Using TAMC to compute sensitivities of tracer concentrations simulated with a simple box model

Thomas Kaminski
Max-Planck-Institut für Meteorologie

Ralf Giering
Department of Earth, Atmospheric and Planetary Sciences
Massachusetts Institute of Technology

December 17, 1998

Using a simple two box model of the atmospheric transport as an example, an introduction to efficient computation of sensitivities is given. These sensitivities are computed by FORTRAN subroutines applying the forward and reverse modes of automatic differentiation, i.e. adjoint and tangent linear models are used. These models are generated automatically from the FORTRAN code of the transport model by the Tangent linear and Adjoint Model Compiler (TAMC). Automatic differentiation and TAMC are briefly introduced. All necessary preparations of the transport model as well as the usage of the adjoint and tangent linear models are described in detail. A set of exercises is provided, some of them need to be run on a computer. The main text, however, is self contained and can be read without doing the exercises.

1 Introduction

Since a few decades, global station networks are being established to monitor the concentrations of various atmospheric constituents. The spatio-temporal variations in observed concentrations of one or more constituents is interpreted in terms of these constituent's sources (and sinks). Thereby the relation between sources and concentrations is simulated by models of the atmospheric transport. More specifically, for this type of studies, the simulated sensitivity of the concentration at observational sites and times with respect to the individual components of the source fields is used to constrain the sources. In case a constituent has well known sources, these can be combined to atmospheric observations either to validate the transport model or to constrain model parameters that are employed in the formulation of physical processes. For the latter purpose, the sensitivity of the simulated concentration with respect to these model parameters is of interest.

Apart from transport modeling, computation of sensitivities is useful to solve other types of problems in earth sciences (see the contributed papers in this volume), but also in different scientific disciplines. Using a simple transport model as an example, this document aims at introducing beginners to a technique for efficient computation of sensitivities (efficiency is an important feature, because many current studies, due to the complexity of the models they use, are limited by computational resources). The basic concept is more general and can be transferred to compute sensitivities of arbitrary models (see e.g. *Giering*, 1998 submitted).

The mathematical representation of the required sensitivities are derivatives of the simulated concentration with respect to sources or model parameters. Starting from the numerical code of the transport model, a second code for computation of these derivatives is constructed. This task is performed automatically by another numerical program called Tangent linear and Adjoint Model Compiler (TAMC, *Giering* (1997)). This source to source translator transforms FORTRAN subroutines for evaluation of a function to FORTRAN subroutines for evaluation of the function's derivative. Therefore it applies a technique called automatic differentiation (*Griewank*, 1989), which is briefly introduced in section 2. TAMC comes with a utility that, especially for beginners, facilitates the usage of the derivative computing code generated by TAMC. A brief description of both TAMC and the utility is given in section 3.

As for most of the examples of the optimization chapter, the simple two

box model Boxmod is employed to demonstrate the computation of sensitivities. Despite its simplicity, Boxmod contains all features that are important in the context of computation of sensitivities. Prescribing hemispheric source estimates provided by *Prinn et al.* (1992), like in the optimization examples Boxmod is used to simulate the methyl chloroform concentration in both hemispheres. In one of the optimization examples, TAMC is applied to compute sensitivities for Boxmod. Hence the present examples can be considered as preliminary to the optimization examples. In section 4, Boxmod and its derivatives are discussed.

This document also comprises a set of exercises. To keep the main text easy to read for those not interested in all exercises, the exercises are presented separately in section 5. For those who want to do the exercises, however, at a specific place in the main text each of the exercises is referred to. The exercises that involve Boxmod, have to be done numerically. For these exercises source code has been prepared. For those readers who want to do the exercises but don't have access to a computer, the output of the program is included in figures. More details about the topics introduced in sections 2 and 3 can be found in *Giering and Kaminski* (1998) and *Giering* (1997). In particular when doing the exercises it is recommended to have these documents available.

2 Automatic Differentiation

Mathematically, transport models can be represented by mappings (or vector valued functions) transforming a vector of sources or model parameters into a vector of concentrations. And the sensitivities are represented by the derivative of this mapping, i.e. by the so called Jacobian matrix of the transport model, whose entries are the partial derivatives of the components of the concentration vector with respect to the components of the vector of sources (or model parameters). The aim of this section is to give a brief introduction into a technique to compute these derivatives, which is called automatic differentiation, see *Giering and Kaminski* (1998) for more details.

There are different approaches for the computation of the abovementioned partial derivatives: They can be approximated by finite differences, which requires multiple runs of the transport model to simulate the differences in the concentration resulting from a change of the components of the source vector. An alternative is the exact computation of the derivatives by automatic differentiation: The mapping that represents the transport model is considered

as a composition of a (in general huge) number of individual mappings which represent the individual steps executed by the transport model. According to the chain rule the derivative of the mapping that represents the transport model is the product of the derivatives of the individual mappings. Since the numerical code of the transport model is simply another representation (in a programming language) of the original mapping, it can be employed for identification of the individual mappings. After interpretation of the code in terms of individual mappings, these are differentiated, and a second code for propagating derivatives according to the chain rule is constructed. Based on a few principles essentially suggested by *Talagrand* (1991), *Giering and Kaminski* (1998) have derived a few simple rules for identification of the individual mappings from the model code and construction of derivative code that operates in reverse mode (see next paragraph).

The chain rule results in a multiple matrix product, which is associative, i.e. (numerically, apart from rounding errors) the order in the evaluation of this product does not matter. Derivative code that evaluates the chain rule in the same order than the transport model code, is said to be working in forward mode (tangent linear code), and derivative code that evaluates the matrix product in the opposite order operates in reverse mode (adjoint code). Note that, although (apart from rounding errors) the results are the same, there may be significant differences in the amount of storage and CPU time required by codes operating in the two modes. The differences depend on the ratio of the number of input variables (dimension of the vector of sources) to the number of output variables (dimension of the vector of concentrations). For a large number of input variables and a small number of output variables, the reverse mode is more efficient. For a small number of input variables and a large number of output variables, the forward mode is more efficient. In the programming exercises are examples in which either mode will be favorable for computation of the sensitivities that are required to answer the respective scientific question. **Have a look at exercise 1 now!**

3 TAMC and TAMLINK

Using the concept of automatic differentiation, the task of constructing adjoint or tangent linear code can be based on simple rules, like those described in *Giering and Kaminski* (1998). These simple rules can be applied automatically, e.g. by source to source translation programs. There are a number

of these programs, e.g. Odyssee (*Rostaing et al.*, 1993), GRESS (*Horwedel*, 1991), or TAMC (*Giering*, 1997) for reverse mode and ADIFOR (*Bischof et al.*, 1992) or TAMC for forward mode (see *Bischof* (URL) for more references). In this section the program TAMC is introduced. The focus lies on features that are necessary to understand the examples, for more details see (*Giering*, 1997) and (*Giering and Kaminski*, 1998).

Adjoint and tangent linear codes propagate derivatives. In the case of adjoint code these derivatives quantify the sensitivity of the output variables with respect to intermediate results and in the case of tangent linear code these derivatives quantify the sensitivity of intermediate results with respect to the input variables. In TAMC there is a one to one correspondence of variables in the model code to variables in the adjoint (adjoint variables) or tangent linear code (tangent linear variables), which hold the derivatives (*Giering and Kaminski*, 1998). Those variables that either have no influence on the output variables or do not depend on the input variables are called passive variables. For passive variables no derivatives need to be propagated (*Giering and Kaminski*, 1998). In contrast, active variables are those that influence the output variables and also depend on the input variables. For each active variable, an adjoint (or tangent linear variable) is declared in the adjoint (or tangent linear) code, to hold the corresponding derivative. To each statement in the code in which active variables are involved, according to simple rules, adjoint (or tangent linear) statements are generated. **Exercises exercise 3 and exercise 4 include identification of active variables**

Whenever the adjoint and tangent linear statements contain variables from the model code (e.g. due to non linear operations), those variables are called required variables, because their values (as right before execution of the statement the derivative statement corresponds to) are required to further propagate the derivatives. In tangent linear code those values can be easily provided (e.g. by inserting all tangent linear statements in the model code, thereby locating each tangent linear statement before the statement it corresponds to). In adjoint code, providing required variables efficiently is one of the major challenges. **Exercises exercise 3 and exercise 4 include identification of required variables**

The TAMC system consists of two parts: a utility that is to be installed on your local computer and the actual software that does automatic differentiation, which is installed on a remote machine. The source code of the utility is provided with the code for the exercises. TAMC is invoked by the utility trough a UNIX shell script **tamc**, which uses a UNIX remote shell to com-

municate with the remote machine. Trough command line options, the user defines the function to be differentiated by naming a FORTRAN subroutine, its input and output variables as well as the files containing the code to be differentiated. Section 4 gives two examples for invoking TAMC.

To execute the generated derivative code, it has to be embedded in a main program. For different applications of derivate code, a software package (TAMLINK) comprising a number of main programs is included in the utility. The code is linked by a second UNIX shell script named **tamlink**. In order to fit to the respective main program TAMLINK expects a set of subroutines with particular names and interfaces. Section 4 gives a number of examples for invoking TAMLINK.

4 Sensitivities of Boxmod

4.1 Boxmod

Boxmod is a simple two box model of the atmospheric transport. An introduction to box models in general and Boxmod in particular is given in the optimization chapter. In this document the focus lies on the important details for automatic differentiation. Despite the simplicity of Boxmod, from this point of view, many features of the structure of larger, three dimensional models that are currently used for transport model studies are contained in Boxmod.

[Figure 1 about here.]

The code of Boxmod is depicted in Figure 1. Using prescribed hemispheric estimates of the sources of methyl chloroform, which have been provided by *Prinn et al.* (1992), and an atmospheric lifetime of 4.7 years (*Houweling et al.*, 1998), Boxmod is simulating the hemispheric concentrations of methyl chloroform. Boxmod has a (single) transport parameter, namely the rate of interhemispheric mixing `mixrate` (one of the optimization exercises is to tune this parameter using sensitivities computed by the adjoint of Boxmod). Unlike more sophisticated transport models, the transport in Boxmod has neither seasonality nor interannual variations. The change in the simulated concentration depends non linearly on the mixing rate and, due to the sink term, it also depends non linearly on the sources. For an inert tracer like CO_2, i.e. in the absence of the sink term that depends on the concentration, the change in the simulated concentration would be linear in the sources.

4.2 Adjoint

[Figure 2 about here.]

[Figure 3 about here.]

This is a good moment to have a look at exercise 2 first and then at exercise 3. Exercise 3 can be solved by applying TAMC to generate code for computation of the sensitivity of the concentration in box 1 at the last time step with respect to the mixing rate, the inverse lifetime, and the source components (see answer 3). Since the function to be differentiated has one output variable and many input variables, the reverse mode is most efficient, i.e. the adjoint of Boxmod is to be constructed. To provide all necessary information about the function to be differentiated to TAMC:

- a subroutine `model` (see Figure 2), which has in its argument list the number of input variables, the vector of input variables `X`, and the output variable `FC`, has been constructed,
- and TAMC is called with the options
  -module model -input X -output FC -reverse.

A particularly important feature for adjoint code generation, which is typical for transport models, is that Boxmod overwrites the current concentration every time step. For applications like exercise 3, in which either the mixing rate or the inverse lifetime are active variables, the concentration `c` is one of the required variables. There are further required variables (or parameters) such as `ny, ntpy, kt2pptv, mixrate`, and `invlif`, but those are not overwritten and, thus, easy to provide. By default TAMC generates a loop to recompute the required values of `c`, but TAMC also provides the alternative of storing the required values on a 'tape', i.e. in memory or in a file. In Figure 2 the storing feature is demonstrated, because recomputation would require a second loop within the adjoint of the main loop (see *Giering and Kaminski*, 1998; *Giering*, 1997), which, computationally, for a three dimensional model would be prohibitively expensive (see e.g. computational cost of the adjoint of TM2 in *Kaminski et al.* (submitted)). Directives are used to make TAMC store and read the required variables by calling special library routines. First an `init` directive is needed to initialize the 'tape', and then right before the statement whose adjoint uses the required variable a `store` directive is inserted (see Figure 2). Before the adjoint code is executed, the required values are recomputed

and stored, and during the execution of the adjoint code they are read (see below).

[Figure 4 about here.]

[Figure 5 about here.]

[Figure 6 about here.]

The adjoint of Boxmod generated by TAMC is depicted in Figures 4 - 6. Figure 4 contains the declaration of required and adjoint variables and the initialization of the local adjoint variables `adc` and `adcnew` to 0. The global adjoint variable `adsrc` is initialized by the subroutine `adzero`. The global adjoint variable `adfc` has to be initialized to 1 before calling `admodel`. The initialization strategy can be understood as a consequence of the concept of locality (for details see definition of locality in *Giering and Kaminski*, 1998): The current values of adjoint variables reflect the derivative of `fc` with respect to the variable in the forward mode the adjoint variable corresponds to. Thereby current refers to the place in the code of Boxmod where the statement to which the adjoint statement corresponds is located. Since `admodel` runs in reverse mode, the end of `model` is the location the initialization part of `admodel` corresponds to. And at the end of `model`, only a change in the variable `fc` could change the function value, all others don't have any impact anymore. Hence their adjoints must be 0. The adjoint computations part in `admodel` (see Figure 5) updates the values of the adjoint variables by executing the adjoints of the statements (see *Giering and Kaminski*, 1998) in `model` in reverse order. At the end of `admodel` the adjoint of `x` holds the derivative of `fc` with respect the `x`, and all other adjoint variables hold 0s.

As discussed above, the correct values of most of the required variables are easy to provide. For instance the values of `mixrate` and `invlif` can be recovered from `x` at the beginning of the adjoint computations part. The computation of `c` is carried out during computation of the value of the function, which TAMC adds by default before the adjoint computations, since it is needed e.g. for use in optimization procedures (generation of code for the value of the function can be avoided by using the directive **-pure**). Reading and writing and the necessary bookkeeping are organized by the subroutines `adstore` and `adresto`.

[Figure 7 about here.]

[Figure 8 about here.]

[Figure 9 about here.]

To actually compute the sensitivity needed to answer exercise 3, `admodel` has to be executed using the correct values of the input variables, i.e. sources, inverse lifetime, and mixing rate. The subroutine can be executed easily after linking the appropriate main program for computing sensitivities by TAMLINK. This main program requires that, besides `model`, three subroutines be provided:

- a subroutine `numbmod` defining the number of input variables (see Figure 7),
- a subroutine `initmod` doing all necessary initialization, in particular setting the values of the input variables (see Figure 8),
- and a subroutine `postmod` doing the post processing (see Figure 9).

To link the main program for computation of the sensitivity in reverse mode, TAMLINK is to be invoked with the command line option **-adjoint**.

4.3 Tangent linear model

[Figure 10 about here.]

This is a good moment to have a look at exercise 4. exercise 4 can be solved by applying TAMC to generate code for computation of the sensitivity of the concentration in both boxes and all years with respect to the inverse lifetime and the mixing ratio (see answer 4). Since the function to be differentiated has two input variables and many output variables, the forward mode is most efficient, i.e. the tangent linear of Boxmod is to be constructed. To provide all necessary information about the function to be differentiated to TAMC

- a subroutine `func` (see Figure 10), which has in its argument list the number of input variables, the vector of input variables `X`, the number of output variables, and the output variable `Y` has been constructed,
- and TAMC is called with the options
  -module func -input X -output Y -ldg -jacobian 20 -forward
  (the **-jacobian** option sets the number of output variables, default is one and the **-ldg** extends the parameter list of `g_func`).

[Figure 11 about here.]

[Figure 12 about here.]

The tangent linear code of Boxmod is depicted in Figures 11 and 12. The tangent linear variables are marked by the prefix `g_`. Recall that they hold the derivative of the (active) variable they correspond to with respect to the input variable `x`. Figure 4 contains the declaration of forward code and tangent linear variables. Note that unlike adjoint variables, tangent linear variables do not have to be initialized, because, except for the tangent linear variables corresponding to the input variable `x`, they are not referenced before being set by at least one tangent linear statement in `g_func`. Each statement from the function evaluation is preceded by its tangent linear statement. Thanks to this scheme, correct values of all required variables are readily provided. As a comfortable side effect, tangent linear code is much better readable than adjoint code. While the function part of `_g` computes `y`, the derivative part computes `g_y`

[Figure 13 about here.]

[Figure 14 about here.]

[Figure 15 about here.]

As for the adjoint code, to actually compute the sensitivity needed to answer exercise 4, `g_func` has to be executed using the correct values of the input variables, i.e. sources, inverse lifetime, and mixing rate. The subroutine can be executed easily after linking the appropriate main program for computing sensitivities by TAMLINK. This main program requires that, besides `func`, three subroutines be provided:

- a subroutine `setfunc` defining the number of input and output variables (see Figure 13),
- a subroutine `initfunc` doing all necessary initialization, in particular setting the values of the input variables (see Figure 14),
- and a subroutine `postfunc` doing the post processing (see Figure 15).

Note that unlike the scalar valued function of the previous example, here a vector valued function has been differentiated, which is the reason for the differences in the argument lists of the subroutines. To link the main program for computation of the sensitivity in forward mode, TAMLINK is to be invoked with the command line option **-forward**.

5 Exercises

5.1 Questions

1. Imagine you have a simple transport model, and for computing the derivative you need, the mapping defined by your transport model can be decomposed to only four individual mappings (i.e. the code of your transport model is indeed very simple). Their respective Jacobians are (in the order of the transport model using the convention <number of rows> × <number of columns>) 2×5, 3×5, 3×3, and 1×3 matrices, i.e. the derivative of the transport model is a 1×5 matrix.

   (a) Make a drawing of the matrix product and of its evaluation in forward and reverse modes!

   (b) What is the largest matrix needed to hold a (n intermediate) result in forward and reverse modes?

   (c) How many additions and multiplications are needed for the evaluation of the product in forward and reverse modes?

2. Compile and run boxmod (boxmod is in file boxmod_0.F).

3. Imagine you perform a unit change of the mixing rate, the inverse lifetime, or any of the source components. Which change (to first order, since we want to use first derivatives) has the largest impact on the simulated concentration in box 1 at the end of year 10 (this is an application for reverse mode)?
   Which are the active variables?
   Which are the required variables?
   Detailed instructions:

   (a) Transform the code in boxmod_0.F to a form that defines the concentration at the end of year 10 as a function (in the code: subroutine) of all sources (and thus can be recognized by TAMC) and that can be recognized by TAMLINK. Having a look in the directory ../tamc/examples might help!

   (b) Invoke 'tamlink -cost' to check whether this modified code runs and computes the correct value for the concentration in year 10!

   (c) Invoke TAMC to compute the derivate of this function!

(d) Check this derivative against finite differences by using 'tamlink -check'!

(e) Run the adjoint code by using 'tamlink -adjoint'!

4. The concentration in which box and year will be strongest effected by a unit change of the inverse lifetime or the mixing ratio (this is an application for forward mode)?
   Which are the active variables?
   Which are the required variables?
   Detailed instructions:

   (a) Transform the code in boxmod_0.F to a form that defines the concentration in all years and both boxes as a function (in the code: subroutine) of both sources in the first year (and thus can be recognized by TAMC) and which can be recognized by TAMLINK. Having a look in the directory ../tamc/examples might help!

   (b) Invoke TAMC to compute the derivate of this function!

   (c) Check this derivative against the derivative computed in reverse mode by using 'tamlink -compare'!

5.2 Answers

For most of the numerical exercises targets for 'make' and code have been prepared. In this case you'll get the answer by entering 'make <target>'. Most of the answers are discussed in the main text, right after the reader is asked to do the exercise.

[Figure 16 about here.]

[Figure 17 about here.]

[Figure 18 about here.]

1. (a) Have a look at Figure 16!

   (b) In forward mode it is the 3×5 matrix after the first product has been evaluated, and in reverse mode it is the 1×5 matrix holding the final result.

(c) In forward mode, the number of multiplications is:

$$\begin{aligned} n_m &= 3 \times 2 \times 5 \\ &+ 3 \times 3 \times 5 \\ &+ 1 \times 3 \times 5 \\ &= 90, \end{aligned}$$

and the number of additions is:

$$\begin{aligned} n_a &= 3 \times 2 - 1 \times 5 && (1) \\ &+ 3 \times 3 - 1 \times 5 && (2) \\ &+ 1 \times 3 - 1 \times 5 && (3) \\ &= 55. && (4) \end{aligned}$$

In reverse mode, the number of multiplications is:

$$\begin{aligned} n_m &= 1 \times 3 \times 3 \\ &+ 1 \times 3 \times 2 \\ &+ 1 \times 2 \times 5 \\ &= 25, \end{aligned}$$

and the number of additions is:

$$\begin{aligned} n_a &= 1 \times 3 - 1 \times 3 \\ &+ 1 \times 3 - 1 \times 2 \\ &+ 1 \times 2 - 1 \times 5 \\ &= 15. \end{aligned}$$

2. Enter: 'make run'!

3. The active variables are `x, mixrate, invlif, src, c_new, c`, and `fc`. The required variables are `c, ny, ntpy, kt2pptv, mixrate`, and `invlif`.

   (a) Have a look at Figures 2 and 7 - 9!

   (b) Enter: 'make cost'!

   (c) Enter: 'make boxmod_1_ad.f'!

(d) Enter: 'make check'!

(e) Enter: 'make adjoint' or have a look at Figure 17!

A unit change of the inverse lifetime has the largest impact on the simulated concentration in box 1 at the end of year 10 (-424 pptv $\times$ year). Note that the answer depends on what units we choose for the input and output variables!

4. The active variables are `x, g_p_, mixrate, invlif, c_new, c`, and `y`. The required variables are `c, ny, ntpy, kt2pptv, mixrate`, and `invlif`.

(a) Have a look at Figure 10!

(b) Enter: 'make forward' or have a look at Figure 18!

(c) Enter: 'make jacobian'!

The concentration in box 1 after year 10 is effected strongest by a unit change of the inverse lifetime ((-424 pptv $\times$ year). The concentrations in both boxes after year 10 are effected equally strong by a unit change of the mixing rate (± 10.4 pptv $\times$ year). This time the answers does not depend on the units chosen.

References

Bischof, C., A collection of automatic differentiation tools, URL=`http://www.mcs.anl.gov/Projects/autodiff/AD_Tools/index.html`, URL.

Bischof, C., A. Carle, G. Corlies, A. Griewank, and P. Hovland, ADIFOR: Generating derivative codes from FORTRAN programs, *Scientific Programming*, *1(1)*, 11–29, 1992.

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

Giering, R., Tangent linear and adjoint models, in *Inverse modeling of biogeochemical cycles*, edited by P. Kasibhatla, and P. J. Rayner, American Geophysical Union, 1998 submitted.

Giering, R., and T. Kaminski, Recipes for Adjoint Code Construction, 1998, in press ACM Trans. On Math. Software.

Griewank, A., On automatic differentiation, in *Mathematical Programming: Recent Developments and Applications*, edited by M. Iri, and K. Tanabe, pp. 83 – 108, Kluwer Academic Publishers, 1989.

Horwedel, J. E., GRESS: A preprocessor for sensitivity studies on Fortran programs, in *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, edited by A. Griewank, and G. F. Corliss, pp. 243–250, SIAM, Philadelphia, Penn., 1991.

Houweling, S., F. Dentener, and J. Lelieveld, The impact of nonmethane hydrocarbon compounds on tropospheric photochemistry, *J.Geophys.Res.*, (D9), 10673–10696, 1998.

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.*, submitted.

Prinn et al., Global average concentration and trend for hydroxil radical deduction from ALE/GAGE trichloroethane (methyl chloroform) data for 1987–1990, *J.Geophys.Res.*, *97*, 2445–2461, 1992.

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

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

Acknowledgments

The original version of Boxmod has been programmed by Peter Rayner. Ron Prinn provided the source estimates of methyl chloroform. This work was supported in part by the Commission of the European Communities under contract EV5V-CT92-0120.

List of Figures

```
      program boxmod
      implicit none
c parameters
      integer ny !  number of years
      integer ntpy !  number of time steps per year
      parameter(ny=10,ntpy=50)
      real src(2,ny) !  sources
      real mixrate, invlif !  1/mixing rate; 1/life time
      real kt2pptv !  conversion source to concentration
      integer i,l !  loop counters
      real c(2),cnew !  concentrations
      integer y !  year of source
      real src_n,src_s,tot !  percentages in nh and sh, total in kt
c initialize sources
c read three comment lines plus the 1977 record,
c i.e.  start with the 1978 sources
      open(unit=1,file='src_CH3CCl3.d',status='old')
      read(1,'(////)')
      do i=1,ny
         read(1,'(8x,i4,2(f6.3),f6.1)') y,src_n,src_s,tot
         src(1,i) =src_n*tot
         src(2,i) =src_s*tot
      end do
      close(1)
c initialize transport and sink
      mixrate = 1.  !  mixing rate in 1/year
      invlif = 1./4.7 !  invers lifetime in 1/year
                                    !  sander houweling found 4.7 years
c conversion for kt to pptv within a hemisphere
      kt2pptv = 0.471 * 2 * 12/133.5
c initialize concentration with values for 1978
      c(1) = 84.  !  northern box
      c(2) = 60.  !  southern box
c output
      write(*,'(a50)') 'The simulated concentration is :  '
      write(*,'(a10,2(a20))'),'year','box1','box2'
      write(*,'(i10,2(12x,f8.4))')
     .      0,c(1),c(2)
c calculate concentrations with forward differencing box model
c and add contribution to misfit function every year
c (cnew stores the new value of c(1) because the old is need for
c computation of c(2) )
      do i=1,ny
         do l=1,ntpy
         cnew = c(1) + 1./ntpy *
     .            ( kt2pptv*src(1,i) - (c(1)-c(2))
     .            * mixrate - invlif*c(1) )
         c(2) = c(2) + 1./ntpy *
     .            ( kt2pptv*src(2,i) - (c(2)-c(1))
     .            * mixrate - invlif*c(2) )
         c(1) = cnew
         enddo
c output
         write(*,'(i10,2(12x,f8.4))')
     .         i,c(1),c(2)
      enddo
      end
```

Figure 1: The code of Boxmod.

```
C================================================
C The subroutine "MODEL" is called by the optimization procedure.
C It has to calculate the cost function "FC"
C depending on the control vector "X(N)".
C================================================
      SUBROUTINE MODEL( N, X, FC )
      implicit none
      INTEGER N
      REAL X(N), FC

#include "boxmod.h"
      integer i,j,l                !  loop counters
      real c(2),cnew               !  concentrations

c tamc directive to initialize a tape for the trajectory
c cadj init tape1 = 'trajectory'
c alternatively memory can be used
cadj init tape1 = MEMORY

c copy control variables
      mixrate = x(1)
      invlif = x(2)
      do i=1,ny
         do j=1,2
           src(j,i)=x(2+i+(j-1)*ny)
         enddo
      end do
c initialize concentration with values for 1978
      c(1) = 84.                   !  northern box
      c(2) = 60.                   !  southern box

c calculate concentrations with forward differencing box model
c and add contribution to misfit function every year
c (cnew stores the new value of c(1) because the old is need for
c computation of c(2) )
      do i=1,ny
         do l=1,ntpy
cadj store c = tape1
           cnew = c(1) + 1./ntpy *
     .             ( kt2pptv*src(1,i) - (c(1)-c(2))
     .             * mixrate - invlif*c(1) )
           c(2) = c(2) + 1./ntpy *
     .             ( kt2pptv*src(2,i) - (c(2)-c(1))
     .             * mixrate - invlif*c(2) )
           c(1) = cnew
         enddo
      enddo
c set output variable
      fc=c(1)
      END
```

Figure 2: Subroutine `model` defining the function to be differentiated in exercise 3. The subroutine has been constructed by rearranging the code of Boxmod. The header file `boxmod.h` is depicted in Figure 3.

```
c header file for boxmod
c parameters
     integer ny !  number of years
     integer ntpy !  number of time steps per year
     parameter(ny=10,ntpy=50)
c variables
     real src(2,ny) !  sources
     real mixrate, invlif !  1/mixing rate; 1/life time
     real kt2pptv !  conversion source to concentration
     common /vars / mixrate, invlif, src, kt2pptv
c
c values of control variables:
     real src0(2,ny) !  sources
     real mixrate0, invlif0 !  1/mixing rate; 1/life time
     common /cvars / mixrate0, invlif0, src0
```

Figure 3: Header file `boxmod.h` for subroutines `numbmod`, `model`, `initmod` and `postmod`.

```
      subroutine admodel( n, x, fc, adx, adfc )
C*****************************************************************
C*****************************************************************
C** This routine was generated by the **
C** Tangent linear and Adjoint Model Compiler, TAMC 4.97 **
C*****************************************************************
C*****************************************************************
      implicit none
C=============================================
C define parameters
C=============================================
      integer ntpy
      parameter ( ntpy = 50 )
      integer ny
      parameter ( ny = 10 )
C=============================================
C define common blocks
C=============================================
      common /advars/ admixrate, adinvlif, adsrc
      real adinvlif, admixrate, adsrc(2,ny)
      common /vars/ mixrate, invlif, src, kt2pptv
      real invlif, kt2pptv, mixrate, src(2,ny)
C=============================================
C define arguments
C=============================================
      integer n
      real adfc, adx(n), fc, x(n)
C=============================================
C define local variables
C=============================================
      real adc(2), adcnew, c(2), cnew
      integer i, ip1, j, l
C---------------------------------------------
C RESET GLOBAL ADJOINT VARIABLES
C---------------------------------------------
      call adzero
C---------------------------------------------
C RESET LOCAL ADJOINT VARIABLES
C---------------------------------------------
      do ip1 = 1, 2
         adc(ip1) = 0.
      end do
      adcnew = 0.
```

Figure 4: The adjoint and modified forward codes of Boxmod, declaration and initialization of adjoint variables. TAMC output has been slightly edited to fit better in the figure.

```
C-----------------------------------------------
C ROUTINE BODY
C-----------------------------------------------
C-----------------------------------------------
C FUNCTION AND TAPE COMPUTATIONS
C-----------------------------------------------
     mixrate = x(1)
     invlif = x(2)
     do i = 1, ny
        do j = 1, 2
          src(j,i) = x(2+i+(j-1)*ny)
        end do
     end do
     c(1) = 84.
     c(2) = 60.
     do i = 1, ny
        do l = 1, ntpy
          call adstore( 'memory_1_model_c',16,c,8,2,1+((-1)+i)*ntpy+l-1)
          cnew = c(1)+1./ntpy*(kt2pptv*src(1,i)-(c(1)-c(2))*mixrate-invlif*c(1))
          c(2) = c(2)+1./ntpy*(kt2pptv*src(2,i)-(c(2)-c(1))*mixrate-invlif*c(2))
          c(1) = cnew
        end do
     end do
     fc = c(1)
C-----------------------------------------------
C ADJOINT COMPUTATIONS
C-----------------------------------------------
     mixrate = x(1)
     invlif = x(2)
     adc(1) = adc(1)+adfc
     adfc = 0.
     do i = ny, 1, -1
        do l = ntpy, 1, -1
          call adresto( 'memory_1_model_c',16,c,8,2,1+((-1)+i)*ntpy+l-1 )
          adcnew = adcnew+adc(1)
          adc(1) = 0.
          adinvlif = adinvlif-adc(2)*1./float(ntpy)*c(2)
          admixrate = admixrate-adc(2)*1./float(ntpy)*(c(2)-c(1))
          adsrc(2,i) = adsrc(2,i)+adc(2)*1./float(ntpy)*kt2pptv
          adc(1) = adc(1)+adc(2)*1./float(ntpy)*mixrate
          adc(2) = adc(2)*(1-1./float(ntpy)*(mixrate+invlif))
          adc(2) = adc(2)+adcnew*1./float(ntpy)*mixrate
          adc(1) = adc(1)+adcnew*(1-1./float(ntpy)*(mixrate+invlif))
          adinvlif = adinvlif-adcnew*1./float(ntpy)*c(1)
          admixrate = admixrate-adcnew*1./float(ntpy)*(c(1)-c(2))
          adsrc(1,i) = adsrc(1,i)+adcnew*1./float(ntpy)*kt2pptv
          adcnew = 0.
        end do
     end do
     adc(2) = 0.
     adc(1) = 0.
     do i = 1, ny
        do j = 1, 2
          adx(2+i+(j-1)*ny) = adx(2+i+(j-1)*ny)+adsrc(j,i)
          adsrc(j,i) = 0.
        end do
     end do
     adx(2) = adx(2)+adinvlif
     adinvlif = 0.
     adx(1) = adx(1)+admixrate
     admixrate = 0.
     end
```

Figure 5: The adjoint and modified forward codes of Boxmod, function computation including computation and storing of c as well as adjoint computations including reading of c. TAMC output has been slightly edited to fit better in the figure.

```
      subroutine adzero
C******************************************************************
C******************************************************************
C** This routine was generated by the **
C** Tangent linear and Adjoint Model Compiler, TAMC 4.97 **
C******************************************************************
C******************************************************************
      implicit none
C================================================
C define parameters
C================================================
      integer ny
      parameter ( ny = 10 )
C================================================
C define common blocks
C================================================
      common /advars/ admixrate, adinvlif, adsrc
      real adinvlif, admixrate, adsrc(2,ny)
C================================================
C define local variables
C================================================
      integer ip1, ip2
      admixrate = 0.
      adinvlif = 0.
      do ip2 = 1, ny
         do ip1 = 1, 2
           adsrc(ip1,ip2) = 0.
         end do
      end do
      end
```

Figure 6: The adjoint and modified forward codes of Boxmod, subroutine `adzero` for initialization of global adjoint variables. TAMC output has been slightly edited to fit better in the figure.

```
C================================================
C This subroutine sets the number
C of control variables
C================================================
      SUBROUTINE NUMBMOD( N )
      implicit none
#include "boxmod.h"
      INTEGER N
      n = 2+2*ny

      END
```

Figure 7: Code of Subroutine `numbmod`. `numbmod` is one of the subroutines needed by TAMLINK to link `admodel` to a main program for computation of the sensitivity. The header file `boxmod.h` is depicted in Figure 3.

```
C=================================================
C The subroutine "INITMOD" is called before the
C optimization.  It must set a first guess
C of the parameter vector.
C It may also contain the initialization of
C the model.
C=================================================
     SUBROUTINE INITMOD( N, X )
     implicit none

     INTEGER N
     REAL X(N)
#include "boxmod.h"

     integer i,j                   !  loop counter
     integer y                     !  year of source
     real src_n,src_s,tot          !  percentages in nh and sh, total in kt
c initialize sources
c read three comment lines plus the 1977 record,
c i.e.  start with the 1978 sources
     open(unit=1,file='src_CH3CCl3.d',status='old')
     read(1,'(////)')
     do i=1,ny
        read(1,'(8x,i4,2(f6.3),f6.1)') y,src_n,src_s,tot
        src0(1,i) =src_n*tot
        src0(2,i) =src_s*tot
     end do
     close(1)

c initialize transport and sink
     mixrate0 = 1.                 !  mixing rate in 1/year
     invlif0 = 1./4.7              !  invers lifetime in 1/year
                                   !  sander houweling found 4.7 years

c conversion for kt to pptv within a hemisphere
c we use 0.471 to transform from
c gigatons of carbon in co2 to ppmv in the entire atmosphere
c the ratio of the molecular weights of carbon and CH3CCl3
c should be something like 12/133.5
c replacing Gt by kt and ppmv by pptv cancels out
     kt2pptv = 0.471 * 2 * 12/133.5

c set first guess of (potential) control vars
     x(1) = mixrate0
     x(2) = invlif0
     do i=1,ny
        do j=1,2
          x(2+i+(j-1)*ny)=src0(j,i)
        enddo
     end do
     END
```

Figure 8: Code of Subroutine `initmod`. `initmod` is one of the subroutines needed by TAMLINK to link `admodel` to a main program for computation of the sensitivity. The header file `boxmod.h` is depicted in Figure 3.

```
C================================================
C The subroutine "POSTMOD" is called after
C the optimization.
C It should contain the output of the results.
C================================================
     SUBROUTINE POSTMOD( N, X, FC, ADX )
     implicit none

     INTEGER N
     REAL X(N), FC, ADX(N)
     INTEGER I

#include "boxmod.h"
     write(*,'(a25,3(2x,f13.4))')
    .     'The value of fc is :  ',fc
     write(*,'(a25)') 'Its derivatives are :  '
     write(*,'(a55,1(2x,f13.4))')
    .     'with resp.  to mixing rate :  ', adx(1)
     write(*,'(a55,1(2x,f13.4))')
    .     'with resp.  to inv.  lifetime :  ', adx(2)
     write(*,'(a55)') 'with resp.  to sources :  '
     write(*,'(a10,2(a30))'),'year','box1','box2'
     do i=1,ny
        write(*,'(i10,2(22x,f8.4))')
    .         i,adx(2+i),adx(2+i+ny)
     end do
     END
```

Figure 9: Code of Subroutine `postmod`. `postmod` is one of the subroutines needed by TAMLINK to link `admodel` to a main program for computation of the sensitivity. The header file `boxmod.h` is depicted in Figure 3.

```
C================================================
C This is the top level routine,
C it has to calculate the dependent variables Y(M)
C out of the independent variables X(N)
C================================================
     SUBROUTINE FUNC( N, X, M, Y )
     INTEGER N, M
     REAL X(N), Y(M)

#include "boxmod.h"
     integer i,j,l                !  loop counters
     real c(2),cnew               !  concentrations

c tamc directive to initialize a tape for the trajectory
c (adjoint code is generated to check tangent linear code)
c cadj init tape1 = 'trajectory'
c alternatively memory can be used
cadj init tape1 = MEMORY
c copy control variables
     mixrate = x(1)
     invlif = x(2)
c initialize concentration with values for 1978
     c(1) = 84.                   !  northern box
     c(2) = 60.                   !  southern box

c calculate concentrations with forward differencing box model
c and add contribution to misfit function every year
c (cnew stores the new value of c(1) because the old is need for
c computation of c(2) )
     do i=1,ny
        do l=1,ntpy
cadj store c = tape1
          cnew = c(1) + 1./ntpy *
    .             ( kt2pptv*src(1,i) - (c(1)-c(2))
    .             * mixrate - invlif*c(1) )
          c(2) = c(2) + 1./ntpy *
    .             ( kt2pptv*src(2,i) - (c(2)-c(1))
    .             * mixrate - invlif*c(2) )
          c(1) = cnew
        enddo
c save output
        do j=1,2
          y(j+(i-1)*2) = c(j)
        enddo
     enddo
     END
```

Figure 10: Subroutine `func` defining the function to be differentiated in exercise 4. The subroutine has been constructed by rearranging the code of Boxmod. The header file `boxmod.h` is depicted in Figure 3.

```
      subroutine g_func( n, x, m, y, g_p_, g_x, ldx, g_y, ldy )
C****************************************************************
C****************************************************************
C** This routine was generated by the **
C** Tangent linear and Adjoint Model Compiler, TAMC 4.97 **
C****************************************************************
C****************************************************************
      implicit none
C=============================================
C define parameters
C=============================================
      integer g_pmax
      parameter ( g_pmax = 20 )
      integer ntpy
      parameter ( ntpy = 50 )
      integer ny
      parameter ( ny = 10 )
C=============================================
C define common blocks
C=============================================
      common /g_vars/ g_mixrate, g_invlif
      real g_invlif(g_pmax)
      real g_mixrate(g_pmax)
      common /vars/ mixrate, invlif, src, kt2pptv
      real invlif
      real kt2pptv
      real mixrate
      real src(2,ny)
C=============================================
C define arguments
C=============================================
      integer g_p_
      integer ldx
      integer ldy
      integer m
      integer n
      real g_x(ldx,n)
      real g_y(ldy,m)
      real x(n)
      real y(m)
C=============================================
C define local variables
C=============================================
      real c(2)
      real cnew
      real g_c(g_pmax,2)
      real g_cnew(g_pmax)
      integer g_i_
      integer i
      integer j
      integer l
C---------------------------------------------
C CHECK PACT LOWER EQUAL PMAX
C---------------------------------------------
      if (g_p_ .gt.  g_pmax) then
        stop 'error :  pact is greater than pmax'
      endif
```

Figure 11: The tangent linear and forward code of Boxmod, declaration and initialization of tangent linear variables. TAMC output has been slightly edited to fit better in the figure.

```
C-----------------------------------------------
C TANGENT LINEAR AND FUNCTION STATEMENTS
C-----------------------------------------------
     do g_i_ = 1, g_p_
       g_mixrate(g_i_) = g_x(g_i_,1)
     end do
     mixrate = x(1)
     do g_i_ = 1, g_p_
       g_invlif(g_i_) = g_x(g_i_,2)
     end do
     invlif = x(2)
     do g_i_ = 1, g_p_
       g_c(g_i_,1) = 0.
     end do
     c(1) = 84.
     do g_i_ = 1, g_p_
       g_c(g_i_,2) = 0.
     end do
     c(2) = 60.
     do i = 1, ny
       do l = 1, ntpy
         do g_i_ = 1, g_p_
           g_cnew(g_i_) = g_c(g_i_,2)*1./float(ntpy)*mixrate+g_c(g_i_,
    $1)*(1-1./float(ntpy)*(mixrate+invlif))-g_invlif(g_i_)*1./
    $float(ntpy)*c(1)-g_mixrate(g_i_)*1./float(ntpy)*(c(1)-c(2))
         end do
         cnew = c(1)+1./ntpy*(kt2pptv*src(1,i)-(c(1)-c(2))*mixrate-invlif*c(1))
         do g_i_ = 1, g_p_
           g_c(g_i_,2) = g_c(g_i_,2)*(1-1./float(ntpy)*(mixrate+invlif)
    $)+g_c(g_i_,1)*1./float(ntpy)*mixrate-g_invlif(g_i_)*1./float(ntpy)
    $*c(2)-g_mixrate(g_i_)*1./float(ntpy)*(c(2)-c(1))
         end do
         c(2) = c(2)+1./ntpy*(kt2pptv*src(2,i)-(c(2)-c(1))*mixrate-invlif*c(2))
         do g_i_ = 1, g_p_
           g_c(g_i_,1) = g_cnew(g_i_)
         end do
         c(1) = cnew
       end do
       do j = 1, 2
         do g_i_ = 1, g_p_
           g_y(g_i_,j+(i-1)*2) = g_c(g_i_,j)
         end do
         y(j+(i-1)*2) = c(j)
       end do
     end do
     end
```

Figure 12: The tangent linear and forward code of Boxmod, function and tangent linear computations. TAMC output has been slightly edited to fit better in the figure.

```
C================================================
C This subroutine sets the number
C of independent and dependent variables
C================================================
      SUBROUTINE SETFUNC( N, M )
      implicit none
#include "boxmod.h"
      INTEGER N, M
      n = 2
      m = 2*ny
      END
```

Figure 13: Code of Subroutine `setfunc`. `setfunc` is one of the subroutines needed by TAMLINK to link `g_func` to a main program for computation of the sensitivity. The header file `boxmod.h` is depicted in Figure 3.

```
C================================================
C The subroutine INITFUNC
C must set the independent variables
C================================================
     SUBROUTINE INITFUNC( N, X )
     implicit none

     INTEGER N
     REAL X(N)
#include "boxmod.h"

     integer i                    !  loop counter
     integer y                    !  year of source
     real src_n,src_s,tot         !  percentages in nh and sh, total in kt
c initialize sources
c read three comment lines plus the 1977 record,
c i.e.  start with the 1978 sources
     open(unit=1,file='src_CH3CCl3.d',status='old')
     read(1,'(////)')
     do i=1,ny
        read(1,'(8x,i4,2(f6.3),f6.1)') y,src_n,src_s,tot
        src(1,i) =src_n*tot
        src(2,i) =src_s*tot
     end do
     close(1)

c initialize transport and sink
     mixrate0 = 1.                !  mixing rate in 1/year
     invlif0 = 1./4.7             !  invers lifetime in 1/year
                                  !  sander houweling found 4.7 years

c conversion for kt to pptv within a hemisphere
c we use 0.471 to transform from
c gigatons of carbon in co2 to ppmv in the entire atmosphere
c the ratio of the molecular weights of carbon and CH3CCl3
c should be something like 12/133.5
c replacing Gt by kt and ppmv by pptv cancels out
     kt2pptv = 0.471 * 2 * 12/133.5

c set first guess of control vars
     x(1) = mixrate0
     x(2) = invlif0
     END
```

Figure 14: Code of Subroutine `initfunc`. `initfunc` is one of the subroutines needed by TAMLINK to link `g_func` to a main program for computation of the sensitivity. The header file `boxmod.h` is depicted in Figure 3.

```
C================================================
C The subroutine "POSTFUNC" is called at last
C It should contain the output of the results.
C================================================
     SUBROUTINE POSTFUNC( N, X, M, Y, GDX, LDX )
     implicit none

     INTEGER N, M, LDX
     REAL X(N), Y(M), GDX(LDX,M)

#include "boxmod.h"
     integer i                    !  loop counters

     write(*,'(a25)') 'The sensitivities are :  '
     write(*,'(a10,2(a20))'),'year','c box1','c box2'
     write(*,'(a10,4(a10))'),' ',
    .     'mixrate','1/lifet','mixrate','1/lifet'
     do i=1,ny
        write(*,'(i10,4(2x,f8.2))')
    .         i,gdx(1,1+(i-1)*2),gdx(2,1+(i-1)*2),
    .         gdx(1,2+(i-1)*2),gdx(2,2+(i-1)*2)
     end do

     END
```

Figure 15: Code of Subroutine `postfunc`. `postfunc` is one of the subroutines needed by TAMLINK to link `g_func` to a main program for computation of the sensitivity. The header file `boxmod.h` is depicted in Figure 3.

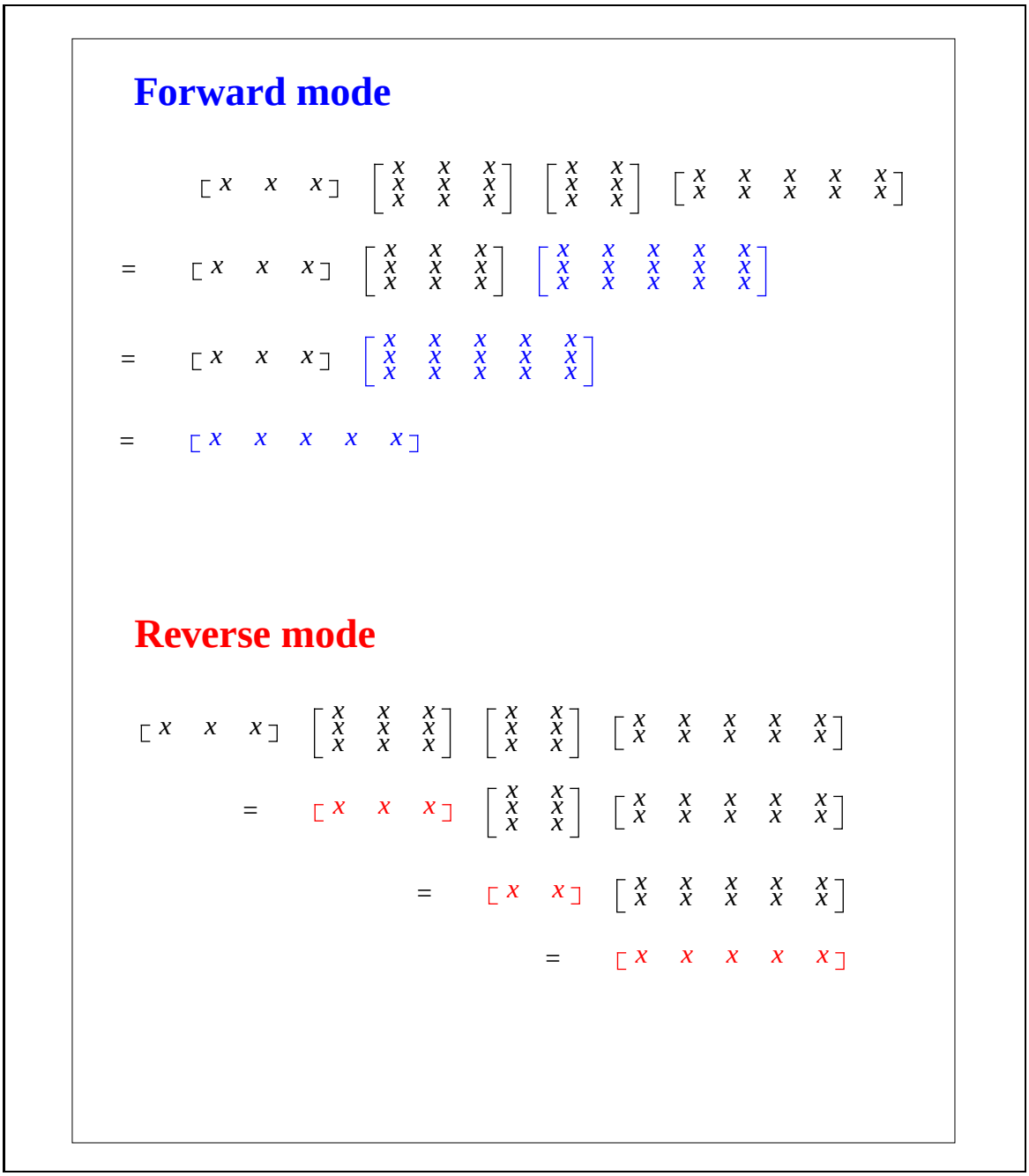

Figure 16: Example of forward and reverse mode illustrating the differences in the storage requirements and in the number of operations: The same matrix product, whose result has 1 row and 5 columns, is evaluated in forward mode, i.e. from right to left (top), and in reverse mode, i.e. from left to right (bottom). In forward mode the matrices holding the intermediate results have 5 columns, while in reverse mode they have 1 row.

```
================================================
      COMPUTATION OF FUNCTION AND DERIVATIVES
      IN REVERSE MODE
================================================
      The value of fc is :       125.8274
      Its derivatives are :
          with resp. to mixing rate :      -10.3982
          with resp. to inv. lifetime :    -424.1904
          with resp. to sources :
          year                box1                box2
            1                 0.0056              0.0056
            2                 0.0069              0.0069
            3                 0.0086              0.0086
            4                 0.0106              0.0106
            5                 0.0132              0.0132
            6                 0.0163              0.0163
            7                 0.0202              0.0201
            8                 0.0251              0.0248
            9                 0.0327              0.0291
           10                 0.0554              0.0211
```

Figure 17: Sensitivities computed by adjoint model to Boxmod.

==========

COMPUTATION OF FUNCTION AND JACOBIAN
IN FORWARD MODE

==========

The sensitivities are :

year	c box1		c box2	
	mixrate	1/lifet	mixrate	1/lifet
1	–8.06	–71.67	8.06	–63.61
2	–8.64	–131.99	8.64	–123.34
3	–8.77	–185.58	8.77	–176.81
4	–8.49	–232.22	8.49	–223.73
5	–8.54	–272.68	8.54	–264.15
6	–9.12	–308.90	9.12	–299.78
7	–9.44	–341.67	9.44	–332.23
8	–9.53	–371.27	9.53	–361.74
9	–9.81	–398.41	9.81	–388.60
10	–10.40	–424.19	10.40	–413.79

Figure 18: Sensitivities computed by tangent linear model to Boxmod.