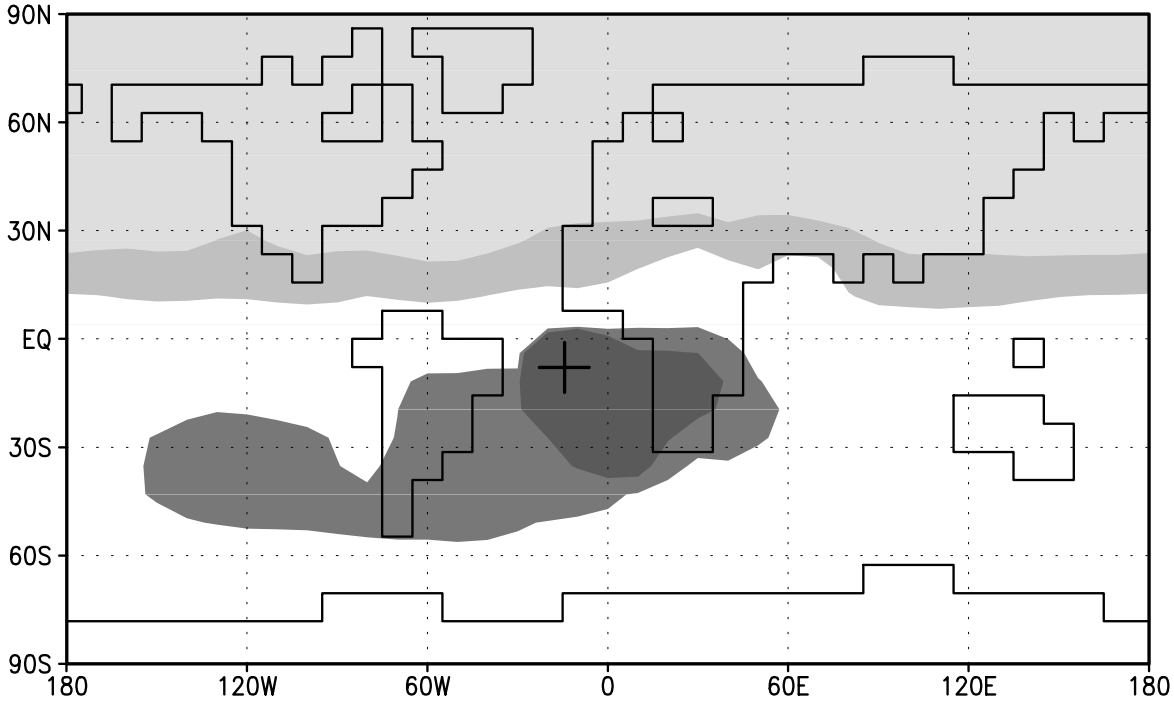
M&D Modelle & Daten

Technical Report

No. 1

Support for Climate- and Earth System Research at the Max Planck Institute for Meteorology, Hamburg

The Adjoint of TM2



by

Michael Voßbeck • Thomas Kaminski

Ralf Giering • Martin Heimann

Hamburg, Dezember 2001

Support for Climate- and Earth System Research at the Max Planck Institute for Meteorology, Hamburg

Postal:

Modelle und Daten

Max-Planck Institut für Meteorologie

Bundesstrasse 55
D-20146 Hamburg
Germany

Office:

Tel.: +49 - (0)40 411 73 - 397

Fax: +49 - (0)40 411 73 - 476

e-mails:

Office: mad_office@dkrz.de

Data Section: data@dkrz.de

Model Section: model@dkrz.de

Internet:

Home Page: http://www.mpimet.mpg.de/Depts/MaD/

Cover figure: Graphical representation of the sensitivity matrix for the mean annual CO_2 concentration at the station Ascension Island (8° S, 14° W) in the South Atlantic ocean.The gray shading in a particular model grid cell indicates the magnitude of the concentration response at the Ascension island station generated by a local annual unit source (see Kaminski et al.,1999a).

Technical Report

No. 1

Gruppe Modelle & Daten

Support for Climate- and Earth System Research at the Max Planck Institute for Meteorology, Hamburg

The Adjoint of TM2

Michael Voßbeck[1,2], Thomas Kaminski[1,3], Ralf Giering[3], and Martin Heimann[4]

[1]Max-Planck-Institut für Meteorologie
Bundesstraße 55, D-20146 Hamburg

[2]Technische Universität Berlin
Straße des 17. Juni 136, D-10623 Berlin

[3]FastOpt
Martinistraße 21, D-20251 Hamburg

[4]Max-Planck-Institut für Biogeochemie
Postfach 100164, D-07701 Jena

Diese Reihe erscheint parallel in gedruckter Form (ISSN 1619-2249)

ISSN 1619-2257

Hamburg, Dezember 2001

Contents

1 SUMMARY PAGE **2**

2 Automatic Differentiation, Adjoint, and Jacobian of TM2 **3**

2.1 TM2 3

2.2 Automatic differentiation 3

2.3 Jacobian matrix 5

3 Modelling software for adjoint (package tm2adj) **7**

3.1 Generation of the Adjoint Model 8

3.2 Running the adjoint 9

4 Modelling software for Jacobian (package tm2jac) **11**

5 Graphics software **15**

6 Acknowledgments **16**

7 Figures **18**

1 SUMMARY PAGE

This document describes the software package of the adjoint to the global three-dimensional atmospheric transport model TM2, which has been generated automatically by means of the Tangent linear and Adjoint Model Compiler (TAMC). Using the example of CO_2 as a passive tracer, the document demonstrates how the adjoint may be used to efficiently compute the transport model's Jacobian matrix, which quantifies the sensitivity of the simulated concentrations at atmospheric monitoring stations with respect to the CO_2 surface exchange fluxes. The software package includes examples of Jacobian matrices for a number of stations, several prescribed surface flux fields, and a module to simulate the concentrations at observational sites by multiplication of the Jacobian matrix with prescribed flux fields. Furthermore the package contains a graphics library based on the GRADS software system for visualisation of the Jacobian, the flux fields, the simulated concentrations, and the maps of the station locations.

2 Automatic Differentiation, Adjoint, and Jacobian of TM2

2.1 TM2

TM2 is a three-dimensional atmospheric transport model, which solves the continuity equation for an arbitrary number of atmospheric tracers on an Eulerian grid spanning the entire globe (see *Heimann* [1995]). It is driven by stored meteorological fields derived from analyses of a weather forecast model or from output of an atmospheric general circulation model. Tracer advection is calculated using the "slopes scheme" of *Russel and Lerner* [1981]. Vertical transport due to convective clouds is computed using the cloud mass flux scheme of *Tiedtke* [1989]. Turbulent vertical transport is calculated by stability dependent vertical diffusion according to the scheme by *Louis* [1979]. Numerically, in each base time step the model calculates the source and sink processes affecting each tracer, followed by the calculation of the transport processes. The spatial structure of the model is a regular latitude-longitude grid and a sigma coordinate system in the vertical. The base "coarse grid" version of the model uses a horizontal resolution of approximately 8^o latitude by 10^o longitude (the horizontal dimension of the grid is $n_g = 36 \times 24$, see Figure 1) and 9 layers in the vertical dimension.

2.2 Automatic differentiation

This section describes how a function composed of a series of elementary functions may be differentiated by use of the chain rule, which constitutes the underlying principle of the automatic differentiation. In the present context elementary functions refer to single statements of the TM2 code. For the automatic generation of the derivative computing code, it is crucial that the Jacobians of the single steps can be constructed according to simple rules. Let

$$\begin{array}{rcl} \mathcal{H} : I\!R^n & \rightarrow & I\!R^m \\ X & \mapsto & Y \end{array}$$

be a function that is composed

$$\mathcal{H} \;=\; \mathcal{H}^K \circ \ldots \circ \mathcal{H}^1 \;=:\; \bigodot_{l=1}^{K} \mathcal{H}^l \tag{1}$$

of K differentiable elementary functions:

$$\begin{array}{rcll} \mathcal{H}^l : I\!R^{n_{l-1}} & \rightarrow & I\!R^{n_l} & (l = 1, ..., K) \\ Z^{l-1} & \mapsto & Z^l \;. & \end{array}$$

Even if $\mathcal{H}$ is not given symbolically, i.e. by a formula, but instead by a numerical algorithm such as the TM2 model code, the Jacobian matrix representing the first derivative of $\mathcal{H}$

$$\frac{\partial \mathcal{H}(X)}{\partial X} := \begin{pmatrix} \frac{\partial \mathcal{H}_1(X)}{\partial X_1} & \cdots & \frac{\partial \mathcal{H}_1(X)}{\partial X_n} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{H}_m(X)}{\partial X_1} & \cdots & \frac{\partial \mathcal{H}_m(X)}{\partial X_n} \end{pmatrix}$$

can be computed using the chain rule of differentiation from the Jacobians of the elementary functions

$$\left.\frac{\partial \mathcal{H}(X)}{\partial X}\right|_{X=X_0} = \left.\frac{\partial \mathcal{H}^K}{\partial Z^{K-1}}\right|_{Z^{K-1}=Z_0^{K-1}} \cdot \ldots \cdot \left.\frac{\partial \mathcal{H}^1}{\partial Z^0}\right|_{Z^0=X_0} \quad . \tag{2}$$

Thereby

$$Z_0^l := \mathcal{H}^l \circ \ldots \circ \mathcal{H}^1(X_0) \qquad (1 \leq l \leq K)$$

denote the intermediate results, through which the derivatives of the elementary functions depend on X_0.

For evaluating the multiple matrix product in Eq. (2) there are many possibilities. Depending on the size of the elementary matrices they differ in the number of operations that have to be performed and in the size of the matrices containing the intermediate derivatives. The most obvious strategies for the computation of the matrix product are the forward and the reverse mode, where forward and reverse refer to the order of the operations in the calculation: Operating in forward mode, the product is evaluated from the right to the left, which means that the product is computed in the same order as for evaluation of $\mathcal{H}$ in Eq. (1). Alternatively, the product can be evaluated from the left to the right, which is denoted as reverse mode, because the order is opposite to the order for evaluation of $\mathcal{H}$ in Eq. (1). In this procedure the intermediate matrices after the l-th step of the composition (1) contain $\left.\frac{\partial(\mathcal{H}^l \circ \ldots \circ \mathcal{H}^1)(X)}{\partial X}\right|_{X=X_0}$ in forward mode and $\left.\frac{\partial(\mathcal{H}^K \circ \ldots \circ \mathcal{H}^{l+1})(Z^l)}{\partial Z^l}\right|_{Z^l=Z_0^l}$ in reverse mode. Thus forward and reverse refer to the directions in which the intermediate derivatives are propagated by the respective algorithm for evaluation of Eq. (2). According to Eq. (2) the forward mode step corresponding to the l-th step of the composition (1) is:

$$\left.\frac{\partial(\mathcal{H}^l \circ \ldots \circ \mathcal{H}^1)(X)}{\partial X}\right|_{X=X_0} = \left.\frac{\partial \mathcal{H}^l}{\partial Z^{l-1}}\right|_{Z^{l-1}=Z_0^{l-1}} \cdot \left.\frac{\partial(\mathcal{H}^{l-1} \circ \ldots \circ \mathcal{H}^1)(X)}{\partial X}\right|_{X=X_0} \quad . \tag{3}$$

With respect to the standard inner product the adjoint matrix of $\frac{\partial \mathcal{H}(X)}{\partial X}$ is simply the transposed matrix. Thus Eq. (2) can be written in the form

$$\left.\frac{\partial \mathcal{H}(X)}{\partial X}\right|_{X=X_0}^* = \left.\frac{\partial \mathcal{H}^1}{\partial Z^0}\right|_{Z^0=X_0}^* \cdot \ldots \cdot \left.\frac{\partial \mathcal{H}^K}{\partial Z^{K-1}}\right|_{Z^{K-1}=Z_0^{K-1}}^* \quad . \tag{4}$$

This means, the reverse mode step corresponding to the l-th step of the composition (1) is performed by multiplying the intermediate matrix $\left.\frac{\partial(\mathcal{H}^K \circ \ldots \circ \mathcal{H}^{l+1})(Z^l)}{\partial Z^l}\right|_{Z^l=Z_0^l}$ by the adjoint of $\left.\frac{\partial \mathcal{H}^l}{\partial Z^{l-1}}\right|_{Z^{l-1}=Z_0^{l-1}}$:

$$\left.\frac{\partial(\mathcal{H}^K \circ \ldots \circ \mathcal{H}^l)(Z^{l-1})}{\partial Z^{l-1}}\right|_{Z^{l-1}=Z_0^{l-1}}^* = \left.\frac{\partial \mathcal{H}^l}{\partial Z^{l-1}}\right|_{Z^{l-1}=Z_0^{l-1}}^* \cdot \left.\frac{\partial(\mathcal{H}^K \circ \ldots \circ \mathcal{H}^{l+1})(Z^l)}{\partial Z^l}\right|_{Z^l=Z_0^l}^* \quad . \tag{5}$$

Therefore the reverse mode is also called adjoint mode.

As is illustrated by Figure 2 for a scalar valued function ($m = 1$) of $n = 5$ variables, in the forward mode all matrices containing intermediate derivatives have n columns, whereas in

the reverse mode they have m rows. Therefore in forward mode the number of operations as well as the storage requirements are proportional to n, whereas in reverse mode they are proportional to m.

In general, the intermediate results Z_0^l of the preceding step are required for the evaluation of the derivatives of the elementary functions (see Eq. 2). While in the forward mode the intermediate results are required in the same order as computed, in the reverse mode they are required in reverse order. Therefore, providing of the intermediate results is more complicated in reverse mode and in general implies extra operations and/or extra storage requirements [*Giering and Kaminski*, 1998]. This has to be taken into account when comparing the efficiency of reverse and forward mode for a particular function $\mathcal{H}$ (see section 3.1).

The Tangent linear and Adjoint Model Compiler [TAMC *Giering*, 1997] is a tool that automatically generates code for the evaluation of the first derivatives of a given function $\mathcal{H}(X)$ represented as a FORTRAN 77 computer program. The TAMC is a source to source translator that accepts essentially FORTRAN 77 code for the evaluation of a function and generates code for evaluation of its Jacobian. As requested by the user, the generated code operates either in forward or reverse mode. The schemes for forward or reverse mode essentially represent implementations of the general rules (3) and (5) respectively described above. This implementation is not unique: The scheme chosen for the TAMC is based on a few principles [*Giering and Kaminski*, 1998], which essentially have been suggested by *Talagrand* [1991]. Rigorous application of these principles yields rules for differentiating the single statements a code is composed of. These simple rules can be applied automatically by source to source translators like TAMC or similar tools such as Odyssée [*Rostaing et al.*, 1993].

2.3 Jacobian matrix

Section 2.1 has introduced the transport model TM2, and section 2.2 has introduced the concept of Automatic differentiation (AD). In this section we explain how this concept is applied to derive an adjoint of TM2. Note that in using this (standard) terminology an important aspect of the concept of AD is obscured, i.e. there does not exist an adjoint to a "transport model", but merely to a function (which of course contains the model). Hence, before generating an adjoint, a function has to be defined, which includes the definitions of the input and output variables and the setup of the transport model. This is illustrated in this section. More details are to be found in *Kaminski et al.* [1999a].

The function to be differentiated maps a vector of CO_2 exchange fluxes onto a vector of simulated CO_2 concentrations. While the fluxes are defined on every TM2 surface level grid cell, the concentrations are taken at a number of particular locations in the model grid. For input and output variables a temporal resolution of one month is used. The model is driven with the meteorological analyses of a particular year (fields for the year 1987 are provided). The model is driven with the same analysis fields for 4 years, and the same flux fields are repeated year by year. In the fourth year, the simulated concentrations contain a spatially varying offset, a global trend, and a spatially varying seasonal cycle. When using the cyclostationary mean flux field over a given multi-year target period, these concentration values are to be interpreted as one representation of the

ensemble of the mean quantities over the target period. To compute the concentrations at a particular location, horizontal interpolation in the appropriate vertical level of TM2 is used.

In summary, this defines a function, the derivative of which is represented by the Jacobian matrix mapping a vector of 12 (month) $\times$ 36 $\times$ 24 (horizontal TM2 grid) flux components onto a vector of 12 (month) $\times$ `nsta` concentration components, where `nsta` denotes the number of sites.

Note that this Jacobian contains all transport information relevant to this particular model setup, i.e. the multiplication of the Jacobian with a given flux field replaces running the model with this flux field as input. Moreover, the Jacobian is a most powerful tool that allows decomposing a given feature of the simulated monthly mean concentrations at the station locations such as the annual mean, a monthly mean, or the seasonal cycle with respect to the modelled contributions by any component of the flux vector. Examples are given in *Kaminski et al.* [1996, 1999a]. Moreover, the Jacobian allows high resolution atmospheric transport inversions on the full grid of TM2 [*Kaminski et al.*, 1999b]. This approach circumvents aggregation errors, which arise in in coarse grid large region in versions [*Kaminski et al.*, 2001].

3 Modelling Software for Adjoint (package tm2adj)

For the model setup of TM2 described in section 2.3, using the concepts described in section 2.2, the adjoint has been generated automatically by means of TAMC version 4.62. To ensure an accurate interpretation by (that version of) the TAMC and to fit within the modelling framework provided with TAMC [*Giering*, 1997], the structure of the model code had to be slightly rearranged. Since TAMC is a software tool under development, both the program itself and the modelling framework keep on changing. Hence, rather than providing the model source code suitable for interpretation by version 4.62, we provide the generated derivative code and the complete framework needed to compile, link and run this code. It is contained in the software package **tm2adj**, which contains all input to compute the adjoint of TM2. The output generated by the adjoint are Jacobian matrices mapping exchange CO_2 fluxes onto station data as described in section 2.3.

tm2adj is only run on a Cray C90 supercomputer, because it uses huge datasets of preprocessed meteorological input data in Cray binary format. Hence we have arranged a single **Make.host** file. In order to port the code on a different machine this file would have to be modified as well as the binary input data would have to be regenerated.

Besides the files **Makefile** and **Make.host**, **tm2adj** contains a few sample job control scripts and a number of subdirectories. In the following we first describe the subdirectories. Then we discuss the adjoint model and its generation (see section 3.1). And finally we describe one of the job control scripts that run the adjoint (see section 3.2).

jacobian The directory contains a Jacobian matrix in **.tm** format (see Figure 10), which has been previously generated and is used to check a new installation as well as a **Makefile**.

libio Contains the Fortran source code of a number of very simple routines for reading and writing data in various formats. Not all of the routines are actually needed. Executing **make** generates a library **tm2adj/libio.a**, which is linked to the main programs in **tm2adj/prgs** that run the adjoint and postprocess generated Jacobians.

libtamc Contains the Fortran and C source code of a number of routines used by the TAMC generated source code. The library is linked to the main programs in **tm2adj/prgs** that call the adjoint.

meteo Contains two files with the preprocessed meteorological driving fields used by TM2. The files have been prepared from analyses of the ECMWF forecast model for 1987 by the preprocessing package described in *Heimann* [1995]. They are Cray binary format. The fields are available to the model every 12 hours and comprise, in total, 5 fields defined on the entire grid, i.e., $720 \times 5n_g$, with $n_g = 720 \times 36 \times 12 \times 9$ being the grid dimension.

model Contains two files of the Fortran sources of the modified forward and the adjoint codes, respectively. It is necessary to run the modified forward code, because it contains subroutine calls to arrange storing of required variables, which then are read via subroutine calls in the adjoint code (see details in section 3.1).

prgs The directory contains two programs that run the adjoint and two programs that postprocess the generated Jacobians. The program **prgmcheck.F** compares the derivative of the simulated concentration at the stations with respect to a January exchange flux in a particular grid cell in the northern hemisphere to an approximation of that derivative by finite differences. The program **prgmrev.F** runs the adjoint. The program **msplit_QSC** splits up the adjoints output file into the Jacobian matrices belonging to individual stations. The program **msplit_TD** does the same for the adjoint's setup with interannually varying exchange fluxes.

station The directory contains 6 examples of station localisation files (see section 5) of networks.

3.1 Generation of the Adjoint Model

As is obvious from Eq. (5), the intermediate results Z_0^l (required variables) have to be provided for the adjoint run. Unlike many other adjoint applications in meteorology and oceanography, in (offline) transport models many of the required variables quantify the dynamic state of the atmosphere, which is computed from the preprocessed meteorological fields. These required variables do not depend on the control variables, i.e. in our case the sources and sinks. In the terminology of adjoint code construction they are called passive variables. Hence, in principle, they could be computed and stored once and then be read during each adjoint run. Since this would require disk space of about 1.3 gigawords (GW), (on a Cray C90) it is more efficient to recompute the required values during every adjoint run. In order to reduce these storage requirements during the adjoint run it is favorable to include a so-called checkpointing scheme [*Griewank*, 1991] in the adjoint model: In a first integration of TM2 the state of the model is saved at checkpoints in intervals of 6 days on disk. During the adjoint run the checkpoints are used as starting points for recomputation and storing of all required values for the six day interval in a second file. Finally, for the adjoint computations these stored values are read. The storage requirements are reduced considerably at the cost of an additional model integration. This checkpointing scheme also is implemented automatically by the TAMC.

One level of checkpointing is applied by splitting the main time loop in an inner and an outer loop (subroutines `func` and `tracer`). In the outer loop the air mass (variable `m`) and the mixing ratio and its slopes (variables `rm, rxm, rym` and `rzm`) are stored. Furthermore, for computation of monthly mean concentrations, an accumulated mixing ratio (variable `rmacc`) and two scalar values (variables `rtloc` and `rxmod`) are stored. The variables `m, rm, rxm, rym, rzm` and `rmacc` are defined on the entire model grid, i.e. they are of dimension $36 \times 24 \times 9$ and thus per checkpoint $6 \times 36 \times 24 \times 9 + 2 = 46558$ W are to be stored.

In the inner loop some quantities of the computation of the advective (variables `am, bm, cm` and `sbm`) and convective (variable `conv`) air flux are stored. For the period in which data are available the air mass is stored in order to compute faster the mass mixing ratio from the tracer mass. Also three scalar values values (variables `ihelp, rtloc` and `rxmod`) are stored. Further, within the advection subroutine `advect` the air mass (variable `m`) is stored seven times (before any of the seven calls of the advective subroutines `dynamu, dynamv` and `dynamz`). The variables `m, am, bm, cm` and `sbm` are defined on the entire

model grid, the dimension of `conv` is that of model horizontal grid times the vertical dimension of 9. Hence for the spin up period per time step $(7+9+4)\times 36\times 24\times 9 = 155520$ W have to be stored, while for the period with data per time step $(7+9+4+1)\times 36\times 24\times 9+3 = 163299$ W have to be stored.

There is one checkpoint every 6 days. The model has 360 days per year and runs for 4 years including spin up. Hence there are 240 checkpoints and a file `adtape` of $240\times 46558 \approx 11.2$ MW (mega words) is written. The integration time step is 4 hours. Hence in 6 days there are 36 time steps. In the period with data, a file `intape` of size $36 \times 163299 \approx 5.9$ MW is written.

3.2 Running the adjoint

In earlier versions the adjoint was constructed to compute the derivative of a scalar valued function specifically quantifying the misfit of simulated concentrations to observed ones. Thereby the adjoint could be run in a variety of modes, e.g. to perform an optimisation of the input exchange fluxes or to generate "synthetic" or "pseudo" data. Since the Jacobian contains all the necessary transport information, many of these modes are not used anymore and thus are not dicussed here. The only two modes that have 'survived' are denoted by `mre` for computing the Jacobian and `mch` for computing the Jacobian and checking the derivative with respect to a control variable by comparison with a finite difference approximation.

An example of a job control script that performs such a check is depicted in Figures 3, 4, and 5. The `QSUB` commands on top interact with Cray's network queuing system (see manual pages of **qsub**). Next is a group of environment variables including the two `modes`. For testing purposes a testing version that runs for only 12 days is selected by setting the variable `test` to `on`. `year` denotes the year of the meteorological data (driving fields for 87 are provided). `mattype` describes whether to use the setup described in section 2.3 (`QSC`) or one where the fluxes can change from year to year over the (four year) simulation period. The next two variables denote the file containing the stations for which the Jacobian is to be computed and their number. Next is the path, where **tm2adj** is installed. The last two variables denote an identifier for the run and the disk where the working directory for the run is to be created which contains all intermediate files.

In the following a few of the standard TM2 variables are set (see *Heimann* [1995]). Changing those might make some adaptations in the code necessary.

Next, a number of variables determining the data side of the model are set, again, changing those might require adaptations in the code. These variables denote the number of output values per site, the number of output values over all, and the time interval within the simulation period at which these values are taken. A number of path names are set. The working directory, station definition file, the main programs and the libraries are prepared, the model is compiled and linked to the libraries and the appropriate main program. Using preprocessor directives, a number of the settings in the variables are passed to model.

Before executing the program, the Cray **assign** feature is used to arrange writing the Jacobian in **ieee** format suitable for visualisation by GrADS (**.gad**). The program first

reads a `namelist` of variables that are specific to the adjoint and then the standard TM2 input containing a list of the driving meteorological fields as well as a second `namelist`. Finally a main program is called that splits the output into Jacobians for any station.

A **.log** file of the corresponding model run is depicted in Figures 6 and 7. It shows the compiler's output followed by the variables of the additional namelist and the standard TM2 one. Then it shows the comparison of the derivative (column 5) of all 12 (monthly mean) output concentrations (at station MLO, columns 7 and 8) with respect to the input variable no. 507 (column 2) to a finite difference approximation (column 4) with stepsize 0.1 KgC/gridcell/year. The adjoint model run terminates with some statistics. The final lines show the split of the Jacobian.

4 Modelling Software for Jacobian (package tm2jac)

This section describes the software package **tm2jac**, which is designed to demonstrate the multiplication of a jacobian matrix with a flux field (see section 2.3). The package is intended to be a starting point, which the user can extend by simple modifications. The structure of the package consists of a main directory, **tm2jac**, and a number of subdirectories that contain the necessary input data, the source code and auxillary information. In addition the subdirectory **tm2jac/libgrads**, which is briefly described in section 5, contains the graphics software to visualise all quantities of interest. Since the package is intended to run on a variety of unix type computer platforms, all data are stored in an ascii format (extension **.tm**).

The package structure resembles that of the TAMC utility [TAMC *Giering*, 1997]. It is not feasible to construct such a package in an absolutely portable way for all of the existing variety of Unix platforms. To simplify portability and handling of **tm2jac**, and to ensure the transparency of all commands executed on the Unix level, we employ the Unix **make** command [see, e.g, *Oram and Talbott*, 1991]. **make** renders the package to a large extend self explaining.

The actions that **tm2jac** is designed to take are documented by the targets of the **Makefile** it contains (see Figures 8 and 9). The targets are built by the command line input

make <target>

If the target is omitted, then **make** picks the first target in the **Makefile**. To see the actions without executing them, use the option **-n**. Most of these targets are built by invoking **make** recursively in some of the subdirectories. We document this package by first a brief description of these targets and further below by a description of all subdirectories.

Make.host Tries to identify the architecture of the local machine using the script **bin/getarch**, and then copies the corresponding **Make.<architecture>** file from the directory **tm2jac/config** to **tm2jac/Make.host**. If there is no appropriate **Make.host** file for the local architecture, a default host file is copied. In case the machine or the environment have settings different from those expected by the **Make.host** file, it will have to be modified.

install Generates the library **tm2jac/libio.a**, by calling **make** in **tm2jac/libio**. Then by calling **make** in **tm2jac/mult** it compiles the main program **tm2jac/mult** and links it to **tm2jac/libio.a**. Eventually, it executes **tm2jac/mult/mult**, to multiply the Jacobian for the station MLO with the flux field **flux/apo_jgr.tm** yielding a time series of concentrations **conc/c0mod_MLO_apo_jgr.set**.

check Compares the result of the multiplication performed by the target **install**, i.e. **conc/c0mod_MLO_apo_jgr.set**, to a file of previously generated concentrations **conc/tst.set** using the Unix **diff** command.

clean Removes backups made by the editor. Invokes **make clean** recursively in all subdirectories.

scratch Reconstructs the initial status of **tm2jac** by removing all files that have been generated from the primary source and data files (such as executables or libraries). Invokes **make scratch** recursively in all subdirectories.

tar First **make** removes all files that can be reconstructed by building the target **scratch**. Then it creates a tar file named **../tm2jac.tar** that includes all remaining files.

tar.gz, tar.Z Gzips or compresses the **tarfile**.

As an example, the command line input

make check

does all the installation plus a check of the simulated concentrations.

tm2jac contains a number of subdirectories which are described in detail below (see the directory tree shown in Figure 12):

bin Contains the scripts **getarch** and **mvgrads**. **getarch** is called without any arguments and returns the architecture of the machine it is executed on. It is used during the installation of the package in order to choose the proper **Make.host** file from the directory **config**. **mvgrads** is called to rename the pairs of files required by the graphics software for each data item as mentioned in the description of **tm2jac/convert**. The arguments are the old kernel <**oldbasename**> and the new kernel <**newbasename**> of the filenames.

conc The directory contains 2 examples of concentration time series in **.set** format (a simple ASCII format, which is also used by the graphics program Xvgr [*Turner*, 1998]), a number of **.ipt** files serving as input to the graphics package, and a **Makefile**. All figures of the concentration time serieses presented in **graphics.ps** can be reconstructed by entering **make doc-bw "PIC=<number>"** in the command line. Colored versions of them are derived by entering **make doc "PIC=<number>"**. The concentration time series **check.set** contains the result of the same matrix multiplication that should be generated by the target **check** in **tm2jac/Makefile**, i.e. the product of the flux field **apo_jgr.gad** (see **fluxes**) with the Jacobian for the station MLO (see **jacobian**). The first column refers to the month and the second has the concentration. It is used to check the installation on a new machine. The example concentration time series **cobs_81-86.set** is the one used for the inversion described in *Kaminski et al.* [1999b], It consists of monthly mean values and uncertainties derived form observed CO_2 concentrations [*Globalview–CO2*, 1996]. All concentrations are in parts per million volume (ppmv). There are records for the 34 stations named in **station/staloc_obs.d** and in the same order. In each record each row contains month, concentration, and uncertainty. The targets of the **Makefile** generate and visualise **.gad** and **.ctl** files of the concentrations.

config Contains a variety of **Make.host** files, with machine specific settings. The appropriate Makefile might have to be be adapted to local particularities. The naming convention is provided in **Make.<architecture>**

convert For portability reasons, all data are provided in the ASCII formats **.tm** (described in Figure 10) and **.set** (see **conc**). The graphics software (see section 5) based on GrADS, however, uses pairs consisting of a binary direct access file (suffix **.gad**) and an ASCII description file (suffix **.ctl**). The directory contains the Fortan programs **set2grads.F**, **tmflux2grads.F**, and **tmjac2grads.F** to convert data files for observed concentrations, fluxes, and jacobians respectively to the input format required by GrADS.

doc Contains a postscript version **techrep.ps** of the present document that you are reading. Furthermore there is a second postscript document **graphics.ps**, which explains the graphics software in detail. Since this is a black and white document also, which cannot demonstrate all features of our graphics package, we put a few color pictures in this directory and refer to them in **graphics.ps**.

flux The directory contains 3 examples of flux fields in **.tm** format (see Figure 10), a number of **.ipt** files serving as input to the graphics package, and a **Makefile**. The examples are the a priori (apr_jgr.tm) and the a posteriori (apo_jgr.tm) flux fields of an inversion by *Kaminski et al.* [1999b] and a flux field computed by the terrestrial biosphere model SDBM of *Knorr and Heimann* [1995]. All seven GrADS created figures corresponding to these examples, which are documented in **graphics.ps**, can be reconstructed by entering **make doc-bw "PIC=<number>"** in the command line. Colored versions can be created by entering **make doc "PIC=<number>"**. The flux fields consist of 36 × 24 monthly values on the TM2 grid (see Figure 1 for order of the grid), the order of the dimensions is defined in Figure 10. The unit of the monthly fluxfields is **kg Carbon/gridcell/year**. The model uses the calendar length of the months, i.e. Although the model internally uses 31,28,31,30,31,30,31,31,30,31,30,31 days for January to December, this is internally adjusted, such that a constant flux throughout the year is represented with the same numerical value for all months. Thus, in order to compute the monthly flux total of a particular month, the numbers in the flux fields have to be scaled by $n_{daymon}/365$, where n_{daymon} is the number of days in the particular month. The target of the **Makefile** generate **.gad** and **.ctl** files of the fluxes and then visualise these.

jacobian The directory contains 10 examples of Jacobian matrices in **.tm** format (see Figure 10), a number of **.ipt** files serving as input to the graphics package, and a **Makefile**. The Jacobians quantify the derivative of a simulated concentration with respect to a flux, for which we use our concentration and flux unit, i.e. *1 ppmv/(kg Carbon/gridcell/year)*. The target of the **Makefile** generate **.gad** and **.ctl** files of the Jacobians and then visualise these. Some examples are shown in **graphics.ps** and can be recreated by entering **make doc-bw "PIC=<number>"** in the command line (colored pictures can be achieved by entering **make doc "PIC=<number>"**).

libgrads Contains the graphics library based on GrADS [*Doty*, 1995]. It is briefly described in section 5, a detailed description can be found in the postscript document **graphics.ps**.

libio Contains the Fortran source code of a number of very simple routines for reading and writing data in various formats. Not all of the routines are actually needed.

Executing **make** generates a library **tm2jac/libio.a**, which is linked to the main programs in **tm2jac/mult** and **tm2jac/convert**, when executing **make** in their respective directories.

mult Contains the Fortran program **mult.F**, which carries out the matrix multiplication and a **Makefile** with targets:

matmult First **make** compiles **mult.F**, links the file **tm2jac/flux/apo_jgr.tm** to the input file **fluxes.tm** (see Figure 10), the station localisation file **tm2jac/station/staloc_MLO.d** to file **staloc.d** and the directory **tm2jac/jacobian** to the output directory **jacobian**. Then it runs the executable **mult**, runs **tm2jac/mvgrads** to incorporate the identifier **_MLO_apo_jgr** in the names of the generated **.gad** and **.ctl** files, moves all the output to **tm2jac/conc**, and removes the links from the beginning.

mmult Demonstrates the same type of matrix multiplication for different Jacobians and different flux fields by multiple builds of the target **matmult** by invoking **make** with different settings of the macros of the **Makefile**: `STA` for the station and `FLUX` for the flux field.

station The directory contains 7 examples of station location files of networks (for more information about this topic see **graphics.ps**), a number of **.ipt** files serving as input to the graphics package (see section 5), and a **Makefile**. The target of the **Makefile** visualise the respective networks. In **graphics.ps** you find 2 examples of network maps, that can be regenerated by entering **make doc-bw "PIC=<number>"** in the command line.

5 Graphics Software

The graphics software for the TM2 adjoint modelling system was designed as a compromise between two partly contradictory requirements: (1) to quickly obtain an impression of the orders of magnitudes and general structure of the computed quantities and (2) to generate high quality pictures, which should be both, reproducible and easily adaptable to slight changes in either the displayed data set or the desired graph layout.

Our graphics software is based on GrADS [*Doty*, 1995], which is available free of charge via the world wide web. To meet our requirements, we extended GrADS by **libgrads**, a library of GrADS scripts. Some of these, which we call "top level scripts", may be invoked directly by the user. With these scripts, surface flux fields, differences (sums, quotients) of surface flux fields, Jacobians, concentration time serieses, and global maps indicating the locations of observational sites, may be displayed. The requested information about the data and the layout may be entered at the beginning in the GrADS command window, or transferred to the scripts by special input files. The top level scripts also allow a simple menu control for browsing through various aspects of the data or for printing.

Further features are:

automatic level setting The contouring levels for flux fields and Jacobian matrices are set automatically. The user has the choice of linear or logarithmic scaling. This feature is extremely convenient whenever a large number of plots are to be generated. Alternatively, for high quality plots, the user may customize the levels interactively in the GrADS command window. Automatic level setting also applies to the axes of all line graphs.

center For `grfill` (color filled grid boxes) and `shaded` plots (color filled isolines), it is possible to emphasize a a single (neutral) value within the range of the quantity to be displayed, which we name `center`. A small band around `center` will appear white on the plot, if center actually lies between the minimum and maximum of the quantity to be displayed.

TM2 world map We use a special world map, on which coast lines are defined by the land-sea-mask of the TM2 grid (see section 2.1), when displaying station maps, flux fields, or Jacobian matrices.

colors As an extension to the GrADS built in rainbow color palette, additional color palettes including one greyscale palette are available for `shaded` and `grfill` pictures.

In addition, there are exist additonal scripts for the the setup of the data (**.gad**) and descriptor (**.ctl**) files. For displaying network maps of observational sites, special files detailing the station locations are needed. A detailed description of the graphics package, including instructions for usage and various examples, can be found in **graphics.ps** in the subdirectory **doc**.

6 Acknowledgments

The authors thank Peter Rayner for valuable comments on the manuscript.

This work was supported by the Bundesministerium für Bildung und Forschung (BMBF) under contract number 01LA9898/9.

References

Doty, B., *Grid Analysis and Display System, V1.5.1.12*, 1995, unpublished, available from `http://grads.iges.org/grads`.

Giering, R., *Tangent linear and Adjoint Model Compiler , Users manual*, 1997, unpublished, available from `http://puddle.mit.edu/~ralf/tamc`.

Giering, R., and T. Kaminski, Recipes for Adjoint Code Construction, *ACM Trans. Math. Software*, *24*, 437–474, 1998.

Globalview–CO2, *Cooperative Atmospheric Data Integration Project - Carbon Dioxide* [CD-ROM], NOAA/CMDL, Boulder, Colo., 1996.

Griewank, A., Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation, *Preprint MCS–P228–0491*, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439–4801, 1991.

Heimann, M., The global atmospheric tracer model TM2, *Tech. Rep. 10*, Max-Planck-Institut für Meteorologie, Hamburg, Germany, 1995.

Kaminski, T., R. Giering, and M. Heimann, Sensitivity of the seasonal cycle of CO_2 at remote monitoring stations with respect to seasonal surface exchange fluxes determined with the adjoint of an atmospheric transport model, *Phys. Chem. Earth*, *21*, 457–462, 1996.

Kaminski, T., M. Heimann, and R. Giering, A coarse grid three-dimensional global inverse model of the atmospheric transport, 1, Adjoint model and Jacobian matrix, *J. Geophys. Res.*, *104*, 18,535–18,553, 1999a.

Kaminski, T., M. Heimann, and R. Giering, A coarse grid three-dimensional global inverse model of the atmospheric transport, 2, Inversion of the transport of CO_2 in the 1980s, *J. Geophys. Res.*, *104*, 18,555–18,581, 1999b.

Kaminski, T., P. Rayner, M. Heimann, and I. Enting, On aggregation errors in atmospheric transport inversions, *J. Geophys. Res.*, *106*, 4703–4715, 2001.

Knorr, W., and M. Heimann, Impact of drought stress and other factors on seasonal land biosphere CO_2 exchange studied through an atmospheric tracer transport model, *Tellus, Ser. B*, *47*, 471–489, 1995.

Louis, J. F., A parametric model of vertical eddy fluxes in the atmosphere, *Boundary Layer Meteorol.*, *17*, 187–202, 1979.

Oram, A., and S. Talbott, *Managing Projects with make*, second ed., O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1991, a unique text on using `make` for software development.

Rostaing, N., S. Dalmas, and A. Galligo, Automatic differentiation in Odyssée, *Tellus, Ser. A*, *45*, 558–568, 1993.

Russel, G. L., and J. A. Lerner, A new finite-differencing scheme for the tracer transport equation, *J. Appl. Meteorol.*, *20*, 1483–1498, 1981.

Talagrand, O., The use of adjoint equations in numerical modelling of the atmospheric circulation, in *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, edited by A. Griewank and G. F. Corliss, pp. 169–180, SIAM, Philadelphia, Pa., 1991.

Tiedtke, M., A comprehensive mass flux scheme for cumulus parameterization in large-scale models, *Mon. Weather Rev.*, *117*, 1779–1800, 1989.

Turner, P., *xmgr*, 1998, unpublished, available from `http://plasma-gate.weizmann.ac.il/Xmgr`.

7 Figures

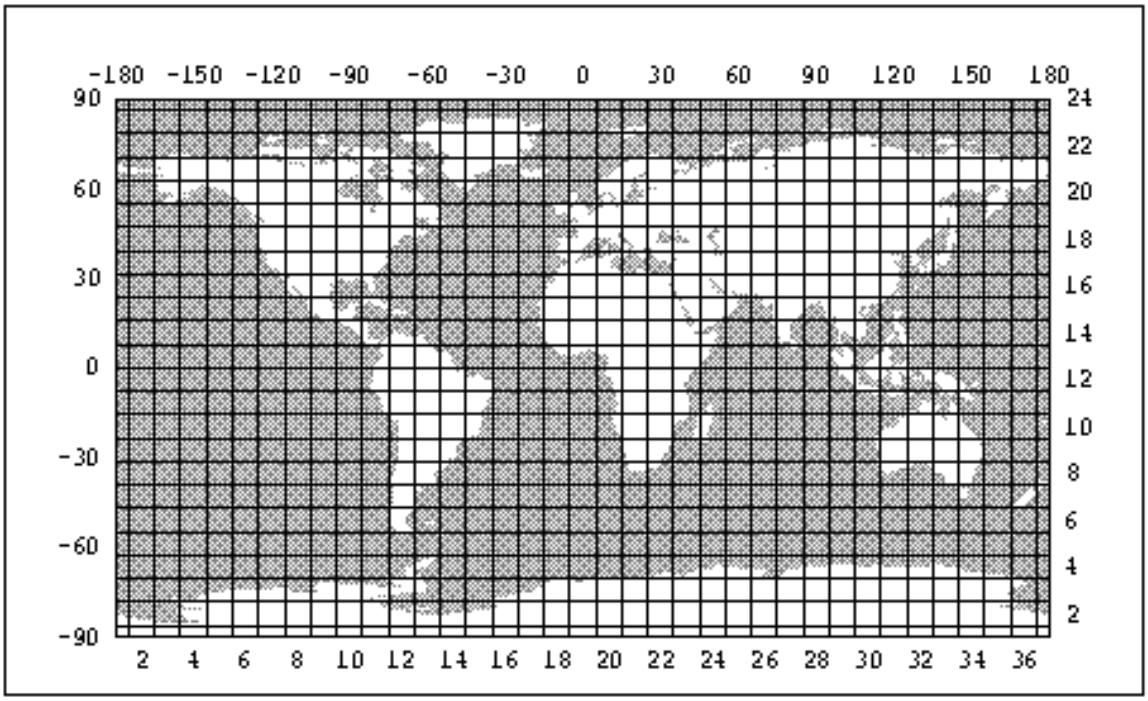


Figure 1: The definition of the TM2 "coarse grid", taken from *Heimann* [1995]. The numbers on the top and left border denote longitude and latitude in degrees, respectively. The numbers on the bottom and right border denote the longitudinal (i) and latidudinal (j) indices of the gridboxes. In the zonal direction the longitudes (in degree) ϕ_i of the centers of the gridboxes are located at

$$\phi_i = -180 + (i-1)dx, \quad i = 1, \ldots, im$$

with a grid spacing of $dx = \frac{360}{im}$ and where west longitudes are designated to be negative. In the meridional direction the jm grid boxes have an extent (in degrees) of $dy = \frac{180}{jm-1}$. They are spaced such that the latitudes (in degrees) θ_j of the grid box centers are located at

$$\theta_j = -90 + (j-1)dy, \quad j = 1, \ldots, jm$$

(Southern latitudes are designated to be negative). The first and last grid box (indices $j = 1$ and $j = jm$) are centered on the poles and have a meridional extent of $\frac{dy}{2}$. These polar boxes are not divided in the zonal direction and are referenced with zonal index $i = 1$ (for indices $j = 1$ and $j = jm$ the elements with zonal indices $i > 1$ are undefined.).

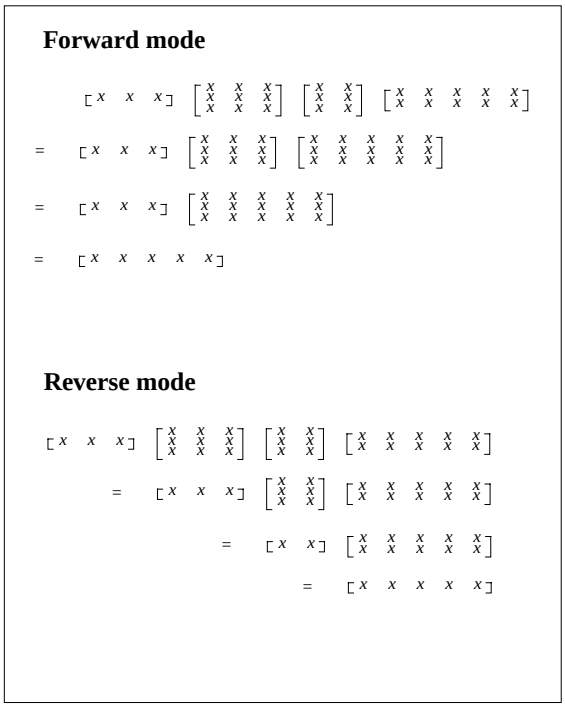


Figure 2: Example of forward and reverse modes illustrating the differences in the storage requirements and in the number of operations: The same matrix product, whose result has one row and five columns, is evaluated in forward mode, i.e., from right to left (top), and in reverse mode, i.e., from left to right (bottom). In forward mode the matrices holding the intermediate results have five columns, while in reverse mode they have one row.

```
#=======================================================================
# Job to run tm2's adjoint on cray,
# in order to compute Jacobian matrix
#=======================================================================
# QSUB -eo
# QSUB -r mcheck.job
# QSUB -o mcheck_qsc_tst_mlo_87.log
# QSUB -lT 100
# QSUB -lM 4MW
#================================================
# set variables
#================================================
#============================
# variables that are likely to be changed from run to run
#============================
#                                     Define mode of run
#                                      mrev   : comp. Jacobian
#                                      mcheck : check derivatives
# and comp. Jacobian
setenv mode "mcheck"
# choose for a short test run
#  by setting test "on"
setenv test "on"
# set year of meteo data
setenv year  87
# set type of Jacobian
# TD: time dependent
# QSC: quasi stat.
setenv mattype "QSC"
# set file with station locations
setenv stafile "staloc_MLO.d"
# set number of stations
setenv numsta "1"
# set path for main directory
set adjhome="/pf/m/m211089/pfregen/tm2adj/tm2adj"
#  Define run-code
setenv run "${mode}_qsc_tst_mlo_${year}"
# set working disc
setenv tmp "$MFHOME"
#============================
# set some control parameters for TM2
#============================
# for test perform short run
if ( $test == "on" ) then
setenv nyear 0
setenv endday 13
setenv stayear $year
setenv new newday
else
setenv nyear 4
setenv endday 1
@ stayear = $year + 1 - $nyear
setenv new newmonth
endif
# End of run
@ endyear = $stayear + $nyear
```

Figure 3: First part of the job control script **mcheck_qsc_tst_mlo_87.job**, which checks the Jacobian computed by a 12 day test run.

```
#============================
# set variables for the adjoint
#============================
# set no of time points p. station
setenv numtloc "12"
# compute no of output variables
@ ncf = $numtloc * $numsta
#
# Start and end of time window
# for assimilation
setenv asstayear $year
@ asendyear = $asstayear + 1
#============================
# set pathes
#============================
#  save directory for Jacobian
setenv matdir "$adjhome/jacobian"
#                                       Working directory
setenv workdir "$tmp/run_$run"
# Directory for Station Locations
setenv stadir "$adjhome/station"
# meteorological driving data
setenv datadir "$adjhome/meteo"
# main programs from tamc utility
setenv prgdir "$adjhome/prgs"
# source code of model
setenv moddir "$adjhome/model"
# tamc library
setenv libtamcdir "$adjhome/libtamc"
# i/o modules library
setenv libiodir "$adjhome/libio"
#================================================
# prepare run
#================================================
# create workdir
if ( ! -d $workdir ) then
mkdir $workdir;
else
cd $workdir; rm -f *
endif
# get station locations
cd $workdir ; ln -s $stadir/$stafile station.d
# compile main programs
cd $prgdir; make all
# make libraries
cd $libtamcdir; make all
cd $libiodir; make all
#================================================
# make, compile, and link model
#================================================

cd ${moddir}; make all "MODPPFLAGS = -Wp"-F" -DNUMSTA=$numsta -DNCF=$ncf -D$mattype -DNEW=$new -DSTAYEAR=19$stayear" "MODEL = $workdir/mat_chk" \
  "OPTI_FLAGS = ${prgdir}/prg${mode}.o" ; cd $workdir
```

Figure 4: Second part of the job control script **mcheck_qsc_tst_mlo_87.job**, which checks the Jacobian computed by a 12 day test run.

```
#=================================================
# run model
#=================================================
echo "TM-Run:  $run    `date`"
# Jacobian is written for grads
# in binary format of sun
assign -R
assign -N ieee -F null u:55
mat_chk <<endinput
 &cfin
 cfscal=1.
 ncfdum=${ncf}
 istafirst=1
 &end
 &adin
 exp='${mode}'
 ntauinn=518400
 idateicost=19${asstayear},1,1,0,0,0
 idateecost=19${asendyear},1,1,0,0,0
 &end

TM2 run ${run}

$datadir/wind_${year}.b
$datadir/sub_${year}.b
$moddir/plandf.d
 &inputz
 icalendo=3
 istart=1
 idatei=19${stayear},1,1,0,0,0
 idatee=19${endyear},1,${endday},0,0,0
 noindc=2
 indc(1,1)=1,indc(2,1)=1,indc(3,1)=1
 indc(1,2)=1,indc(2,2)=48,indc(3,2)=1
 ncheck=0
 fscale(1)=2.4167e6
 ndiagp1=0
 ndiagp2=0
 nwrite=0
 ndyn=14400
 nconv=14400
 nchem=14400
 nsrce=14400
 czeta=1.
 czetak=1.
 cdebug=.false.
 &end
 &gsource
 totsrc=1.
 &end
endinput
echo "TM model run completed --- `date`"
#=================================================
#  save output
#=================================================
ln -s ${matdir} matdir
ln -s mat.ext matrix.ext

cd ${prgdir}; make "SPLIT = ${workdir}/msplit_${mattype}" "SPLITOBJ = msplit_${mattype}.o" ${workdir}/msplit_${mattype}; cd $workdir

# program uses previous assign
#   to write in .gad format!
echo $year | msplit_${mattype}
echo "End of Job --- `date`"
exit
```

Figure 5: Third part of the job control script **mcheck_qsc_tst_mlo_87.job**, which checks the Jacobian computed by a 12 day test run.

```
all: No action was taken.

f90 -ltamc -lio -L. -L.. /pf/m/m211089/pfregen/tm2adj/tm2adj_1.6a/prgs/prgmcheck.o -o /mf/m/m211089/run_mcheck_qsc_tst_mlo_87_a/mat_chk -Wp-F \
 -DNUMSTA=1 -DNCF=12 -DQSC -DNEW=newday -DSTAYEAR=1987 mat_chk_ad.F mat_chk_cf.F


mat_chk_ad.F:
mat_chk_cf.F:
rm -f mat_chk_ad.o mat_chk_cf.o
TM-Run:  mcheck_qsc_tst_mlo_87_a    Sat Jul  8 21:48:34 MDT 2000
 &CFIN  CFSCAL = 1., NCFDUM = 12, ISTAFIRST = 1 /
 &ADIN  EXP = mch, NTAUINN = 518400, IDATEICOST = 1987,  2*1,  3*0, IDATEECOST = 1988,  2*1,  3*0 /
1
     Global Atmospheric Tracer Model TM2, Version  8.5
 Max-Planck-Institut fuer Meteorologie, D-2000     Hamburg 13

 TM2 run mcheck_qsc_tst_mlo_87_a
                                                                              8.5
 ------------------------------------------------------------------------------
 reading massflux fields from file :
 /pf/m/m211089/pfregen/tm2adj/tm2adj_1.6a/meteo/wind_87.b
 reading subscal info fields from file :
 /pf/m/m211089/pfregen/tm2adj/tm2adj_1.6a/meteo/sub_87.b
 reading landfraction info field from file :
 /pf/m/m211089/pfregen/tm2adj/tm2adj_1.6a/model/plandf.d
  &inputz
  icalendo=3
  istart=1
  idatei=1987,1,1,0,0,0
  idatee=1987,1,13,0,0,0
  noindc=2
  indc(1,1)=1,indc(2,1)=1,indc(3,1)=1
  indc(1,2)=1,indc(2,2)=48,indc(3,2)=1
  ncheck=0
  fscale(1)=2.4167e6
  ndiagp1=0
  ndiagp2=0
  nwrite=0
  ndyn=14400
  nconv=14400
  nchem=14400
  nsrce=14400
  czeta=1.
  czetak=1.
  cdebug=.false.
  &end
 1987-jan- 1   0: 0: 0    read massfluxes
 &GSOURCE  TOTSRC = 1. /
 control parameters:

 &INPUTZ  ISTART = 1, NDYN = 14400, NCONV = 14400, NDIAG = 43200, NCHEM = 14400, NSRCE = 14400, NREAD = 43200, NWRITE = 0, NINST = 0
  NCHECK = 0, NDIFF = 0, ICALENDO = 3, IYEARO = 1980, IDATEI = 1987,  2*1,  3*0, IDATEE = 1987,  1,  13,  3*0, IDATET = 1980,  2*1,
 3*0, NDIAGP1 = 0, NDIAGP2 = 0, CZETA = 1., CZETAK = 1., LIMITS = F, DSCALE = 250000., FSCALE = 2416700., NOINDC = 2, INDC = 4*1,  48
 1,  24*0, CDEBUG = F /
 1987-jan- 1   0: 0: 0    model start up complete
 station data of co2


  obsinit : no header found on datafile  obsdata
 Read  1  station locations
 from station definition file station.d
 1987-jan- 1   0: 0: 0    read massfluxes
 1987-jan- 1   0: 0: 0    read convinfo
 1987-jan-13   0: 0: 0    model status saved on unit   3
```

Figure 6: First part of logfile **mcheck.log** created by the job control script **mcheck_qsc_tst_mlo_87.job** (see Figures 3, 4, and 5).

```
==========================================
 CHECK OF GRADIENTS
==========================================

 COMP  CVAR     DELTA X    FC1-FC2/EPS     GRAD(FC)   REL DIFF             y2          y1          y2-y1
 1987-jan-13   0: 0: 0    model status saved on unit   3
     1   507 0.100000E+00 0.115202E-12 0.115202E-12 0.100000E+01 0.115202E-13 0.000000E+00 0.115202E-13
     2   507 0.100000E+00 0.224073E-12 0.224073E-12 0.100000E+01 0.224073E-13 0.000000E+00 0.224073E-13
     3   507 0.100000E+00 0.325852E-12 0.325852E-12 0.100000E+01 0.325852E-13 0.000000E+00 0.325852E-13
     4   507 0.100000E+00 0.418258E-12 0.418258E-12 0.100000E+01 0.418258E-13 0.000000E+00 0.418258E-13
     5   507 0.100000E+00 0.398213E-12 0.398213E-12 0.100000E+01 0.398213E-13 0.000000E+00 0.398213E-13
     6   507 0.100000E+00 0.524355E-12 0.524355E-12 0.100000E+01 0.524355E-13 0.000000E+00 0.524355E-13
     7   507 0.100000E+00 0.816740E-12 0.816740E-12 0.100000E+01 0.816740E-13 0.000000E+00 0.816740E-13
     8   507 0.100000E+00 0.911450E-12 0.911450E-12 0.100000E+01 0.911450E-13 0.000000E+00 0.911450E-13
     9   507 0.100000E+00 0.882997E-12 0.882997E-12 0.100000E+01 0.882997E-13 0.000000E+00 0.882997E-13
    10   507 0.100000E+00 0.780736E-12 0.780736E-12 0.100000E+01 0.780736E-13 0.000000E+00 0.780736E-13
    11   507 0.100000E+00 0.242419E-12 0.242419E-12 0.100000E+01 0.242419E-13 0.000000E+00 0.242419E-13
    12   507 0.100000E+00 0.792077E-13 0.792077E-13 0.100000E+01 0.792077E-14 0.000000E+00 0.792077E-14

 ------------------------------------------------------------------------------

 program has terminated           normally.
 1987-jan-13   0: 0: 0    final time
 no of timesteps:      216 cpu (s):           59.66  cpu/step:       0.27618349

 ------------------------------------------------------------------------------

 STOP   executed at line 3751 in Fortran routine 'EXITUS'
 CPU: 59.693s,  Wallclock: 125.618s,  3.0% of 16-CPU Machine
 Memory HWM: 2561386, Stack HWM: 802810, Stack segment expansions: 288
TM model run completed --- Sat Jul  8 21:50:41 MDT 2000
f90 -ltamc -lio -L. -L..  -o /mf/m/m211089/run_mcheck_qsc_tst_mlo_87_a/msplit_QSC msplit_QSC.o
 Enter year of met. data
 Year:  87
 On station definition file station.d are less than 18 station locations
 Skipped the first  0  !
 Read  1  station locations
 from station definition file station.d
End of Job --- Sat Jul  8 21:50:55 MDT 2000
logout
```

Figure 7: Second part of logfile **mcheck.log** created by the job control script **mcheck_qsc_tst_mlo_87.job** (see Figures 3, 4, and 5).

```
#***********************************************************************
#
#               Makefile for TM2 Jacobian package
#                               by T. Kaminski,
#                               most pieces are copied from
#                               TAMC utility by R. Giering
#***********************************************************************

RM              = rm -f

#=======================================================
DIR             = tm2jac_1.6
DOC             = doc
MULT            = mult
CONC            = conc
FLUX            = flux
JACOBIAN        = jacobian
STATION         = station
LIBIO           = libio
CONVERT         = convert
GRADS           = libgrads
BIN             = bin
CONFIG          = config

#=======================================================
all : install

install: Make.host
        ( cd $(LIBIO); $(MAKE) )
        ( cd $(MULT); $(MAKE) )

check: install
        ( cd $(CONC); $(MAKE) check)

tar ../$(DIR).tar : Make.host scratch
        $(RM) Make.host
        cd .. ; tar -cf $(DIR).tar $(DIR)/Makefile $(DIR)/$(FLUX)/* \
                $(DIR)/$(CONFIG)/* $(DIR)/$(JACOBIAN)/* \
                $(DIR)/$(STATION)/* \
                $(DIR)/$(MULT)/* $(DIR)/$(LIBIO)/* $(DIR)/$(CONC)/* \
                $(DIR)/$(GRADS)/* $(DIR)/$(BIN)/* $(DIR)/$(DOC)/* \
                $(DIR)/$(CONVERT)/* $(DIR)/CHANGES $(DIR)/README

tar.gz ../$(DIR).tar.gz : ../$(DIR).tar
        ../$(DIR).tar

tar.Z ../$(DIR).tar.Z : ../$(DIR).tar
        ../$(DIR).tar
```

Figure 8: First part of the **Makefile** for the main directory **tm2jac**.

```
#--------------------------------------
# architecture depending settings
#--------------------------------------
Make.host:
        @( ARCH=`$(BIN)/getarch`;                                \
           if test -f config/Make.$${ARCH};                      \
           then                                                  \
             echo "====================================";       \
             echo "  architecture = $${ARCH}";                   \
             echo "====================================";       \
           else                                                  \
             echo "====================================";       \
             echo "  architecture $${ARCH} not known";           \
             echo "  trying default";                            \
             echo "====================================";       \
             cp config/Make.UNKNOWN config/Make.$${ARCH};        \
           fi;                                                   \
           cp config/Make.$${ARCH} Make.host;                    \
        )

#--------------------------------------
# clean up
#--------------------------------------
clean   : Make.host
        ( cd $(MULT); $(MAKE) clean )
        ( cd $(BIN); $(MAKE) clean )
        ( cd $(DOC); $(MAKE) clean )
        ( cd $(CONFIG); $(MAKE) clean )
        ( cd $(LIBIO); $(MAKE) clean )
        ( cd $(CONVERT); $(MAKE) clean )
        ( cd $(CONC); $(MAKE) clean )
        ( cd $(FLUX); $(MAKE) clean )
        ( cd $(JACOBIAN); $(MAKE) clean )
        ( cd $(STATION); $(MAKE) clean )
        ( cd $(GRADS) ; $(MAKE) clean )
        $(RM) *~ \#*\#

scratch : clean
        ( cd $(MULT); $(MAKE) scratch )
        ( cd $(BIN); $(MAKE) scratch )
        ( cd $(DOC); $(MAKE) scratch )
        ( cd $(CONFIG); $(MAKE) scratch )
        ( cd $(LIBIO); $(MAKE) scratch )
        ( cd $(CONVERT); $(MAKE) scratch )
        ( cd $(CONC); $(MAKE) scratch )
        ( cd $(FLUX); $(MAKE) scratch )
        ( cd $(JACOBIAN); $(MAKE) scratch )
        ( cd $(STATION); $(MAKE) scratch )
        ( cd $(GRADS) ; $(MAKE) scratch )
        $(RM) Make.host
```

Figure 9: Second part of the **Makefile** for the main directory **tm2jac**.

```
...
                                ! dimensions
      integer nmonf,im,jm,lm,ng,nf
      parameter (nmonf=12,im=36,jm=24,nf=im*jm*nmonf)
      integer nmonc
      parameter (nmonc=12)
                                ! seasonal fluxes
      real f(nf)
                                ! transport matrix
      real amatin(nf,nmonc)
...
                                ! Calling sequence for Jacobian
      call rtm(3,'jacobian.tm',matin,im,jm,nmonc*nmonf)
                                ! Calling sequence for flux field
      call rtm(3,'fluxes.tm',f,im,jm,nmonf)
...

C=======================================================================
C     T.K. 25.8.95
CCCADJ SUBROUTINE RTM = CONST
      SUBROUTINE RTM(IUNIT,FILENAME,ARRAY,IM,JM,NMONTH)
C     read file containing the array flux consisting of  least
C     nmonth fields of dimension im x jm
C=======================================================================

C=================================================
c     declaration part
c=================================================

      implicit none

      integer im,jm,nmonth,iunit
      real array(im,jm,*)
      character*(*) filename

      integer idat,i,j,imonth

c=================================================
c     open source file
c=================================================

      open(unit=iunit,file=filename,status='old')

c=================================================
c     read  arrays
c=================================================
      do imonth=1,nmonth
         read(iunit,*) idat
         read(iunit,*) ((array(i,j,imonth),i=1,im),j=1,jm)
         if(idat.eq.imonth) then
c            write(*,*) 'rtm: ',idat,'th month read'
         else
            write(*,*)'error in rtm: reading ',filename,' idat = ',idat,
     .           ' is not equal to imonth= ',imonth
            stop
         endif
      enddo

      close(iunit)
      end
```

Figure 10: Subroutine **rtm.F** that reads the **.tm** files of Jacobians and flux fields together with an example of the calling sequence.

```
...
                              !iunitg: channel connected to gadfile
      integer iunitg
                              !im,jm: number of lons and lats
                              !lm: number of vertical levels
                              !nt: number of time steps
                              !nv: number of gridvariables
      integer im,jm,lm,nt,nv
      real field(im,jm,lm,nv,nt)
                              ! gadfile is "fname.gad"
      character*50 gadfile
...
      call wgad(iunitg,gadfile,field,im*jm*lm*nv*nt)
...
C=======================================================================
C     T.K. 31.7.95
CCCADJ SUBROUTINE WGAD = CONST
      SUBROUTINE WGAD(IUNIT,FILENAME,FIELD,NDIMTOT)
C=======================================================================

C=================================================
c     declaration part
c=================================================
      implicit none

      integer ndimtot,iunit
      real field(ndimtot)
      character*(*) filename

c=================================================
c     open files
c=================================================
      open(iunit,file=filename,form='unformatted',
     &      access='direct',recl=4*ndimtot)

C=================================================
c     writing
c=================================================
      write(iunit,rec=1) field

      close(iunit)
      end
```

Figure 11: Subroutine **wgad.F** that writes the **.gad** files of Jacobians, concentrations and flux fields together with an example of the calling sequence. Note the order of dimensions: longitudes, latitudes, levels, variables, time. The range within the particular dimensions is defined in the corresponding **.ctl** file. Look in **graphics.ps** for a descriptor file of a single flux field in the surface layer of the TM2 grid (see Figure 1), of which the dimension in the zonal direction is 36, in the meridional one 24, and there are 12 months.

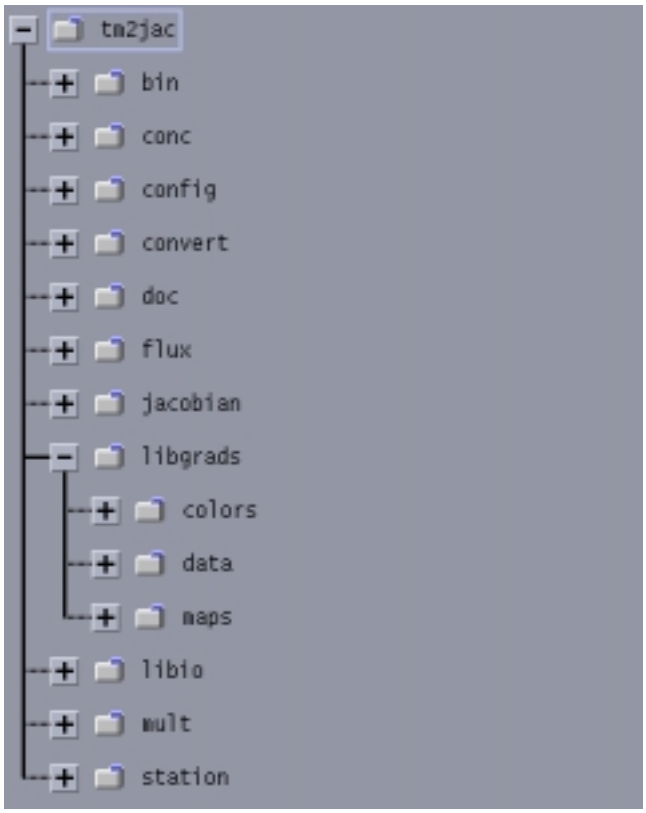


Figure 12: Subdirectory tree of the directory **tm2jac** (see section 4).

ISSN 1619-2257