" ;
        float fluxtop_sw_sd(Time, latitude, longitude) ;
                fluxtop_sw_sd:title = "Solar radiative flux to space total stand
ard deviation over the season" ;
                fluxtop_sw_sd:units = "W.m-2" ;
        float dod(Time, latitude, longitude) ;
                dod:title = "Dust optical depth" ;
                dod:units = "" ;
        float dod_sd(Time, latitude, longitude) ;
                dod_sd:title = "Dust optical depth total standard deviation over
 the season" ;
                dod_sd:units = "" ;
        float temp(Time, altitude, latitude, longitude) ;
                temp:title = "Atmospheric temperature" ;
                temp:units = "K" ;
        float temp_sd(Time, altitude, latitude, longitude) ;
                temp_sd:title = "Atmospheric temperature total standard deviatio
n over the season" ;
                temp_sd:units = "K" ;
        float u(Time, altitude, latitude, longitude) ;
                u:title = "Zonal (East-West) wind" ;
                u:units = "m.s-1" ;
        float u_sd(Time, altitude, latitude, longitude) ;
                u_sd:title = "Zonal (East-West) wind total standard deviation ov
er the season" ;
                u_sd:units = "m.s-1" ;
        float v(Time, altitude, latitude, longitude) ;
                v:title = "Meridional (North-South) wind" ;
                v:units = "m.s-1" ;
        float v_sd(Time, altitude, latitude, longitude) ;
                v_sd:title = "Meridional (North-South) wind total standard devia
tion over the season" ;
                v_sd:units = "m.s-1" ;
        float w(Time, altitude, latitude, longitude) ;
                w:title = "Vertical (down-up) wind" ;
                w:units = "m.s-1" ;
        float w_sd(Time, altitude, latitude, longitude) ;
                w_sd:title = "Vertical (down-up) wind total standard deviation o
ver the season" ;
                w_sd:units = "m.s-1" ;
        float rho(Time, altitude, latitude, longitude) ;
                rho:title = "Atmospheric density" ;
                rho:units = "none" ;
        float rho_sd(Time, altitude, latitude, longitude) ;
                rho_sd:title = "Atmospheric density total standard deviation ove
r the season" ;
                rho_sd:units = "none" ;
        float q2(Time, altitude, latitude, longitude) ;
                q2:title = "Boundary layer eddy kinetic energy" ;
                q2:units = "m2.s-2" ;
        float q2_sd(Time, altitude, latitude, longitude) ;
                q2_sd:title = "Boundary layer eddy kinetic energy total standard
 deviation over the season" ;
                q2_sd:units = "m2.s-2" ;
        float vmr_h2ovapor(Time, altitude, latitude, longitude) ;
                vmr_h2ovapor:title = "H2O vapor volume mixing ratio" ;
                vmr_h2ovapor:units = "mol/mol" ;
        float vmr_h2ovapor_sd(Time, altitude, latitude, longitude) ;
                vmr_h2ovapor_sd:title = "H2O vapor volume mixing ratio total sta
ndard deviation over the season" ;
                vmr_h2ovapor_sd:units = "mol/mol" ;
        float vmr_h2oice(Time, altitude, latitude, longitude) ;
                vmr_h2oice:title = "H2O ice volume mixing ratio" ;
                vmr_h2oice:units = "mol/mol" ;
        float vmr_h2oice_sd(Time, altitude, latitude, longitude) ;
```

```
                vmr_h2oice_sd:title = "H2O ice volume mixing ratio total standar
d deviation over the season" ;
                vmr_h2oice_sd:units = "mol/mol" ;
        float mtot(Time, latitude, longitude) ;
                mtot:title = "total mass of water vapor" ;
                mtot:units = "kg/m2" ;
        float mtot_sd(Time, latitude, longitude) ;
                mtot_sd:title = "total mass of water vapor total standard deviat
ion over the season" ;
                mtot_sd:units = "kg/m2" ;
        float icetot(Time, latitude, longitude) ;
                icetot:title = "total mass of water ice" ;
                icetot:units = "kg/m2" ;
        float icetot_sd(Time, latitude, longitude) ;
                icetot_sd:title = "total mass of water ice total standard deviat
ion over the season" ;
                icetot_sd:units = "kg/m2" ;
}
```

The structure of the file is simillar to the `diagfi.nc` file, except that, as stated before, the average of variables are given for 12 times of the day and that RMS standard deviation are also provided.

Chapter 8

Water Cycle Simulation

To simulate the water cycle with the LMD Generic Model:

- In `callphys.def`, set tracer to true: `tracer=.true.`. In the radiative transfer sub-section, chose an appropriate correlated-k database that includes the effect of water vapour (e.g. `corrkdir=CO2H2Ovar`), and set `varactive=.true.`, `varfixed=.false.`. In the water cycle sub-section you can chose various parameters - see below for a standard example.

```
## Orbit / general options
## ~~~~~~~~~~~~~~~~~~~~~~
# Run with or without tracer transport ?
tracer    = .true.
# Diurnal cycle ?  if diurnal=false, diurnally averaged solar heating
diurnal   = .true.
# Seasonal cycle ? if season=false, Ls stays constant, to value set in "start"
season    = .true.
# Tidally resonant orbit ? must have diurnal=false, correct rotation rate in newstart
tlocked   = .false.
# Tidal resonance ratio ? ratio T_orbit to T_rotation
nres      = 10
# Write some more output on the screen ?
lwrite    = .false.
# Save statistics in file "stats.nc" ?
callstats = .true.
# Test energy conservation of model physics ?
enertest  = .true.

## Radiative transfer options
## ~~~~~~~~~~~~~~~~~~~~~~~~~
# call radiative transfer?
callrad    = .true.
# the rad. transfer is computed every "iradia" physical timestep
iradia     = 4
# call multilayer correlated-k radiative transfer ?
corrk      = .true.
# folder in which correlated-k data is stored ?
corrkdir   = CO2_H2Ovar
# call visible gaseous absorption in radiative transfer ?
callgasvis = .true.
# Include Rayleigh scattering in the visible ?
rayleigh   = .true.
# Characteristic planetary equilibrium (black body) temperature
# This is used only in the aerosol radiative transfer setup. (see aerave.F)
tplanet    = 215.
# Output spectral OLR in 1D/3D?
specOLR    = .false.
# Output global radiative balance in file 'rad_bal.out' - slow for 1D!!
```

```
meanOLR    = .true.
# Variable gas species: Radiatively active ?
varactive  = .true.
# Variable gas species: Fixed vertical distribution ?
varfixed   = .false.
# Variable gas species: Saturation percentage value at ground ?
satval     = 0.0

## Star type
## ~~~~~~~~~
startype = 1
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# The choices are:
#
#     startype = 1 Sol        (G2V-class main sequence)
#     startype = 2 Ad Leo     (M-class, synthetic)
#       startype = 3        GJ644
#       startype = 4        HD128167
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Stellar flux at 1 AU. Examples:
# 1366.0 W m-2 Sol today
# 1024.5 W m-2 Sol today x 0.75 = weak early Sun
# 18.462 W m-2 The feeble Gl581
# 19.960 W m-2 Gl581 with e=0.38 orbital average
Fat1AU = 1024.5

## Tracer and aerosol options
## ~~~~~~~~~~~~~~~~~~~~~~~~~~
# Gravitational sedimentation of tracers (just H2O ice for now) ?
sedimentation = .false.

## Other physics options
## ~~~~~~~~~~~~~~~~~~~~~
# call turbulent vertical diffusion ?
calldifv = .true.
# call convective adjustment ?
calladj  = .true.
# call thermal conduction in the soil ?
callsoil = .true.

#########################################################################
## extra non-standard definitions for Early Mars
#########################################################################

## Tracer and aerosol options
## ~~~~~~~~~~~~~~~~~~~~~~~~~~
# Fixed aerosol distributions?
aerofixed     = .false.
# Varying H2O cloud fraction?
CLFvarying    = .false.
# H2O cloud fraction?
CLFfixval     = 1.0
# number mixing ratio of CO2 ice particles
Nmix_co2      = 100000.
# number mixing ratio of water ice particles
Nmix_h2o      = 100000.

## Water options
## ~~~~~~~~~~~~~
# Model water cycle
water         = .true.
# Model water cloud formation
watercond     = .true.
# Model water precipitation (including coagulation etc.)
waterrain     = .true.
# WATER: Precipitation threshold (simple scheme only) ?
rainthreshold = 0.0011
```

```
# Include hydrology ?
hydrology     = .true.
# H2O snow (and ice) albedo ?
albedosnow    = 0.5
# Maximum sea ice thickness ?
maxicethick   = 0.05
# Freezing point of seawater (degrees C) ?
Tsaldiff      = 0.0
# Evolve surface water sources ?
sourceevol    = .true.

## CO2 options
## ~~~~~~~~~~~
# gas is non-ideal CO2 ?
nonideal      = .false.
# call CO2 condensation ?
co2cond       = .true.
# Set initial temperature profile to 1 K above CO2 condensation everywhere?
nearco2cond   = .false.
```

- You need to compile with at least 2 tracers. If you don't have CO2 clouds, dust or other tracers, compilation is done with the command lines:

  ```
  makegcm -d 64x48x20 -t 2 -p std -b 32x36 newstart

  makegcm -d 64x48x20 -t 2 -p std -b 32x36 gcm
  ```

  Of course, you will also need an appropriate `traceur.def` file indicating you will use tracers `h2o_vap` and `h2o_ice`; if you only run with 2 tracers, then the contents of the `traceur.def` file should be:

  ```
  2
  h2o_ice
  h2o_vap
  ```

  Note that the order in which tracers are set in the `traceur.def` file is not important.

- **Run**

  Same as usual. Just make sure that your start files contains the initial states for water, with an initial state for water vapour / ice in the atmosphere and ice / liquid on the surface.

Chapter 9

1D version of the generic model

The physical part of the model can be used to run 1D radiative-convective simulations (one atmospheric column / globally averaged climate). In practice, the simulation is controlled from a main program called `rcm1d.F` which, after initialization, then calls the master subroutine of the physics `physiq.F90` described in the previous chapters.

9.1 Compilation

- For example, to compile the generic model in 1D with 25 layers, type (in compliance with the makegcm function manual described in section 6.4)

```
makegcm -d 25 -t 1 -b 32x36 -p std rcm1d
```

You can find executable **rcm1d.e** (the compiled model) in the directory from which you ran the makegcm command.

9.2 1-D runs and input files

The 1D model does not use an initial state file (the simulation must be long enough to obtain a balanced state). Thus, to generate a simulation simply type:

```
> rcm1d.e
```

The following example files are available in the `deftank` directory (copy them into your working directory first):

- **callphys.def** : controls the options in the physics, just like for the 3D GCM.
- **z2sig.def** : controls the vertical discretization (no change needed, in general), functions as with the 3D GCM.
- **traceur.def** : controls the tracer names (this file may not be present, as long as you run without tracers (option `tracer=.false.` in callphys.def)
- **run.def** : controls the 1D run parameters and initializations (this is actually file `run.def.1d` the `deftank` directory, which must be renamed `run.def` to be read by the program).

The last file is different from the 3D GCM's `run.def` input file, as it contains options specific to the 1D model, as shown in the example below:

```
#-------------------------------------------------------------------
# Run parameters for the rcm1d.e model
#-------------------------------------------------------------------
```

```
#### Time integration parameters
#
# Initial date (in martian sols ; =0 at Ls=0)
day0=0
# Initial local time (in hours, between 0 and 24)
time=0
# Number of time steps per sol
day_step=48
# Number of sols to run
ndt =400

#### Physical parameters
#
# Surface pressure (Pa)
psurf=7000.
# Gravity (ms^-2)
g=3.72
# Molar mass of atmosphere (g)
mugaz=43.49
# Specific heat capacity of atmosphere?
cpp=744.5
# latitude (in degrees)
latitude=0.0
# orbital distance at perihelion (AU)
periastr=1.558
# orbital distance at aphelion (AU)
apoastr=1.558
# obliquity (degrees)
obliquit=0.0
# Solar zenith angle (degrees)
szangle=60.0

# Albedo of bare ground
albedo=0.2
# Emissivity of bare ground
emis=1.0
# Soil thermal inertia (SI)
inertia=400
# zonal eastward component of the geostrophic wind (m/s)
u=10.
# meridional northward component of the geostrophic wind (m/s)
v=0.
# Initial CO2 ice on the surface (kg.m-2)
co2ice=0
# hybrid vertical coordinate ? (.true. for hybrid and .false. for sigma levels)
hybrid=.false.
# autocompute vertical discretisation? (useful for exoplanet runs)
autozlevs=.false.
% pressure ceiling
pceil=40.0

###### Initial atmospheric temperature profile
#
# Type of initial temperature profile
#         ichoice=1   Constant Temperature:  T=tref
#         ichoice=2   Savidjari profile (as Seiff but with dT/dz=cte)
#         ichoice=3   Lindner (polar profile)
#         ichoice=4   inversion
#         ichoice=5   Seiff  (standard profile, based on Viking entry)
#         ichoice=6   constant T  +  gaussian perturbation (levels)
#         ichoice=7   constant T  + gaussian perturbation (km)
#         ichoice=8   Read in an ascii file "profile"
ichoice=5
# Reference temperature tref (K)
tref=200
# Add a perturbation to profile if isin=1
isin=0
```

```
# peak of gaussian perturbation (for ichoice=6 or 7)
pic=26.522
# width of the gaussian perturbation (for ichoice=6 or 7)
largeur=10
# height of the gaussian perturbation (for ichoice=6 or 7)
hauteur=30.

# some definitions for the physics, in file 'callphys.def'
INCLUDEDEF=callphys.def
```

Note that, just as for the 3D GCM `run.def` file, input parameters may be given in any order, or even not given at all (in which case default values are used by the program).

9.3 Output data

During the entire 1D simulation, you can obtain output data for any variable from any physical subroutine by using subroutine `writeg1d`. This subroutine creates file `g1d.nc` that can be read by GRADS. This subroutine is typically called at the end of subroutine `physiq` .

Example of a call to subroutine `writeg1d` requesting temperature output: (`ngrid` horizontal point, `nlayer` layers, variable `pt` called "T" in K units):

```
CALL writeg1d(ngrid,nlayer,pt,'T','K')
```

Chapter 10

Zoomed simulations

The LMD GCM can use a zoom to enhance the resolution locally. In practice, one can increase the latitudinal resolution on the one hand, and the longitudinal resolution on the other hand.

10.1 To define the zoomed area

The zoom is defined in `run.def`. Here are the variables that you want to set:

- East longitude (in degrees) of zoom center `clon`
- latitude (in degrees) of zoom center `clat`
- zooming factors, along longitude `grossismx`. *Typically 1.5, 2 or even 3 (see below)*
- zooming factors, along latitude `grossismy`. *Typically 1.5, 2 or even 3 (see below)*
- `fxyhypb`: **must be set to "T" for a zoom***, whereas it must be F otherwise*
- extention in longitude of zoomed area `dzoomx`. This is the total longitudinal extension of the zoomed region (degree).
  It is recommended that `grossismx` $\times$ `dzoomx` $< 200^o$
- extention in latitude of the zoomed region `dzoomy`. This is the total latitudinal extension of the zoomed region (degree).
  It is recommended that `grossismy` $\times$ `dzoomy` $< 100^o$
- stiffness of the zoom along longitudes `taux`. 2 is for a smooth transition in longitude, more means sharper transition.
- stiffness of the zoom along latitudes `taux`. 2 is for a smooth transition in latitude, more means sharper transition.

10.2 Making a zoomed initial state

One must start from an initial state archive `start_archive.nc` obtained from a previous simulation (see section 5.10) Then compile and run `newstart.e` **using the** `run.def` **file designed for the zoom**.

After running `newstart.e`. The zoomed grid may be visualized using grads, for instance. Here is a grads script that can be used to map the grid above a topography map:

```
set mpdraw off
set grid off
sdfopen restart.nc
set gxout grid
set digsiz 0
set lon -180 180
d ps
close 1
*** replace the path to surface.nc in the following line:
sdfopen  /u/forget/WWW/datagcm/datafile/surface.nc
set lon -180 180
set gxout contour
set clab off
set cint 3
d zMOL
```

10.3 Running a zoomed simulation and stability issue

- **dynamical timestep** Because of their higher resolution, zoomed simulation requires a higher timestep. Therefore in `run.def`, the number of dynamical timestep per day `day_step` must be increased by more than `grossismx` or `grossismy` (twice that if necessary). However, you can keep the same physical timestep (48/sol) and thus increase `iphysiq` accordingly (`iphysiq = day_step/48`).
- It has been found that when zooming in longitude, on must set `ngroup=1` in `dyn3d/groupeun.F`. Otherwise the run is less stable.
- The very first initial state made with `newstart.e` can be noisy and dynamically unstable. It may be necessary to strongly increase the intensity of the dissipation and increase `day_step` in `run.def` for 1 to 3 sols, and then use less strict values.
- If the run remains very unstable and requires too much dissipation or a too small timestep, a good tip to help stabilize the model is to decrease the vertical extension of your run and the number of layer (one generally zoom to study near-surface process, so 20 to 22 layers and a vertical extension up to 60 or 80 km is usually enough).

Chapter 11

Changing the radiative transfer properties

One of the key advantages of the LMD generic model is the ability to work with arbitrary gas and aerosol mixtures in the radiative transfer. In this chapter we describe how to produce new correlated-k absorption coefficients and implement them in the GCM.

11.1 Producing the high-resolution data

We use the open-source software `kspectrum` to produce line-by-line (LBL) absorption coefficients. `Kspectrum` is freely available online at

```
http://code.google.com/p/kspectrum/
```

See its user manual for general information on installation and basic usage.

To produce LBL data on a grid of pressure and temperature suitable for the GCM, the program `make_composition.F90` is used (available in the `utilities` folder of the main GCM directory). This may be compiled with the script `compile` in the same folder. Once this has been done, the two scripts `prekspectrum` and `postkspectrum` are used to feed `kspectrum` the correct inputs and convert the LBL data to correlated-k coefficients afterward. These scripts require three environment variables to be defined: `DWORK_DIR`, `KSPEC_DIR` and `BANDS_DIR`.

In the following example, we create a database with a mixed CO_2 / H_2O atmosphere where CO_2 is the dominant gas. First, the three environment variables are set as

```
DWORK_DIR=/san/home/rdword/corrk_data/CO2_H2Ovar
KSPEC_DIR=/san/home/rdword/kspectrum/kspec_1
BANDS_DIR=32x36
```

We then create a directory that includes files `Q.dat`, `p.dat` and `T.dat` to define the number of gaseous species and pressure and temperature gridpoints. For each file the first number gives the number of points / species. See the folder `corrk_example` in `utilities` for the example we will describe here.

Typing `prekspectrum` results in the following prompt:

```
Name of atmosphere / planet:
```

The planet name is for reference only and does not affect the results. After this, the values of the temperature, pressure and variable gas (H2O) grids are displayed, and you are asked for the CO2 mixing ratio:

```
 Correlated-k temperature grid:
 1.  100.0 K
 2.  150.0 K
 3.  200.0 K
 4.  250.0 K
 5.  300.0 K
 6.  350.0 K
 7.  400.0 K

 Correlated-k pressure grid (mBar):
 1. 1 x 10 ^-3 mBar
 2. 1 x 10 ^-2 mBar
 3. 1 x 10 ^-1 mBar
 4. 1 x 10 ^ 0 mBar
 5. 1 x 10 ^ 1 mBar
 6. 1 x 10 ^ 2 mBar
 7. 1 x 10 ^ 3 mBar
 8. 1 x 10 ^ 4 mBar
 9. 1 x 10 ^ 5 mBar

 nmolec=           2
 Temperature layers:             7
 Pressure layers:                9
 Mixing ratio layers:            7
 Total:                        441

 Please enter vmr of CO2
```

We chose `1.0` as there are no other gases (the mixing ratio is automatically changed to take into account the variable gas). After `prekspectrum` exits, we can view the resulting `composition.in` file stored in the `data/` directory of `kspectrum`:

```
Atmospheric composition input data file for planet: Zarmina
Number of atmospheric levels: 441
Number of molecules:   2

      z (km)      /    P (atm)      /  T (K)    /  x[CO2]    /  x[H2O]
  0.000000000E+00  0.986923267E-06  0.100E+03  0.99999E+00  0.10000E-06
  0.000000000E+00  0.986923267E-06  0.150E+03  0.99999E+00  0.10000E-06
  0.000000000E+00  0.986923267E-06  0.200E+03  0.99999E+00  0.10000E-06
  ...
```

Typing `run_kspectrum` in the `kspectrum` directory then submits the process as a batch job. Beware: calculating LBL coefficients for multiple gases and several hundred p, T values can take several weeks at current processing speeds!

11.2 Performing the correlated-k conversion

Once the LBL data is calculated, it's time to convert it to correlated-k format. We do this using a program `generate_kmatrix.F90` which is also stored in the `utilities` folder and is called by `postkspectrum`. In addition to the data generated by `kspectrum` and the original `.dat` files, it requires definition of the spectral bands to be used in the GCM. In this example we use a folder `32x36`, containing files `narrowbands_VI.in` and `narrowbands_IR.in`. These files define the number and widths all all bands in the

visible and infrared, respectively. They can of course be modified depending on blackbody temperatures and the tradeoff required between model speed and accuracy - the examples given provide accurate results for planets around Sun-like or M-class stars with surface temperatures in the 200-350 K range. `postkspectrum` moves the LBL database to the `DWORK_DIR` directory along with the script `run_kmatrix`. When `run_kmatrix` is submitted in batch mode, it calls `generate_kmatrix.exe` automatically for both the visible and the infrared. Correlated-k conversion is much quicker than the LBL calculation - for this database on current (2011) systems it should take only a few hours.

11.3 Implementing the absorption data in the GCM

To use our new correlated-k coefficients, we symbolically link the correlated-k folder to the `datagcm` directory defined in the GCM file `phystd/datafile.h` (it's best to avoid copying the data directly due to space considerations). All that is left is to change `corrkdir` in `callphys.def` to the correct name (`CO2_H2Ovar` in this example). Provided that we compile the GCM with the correct number of bands, e.g.

```
makegcm -d 32x32x20 -t 1 -b 32x36 -p std gcm
```

it will run automatically with the new radiative transfer. The GCM checks the radiative transfer data on initialization vs. the values given in `gases.def`, to verify that thermodynamic values (e.g. μ_{gas}, c_p) match the correlated-k data in the model.

Bibliography