SENSITIVITY ANALYSIS OF ODE VIA AUTOMATIC DIFFERENTIATION

by

Adrian Sandu

A thesis submitted in partial fulfillment of the
requirements for the Master of Science
degree in Computer Science in the
Graduate College of The
University of Iowa

August 1997

Thesis supervisor: Professor Florian A. Potra

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

MASTER'S THESIS

This is to certify that the Master's thesis of

Adrian Sandu

has been approved by the Examining Committee
for the thesis requirement for the Master of
Science degree in Computer Science at the
August 1997 graduation.

Thesis committee: _____
Thesis supervisor

_____
Member

_____
Member

To my mother and to the memory of my father

ACKNOWLEDGMENTS

This work was supported in part by grants from DOE (DE–FG02–94ER61855), NASA (NAGW–2428) and the Center for Global and Regional Environmental Research.

I want to thank professors Florian Potra and Gregory Carmichael for opening the issue of automatic differentiation as a sensitivity analysis tool in air quality models.

My special thoughts go to my wife Corina for her understanding, encouragement, support and affection, and to my daughter Andreea, for giving a purpose to all this.

ABSTRACT

Sensitivity analysis of numerical ODE solutions via automatic differentiation is discussed. Black box approach is theoretically analysed and compared to other techniques : indirect, direct and direct decoupled. Automatic differentiation techniques are used in the sensitivity analysis of a comprehensive atmospheric chemical mechanism. Specifically, ADIFOR2.0 software is used to calculate the sensitivity of ozone with respect to all initial concentrations (of 84 species) and all reaction rate constants (178 chemical reactions) for six different chemical regions. Numerical aspects of the application of ADIFOR2.0 are also presented. Automatic differentiation is shown to be a powerful tool for sensitivity analysis in environmental models.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Preliminaries

Consider the following initial value problem:

$$\begin{aligned} \frac{dc_i(t)}{dt} &= f(t, c_1, ..., c_n, \beta_1, ..., \beta_m) \ , \\ c_i(t_0) &= c_i^0 \ , \qquad i = 1 \cdots n \end{aligned} \tag{1.1}$$

where $\beta_j,\ j = 1, ..., m$ are the parameters of the system (for example, reaction rate constants, etc).

For example, if $c_i(t)$ is the concentration of the i^{th} species, the kinetics of a chemical system is described as an initial value problem in matrix production - destruction form:

$$\frac{dc(t)}{dt} = P(c(t)) - D(c(t)) \cdot c(t) \tag{1.2}$$

where $P \in \Re^n,\ D \in \Re^{n \times n},\ D = diag(D_i)$ are the production and the destruction terms, respectively.

In many scientific applications it is of interest to calculate the derivatives of the solution of (1.1) with respect to different parameters. Such derivatives depend on time and are called *sensitivity coefficients*.

The *local sensitivity coefficients* are defined as:

$$s_{i,j}(t) = \frac{\partial c_i}{\partial \alpha_j} \tag{1.3}$$

where α_j represent either the initial values c_j^0 or the parameter β_j. The term *local*

refers to the fact that these sensitivities describe the system around a given set of values for the parameters α. The system being considered to respond linearly for small perturbations, $s_{i,j}$ measures the ratio between the effect (*absolute* variation of the output Δc_i) and the cause (*absolute* variation of the input $\Delta \alpha_j$).

It is sometimes useful to consider the ratio between the *relative* variation of the output and the *relative* variation of the input:

$$s^*_{i,j} = \frac{\alpha_j}{c_i} \cdot s_{i,j} = \frac{\partial \ln(c_i)}{\partial \ln(\alpha_j)} \approx \frac{\Delta c_i}{c_i} \left(\frac{\Delta \alpha_j}{\alpha_j} \right)^{-1}$$

where $s^*_{i,j}$ is the *normalized sensitivity coefficient*. The normalized coefficients have the advantage of being adimensional, hence useful when comparing sensitivities with respect to parameters whose absolute values are orders of magnitude apart. If $\Delta \alpha_j = \alpha_j$ then $\alpha'_j = \alpha_j + \Delta \alpha_j = 2\alpha_j$ and we have the following interpretation: $s^*_{i,j}$ is the relative variation in c_i when the parameter α_j doubles its value, if the system responded linearly.

The disadvantage of the normalized coefficients is that they are not defined for $c_i = 0$. To overcome this drawback, one may consider the *semi-normalised sensitivities* $s^+_{i,j}$, representing the absolute variation in c_i when α_j doubles its value (if the system responded linearly):

$$s^+_{i,j} = \alpha_j \cdot s_{i,j} = \frac{\partial c_i}{\partial \ln(\alpha_j)} \approx \Delta c_i \left(\frac{\Delta \alpha_j}{\alpha_j} \right)^{-1}$$

1.2 Computational methods for sensitivity analysis

There are many ways to compute sensitivities. In this paper we employ three different methods to compute the local sensitivity coefficients - namely indirect, direct and forward mode automatic differentiation.

Conceptually, all three are equivalent, in the sense that a small perturbation

of *a certain input* is propagated forward through the system, and the corresponding deviation of *all outputs* is estimated. Thus, all the methods described below may be called forward propagation methods. They are effective when the sensitivity of all (or many) outputs with respect to one (or few) entries are desired.

1.2.1 Indirect Method ("Brute-Force" Approach)

Equation (1.1) is solved for two sets of parameters $\alpha_1, ..., \tilde{\alpha_j}, ..., \alpha_m$ and $\alpha_1, ..., \bar{\alpha_j}, ..., \alpha_m$, and the obtained outputs are $\tilde{c}(t)$ and $\bar{c}(t)$, respectively.

- One-sided difference approach. If $\bar{\alpha_j} = \alpha_j$ and $\tilde{\alpha_j} = \alpha_j + \epsilon$ the following is an approximation of the local sensitivity at the point $\alpha_1, ..., \alpha_j, ..., \alpha_m$:
$$\frac{\partial c_i}{\partial \alpha_j}(t) \approx \frac{\tilde{c_i}(t) - \bar{c_i}(t)}{\epsilon}$$
If the dependence $c(\alpha)$ is smooth enough, then this is a first order approximation of the sensitivity coefficient.

- Central difference approach. If $\tilde{\alpha_j} = \alpha_j + \epsilon$ and $\bar{\alpha_j} = \alpha_j - \epsilon$ the following is an approximation of the local sensitivity at the point $\alpha_1, ..., \alpha_j, ..., \alpha_m$:
$$\frac{\partial c_i}{\partial \alpha_j}(t) \approx \frac{\tilde{c_i}(t) - \bar{c_i}(t)}{2\epsilon}$$
If the dependence $c(\alpha)$ is sufficiently smooth, then this is a second order approximation of the sensitivity coefficient.

The "Brute-Force" approach requires only one (for one-sided differences) or two (for central differences) extra function evaluations for each independent variable with respect to which sensitivities are desired. The main drawback is that the accuracy of the method is hard to analyze. The smaller the perturbation ϵ, the lower the truncation error resulting from the omission of higher order terms (see

the expansion of finite difference formulas in Taylor series), but the higher the loss-of-significance errors, resulting from subtracting two almost equal numbers. At the very best, the brute force approach results in a sensitivity approximation that has half the significant digits of f.

1.2.2 Direct Method (Variational Equations)

By differentiating (1.1) with respect to the vector of parameters we obtain the variational equations:

$$\frac{d}{dt}\nabla c(t) \doteq f_c(t, c, \beta)\nabla c + \nabla f(t, c, \beta) \tag{1.4a}$$

Equivalently (1.2) gives

$$\frac{d}{dt}\nabla c_i(t) \doteq \nabla P^i(c) - \nabla D^i(c) \cdot c_i - D^i \cdot \nabla c_i, \;\; i = 1, ..., n \tag{1.4b}$$

The fact that the sensitivities satisfy (1.4b) can be proved rigorously, see for example [2]. The notation ∇A stands for the sensitivity coefficients vector $\frac{\partial A}{\partial \alpha}$, and " $\doteq$ " represents a vector (element-by-element) assignment.

To obtain $\nabla c_i(t)$, one has to numerically integrate the large system obtained by appending together (1.1) and (1.4b). This method is usually referred as *the direct approach*. The initial values $\nabla c_i(0)$ must be set properly (if $\nabla x = \frac{\partial x}{\partial c_i}$ then $\nabla c_i(0) = 1$, otherwise $\nabla c_i(0) = 0$). There are two main drawbacks of this approach:

- The generation of the variational equations requires significant extra effort;
- The integration of the large appended system may be very time consuming.

1.2.3 Green's Function Method

This method is based on the analytical solution of equation (1.4a). From the solution of (1.1) one can construct the "Green matrix"

$$G(t_1, t_2) = \exp\left(\int_{t_2}^{t_1} f_c(s, c(s))ds\right)$$

The solution of (1.4a) can be expressed as

$$\nabla c(t) = G(t, t_0)\nabla c(t_0) + \int_{t_0}^{t} G(t, s)\nabla f(s, c(s))ds$$

Thus the computation of any sensitivity coefficients is reduced to the computation of $G(t, s)$. In other words once $G(t, s)$ was calculated any number of sensitivity coefficients can be easily determined; if a large number of such coefficients is needed then the method may be efficient. We will not pursue further Green's function method in this thesis.

1.2.4 Adjoint Equations Method

In order to fix the ideas suppose we want to calculate sensitivities with respect to inital values (this is no restriction of generality since sensitivities with respect to any parameters can be viewed as sensitivities with respect to initial values for an enlarged system). Equation (1.4a) propagates the derivatives of $c(t)$ w.r.t. $c(t_0)$ (namely $\nabla c(t)$) forward in time from t_0 to t_f; the results is $\partial c(t_f)/\partial c(t_0)$.

The adjoint approach is to consider the quantities $\Delta c(t) = \partial c(t_f)/\partial c(t)$ and to propagate them backward in time from t_f to t_0.

Supposing the flow of (1.1) is reversible we can invert the time by introducing $\tau(t) = t_f + t_0 - t$. The change of variables $t \to \tau$ in (1.1) gives

$$\begin{aligned} y'(\tau) &= -f(\tau, y(\tau)) \\ y(t_0) &= c(t_f) \end{aligned}$$

The sensitivities of $y(\tau)$ w.r.t. $y(t_0)$ are the solutions of the variational equation

$$\begin{aligned} \nabla y(\tau)' &= -f_y(\tau, y(\tau))\nabla y(\tau) \\ \nabla y(t_0) &= I \end{aligned}$$

which, by coming back to $c(t)$ is the adjoint equation

$$\begin{aligned} \Delta c(t)' &= -f_c(t, c(t))\Delta c(t) \\ \Delta c(t_f) &= I \end{aligned}$$

Integrating this system backwards gives $\Delta c(t_0)$ which are precisely the quantities of interest $\partial c(t_f)/\partial c(t_0)$.

1.2.5 Automatic Differentiation

Automatic differentiation (reffered throughout the thesis as AD). techniques are based on the fact that any function (regardless of its complexity) is executed on a computer as a well-determined sequence of elementary operations like additions, multiplications and calls to elementary (intrinsic) functions such as sin, cos, etc.

By repeatidly applying the chain rule:

$$\frac{\partial}{\partial t} f(g(t))|_{t=t_0} = \left(\frac{\partial f(s)}{\partial s}|_{s=g(t_0)}\right) \cdot \left(\frac{\partial g(t)}{\partial t}|_{t=t_0}\right)$$

to the composition of these elementary operations one can compute, completely automatically, derivatives of f that are correct up to machine precision.

According to how the chain rule is used to propagate derivatives through the computation, one can distinguish two approaches to AD: the "forward" and the reverse "modes" (see [7], [8] for a detailed discussion).

- The Forward Mode is similar to the way in which the chain rule of differential calculus is usually taught. At each computational stage derivatives of the

intermediate variables with respect to input variables are computed. These derivatives are propagated forward through the computational stages. From now on we will refer to forward automatic differentiation as FAD.

- The Reverse Mode computes at each step adjoint quantities - the derivatives of the final result with respect to intermediate variables. To propagate adjoints, one has to be able to reverse the flow of a program, and remember or recompute any intermediate value that nonlinearly impacts the final result.

1.3 Thesis organization

The thesis is organized as follows.

In chapter 2 the principles of automatic differentiation are presented. The Adifor2.0 system, which received the 1995 Wilkinson prize for numerical software, is also briefly introduced.

Chapter 3 develops the theory for computing sensitivities of the solutions of ordinary differential equations (ODE) via automatic differentiation. Both black box approach and direct decoupled method (DDM) are studied. It is shown that the most efficient algorithms are not obtained by black box differentiation, but by a selective application of Adifor plus DDM integration.

In chapter 4 the theory is applied to a comprehensive, real-life chemical mechanism used in the study of tropospheric pollution. Sensitivities with respect to initial values, rate coefficients, temperature, emission rates etc. are computed and tabulated. A qualiative discussion of the results shows the importance and effectiveness of sensitivity analysis in air quality modeling.

Chapter 5 draws conclusions and pinpoints further research directions.

Finally, the chemical mechanism and some of the test codes used for exemplifying the theoretical conclusions are presented in the appendices.

CHAPTER 2
AUTOMATIC DIFFERENTIATION

2.1 Automatic differentiation in FORTRAN

A promising new implementation developed at Argonne National Laboratory and Rice University over the last couple of years and which has recently been awarded the 1995 Wilkinson Prize for Numerical Software is the package ADIFOR (Automatic Differentiation in FORTRAN, see [8]). It adopts a hybrid approach to computing derivatives that is generally based on the forward mode, but uses the reverse mode to compute the derivatives of assignment statements containing complicated expressions. The forward mode acts similarly to the usual application of the chain rule in calculus. At each computational stage derivatives of the intermediate variables with respect to input variables are computed and are propagated forward through the computational stages. On the other hand the reverse mode is based on adjoint quantities representing derivatives of the output with respect to intermediate variables. The adjoints are computed at each node of the computational graph and they are propagated by reversing the flow of the program and by recomputing intermediate values that have a nonlinear impact on the output. It turns out that this approach is related to the adjoint sensitivity analysis used in such various fields as nuclear engineering (see [13]), weather forecast (see [61]) and neural networks (see [93]). Because assignment statements compute generally one dependent variable in terms of several dependent variables, the reverse mode is very

efficient and can be implemented as in-line code.

2.2 Benefits of automatic differentiation

The ADIFOR (Automatic Differentiation in FORtran) system provides automatic differentiation for programs written in FORTRAN 77. According to its authors [8] the ADIFOR approach provides four benefits:

- Ease of use. The user only supplies the source code and indicates the independent and dependent variables.
- Portability. ADIFOR produces FORTRAN 77 code.
- Efficiency. ADIFOR-generated derivative code usually outperforms divided-difference approximations.
- Extensibility. ADIFOR employs a consistent subroutine-naming scheme.

ADIFOR requires the user to supply the FORTRAN source code for the function value and for all lower level subroutines as well as a list of the independent and dependent variables in the form of parameter lists or common blocks. ADIFOR then determines which other variables throughout the programs are to be differentiated, and augments the original code with derivative statements. The augmented code is then optimized by eliminating unnecessary operations and temporary variables. The FORTRAN code generated by ADIFOR requires no run-time support and therefore can be ported between different computing environments.

2.3 Automatic differentiation

as a source transformation

Traditionally two modes of AD (automatic differentiation) have been developed: the forward and the reverse modes. The forward mode accumulates the derivatives of intermediate variables w.r.t. independent variables, whereas the reverse mode propagates the derivatives of the final values with respect to intermediate variables. In either case automatic differentiation produces code that (in the absence of floating point exceptions) computes the values of the analytical derivatives accurate up to machine precision.

We illustrate the differences on the following fragment of code (example taken from [8]). We need to compute the derivatives $\partial y(1:2)/\partial x(1:n)$.

```
y(1) = 1.0
y(2) = 1.0
do i=1,n
    if (x(i) > 0.0) then
       y(1) = x(i)*y(1)**2
    else
       y(2) = x(i)*y(2)**2
    endif
enddo
```

2.3.1 The forward mode

To apply automatic differentiation the code is first rewritten such that only elementary unary and binary operations appear:

```
y(1) = 1.0
```

```
y(2) = 1.0
do i=1,n
    if (x(i) > 0.0) then
       temp = x(i)*y(1)
       y(1) = temp*y(1)
    else
       temp = x(i)*y(2)
       y(2) = temp*y(2)
    endif
enddo
```

The are n independent variables, so each derivative will be a n-dimensional vector. The results is the matrix g_y (the "gradient" of y):

$$g\_y = \begin{bmatrix} \frac{\partial y(1)}{\partial x(1)} & \cdots & \frac{\partial y(1)}{\partial x(n)} \\ \frac{\partial y(2)}{\partial x(1)} & \cdots & \frac{\partial y(2)}{\partial x(n)} \end{bmatrix}$$

The derivatives of $x(1:n)$ (which form a $n \times n$ matrix) are initialized to the identity matrix. All other derivatives are initialized to zero. The generated code is

```
y(1) = 1.0              ! original initial value
do j=1,n
  g_y(1,j) = 0.0        ! derivatives of y(1)
end do
y(2) = 1.0
do j=1,n
  g_y(2,j) = 0.0        ! derivatives of y(2)
end do
C Initialize the derivatives of x
```

```
do i=1,n
  do j=1,n
    if (i.eq.j) then
      g_x(i,j) = 1.0
    else
      g_x(i,j) = 0.0
    end if
  end do
end do


do i=1,n
  if (x(i) > 0.0) then
    do j=1,n                                        ! inserted code
      g_temp(j) = g_x(i,j)*y(1) + x(i)*g_y(1,j)  ! for derivatives
    end do                                          ! of temp
    temp = x(i)*y(1)
    do j=1,n                                        ! inserted code
      g_y(1,j) = g_temp(j)*y(1) + temp*g_y(1,j)  ! for derivatives
    end do                                          ! of y(1)
    y(1) = temp*y(1)
  else
    do j=1,n
      g_temp(j) = g_x(i,j)*y(2) + x(i)*g_y(2,j)
    end do
    temp = x(i)*y(2)
```

```
    do j=1,n
      g_y(2,j) = g_temp(j)*y(2) + temp*g_y(2,j)
    end do
    y(2) = temp*y(2)
  endif
enddo
```

2.3.2 The reverse mode

The reverse mode of AD computes adjoints - the derivatives of the final result with respect to an intermediate quantity.

Suppose that x is the independent variable, y is the result of some computations and v, w, s are intermediate quantities. Let $\bar{\xi} = \partial y / \partial \xi$ denote the adjoint of the variable ξ. At some point in the program we have [8]

$$s = f(v, w)$$

Then

$$\frac{\partial y}{\partial v} = \frac{\partial y}{\partial s}\frac{\partial s}{\partial v}$$

which (considering the fact that y depends on v through multiple computational channels) leads to the generated code

$$\bar{v} += \frac{\partial s}{\partial v}\bar{s}$$

Since $\partial s / \partial v$ is (in general) a function of some arguments, one must remember the values of the operands in order to be able to compute it. Thus, in order to apply reverse mode AD one needs to reverse the flow of the program, must remember intermediate values that were overwritten and trace how branches were taken.

The example code prepared for reverse AD and the transformed code are

described in detail in [8]. Here we will just note that by the linearity of differentiation the adjoints are related as

$$\left[\bar{x(1)}, \ldots, \bar{x(n)}\right] = \left[\bar{y(1)}, \bar{y(2)}\right] \begin{bmatrix} \frac{\partial y(1)}{\partial x(1)} & \cdots & \frac{\partial y(1)}{\partial x(n)} \\ \frac{\partial y(2)}{\partial x(1)} & \cdots & \frac{\partial y(2)}{\partial x(n)} \end{bmatrix}$$

Reverse mode code allows to compute arbitrary linear combinations of the rows of the Jacobian. If $\bar{y}$ is initialized to d then the result of the generated code is

$$\frac{\partial d^T y(x)}{\partial x}$$

It is a much more involved process to generate reverse mode code. Extra memory is required (in the first case as many locations as there are floating point operations in the original source). However the running time is about m times that of the function when computing m linear combinations of the rows of the Jacobian.

2.3.3 The hybrid approach

ADIFOR employs a hybrid approach to AD, namely for each statement it accumulates the partial derivatives of the variable on the LHS with respect to the RHS and then applies the forward mode to propagate the total derivatives according to the chain rule. Instead of simply producing the Jacobian J ADIFOR computes $J * S$ where S is the "seed matrix". The cost is proportional to the number of columns of S that are computed in one run. Typically ADIFOR generated code runs 2-4 times faster than one-sided divided differences when more than 5-10 derivatives are computed at one time.

2.4 The ADIFOR2.0 system

Adifor2.0 system has three major components:

- Preprocessor. Parses the code, performs certain node normalizations, determines which variables have to be augmented with derivative objects and generates derivative code with templates at call sites of F77 intrinsics and calls SparseLinC routines.

- ADIntrinsics system. Expands calls to Fortran 77 intrinsic templates to Fortran 77 code guided by a template library defining how each intrinsic is to be translated.

- SparseLinC library. Provides transparent support of sparsity in derivative computations.

The relations among these components is shown in figure 2.1 (from [8]).

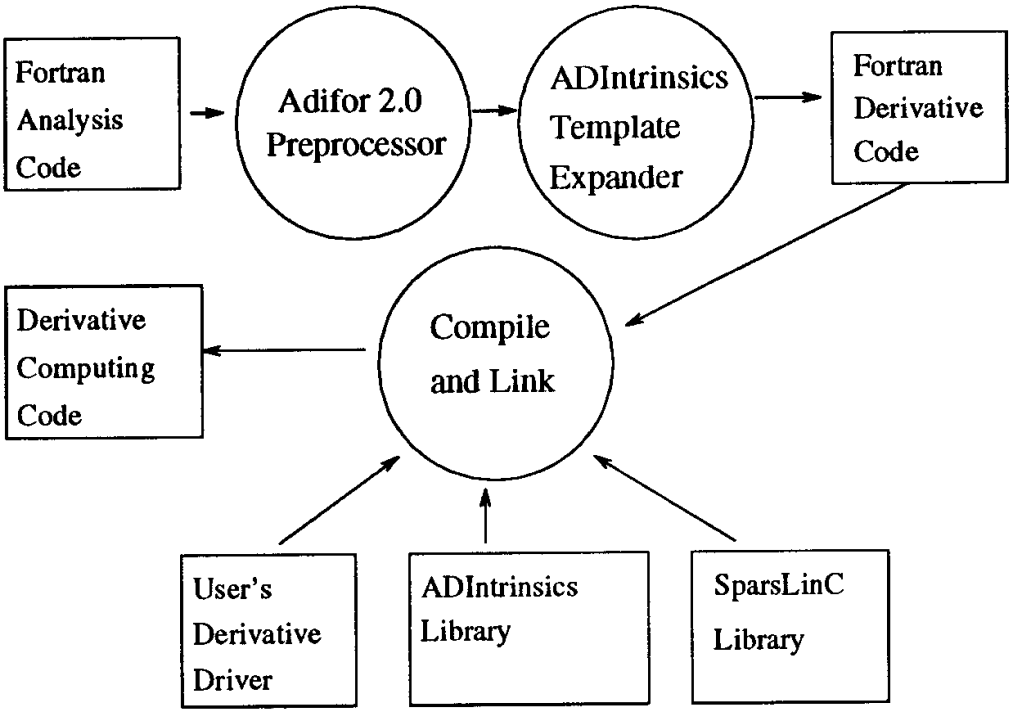


Figure 2.1. Overview of Adifor2.0 system (from [8]).

CHAPTER 3
THEORY

3.1 Applying automatic differentiation on ODE integrators

The numerical simulation of the chemical transformations consists in integrating (advancing in time) a system of coupled, stiff ordinary differential equations that models the chemical rate equations. It is of interest to ask how automatic differentiation can be used to obtain sensitivities of concentrations with respect to different parameters, given the rate equations and a numerical routine that solves them. The simplest approach would be to apply AD directly on the numerical integrator; we will call this "the black box approach". A more refined technique, which necessitates the expert (user) intervention, would be to generate the sensitivity (variational) equations via automatic differentiation, then to integrate them using a user-defined technique, for example employing the direct decoupled method. Figure 3.1 gives a visual description of these two possibilities. In short, black box FAD computes the sensitivities of the numerical solution, while the variational approach gives a numerical solution to the sensitivity equations.

The black box approach is simpler, while the variational equations approach is more flexible. In the next sections we will investigate both approaches.

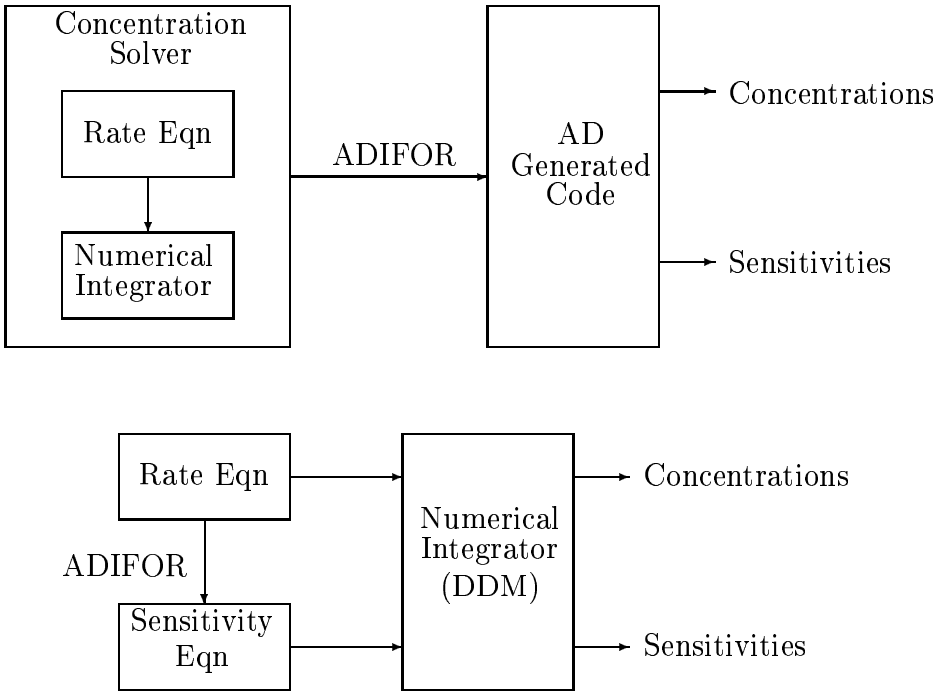


Figure 3.1. Alternative ways of using automatic differentiation for calculating chemical sensitivities. The black box approach (upper figure) is completely automatical, while the variational equation approach (lower figure) requires user intervention.

3.1.1 One example

Consider the following system (the "Brusselator" from [45], modeling a chemical system with a limit cycle)

$$\begin{cases} y_1' &= 1 + y_1^2 y_2 - 4y_1 \\ y_2' &= 3y_1 - y_1^2 y_2 \\ y_1(t_0) &= 1 \\ y_2(t_0) &= 1 \end{cases} \tag{3.1}$$

We will denote by Ψ the derivatives with respect to initial values

$$\Psi(t) = \frac{\partial[y_1(t), y_2(t)]}{\partial[y_1(t_0), y_2(t_0)]}$$

Ψ satisfies the variational equations

$$\begin{cases} \Psi'(t) &= \begin{bmatrix} 2y_1y_2 - 4 & y_1^2 \\ 3 - 2y_1y_2 & -y_1^2 \end{bmatrix} \Psi(t) \\ \Psi(t_0) &= I \end{cases}$$

In figure 3.2 exact solutions are plotted, together with the absolute sensitivity coefficients Ψ computed by the direct method (i.e. the equations for y and Ψ were integrated *together* as a six-dimensional system). The numerical solver used is VODE with very tight tolerances $atol = rtol = 10^{-12}$.

Sensitivities were estimated using the indirect approach with different perturbations of the initial values ranging between $10^{-2} - -10^{-7}$ (recall that the initial values are equal to 1). Results are reported in figure 3.3; it shows that the value 10^{-4} gives optimal accuracy while for other perturbations the error is much larger. Thus, the results of the indirect method are influenced by the interplay between asymptotics and numerical errors; more exactly, the smaller the perturbation the better the asymptotic approximation of the sensitivities, but also the larger loss-of-significance numerical errors.

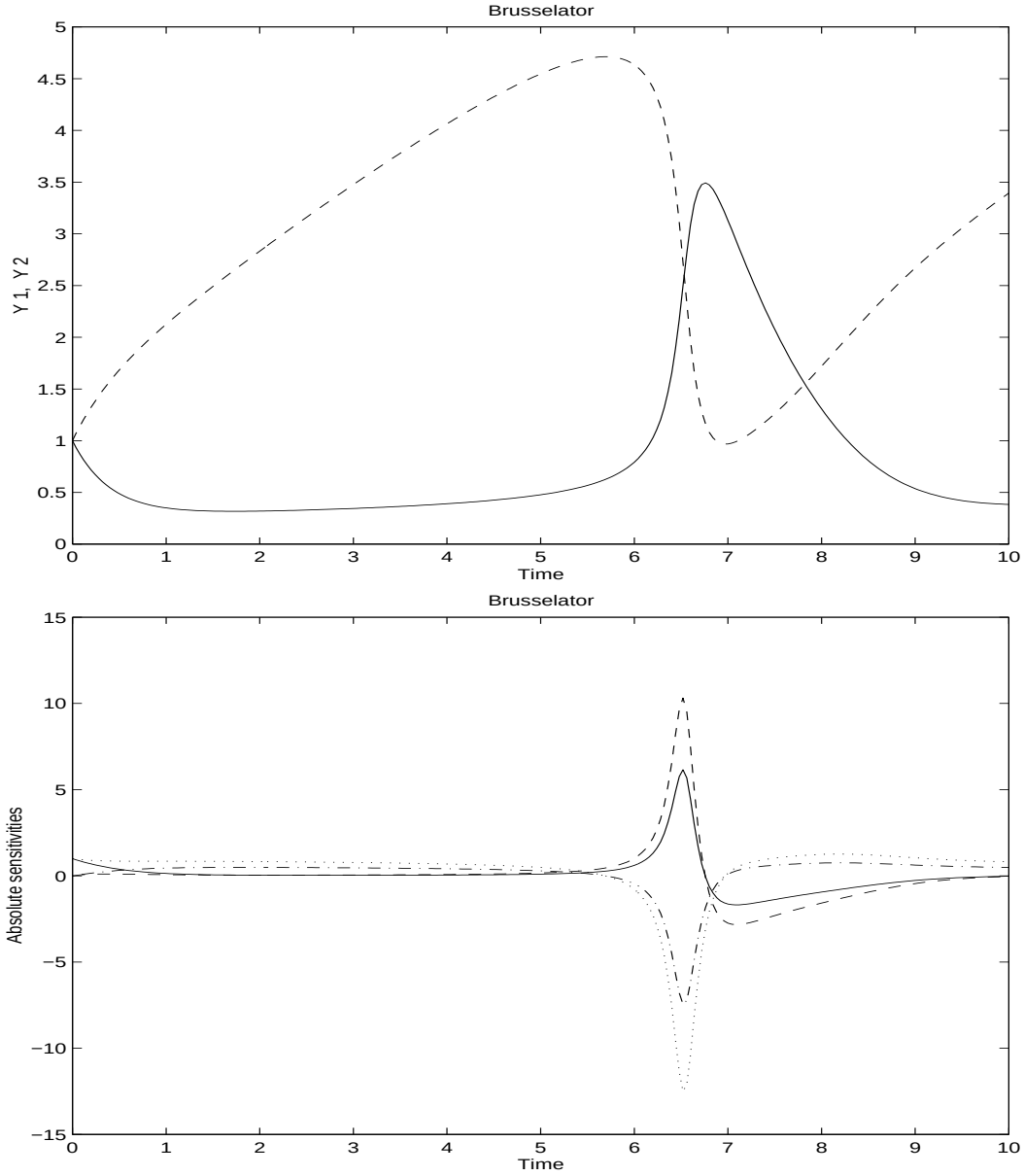


Figure 3.2. Exact solution of brusselator (upper figure); y_1 (solid) and y_2 (dashed). Exact absolute sensitivities (lower figure); the lines are $\partial y_1/\partial y_1^0$ (solid), $\partial y_1/\partial y_2^0$ (dashed), $\partial y_2/\partial y_1^0$ (dash-dotted), $\partial y_2/\partial y_2^0$ (dotted). Note that the peaks of the sensitivity w.r.t. initial values appear when the solution is changing most rapidly.

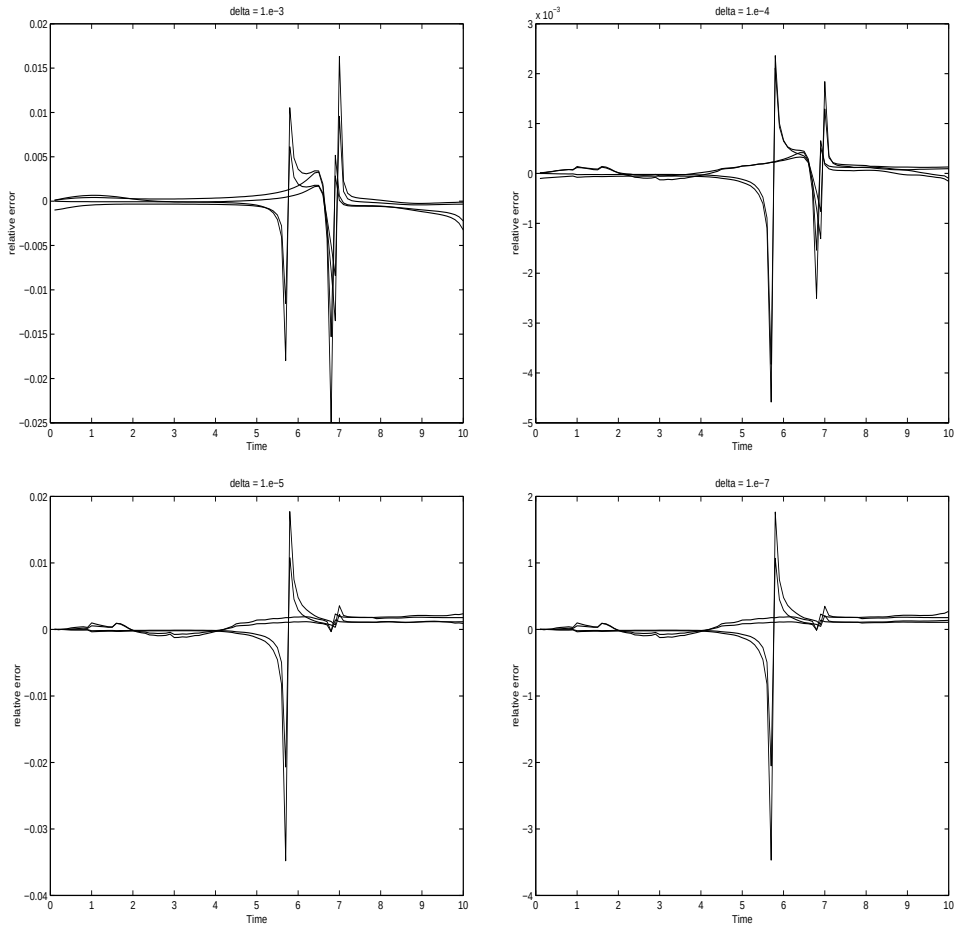


Figure 3.3. Relative errors for absolute sensitivity coefficients computed by finite differences. The perturbation of the initial values is 10^{-d} with $d = 3,4,5,7$ (from upper left to lower right). Smaller perturbations can increase the error.

3.1.2 Black-box approach

Given the initial value problem

$$c' = f(t, c, \alpha), \qquad c(t_0) = c_0 \tag{3.2}$$

one would like to numerically compute the solution in time $c(t)$ and also to find the sensitivity of this solution with respect to the parameters $\alpha \in \Re^m$. Note that with the transformation:

$$z = c - c_0 \ ,$$

the system (3.2) can be rewritten as

$$z' = f(t, z + c_0) = \tilde{f}(t, z, c_0), \qquad z(t_0) = 0 \ . \tag{3.3}$$

In other words the dependence of the solution w.r.t. the initial values can be reduced to the dependence of solution w.r.t. a parameter.

Similarly the dependence of the solution of (3.2) w.r.t. the parameter α can be reduced to a dependence of initial values by adding differential equations for α

$$\begin{cases} c' &= f(t, c, a) \ , & c(t_0) = c_0 \\ a' &= 0 \ , & a(t_0) = \alpha \ . \end{cases}$$

Since the derivatives of the solution w.r.t. the initial values and w.r.t. parameters are conceptually the same thing we will consider in what follows that the system depends explicitly on α.

Suppose for the sake of discussion that we solve (3.2) numerically with a one-step, first order consistent method, expressed in Henrici notation as:

$$c_{n+1} = c_n + h_n \cdot \Phi(t_n, c_n, c_{n+1}, \alpha, h_n) \tag{3.4}$$

where $c_k = c(t_k) = c(t_0 + \sum_i h_i)$ and Φ is supposed to be smooth. The method is applied with a predefined selection of the stepsize h_i (either using a constant step

or a prescribed step – for example, take smaller steps during sunrise and sundown intervals; it is essential that h does not depend on c, as is the case with inline step controllers, typical for all popular variable step-size codes). The consistency of the method implies that $\Phi(t, c, c, \alpha, 0) = f(t, c, \alpha)$ and hence, since Φ is supposed to be smooth in all its arguments we have:

$$\begin{aligned} \lim_{h\to 0} \nabla\Phi(t, c_n, c_{n+1}, \alpha, h) &\doteq \nabla \lim_{h\to 0} \Phi(t, c_n, c_{n+1}, \alpha, h) \\ &\doteq \nabla\Phi(t, c_n, c_n, \alpha, 0) \\ &\doteq f_c(t, c_n, \alpha) \cdot \nabla c_n + f_\alpha(t, c_n, \alpha) \end{aligned} \tag{3.5}$$

The sensitivities $\nabla c = \partial c/\partial\alpha$ obey the variational equations:

$$\begin{aligned} \frac{d}{dt}\nabla c &\doteq f_c(t, c, \alpha) \cdot \nabla c + f_\alpha(t, c, \alpha) \\ &\doteq \nabla\Phi(t, c, c, \alpha, 0) \end{aligned} \tag{3.6}$$

Forward automatic differentiation applied on (3.4) will generate:

$$\begin{aligned} \nabla c_{n+1} \doteq\ & \nabla c_n + h_n \cdot \Phi_{c_n}(t_n, c_n, c_{n+1}, \alpha, h_n) \cdot \nabla c_n \\ & + h_n \cdot \Phi_{c_{n+1}}(t_n, c_n, c_{n+1}, \alpha, h_n) \cdot \nabla c_{n+1} \\ & + h_n \cdot \Phi_\alpha(t_n, c_n, c_{n+1}, \alpha, h_n) \end{aligned} \tag{3.7}$$

The idea is to consider (3.7) a numerical method applied to (3.6). By developing (3.7) in power series over h_n we get

$$\begin{aligned} \nabla c_{n+1} &\doteq \nabla c_n + h_n \cdot (\Phi_{c_n} + \Phi_{c_{n+1}})(t_n, c_n, c_n, \alpha, 0) \cdot \nabla c_n + \\ &\quad h_n \cdot \Phi_\alpha(t_n, c_n, c_n, \alpha, 0) + O(h_n^2) \\ &\doteq \nabla c_n + h_n \nabla\Phi(t_n, c_n, c_n, \alpha, 0) + O(h_n^2) \\ &\doteq \nabla c_n + h_n \cdot \frac{d}{dt}\nabla c_n + O(h_n^2) \end{aligned}$$

Hence by forward automatic differentiating (3.4) we obtain (3.7), a first order consistent method for solving the variational equation.

The above remark leads us to the following conclusion: Black box automatic differentiation implicitly generates the sensitivity equations. More exactly, suppose we start with a code that solves (3.2) using a numerical method A. The time discretization A of the initial equation (3.2) is transformed by black box FAD into a time discretization of the variational equation (3.6). This represents a new integration method B. We say that B is the derivative of the method A. If the method B is convergent, then the resulting code will be able to consistently calculate sensitivities. Thus, the black box FAD is equivalent to the direct method, since the resulting code applies a numerical integrator (B) to the variational equations (3.6).

In the above we proved that, if the method A is consistent with (3.2) then the differentiated method B is consistent with (3.6).

In addition, the accuracy of the computed sensitivity coefficients will be given by the accuracy of the method B (which integrates the variational equations); this in its turn depends on the method A; hence, although ADIFOR generates derivatives accurate up to machine precision, the accuracy of the sensitivities will be limited by the accuracy of the underlying numerical method.

As a numerical illustration, we have computed the errors in the numerical solution obtained with Merson's method and the sensitivities obtained by applying ADIFOR on Merson's integrator. Results are presented in figure 3.4. The reference solution is given by VODE, applied on the variational equations with very tight tolerances ($atol = rtol = 10^{-12}$). The relative errors in the numerical solution are of the order 10^{-4}, while the relative errors in the sensitivity coefficients are of the order 10^{-3}. This illustrates our conclusion that the accuracy of the sensitivity

coefficients depends on the underlying numerical method.

3.1.2.1 Variable step size algorithms

If the algorithm uses an error controller that adjusts the stepsize based on local error estimates, then h_n depends on c (and hence on α) and (3.7) becomes:

$$\begin{aligned}\nabla c_{n+1} &= \nabla c_n + h_n \cdot \Phi_{c_n}(t_n, c_n, c_{n+1}, \alpha, h_n) \cdot \nabla c_n \\ &\quad + h_n \cdot \Phi_{c_{n+1}}(t_n, c_n, c_{n+1}, \alpha, h_n) \cdot \nabla c_{n+1} + h_n \cdot \Phi_\alpha(t_n, c_n, \alpha, h_n) \\ &\quad + (h_n \cdot \Phi_h(t_n, c_n, \alpha, h_n) + \Phi(t_n, c_n, \alpha, h_n)) \cdot \nabla h_n\end{aligned}$$

The last term, and the eventual stepsize rejections may create problems when the algorithm is automatically differentiated. This is so because of the extra term implying the sensitivity of the stepsize. Consider for example the asymptotic step controller:

$$\begin{aligned}h_n &= h_{n-1} \cdot \left(\frac{||c_n - \hat{c_n}||}{tol} \right)^{\frac{1}{p+1}} \\ \nabla h_n &= \nabla h_{n-1} \cdot \left(\frac{||c_n - \hat{c_n}||}{tol} \right)^{\frac{1}{p+1}} \\ &\quad + \frac{h_{n-1} \cdot tol^{-1/p+1}}{p+1} \cdot (||c_n - \hat{c_n}||)^{-\frac{p}{p+1}} \cdot \nabla(||c_n - \hat{c_n}||)\end{aligned}$$

Numerical experiments with the variable step-size Merson's method applied on brusselator did not reveal this problem. In fact, setting $\nabla h = 0$ at each step increased the numerical error in the sensitivities. I do not understand at this point why it is so.

3.1.2.2 Implicit methods

If an implicit method is used for solving (3.2) then at each time step a nonlinear system of algebraic equations has to be solved. This is usually done by simplified

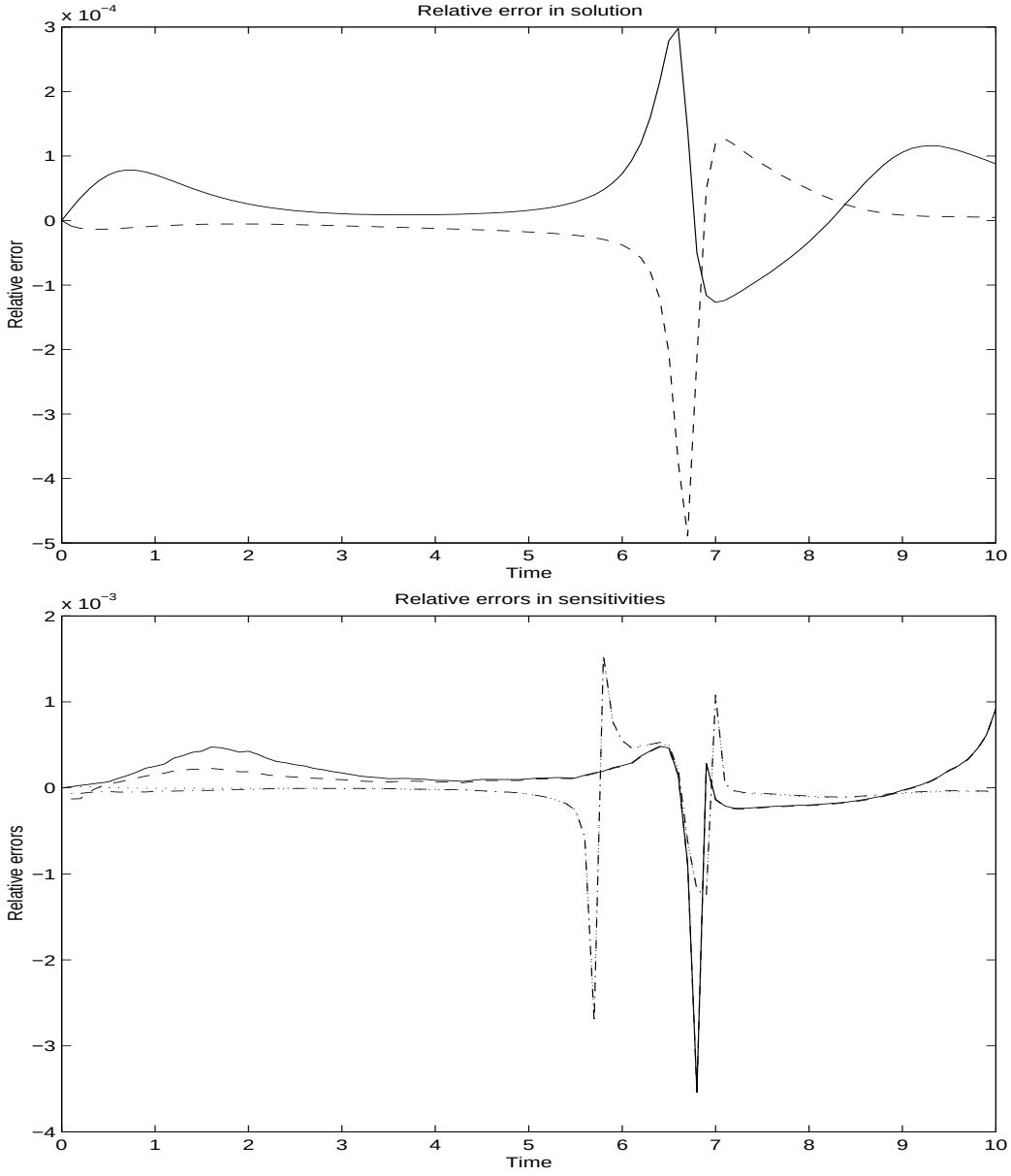


Figure 3.4. Relative errors for the solution of brusselator (upper figure) and for absolute sensitivity coefficients computed by Adifor2.0 (lower figure); the lines are $\partial y_1/\partial y_1^0$ (solid), $\partial y_1/\partial y_2^0$ (dashed), $\partial y_2/\partial y_1^0$ (dash-dotted), $\partial y_2/\partial y_2^0$ (dotted). This illustrates the fact that the accuracy of the sensitivity coeficients depends on the numerical method.

Newton iterations.

Black box FAD will transform the nonlinear system in concentrations into a nonlinear system in concentrations and sensitivities, and will add to the iterative solution for concentrations an iterative solution for sensitivities. For example consider the fixed step-size method

$$
\begin{aligned}
c_{n+1} &= c_n + h\Phi(t, c_{n-k}, ..., c_{n+1}, \alpha, h) && \text{(3.8a)}\\
\nabla c_{n+1} &\doteq \nabla c_n + h \sum_{i=-1}^{i=k} \Phi_{c_{n-i}}(t, c_{n-k}, ..., c_{n+1}, \alpha, h) \cdot \nabla c_{n-i} && \text{(3.8b)}\\
&\quad + h\Phi_\alpha(t, c_{n-k}, ..., c_{n+1}, \alpha, h)
\end{aligned}
$$

Both formulas (3.8a-3.8b) are implicit since they involve $\Phi(..., c_{n+1})$ and $\Phi_c(c_{n+1})$, respectively. The code generated by black box FAD will consist of an iterative solution of (3.8a) and (3.8a) coupled together. An implicit solution of (3.8a-3.8b) requires the factorization of the matrix

$$
\begin{bmatrix}
I_n - h\Phi_{c_{n+1}} & 0 & 0 & \cdots & 0 \\
-\Phi_{c_{n+1},c_{n+1}}(\nabla c)_1 - \Phi_{c_{n+1},\alpha_1} & \left(I_n - h\Phi_{c_{n+1}}\right) & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
-\Phi_{c_{n+1},c_{n+1}}(\nabla c)_m - \Phi_{c_{n+1},\alpha_m} & 0 & 0 & \cdots & \left(I_n - h\Phi_{c_{n+1}}\right)
\end{bmatrix} \qquad (3.9)
$$

(where n is the dimension of the initial system and m is the number of coefficients to be calculated). Solved directly, this factorization requires $O\left(n^3(m+1)^3\right)$ operations. Of course, the inverse of (3.9) involves only the matrix

$$\left(I_n - h\Phi_{c_{n+1}}\right)^{-1}$$

so the factorization should really require only $O\left((m+1)n^3\right)$ operations.

If (3.8a) is solved first, and the obtained value for c_{n+1} is then used in (3.8b), this becomes a linear system in ∇c_{n+1}; hence the sensitivities can be obtained with

only one matrix decomposition (of $I - h\Phi_{c_{n+1}}$ for all α_i) and without any iterations (the Direct Decoupled Method from [34]).

To fix the ideas, consider the model problem (with $y \in \Re^n$)

$$y' = f(y)$$

together with the variational equations

$$\Psi' = I_n \bigotimes J(y)\Psi$$

where $\Psi = dy(t)/dy(t_0)$ and $J = f_y$.

Implicit Euler discretization of the above equations reads

$$\begin{cases} y_{n+1} &= y_n + hf(y_{n+1}) \\ \Psi_{n+1} &= \Psi_n + hI_n \otimes J(y_{n+1})\Psi_{n+1} \end{cases}$$

The *direct decoupled method* solves the nonlinear equation in y_{n+1} first using quasi-Newton, then evaluates $J(y_{n+1})$ and solves the linear variational equation.

$$\left[\begin{array}{l} Factorize \quad S = I - hJ(y_n) \\ do \qquad (until \quad \|\Delta y\| \quad small \quad enough) \\ \qquad\quad S\Delta y = y_{n+1}^k - y_n^k - hf(y_{n+1}^k) \\ \qquad\quad y_{n+1}^{k+1} = y_{n+1}^k - \Delta y \\ \qquad\quad k = k + 1 \\ enddo \\ Factorize \quad S = I - hJ(y_{n+1}) \\ Solve \quad I_n \bigotimes S\Psi_{n+1} = \Psi_n \end{array}\right.$$

Note that the solution for Ψ_{n+1} requires only n (or in general m) backsubstitutions with the same matrix S; the decomposition of S will be used in the next step for solving the equation in y_{n+2}.

The *direct (coupled) method* will attempt to solve the big, nonlinear system together, iterating for both the concentrations and the sensitivities.

With FAD the following takes place. For a system

$$Ax = b$$

the sensitivities obey

$$A(\nabla x) = \nabla b - (\nabla A)x$$

The fact that both systems share the same matrix is exploited with DDM. On the other hand Adifor will generate code for each component of ∇x, which is equivalent to repeating the decomposition of A m times (in our example n times).

Thus, we may conclude that *black box FAD is more effective than the direct (coupled) method, but less effective than the direct decoupled method (DDM).* Formally the CPU times of the three methods obey the inequality

$$T_{Direct} \geq T_{Adifor} \geq T_{DDM} \ .$$

To illustrate this we have employed the Brusselator example discretized with the implicit Euler method. We considered the solution of the sensitivities by the Adifor generated code and using the direct decoupled method. The results are given in table 3.1. While with DDM the computation of the 4 sensitivity coefficients increases the CPU time by only 50%, with Adifor the CPU time increases linearly with the number of independent parameters (more than this in our small example).

Since the AD generated code uses iterations for the sensitivity coefficients also, the convergence of the sensitivity iterations adds to the problems mentioned above. The aspects of derivative convergence are thoroughly examined in [42]; it is shown that, if the concentration iterations are convergent, under natural hypothesis, the new sensitivity iterations are also convergent.

Table 3.1. Comparison of different methods for sensitivity calculation, brusselator example. Direct decoupled method (DDM) is notably faster than black box FAD. FAD time grows almost linearly with the number of input parameters.

Method	Mflops	CPU time	max Conc err	max Sen err
Impl Euler	24	3.6 sec	5E-4	–
Indirect	72	10.8 sec	–	4.E-3
Adifor	120	12.3 sec	5E-4	4E-3
DDM	34	4.8 sec	5E-4	5E-3

3.1.3 Stability

The equations and the variational equations share the same subset of eigenvalues. This can be seen from the fact that the variational system (with a one-dimensional parameter α and $\Psi = \partial y/\partial\alpha$)

$$\begin{aligned} y' &= f(y) \\ \Psi' &= f_y(y)\Psi + f_\alpha(y) \end{aligned}$$

has the Jacobian

$$\begin{bmatrix} f_y(y) & 0 \\ f_{yy}(y,\Psi) + f_{\alpha y}(y) & f_y(y) \end{bmatrix}.$$

Thus, it is reasonable to conjecture that, if the original method is stable, the differentiated method is also stable. This is certainly true for linear multistep and Runge Kutta methods which are invariant under differentiation, as we shall see shortly.

3.1.3.1 One step methods

Let A be a one-step method consistent of order p; its local error has an asymptotic expansion of the form:

$$c(t+h) - c(t) - h \cdot \Phi(t, c(t), h) = \sum_{i=p+1}^{N+1} d_i(t, \alpha) \cdot h^i + O\left(h^{N+2}\right)$$

By differentiating the above relation:

$$\nabla c(t+h) - \nabla c(t) - h \cdot \nabla\Phi(t, c(t), h) = \sum_{i=p+1}^{N+1} \nabla d_i(t, \alpha) \cdot h^i + O\left(h^{N+2}\right)$$

which shows that, if the d_i are smooth in α, the differentiated method B has the same order of consistency.

Also, if the global error of the method satisfies

$$c(t_0 + nh) - c_n = \sum_{i=p+1}^{N+1} \left(b_i(t, \alpha) + \beta_i\right) \cdot h^i + O\left(h^{N+2}\right)$$

then

$$\nabla c(t_0 + nh) - \nabla c_n\Phi(t, c(t), h) = \sum_{i=p+1}^{N+1} \nabla\left(b_i(t, \alpha) + \beta_i\right) \cdot h^i + O\left(h^{N+2}\right)$$

Thus if the initial method is convergent of order p and if the expansion coefficients b_i and the perturbations β_i are smooth in α, then the differentiated method converges with the same order.

The above considerations show that although automatic differentiation produces derivatives that are exact up to the machine precision, the accuracy of the sensitivities computed by the black box FAD depends on the accuracy of the underlying numerical integration method (in particular depends on its order).

3.1.3.2 Runge-Kutta methods

A s-stage Runge-Kutta method is defined as

$$y_1 \quad = \quad y_0 + \sum_{i=1}^{s} b_i k_i$$

$$k_i = hf\left(t_0 + c_i h,\ y_0 + \sum_{j=1}^{s} a_{ij} k_j,\ \alpha\right)$$

Applying FAD yields

$$\begin{aligned}
\nabla y_1 &= \nabla y_0 + \sum_{i=1}^{s} b_i \nabla k_i \\
\nabla k_i &= hf_y\left(t_0 + c_i h,\ y_0 + \sum_{j=1}^{s} a_{ij} k_j,\ \alpha\right)\left(\sum_{j=1}^{s} a_{ij} \nabla k_j\right) \\
&\quad + hf_\alpha\left(t_0 + c_i h,\ y_0 + \sum_{j=1}^{s} a_{ij} k_j,\ \alpha\right)
\end{aligned}$$

This relation is exactly the original Runge-Kutta method applied to the variational equations.

Two aspects of black-box FAD of RK methods are worth mentioning:

- The system in $k_i,\ \nabla k_i$ has dimension $s(m+1)n$;
- The error control of the resulting code will adjust the stepsize according to the error in the concentrations only; user intervention is needed to control the sensitivity errors also.

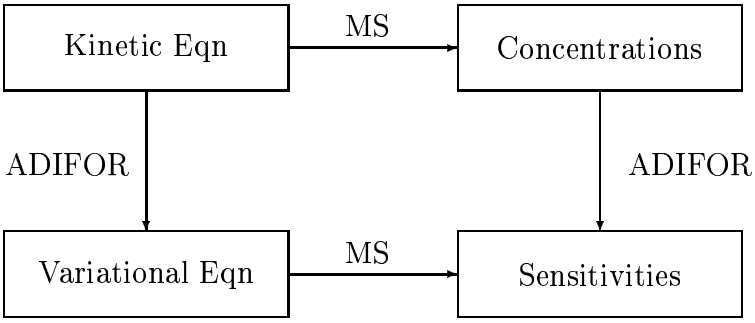
3.1.3.3 Linear Multistep methods

By differentiating the linear multistep method applied on the rate equation (3.2) one obtains the same method, applied on sensitivity equations (3.6):

$$\sum_{i=0}^{k} \alpha_i \cdot c_{n+i} = h \cdot \sum_{i=0}^{k} \beta_i \cdot f_{n+i} \tag{3.10a}$$

$$\sum_{i=0}^{k} \alpha_i \cdot \nabla c_{n+i} = h \cdot \sum_{i=0}^{k} \beta_i \cdot ((f_c)_{n+i} \nabla c_{n+i} + (f_\alpha)_{n+i}) \tag{3.10b}$$

Hence, the following diagram commutes for multistep methods (provided that sensitivity iterations converge if the multistep method is implicit).



3.1.4 Black box forward automatic differentiation of the QSSA method

As an illustration, we consider in detail the integration of (1.2) using the Quasi- Steady- State- Approximation (QSSA) technique (see [49]). This method solves (1.2) with the first order consistent formula:

$$c(t_0 + h) = e^{-D_0 \cdot h} \cdot \left(c_0 - D_0^{-1} \cdot P_0\right) + D_0^{-1} \cdot P_0 \tag{3.11}$$

where (3.11) is applied on the typical integration interval $[t_0, t_0 + h]$, $c_0 = c(t_0)$, $P_0 = P(c_0)$ and $D_0 = D(c_0)$. Because of the diagonal form of D, (3.11) can be decomposed componentwise, and the implementation of the QSSA step looks like this:

$$\begin{aligned} &\textit{call compute } (\textit{in}: \ t_0, \ \alpha, \ c_0; \ \textit{out}: \ P_0, \ D_0) \\ &\textit{do} \ \ i = 1, n \\ &\quad c_i(t_0 + h) = e^{-D_0^i \cdot h} \cdot \left(c_{0,i} - \frac{P_0^i}{D_0^i}\right) + \frac{P_0^i}{D_0^i} \\ &\textit{end do} \end{aligned} \tag{3.12}$$

The subroutine *compute* calculates the production and destruction terms at $t,\ c$.

From (3.12), the FAD will generate the algorithm:

call g_compute $(in:\ t_0,\ \alpha,\ c_0,\ \nabla c_0;\ out:\ P_0,\ D_0, \nabla P_0, \nabla D_0)$

do $i = 1, n$

$$c_i(t_0 + h) = e^{-D_0^i \cdot h} \cdot \left(c_{0,i} - \frac{P_0^i}{D_0^i} \right) + \frac{P_0^i}{D_0^i} \tag{3.13a}$$

$$\begin{aligned} \nabla c_i(t_0 + h) \doteq & -h \cdot \nabla D_0^i \cdot e^{-D_0^i h} \cdot \left(c_{0,i} - \frac{P_0^i}{D_0^i} \right) \\ & + e^{-D_0^i h} \cdot \left(\nabla c_{0,i} - \frac{\nabla P_0^i}{D_0^i} + \frac{P_0^i \cdot \nabla D_0^i}{(D_0^i)^2} \right) \\ & + \frac{\nabla P_0^i}{D_0^i} - \frac{P_0^i \cdot \nabla D_0^i}{(D_0^i)^2} \end{aligned} \tag{3.13b}$$

end do

Equation (3.13b) is a numerical method applied to (1.4b). From (3.13b) it is clear that:

$$\begin{aligned} \nabla c(h)|_{h=0} &= \nabla c_0 \\ \frac{d}{dh} \nabla c_i(t_0 + h)|_{h=0} &= -\nabla D_0^i \cdot \left(c_{0,i} - \frac{P_0^i}{D_0^i} \right) \\ & \quad - D_0^i \cdot \left(\nabla c_0 - \frac{\nabla P_0^i}{D_0^i} \right) + \frac{P_0^i \cdot \nabla D_0^i}{(D_0^i)^2} \\ &= -\nabla D_0^i \cdot c_{0,i} - D_0^i \cdot \nabla c_{0,i} + \nabla P_0^i \end{aligned}$$

and by comparing this with (1.4b) we conclude that (3.13b) is an order 1 consistent method applied to the variational equations.

3.1.4.1 The steady-state equations

Many codes use the steady-state approximation for the radical differential equations. More exactly, let $c = [l, r]$ with l denoting the long-lived species and r the radical species and let $\Pi(l, r)$, $\Delta(l, r)$ be the production and destruction terms for radicals:

$$\frac{dr(t)}{dt} = \Pi(l(t), r(t)) - \Delta(l(t), r(t)) \cdot r$$

Consider the radicals to vary so rapidly, that they are at steady state. Then $\frac{dr(t)}{dt} \approx 0$ and

$$r(t) \approx \frac{\Pi(l(t), r(t))}{\Delta(l(t), r(t))} \tag{3.14}$$

This algebraic equation is commonly solved by fixed-point (functional) iterations, keeping $l(t) = l_0 = constant$:

$do\ k = 1, No\_of\_iterations$

$call\ radical\ (in:\ \alpha,\ \beta,\ r^k,\ l_0;\ out: \Pi^k,\ \Delta^k)$

$do\ i = 1, No\_of\_radicals$

$$r_i^{k+1} = \frac{\Pi_i^k}{\Delta_i^k}$$

$end\ do\ i$

$end\ do\ k$

where the superscript k refers to the iteration number.

By forward automatic differentiation, this becomes:

$do\ k = 1, No\_of\_iterations$

$call\ g\_radical\ (in:\ \alpha,\ \beta,\ r^k,\ l_0,\ \nabla\ \beta,\ \nabla r^k,\ \nabla l_0;$

$out:\quad \Pi^k,\ \Delta^k,\ \nabla\Pi^k,\ \nabla\Delta^k)$

$do\ i = 1, No\_of\_radicals$

$$r_i^{k+1} = \frac{\Pi_i^k}{\Delta_i^k} \tag{3.15a}$$

$$\nabla r_i^{k+1} \doteq \frac{\nabla \Pi_i^k}{\Delta_i^k} - \frac{\Pi_i^k \nabla \Delta_i^k}{(\Delta_i^k)^2} \tag{3.15b}$$

end do i

end do k

It is interesting to look at (3.15b) in relation with (1.4b). Since the radicals are supposed to be at steady-state, the sensitivities ∇r are constant in time and hence (1.4b) becomes:

$$\nabla r_i \doteq \frac{\nabla \Pi_i}{\Delta_i} - r_i \cdot \frac{\nabla \Delta_i}{\Delta_i} \doteq \frac{\nabla \Pi_i}{\Delta_i} - \frac{\Pi_i \nabla \Delta_i}{\Delta_i^2}, \quad i = 1, ..., n \tag{3.16}$$

This reveals that the FAD generated (3.15b) is a fixed point iteration to solve the nonlinear system (3.16). For the conditions under which these iterations converge we refer again to [42].

3.1.4.2 Conclusions for automatically differentiated QSSA

Black box application of forward automatic differentiation on a fixed step-size QSSA algorithm (3.11) is equivalent to solving the variational equations (1.4b) with the first order method given by the formula (3.13b). If the steady state approximation is used for radicals, and the resulting nonlinear equations are solved by fixed point iterations, then forward automatic differentiation will result in an algorithm that uses the steady state assumption on the variational equation and solves the nonlinear equations for ∇r by a fixed point scheme.

We point out again here that the errors of the numerical scheme (3.13b) will dominate those induced by the automatic differentiation process. Hence, it is reasonable to expect that the computed sensitivity coefficients are at most as accurate

as the computed concentrations.

3.2 Sensitivity calculation via FAD-generated variational equations

3.2.1 General setting

As shown by our previous considerations, automatic differentiation implicitly generates the linearized equations (1.4b). To be more precise, let the rate equations be described by the subroutine:

subroutine compute $(in:\ t,\ \alpha,\ c;\ out:\ cdot)$

where α are the parameters, c is the vector of concentrations and $cdot = dc/dt$ at given input arguments. Forward automatic differentiation will generate:

subroutine g_compute $(in:\ t,\ \alpha,\ c,\ \nabla c;\ out:\ cdot, \nabla cdot)$

Under the assumptions that sensitivities obey (1.4b) and that dc/dt is a smooth function of t and α the subroutine *g_compute* completely describes (1.4b) because:

$$\nabla cdot = \nabla \frac{dc}{dt} = \frac{d}{dt} \nabla c.$$

It is therefore possible to combine the advantages of FAD with efficient numerical schemes if we firstly generate (1.4b) via automatic differentiation, then solve the variational system using an integrator of choice. This approach should work:

- better than the black box automatic differentiation, which is in principle equivalent to the direct approach, with a fixed method for integrating the system; the hybrid approach allows the use of variable step-size algorithms and enables the decoupled integration of sensitivity equations ([34]);

- easier than the usual implemention of the direct (decoupled) method where the variational equations are derived either by hand or by using symbolic manipulation.

3.2.2 Direct decoupled method with dedicated algorithms

Dedicated algorithms are explicit methods for the numerical solution of the rate equations (3.2) which make use of their special (production-destruction) form; examples of dedicated methods are QSSA, Young and Boris and Two-Step (see [91]).

Looking at the variational equation (1.4b) we remark that it is also formulated in production - destruction form:

$$\begin{aligned} \frac{d}{dt}\nabla c_i(t) &\doteq \mathcal{P}^i(c) - \mathcal{D}^i \cdot \nabla c_i, \quad i = 1, ..., n\ , \quad \text{where:} \\ \mathcal{P}^i(c) &\doteq \nabla P^i(c) - \nabla D^i(c) \cdot c_i, \quad i = 1, ..., n\ , \\ \mathcal{D}^i(c) &= D^i(c), \quad i = 1, ..., n\ . \end{aligned}$$

Since dedicated integrators perform the computations in a componentwise manner, their use leads to a natural decoupling between the concentration and the sensitivity equations. Thus, we can extend the direct decoupled method of [34] to dedicated integrators.

In the case of using QSSA-type methods, since their main computational load (besides function evaluation) consists of calculating the exponentials $exp(-D^i \cdot \Delta t)$ and since $\mathcal{D}^i(c) = D^i(c),\ i = 1, ..., n$, the algorithm can be optimized, in the sense that the exponentials have to be evaluated only once per component per time step (and not once per sensitivity coefficient per time step).

CHAPTER 4
RESULTS

4.1 Introduction

In this chapter we demonstrate the use of ADIFOR in the sensitivity analysis of a comprehensive gas phase chemical box model. Automatic differentiation is used to calculate first order sensitivities with respect to initial conditions and chemical reaction rate constants for a wide range of chemical conditions corresponding to the IPCC (Intergovernmental Panel on Climate Change) Chemistry Intercomparison study (see [70]).

Dedicated algorithms are explicit methods for the numerical solution of the rate equations (3.2) which make use of their special (production-destruction) form; examples of dedicated methods are QSSA, Young and Boris and Two-Step (see [91]).

Looking at the variational equation (1.4b) we remark that it is also formulated in production - destruction form:

$$\begin{aligned} \frac{d}{dt}\nabla c_i(t) &\doteq \mathcal{P}^i(c) - \mathcal{D}^i \cdot \nabla c_i, \quad i = 1, ..., n \ , \quad \text{where:} \\ \mathcal{P}^i(c) &\doteq \nabla P^i(c) - \nabla D^i(c) \cdot c_i, \quad i = 1, ..., n \ , \\ \mathcal{D}^i(c) &= D^i(c), \quad i = 1, ..., n \ . \end{aligned}$$

Since dedicated integrators perform the computations in a componentwise manner, their use leads to a natural decoupling between the concentration and the sensitivity equations. Thus, we can extend the direct decoupled method of [34] to

dedicated integrators.

In the case of using QSSA-type methods, since their main computational load (besides function evaluation) consists of calculating the exponentials $exp(-D^i \cdot \Delta t)$ and since $\mathcal{D}^i(c) = D^i(c),\ i = 1, ..., n$, the algorithm can be optimized, in the sense that the exponentials have to be evaluated only once per component per time step (and not once per sensitivity coefficient per time step).

4.2 Accuracy of different methods: numerical comparisons

In this subsection we display some numerical results obtained by integrating variational equations with Plain QSSA, Extrapolated QSSA and Symmetric QSSA methods (as described in [55]). We have employed VODE (see [11]) to accurately solve the system of ordinary differential equations, as well as the corresponding variational system. We consider the "exact" solution to be that obtained by integrating the variational system with VODE, rtol=10^{-6}, atol=10^{-18}.

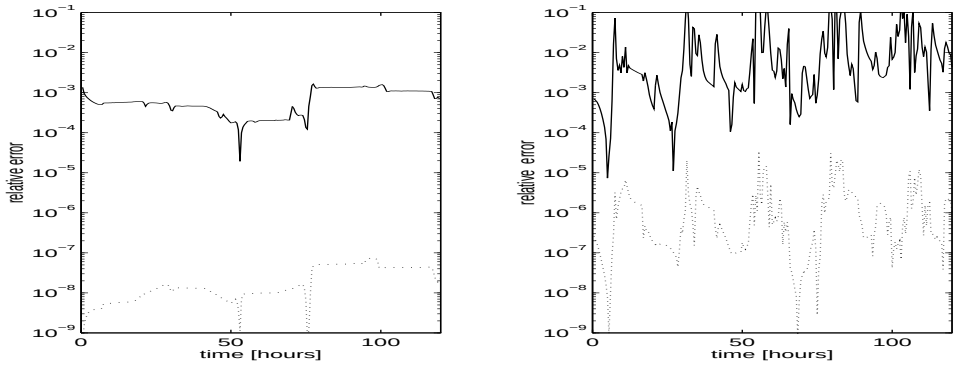


Figure 4.1. Relative errors in computed concentration C (dots) and in brute force $\partial C/\partial[NO]_0$ (solid) for $C = O_3$ (left) and $C = OH$ (right).

Figure 4.1 compares the relative errors in the computed concentration and the corresponding brute-force sensitivity coefficient $\partial/\partial[NO]_0$. The integration was done in double precision with VODE (rtol=10^{-5}). Figure 4.1 illustrates a point that we made earlier, namely that, for the brute force approach, the number of significant digits in the computed sensitivity coefficient is at most half of the number of significant digits in the value of the function (here, the value of the concentration c).

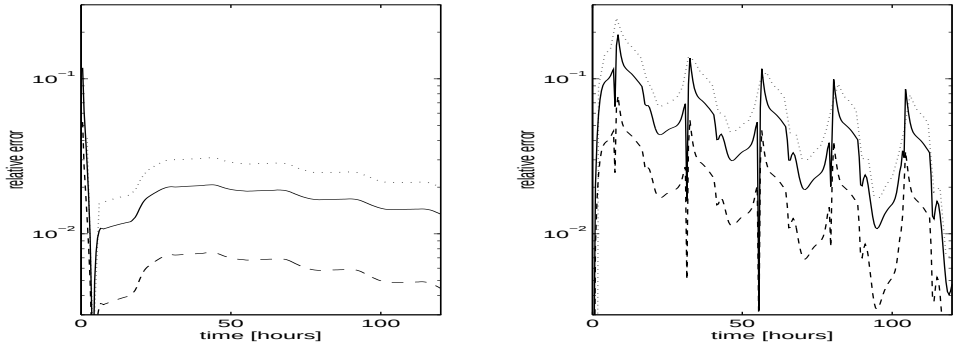


Figure 4.2. Relative errors in $\partial[O_3]/\partial[NO]_0$ (left) and $\partial[NO]/\partial[NO]_0$ (right) with brute - force approach (dots), directly differentiated QSSA algorithm (solid), and QSSA integrated variational equations (dash-dashed).

Figure 4.2 compares the relative errors in the sensitivity coefficients $\partial/\partial[NO]_0$ computed by directly differentiating the QSSA code (solid lines), by integrating the variational equations with QSSA (step-size 10 seconds, dash-dashed line) and by brute-force estimating them from the outputs of the QSSA code (step-size 10 seconds). This comparison illustrates the fact that the number of significant digits in the computed sensitivity coefficients is given by the precision of the integration scheme (here QSSA) and not by the precision of automatic differentiation process

(which is of the same order as machine precision). As expected, the brute force approach is the least accurate, while the integration of variational equations seems to be the right choice. The CPU timings corresponding to these numerical experiments are shown (the timings may depend on the size of the code and the amount of memory available. Results presented here were obtained on a HP-UX A 9000/735 workstation with 128 M RAM, using level 1 of optimization) in Table 4.1. The calculation of the sensitivities of all species with respect to one parameter implies the calculation of 84 coefficients (since the model has 84 variable species); the calculation of the sensitivities of all species with respect to all initial values implies the calculation of 84*84 coefficients.

Table 4.1. Timings for different methods.

Method	No of sensitivity coefficients	Normalized time
QSSA	concentrations only	1
Direct FAD on QSSA	$84 * 84$	105
Variational eqns with QSSA	$84 * 84$	82
Variational eqns with Extrapolated QSSA	$84 * 84$	50
Variational eqns with Symmetric QSSA	$84 * 84$	58

4.3 Application of FAD to a comprehensive atmospheric chemical mechanism

4.3.1 The Chemical mechanism

The chemical mechanism used in this study is that presently used in the STEM-II regional scale transport/ chemistry/ removal model (see [14]). This mechanism consists of 86 chemical species and 178 gas phase reactions. The mechanism, based on the work of Lurmann et al. ([58]) and Atkinson et al. ([5]) is representative of those presently being used in the study of chemically perturbed environments. The mechanism represents the major features of the photochemical oxidant cycle in the troposphere and can be used to study the chemistry of both highly polluted (e.g., near urban centers) and remote environments. The photochemical oxydant cycle is driven by solar energy and involves nitrogen oxides, reactive hydrocarbons, sulphur oxides and water vapour. The chemistry also involves naturally occuring species as well as those produced by anthropogenic activities. Many of the chemical reaction rate coefficients vary with the intensity of solar radiation (photolysis rates), and thus follow a strong diurnal cycle. Others vary with temperature.

4.3.2 The IPCC scenarios

To test the robustness of the above numerical algorithms, we have employed six different scenarios (summarised in Table 4.2). These conditions represent various chemical environments ranging from: low NO_x oceanic boundary layer regions

Table 4.2. Initial Conditions for each of the six IPCC scenarios simulated.

Scenario		1	2	3	4	5	6
Elements	Units	Marine	Land	Bio	Free	Plume 1	Plume 2
Altitude	[Km]	0	0	0	8	4	4
Temp	[K]	288.15	288.15	288.15	236.21	262.17	262.17
Pressure	[mbar]	1013.25	1013.25	1013.25	356.5	616.6	616.6
Nitrogen	$[10^{19} cm^{-3}]$	2.55	2.55	2.55	1.09	1.70	1.70
H_2O	[ppb]	10000	10000	10000	500	2500	2500
H_2	[ppb]	500	500	500	500	500	500
H_2O_2	[ppb]	2	2	2	2	2	2
O_3	[ppb]	30	30	30	100	50	50
NO_x	[ppb]	0.01	0.2	0.2	0.1	10	10
HNO_3	[ppb]	0.1	0.1	0.1	0.1	0.1	0.1
CO	[ppb]	100	100	100	100	600	600
CH_4	[ppb]	1700	1700	1700	1700	1700	1700
$NMHC$	[ppb]	0	0	1 (ISOP)	0	0	117.5

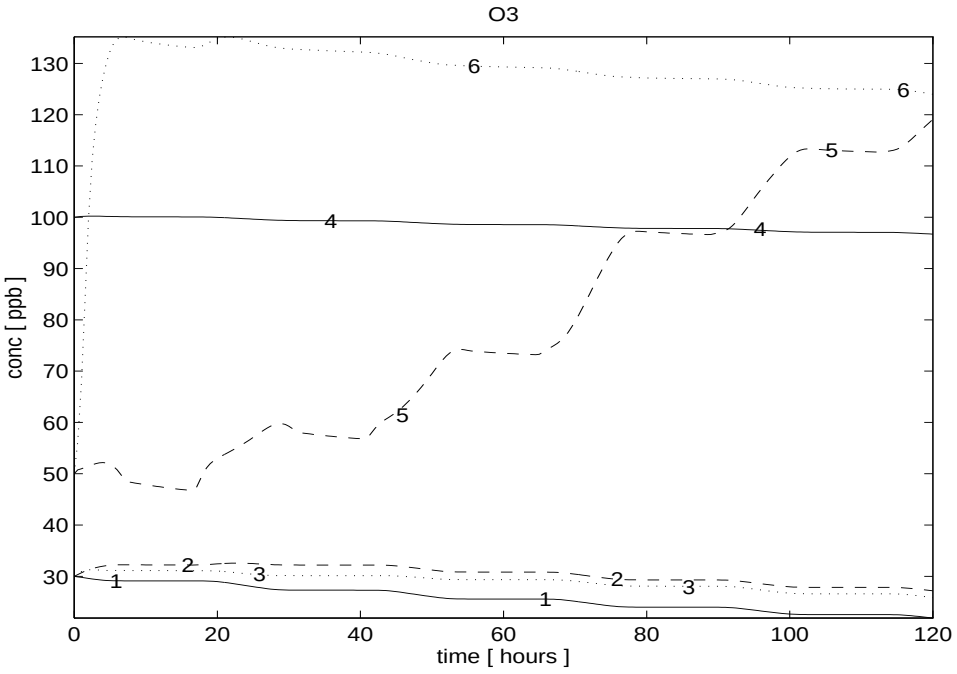


Figure 4.3. The variations of ozone under different IPCC scenarios (see Table 2 for a detailed description).

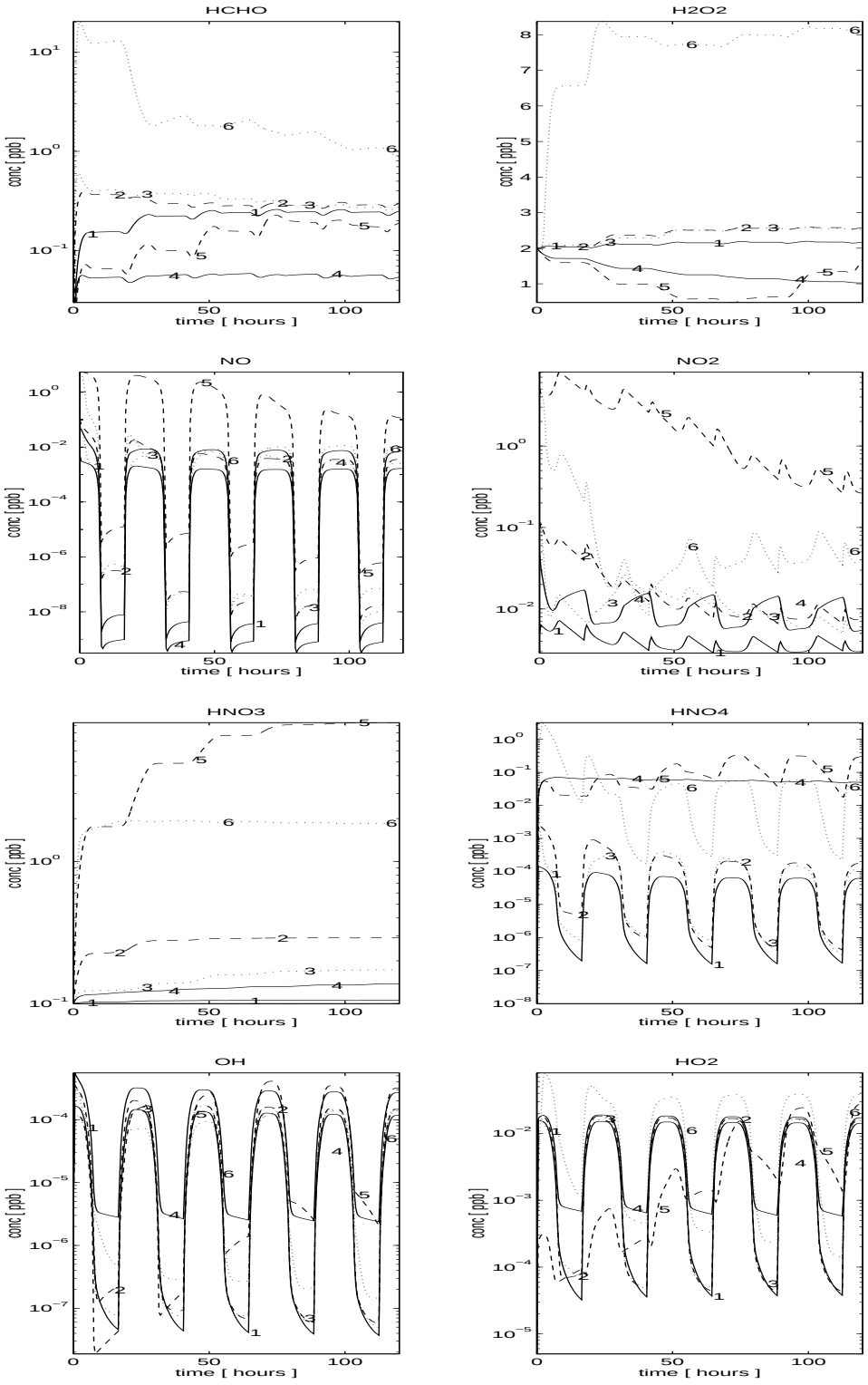
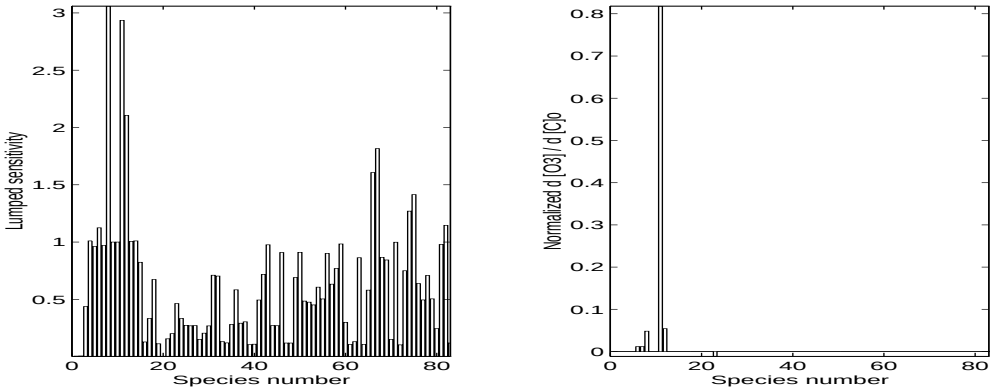


Figure 4.4. Variation of the most important species under IPCC scenarios.

(Marine); high NO_x continental boundary layer regions without (Land) and with isoprene (Bio); dry upper tropospheric regions (Free); biomass burning plumes without (Plume 1) and with (Plume 2) reactive hydrocarbon species. Further details are presented in Chapter 7 of the current WMO Ozone Assesment (see [70]). ADIFOR was used to calculate sensitivities of ozone with respect to initial conditions and reaction rate parameters. In the simulations of these cases the QSSA method with a fixed stepsize of 10 seconds was used. This algorithm is suited for direct automatic differentiation, and is easy to implement when solving (1.4b). Its use leads to results having 1-2 significant digits. The calculated ozone concentrations for the five cases are presented in Figure 4.3, and all other important species in Figure 4.4. In the marine boundary layer case, ozone is continuously destroyed throughout the 5 day period. The land and bio conditions show initially ozone production, followed by a net slight destruction of ozone over the simulation period. In the dry free troposphere (Free) ozone values decrease very slowly. Both plume cases show a large net ozone production. The case (Plume-1) without non-methane hydrocarbons (NMHC) shows a much slower net ozone production rate, and a distinct diurnal behavior. The Plume-2 case (with NMHC) shows a very rapid increase in ozone, followed by a period of slow ozone destruction.

4.3.3 Numerical results and interpretation

The calculated local sensitivities of ozone with respect to the initial conditions of each species for the Marine case are shown in Figure 4.5. Plotted are the normalized sensitivities at 120 hours of simulation. Also shown are the 8 largest (+) sensitivities (indicating ozone production) and (-) sensitivities. Under these



LUMPED		OZONE			
Species	Sensitivity	Species	Sensit > 0	Species	Sensit < 0
HNO_3	3.0591	O_3	8.1813E-1	H_2O_2	-1.12491E-2
O_3	2.9348	CH_4	5.4110E-2	CO	-1.04765E-2
CH_4	2.1060	HNO_3	4.8292E-2	$PRN1$	–1.17844E-6
$RAO2$	1.8158	NO_2	1.1494E-2	$MRO2$	-3.33028E-7
R_3O_2	1.6063	NO	1.1475E-2	$VRO2$	-2.68546E-7
$PRN1$	1.4141	$MAO3$	3.6442E-7	$RAO2$	-1.52807E-7
$CRO2$	1.2698	$KO2$	3.0516E-7	$MVKO$	-8.79144E-8
$MCRG$	1.1467	$MCO3$	2.6008E-7	OH	-7.02106E-8

Figure 4.5. Marine case, lumped sensitivities w.r.t. initial values (left) and sensitivities of ozone w.r.t. initial values (right).

conditions ozone concentrations are most sensitive to the initial concentration of ozone (as expected since this case has a net destruction of ozone). Ozone levels increase with increases in CH_4, $NO_x = NO + NO_2$ and HNO_3, species which both lead to the production of ozone and also help to modulate the HO_2 concentrations which is the principal lose mechanism for ozone under these conditions. Note also that HNO_3 is the principal source of NO_x in this case since its initial condition is an order of magnitude higher than NO_x. The largest negative sensitivity is that with respect to H_2O_2, which is the dominant source of HO_2 radicals.

Also shown are the lumped sensitivities. Lumped sensitivities can help describe the overall effect of a given perturbation. The lumped sensitivity of the system with respect to parameter α_j is defined as:

$$L(\alpha_j) = \sqrt{\sum_{i=1}^{N} \left(\frac{\partial c_i(t)}{\partial(\alpha_j)} \right)^2} \tag{4.1}$$

where $c_i,\ i = 1, ..., N$ are the concentrations of the component species. Since $L(\alpha_j)$, $j = 1, ..., m$ are functions of time, and since we are interested in the global effect of α_j over the system, we employ the mean values of the lumped sensitivity coefficients over the selected time horizon:

$$\bar{L}(\alpha_j) = \frac{1}{t_2 - t_1} \cdot \int_{t_1}^{t_2} L(\alpha_j) \cdot dt \tag{4.2}$$

If the mean sensitivities:

$$S_j = \frac{\bar{L}(\alpha_j)}{N}$$

are far less then one, the system is considered stable with respect to the initial conditions (see [18]).

Numerical experiments with the test problem showed that lumped sensitivities are dominated by ozone sensitivity. This may happen because O_3 concentration is

larger than other variable species. To obtain a more accurate description of the global response of the system with respect to perturbations of initial conditions, we employ the lumped normalized sensitivities:

$$\hat{L}(\alpha_j) = \sqrt{\sum_{i=1}^{N} \left(\frac{\alpha_j}{c_i(t)} \cdot \frac{\partial c_i(t)}{\partial(\alpha_j)} \right)^2} \tag{4.3}$$

where $c_i,\ i = 1, ..., N$ are the concentrations of the component species.

The lumped coefficients are thought of as measures of the global influence of one parameter over the whole system. These quantities are a bit more difficult to interpret. A large value can arise from a few very large individual sensitivities or from many species being sensitive to changes in one individual parameter (as is the case for the radical species). One drawback is that, when a parameter has a large influence over a specific or small number of species, the lumped coefficient may be large, although, from a chemical standpoint, the "global" influence is negligible. As an example, look at Table 4.8. The large lumped sensitivity associated with DMS can be explained by a strong influence of DMS initial concentration over itself, although the effect of this parameter on important species is negligible.

Results for all the cases are summarized in Tables 4.3 and 4.4. The land, bio and free cases are quite similar. One notable difference is the bio case where isoprene is shown to have a large negative sensitivity. This highly reactive species is an important source of peroxyl radicals. The Plume-1 and Plume-2 cases both show large net ozone production and thus behave much differently than the previous cases. In the Plume-1 case, ozone increases with increases in O_3, CO, CH_4, and H_2O_2. Under high NO_x conditions these species are involved in reactions which increase the peoroxyl radical pool, which in turn leads to net ozone production. Ozone decreases with increases in NO_x, because in the absence of $NMHC$ these species scavenge the free radicals and shorten the chain propagation reactions. When sufficient

$NMHC$ are present then ozone production is larger, and increases in NO_x lead to increases in ozone. The importance of reactive hydrocarbons appears both in the production and destruction of ozone. At the time shown in Table 4.3, ozone is in a period of net ozone destruction. Here ozone is being destroyed by reaction with hyroperoxyl radicals. As shown in Figure 4.3, ozone production is very rapid during the first day of the simulation. In this period ozone production is driven by $NMHC$ reactions involving alkenes, aldehydes, isoporene and aromatics. After this initial period, those less reactive species which supply peroxyl radicals contribute to ozone destruction (e.g., aldehyde2, and aromatics and alkanes). NO_x, which both produces ozone and scavenges peroxyl radicals, shows positive sensitivities. The lumped sensitivities are much higher indicating the higher overall reactivity of the system, and the importance of the $NMHC$'s.

The amount of ozone produced in an air mass is a complex function of the absolute amounts of NO_x, the relative amounts (the value of $NMHC/NO_x$) and the meteorological conditions (solar actinic flux, temperature, water vapor, etc.). A net ozone production efficiency e_N has been defined by Lin et. al. (see [57]):

$$e_N = \frac{\partial O_3}{\partial NO_y}$$

where e_N represents the net number of ozone molecules produced per molecule of NO_y consumed. When the above equation is evaluated using observations or model derived estimates of ozone, it represents a lower limit since ozone is deposited at the earth's surface. This metric can be used to help characterize the modeled ozone production efficiencies . The relationships between model calculated O_3 and NO_z are presented in Figure 4.4. The quantity NO_z is defined as:

$$\begin{aligned} NO_z &= NO_y - NO_x \\ &= PRN_1 + PRN_2 + PRPN + HNO_3 + HONO + PAN + \end{aligned}$$

$$TPAN + R_3N_2 + RAN_1 + RAN_2 + N_2O_5 + HNO_4 +$$

$$NO_3 + MPAN + IPAN + INO_2 + MAN_2 + MVN_2$$

and is a metric which reduces variations arising from differences in the age of the air masses being compared (see [67], [66]).

This ozone production efficiency is simply a local sensitivity. This efficiency was calculated using ADIFOR for each of the studied cases. The results are shown in Figure 4.6. The highest ozone production efficiencies occur for the marine case followed by the land, bio and free conditions. For these cases e_N ranges from 5 to 10. The plume with NMHC shows a value of ≈ 2, while the plume without NMHC has a very small ozone production efficiency. These values can be compared to those measured in the eastern United States during the summer. Olsyzyna et al. ([66]) report a value of 12 (10 for NO_y) at Harvard Forrest, MA, while Trainer et al. ([67]) report an average value of 8.5 at several rural sites.

ADIFOR was also used to calculate the sensitivity of ozone to variations in the reaction rate constants. The results for the individual and lumped sensitivities are shown in Tables 4.6 and 4.8 respectively. For the marine case the largest (+) sensitivities occur for the photolysis reaction of NO_2 and HNO_3 and consumption reactions of HO_2 (i.e., the $HO_2 + HO_2$ reaction and the $NO + HO_2$ type reactions). These reactions consume the peroxyl radical which by reaction with O_3 destroys ozone. The largest (-) sensitivities occur for the major ozone destruction reactions ($O_3 + h\nu$ and $O_3 + HO_2$). Again the land, bio and free cases show similar behavior. The bio case shows a much higher reactivity as measured by the magnitudes of the lumped sensitivities. The plume cases show a different behavior. The case without $NMHC$ (Plume-1) shows that ozone is most positively sensitive to the reaction rate constants for the NO_2 photolysis rate, the NO reaction with HO_2, CO

oxidation by OH and ozone photolysis. The largest negative sensitivities occur for the $NO_2 + OH$ and $O_3 + NO$ reactions. The Plume-2 case shows the importance of $NMHC$ reaction rate constants. For example, ozone has a positive sensitivity with respect to the reaction rate constant for the ethene and alkene oxidation by OH, aldehyde photolysis, and reactions involving PAN (i.e., $PAN \rightarrow MCO_3 + NO_2$). Ozone decreases with increases in reaction rate constants for the $MCO_3 + NO_2$ reaction, and other reactions similar to the Plume-1 case.

ADIFOR also easily yields the sensitivities with respect to other chemical kinetic parameters. For example the sensitivities with respect to changes in emissions are presented in Table 4.10, and sensitivities with respect to temperature are shown in Table 4.9 for the `Bio`, `Free` and `Plume-2` cases. The `Bio` and `Plume-2` cases show similar behaviour with respect to temperature, with large positive sensitivities for NO_X, HNO_3, and negative sensitivities for PAN and all PAN-like compounds. The `Free` case shows the opposite behaviour, and PAN increases with increasing temperature.

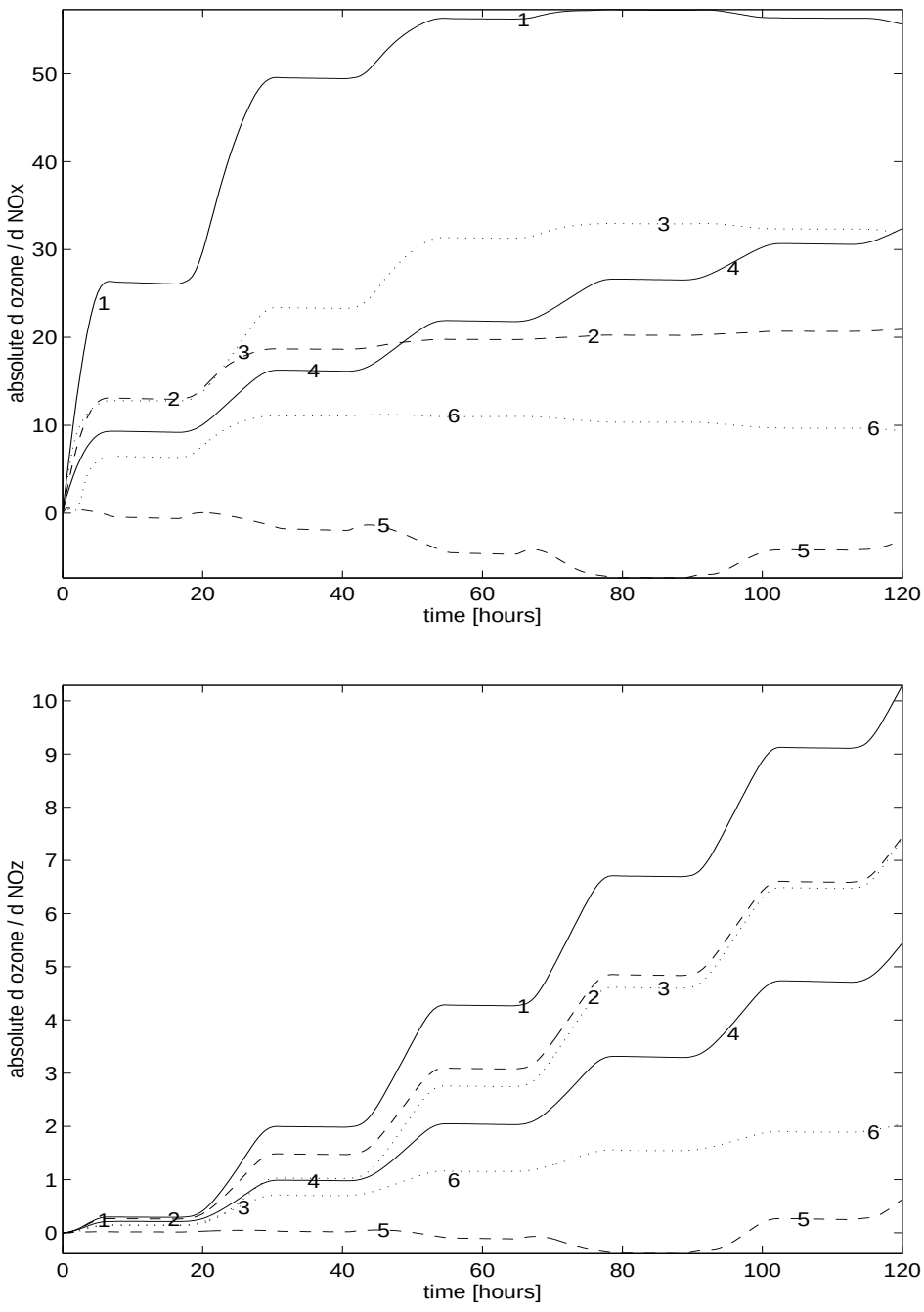


Figure 4.6. Absolute sensitivities of ozone with respect to initial NO_X concentration (upper plot) and with respect to initial NO_Z concentration (lower plot). The variation in time is shown for different IPCC scenarios.

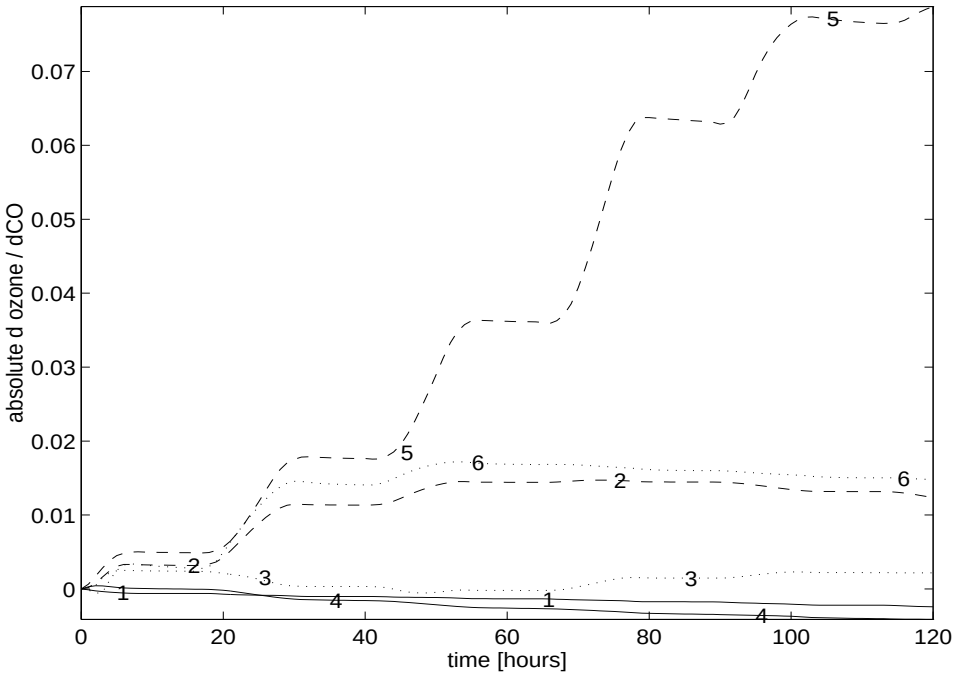


Figure 4.7. Absolute sensitivities of ozone with respect to initial CO concentration. The variation in time is shown for different IPCC scenarios.

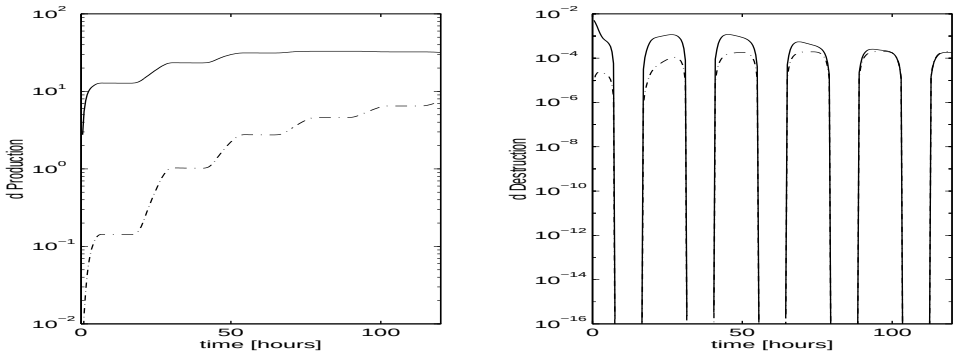


Figure 4.8. Scenario 3 (Bio). Absolute sensitivities of production term (left) and destruction term (right) of ozone w.r.t. initial NO_x concentration (solid) and w.r.t.initial NO_z concentration (dash-dot).

Table 4.3. Normalized sensitivities of O_3 with respect to initial values (at the end of the fifth day); displayed are the values of modulus greater than 1E-3.

$\frac{\partial O_3}{\partial \cdots}$	Marine	Land	Bio	Free	Plume 1	Plume 2
O_3	8.18E-1	5.41E-1	6.40E-1	9.25E-1	3.93E-1	1.52E-1
CH_4	5.41E-2	1.16E-1	8.13E-2	9.27E-3	9.89E-2	8.70E-3
CO	-1.04E-2	3.41E-2	4.05E-3	-5.04E-3	0.31	4.48E-2
HNO_3	4.82E-2	2.84E-2	3.00E-2	5.74E-3	9.20E-4	1.55E-3
NO_2	1.14E-2	7.08E-2	1.10E-1	1.52E-2	-7.84E-2	3.23E-1
NO	1.14E-2	6.97E-2	1.09E-1	1.46E-2	-1.17E-1	3.05E-1
HO_2	-1.12E-2	-7.63E-3	-5.89E-3	-7.32E-3	9.03E-2	-5.04E-3
$ISOP$	-	-	-7.50E-2	-	-	-1.36E-2
$ETHE$	-	-	-	-	-	1.64E-1
ALD_2	-	-	-	-	-	-1.39E-1
$ALKE$	-	-	-	-	-	-9.34E-2
$AROM$	-	-	-	-	-	-6.36E-2
$ALKA$	-	-	-	-	-	-2.51E-2
C_3H_8	-	-	-	-	-	-1.54E-2

Table 4.4. Lumped sensitivities with respect to initial values (at the end of the fifth day); displayed are the values of modulus greater than 1E-3.

$\frac{\partial O_3}{\partial \cdots}$	Marine	Land	Bio	Free	Plume 1	Plume 2
HNO_3	3.05	-	2.00	-	-	-
O_3	2.93	1.77	15	2.52	2.66	-
CH_4	2.10	2.13	8.03	2.34	2.95	-
CO	2.39	1.81	50.8	2.67	3.79	20
RAO_2	1.81	1.70	1.51	1.36	-	-
R_3O_2	1.60	1.59	-	1.31	-	-
$PRN1$	1.41	1.46	-	1.36	-	-
CRO_2	1.25	-	-	-	-	-
$MCRG$	1.14	-	-	-	-	-
NO	-	1.63	4.31	1.66	3.95	15.9
NO_2	-	1.41	4.29	1.36	33	16
C_2H_6	-	1.19	-	-	1.25	-
$ISOP$	-	-	9.00	-	-	4.05
H_2O_2	-	-	2.16	-	1.36	-
DMS	-	-	-	1.26	1.34	
CHO_2	-	-	-	-	1.39	-
$ALKE$	-	-	-	-	-	9.52
ALD_2	-	-	-	-	-	9.46
$AROM$	-	-	-	-	-	6.22
C_3H_8	-	-	-	-	-	5.08
$ALKA$	-	-	-	-	-	3.96

Table 4.5. Normalized sensitivities of O_3 with respect to rate coefficients at the end of the fifth day (Part 1); displayed are the values of modulus greater than 1E-3.

Reactants	Marine	Land	Bio	Free	Plume 1	Plume 2
$NO_2 + h\nu$	6.71E-2	1.84E-1	1.27E-1	- 3.28E-2	3.58E-1	3.84E-1
$CH_4 + OH$	5.40E-2	1.16E-1	8.11E-2	- 9.27E-3	9.88E-2	
$HO_2 + HO_2$	3.48E-2	2.17E-2	2.27E-2	- 1.08E-2	-3.44E-2	-
$HNO_3 + h\nu$	2.94E-2	4.15E-2	2.59E-2	-	4.84E-2	-
$NO + HO_2$	2.45E-2	8.06E-2	6.92E-2	- 2.60E-2	2.02E-1	-
$HNO_3 + OH$	2.45E-2	3.38E-2	2.13E-2	- 3.64E-3	-	-
$MO_2 + NO$	1.85E-2	2.74E-2	2.79E-2	-	-	-
$HO_2 + MO_2$	8.49E-3	-	-	-	-	-
$O_3 + h\nu$	-2.93E-1	-2.87E-1	-2.68E-1	- -2.84E-2	2.67E-1	-1.12E-1
$O_3 + HO_2$	-1.09E-1	-1.26E-1	-1.15E-1	- -3.97E-2	-4.70E-2	-1.89E-1
$NO_2 + OH$	-5.79E-2	-2.00E-1	-1.29E-1	- -1.24E-2	-4.11E-1	-9.10E-2
$NO + O_3$	-3.87E-2	-1.14E-1	-6.98E-2	- -2.20E-2	-3.19E-1	-1.90E-1
$O_3 + OH$	-2.54E-2	-2.75E-2	-2.56E-2	- -1.41E-2	-	
$H_2O_2 + h\nu$	-1.70E-2	-2.35E-2	-1.63E-2	- -7.07E-2	5.60E-2	-
$MO_2 + MO_2$	-1.27E-2	-1.01E-2	-	-	-	-
$CO + OH$	-1.17E-2	3.66E-2	-	- -6.14E-3	3.47E-1	-
$NO_2 + O_3$	-	-1.14E-2	-1.84E-2	-	-	-
$ISOP + OH$	-	-	1.52E-2	-	-	-

Table 4.6. Normalized sensitivities of O_3 with respect to rate coefficients at the end of the fifth day (Part 2); displayed are the values of modulus greater than 1E-3.

Reactants	Bio	Free	Plume 1	Plume 2
PAN	-1.44E-2	-	-	6.77E-2
$HNO_4 + OH$	-	- 9.25E-3	-2.68E-2	-
$HO_2 + OH$	-	- 6.49E-3	-2.05E-2	-
$HNO_4 + h\nu$	-	- 4.33E-3	-	-
$NO_2 + HO_2$	-	- -1.61E-2	-5.97E-2	-
HNO_4	-	-	5.49E-2	-
$HCHO + h\nu$	-	-	-2.34E-2	-
$MCO_3 + NO$	-	-	-	1.99E-1
$ALD_2 + h\nu$	-	-	-	1.85E-1
$ETHE + OH$	-	-	-	1.22E-1
$ALKE + OH$	-	-	-	5.48E-2
$HO_2 + MCO_3$	-	-	-	4.82E-2
$HCHO + OH$	-	-	-	4.07E-2
$MCO_3 + NO_2$	-	-	-	-2.48E-1
$ALD_2 + OH$	-	-	-	-1.64E-1
$HCHO + h\nu$	-	-	-	-4.64E-2
$ETHE + O_3$	-	-	-	-3.75E-2

Table 4.7. Lumped sensitivities with respect to rate coefficients at the end of the fifth day (Part 1); displayed are the values of modulus greater than 1E-3.

Reactants	Marine	Land	Bio	Free	Plume 1	Plume 2
$NO_2 + OH$	2.47	2.60	5.66	- 1.34	2.66	4.71
$O_3 + h\nu$	1.93	1.58	13.9	- 1.38	5.93	6.18
$CH_4 + OH$	1.77	1.86	7.94	- 2.11	2.77	-
$NO_2 + h\nu$	1.71	1.78	6.68	- 2.02	1.81	15
$CO + OH$	1.59	1.54	8.05	- 1.51	1.84	4.47
$HNO_3 + h\nu$	1.59	1.48	-	-	-	-
$HNO_3 + OH$	1.56	1.75	-	-	2.07	-
$HCHO + HO_2$	1.36	1.49	-	- 1.39	1.41	-
$MCO_3 + NO_2$	1.29	1.16	2.92	-	-	12.8
$NO_3 + h\nu$	1.26	1.27	-	- 1.03	-	-
$RAN_2 + OH$	1.23	1.57	-	- 1.42	1.61	-
PAN	1.23	-	5.53	-	-	4.35
$RAO_2 + NO$	1.20	-	-	-	1.19	-
$NO + OH$	1.15	-	-	- 1.02	-	-
$HONO + h\nu$	1.15	-	-	- 1.02	-	-
$R_3O_2 + NO$	1.11	-	-	-	-	-
N_2O_5	-	1.33	-	- 1.08	-	-
$NO_2 + HO_2$	-	1.28	-	- 1.04	-	-
$HO_2 + HO_2$	-	1.28	-	- 9.78E-1	1.56	-
$KET + OH$	-	1.17	-	-	-	-

Table 4.8. Lumped sensitivities with respect to rate coefficients at the end of the fifth day (Part 2); displayed are the values of modulus greater than 1E-3.

Reactants	Land	Bio	Free	Plume 1	Plume 2
$MGLY + h\nu$	1.12	-	-	-	-
$MVK + OH$	-	19.6	-	-	10.4
$MPAN$	-	13.3	-	-	-
$HAC + OH$	-	9.40	-	-	5.00
$MAO_3 + NO_2$	-	6.20	-	-	-
$H_2O_2 + h\nu$	-	5.34	- 1.02	-	
$MACR + OH$	-	4.30	-	-	10.6
$NO + O_3$	-	4.16	- 1.37	1.54	6.82
$MAO_3 + HO_2$	-	3.61	-	-	-
$MAO_3 + NO$	-	2.84	-	-	
$HCHO + h\nu$	-	-	- 1.14	-	-
$ETO_2 + NO$	-	-	- 9.71E-1	-	-
$MO_2 + NO$	-	-	-	1.33	-
$AHO_2 + NO$	-	-	-	1.19	-
$TPAN$	-	-	-	1.18	-
$ALD_2 + OH$	-	-	-	-	10
$MCO_3 + OH$	-	-	-	-	9.69
$ETHE + OH$	-	-	-	-	9.49
$AROM + OH$	-	-	-	-	8.85
$HCHO + h\ \nu$	-	-	-	-	4.91

Table 4.9. Normalized sensitivity coefficients w.r.t. temperature. Shown are the values at the end of the fifth day for some selected species.

Species	Emission `Bio`	`Bio`	`Free`	`Plume-2`
NO	.119E+01	-.477E+01	-.103E+02	.280E+02
NO_2	.434E+01	-.355E+01	-.412E+01	.314E+02
HNO_3	.100E+02	-.212E+01	.105E+01	.423E+01
O_3	.162E+01	-.216E+01	-.438E-01	.101E+01
C_2H_6	-.158E+01	-.740E+00	-.135E+00	-.678E+00
C_3H_8	-.613E+01	-.169E+01	-.457E+00	-.316E+01
$ALKA$	-.513E+01	-.463E+00	-.697E+00	-.985E+01
$HCHO$	-.197E+01	-.286E+00	-.452E+00	-.211E+01
ALD_2	.146E+01	.151E+00	.309E+00	.186E+02
H_2O_2	.162E+01	-.165E+01	.838E-01	.567E+01
$ROOH$	-.360E+01	.194E+01	.166E+02	-.136E+02
$HONO$	.723E+01	-.593E+01	-.106E+02	.369E+02
PAN	-.437E+02	-.140E+03	.421E+01	-.210E+01
$TPAN$	-.230E+01	-.331E-02	.395E+01	-.164E+01
$MPAN$	-.398E+02	-.117E+03	.360E+01	-.230E+01
$IPAN$	-.491E+02	-.124E+03	.351E+01	-.323E+01
KET	-.350E+01	-.138E+01	-.365E+00	.140E+02
$GLYX$	-.117E+01	-.387E-02	.581E+00	.400E+02
$MGLY$	.370E+01	.206E+01	.891E-01	.155E+02
N_2O_5	.832E+01	-.459E+01	-.142E-03	.588E+02
HNO_4	.675E+01	-.434E+01	.173E+01	.340E+02
NO_3	.200E+01	-.103E+01	.439E+00	.277E+02
$ISOP$	-.401E+01	.393E-18	-.134E-13	-.494E-11

Table 4.10. Normalised sensitivity coefficients w.r.t. emission source intensities, `Bio` case. Shown are the values at the end of the fifth day.

Species	E[NO_X]	E[$ISOP$]	Species	E[NO_X]	E[$ISOP$]
NO	.124E+01	-.733E+00	N_2O_5	.207E+01	-.615E+00
NO_2	.144E+01	-.805E+00	HNO_4	.204E+01	-.143E+01
HNO_3	.153E+01	-.137E+01	NO_3	.390E+00	.690E+00
O_3	.527E+00	-.230E+00	$ISOP$	-.942E+00	.219E+01
C_2H_6	-.114E+00	.176E+00	MVK	-.898E+00	.222E+01
C_3H_8	-.538E+00	.833E+00	$MACR$	-.126E+01	.274E+01
$ALKA$	-.521E+00	.806E+00	HAC	-.270E+00	.159E+01
ACO_2	.272E+00	.715E+00	$MGGY$	.407E+00	.741E+00
ALD_2	-.860E+00	.265E+01	$MPAN$	.766E+00	.620E+00
H_2O_2	.419E+00	-.738E-01	$IPAN$	.122E+01	.223E+00
$ROOH$	-.102E+01	.199E+01	INO_2	-.286E+01	.520E+01
$HONO$	.243E+01	-.252E+01	MAN_2	-.177E+01	.408E+01
PAN	.102E+01	.442E+00	MVN_2	-.141E+01	.356E+01
KET	-.264E+00	.407E+00	$MACA$	-.321E+01	.560E+01
$MGLY$	.127E+00	-.298E+00	$PYVA$	-.192E+00	.166E+01
RAN_2	-.915E+00	.142E+01	OH	.114E+01	-.137E+01
RAN_1	-.623E+00	.676E+00	HO_2	.459E+00	-.334E+00

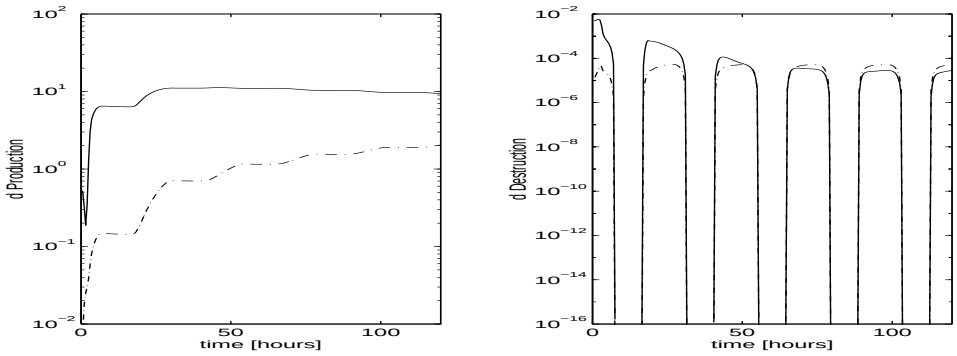


Figure 4.9. Scenario 6. Absolute sensitivities of production term (left) and destruction term (right) of ozone w.r.t. initial NO_x concentration (solid) and w.r.t.initial NO_z concentration (dash-dot).

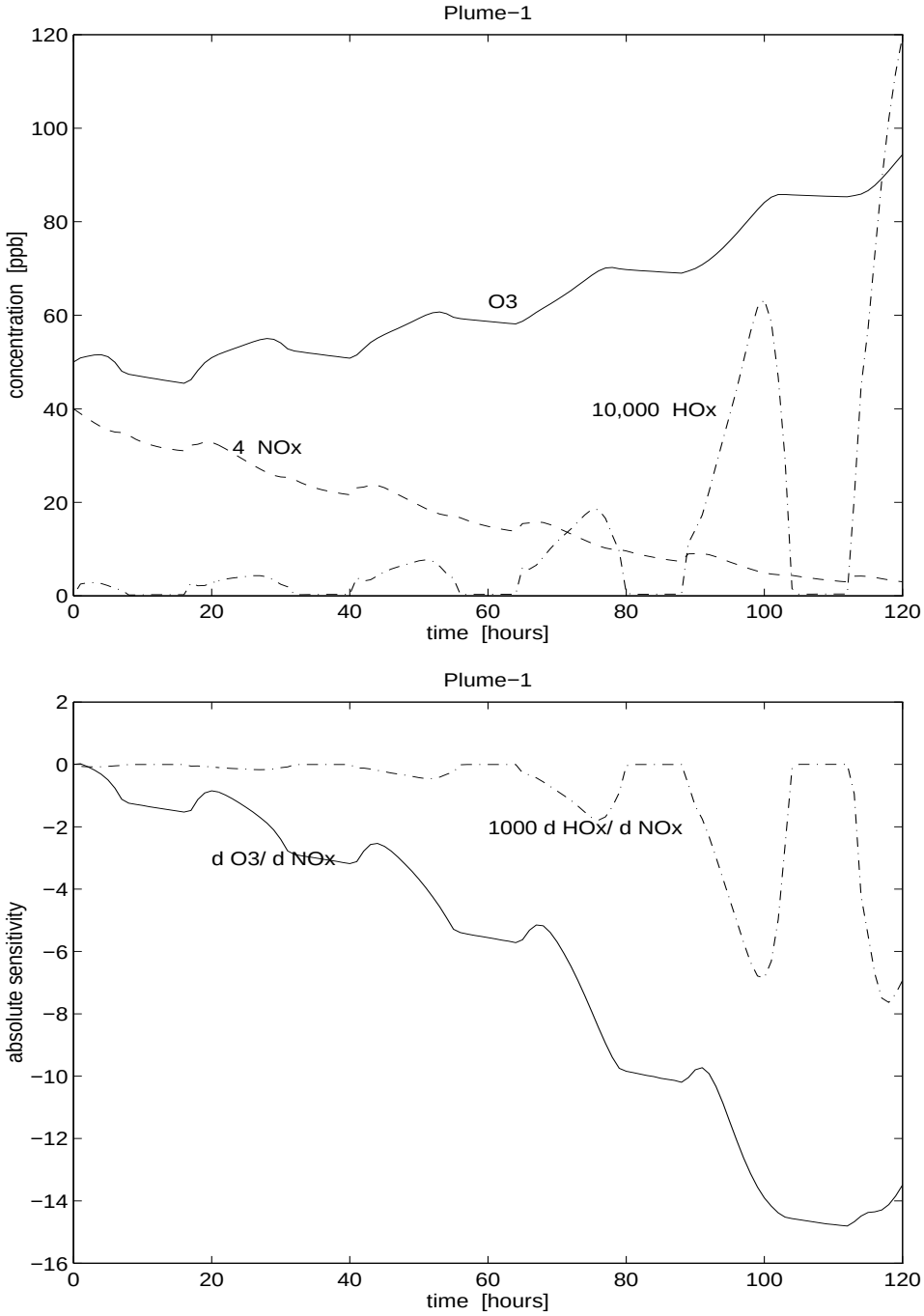


Figure 4.10. Plume-1 scenario. Time variation of O_3 (solid), NO_X (dashed, magnified 4 times) and HO_X (dash-dots, magnified 10^4 times) (upper plot) and absolute sensitivities with respect to initial NO_X concentration for O_3 (solid) and HO_X (dash-dots, magnified 10^3 times) (lower plot).

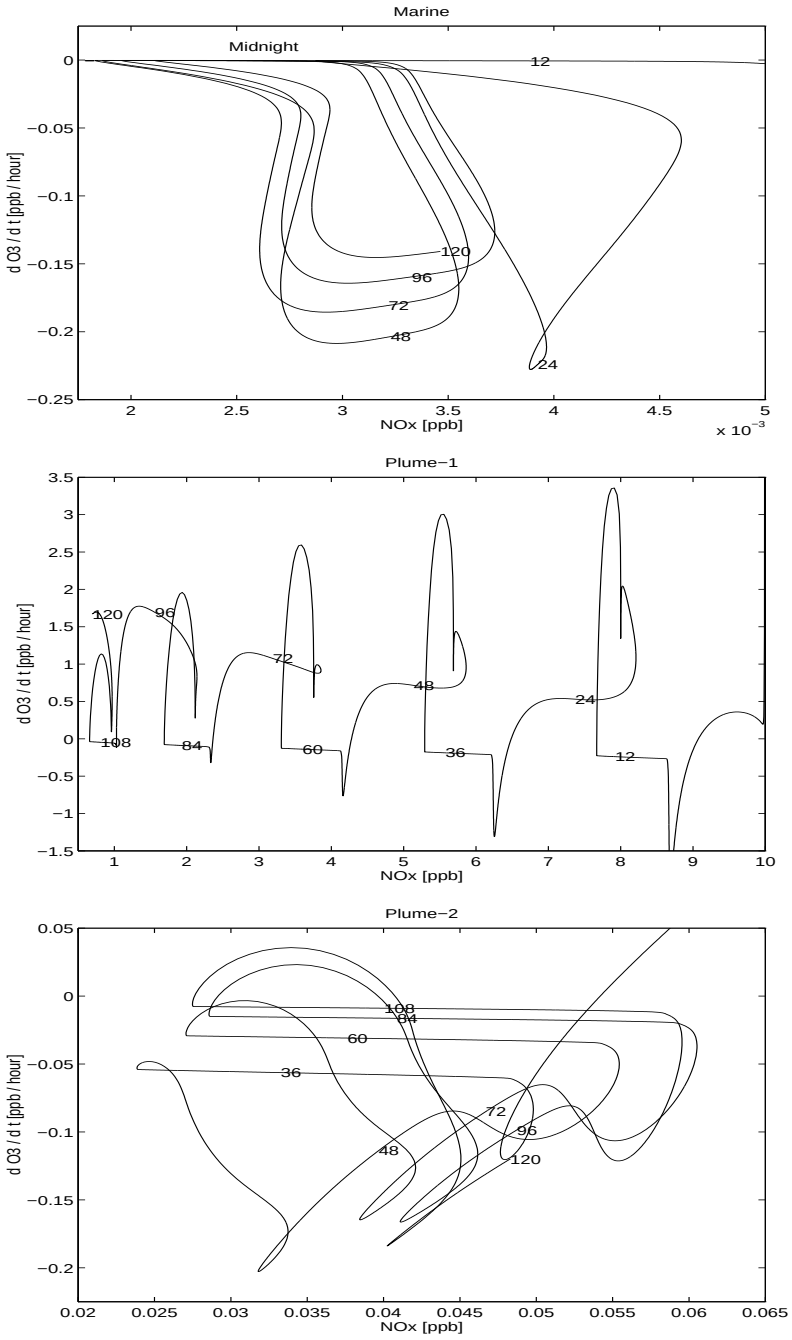


Figure 4.11. Time derivative of ozone versus NO_X for Marine (upper frame), Plume-1 and Plume-2 (lower frame) scenarios. The parameters on the curves represent the simulation time in hours. Since simulation starts at 12:00 pm, multiples of 24 represent local noon, while multiples of 24 plus 12 represent local midnight.

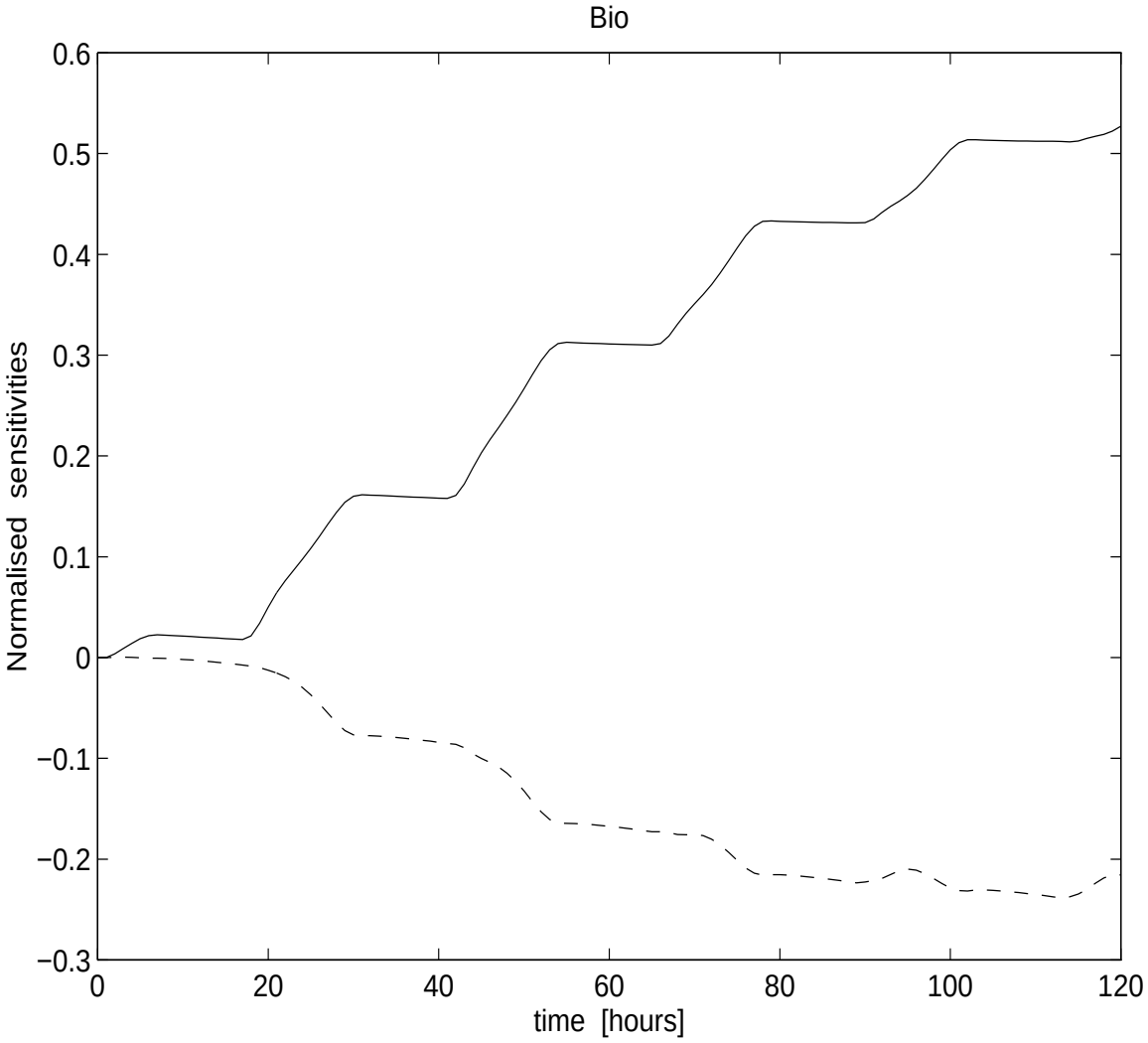


Figure 4.12. Bio scenario. Time evolution of the ozone sensitivities with respect to the strength of $NO_\times$ emission source (solid), and with respect to the strength of the isoprene emission source (dashed). Normalized coefficients are plotted.

CHAPTER 5
CONCLUSIONS

5.1 Results of this thesis

Automatic differentiation is a new and rapidly emerging software technology that enables the computation of derivatives from any mathematical relation described by a subroutine. The main strengths of AD are: calculation of derivatives without user intervention (in fact the user can ignore completely the meaning of the code at hand); accuracy limited by machine precision only; minimal number of arithmetic operations.

The principles of automatic differentiation and the Adifor2.0 system are presented. Several example codes are shown, and the difference between direct and reverse modes is discussed.

One problem of particular importance is the calculation of sensitivity coefficients: given an ordinary differential equation one needs the (partial) derivatives of the solution with respect to parameters that influence the solution (e.g. initial values). If the differential equation is integrated numerically, then numerical approximations of the sensitivities can be obtained via AD.

Two different strategies of using AD are analyzed in the thesis:

- Black box approach. The discretized differential equation is processed by Adifor2.0; the resulting code is shown to be a consistent discretization of the variational equations. The order of consistency and the convergence of

the resulting discretization are treated; several cases are studied in detail (multistep, Runge-Kutta and QSSA);

- Decoupled approach. The derivative function is processed by Adifor2.0, giving the variational equations; these are then discretized along the lines of the direct decoupled method.

It is shown that the black box approach is more direct, but also leads to a more expensive code (from the point of view of number of arithmetic operations) when an implicit discretization is used. Thus we recommend (whenever possible) that the decoupled approach be employed. The gains in efficiency might be critical for large scale problems, like the ones resulting from air quality models.

Adifor2.0 has been succesfully used in the sensitivity analysis of a comprehensive tropospheric chemistry model. Automatic differentiation appears to be a valuable tool for sensitivity analysis of atmospheric chemistry models. In this paper we focused solely on the chemical equations. However the method applies to to the coupled transport/ chemistry problems as well. We are presently applying this technique to the STEM-II model ([14]).

A valuable aspect of employing automatic differentiation for sensitivity analysis studies is that the atmospheric chemistry/ transport/ removal models are permanently subject to modifications and improvements. For routinely performing sensitivity analysis this means that, whenever a modification is performed in the model, the corresponding adjustment be made in the variational equations. Since the slightest mistake in the generation of variational equations could lead to useless results, one needs to thoroughly check for their correctness. Both the issues of

- easily generating the sensitivity equations and

- making sure they are error free

can be directly and succesfully addressed by the use of automatic differentiation.

5.2 Further research directions

Sensitivity analysis is of utmost importance in the qualitative understanding of the interdependencies arising in air quality models. There are a number of possible (future) applications.

Consider the atmospheric mass balance equation

$$\mathbf{c}_t + \nabla(u(\mathbf{x})\mathbf{c}) = \nabla(K(\mathbf{x})\nabla\mathbf{c}) + R(\mathbf{c}) + E(\mathbf{x})$$

where $\mathbf{c}$ is the vector of concentrations, $\nabla(u\mathbf{c})$ the advective flux due to the windfield u, $\nabla(K\nabla\mathbf{c})$ the diffusive flux, $R(\mathbf{c})$ the rate of chemical tranformations and $E(\mathbf{x})$ the intensity of emission sources.

A real-life problem is the following: pollutants emitted in one area (say, East Coast) affect (via advection and chemical transformations) other areas (say Mid-west). Thus, one would like to minimize the pollutant concentration at a point $\mathbf{x_0}$ by changing the emissions at several points $\mathbf{y_i}, 1 \leq i \leq n$:

$$\begin{aligned} \min \qquad & \mathbf{c}(\mathbf{x_0}) \\ & low \leq E(\mathbf{y_1}), \cdots, E(\mathbf{y_n}) \leq high \end{aligned}$$

To solve such an optimization problem one needs the derivatives of the solution $\mathbf{c}(t, \mathbf{x_0})$ with respect to the emission sources $E(\mathbf{y_1}), \cdots, E(\mathbf{y_n})$; in the language of this thesis, we need the derivative of the solution of a partial differential equation (PDE) with respect to some parammeters of the equation. Automatic differentiation can be very useful here; an extension of the theory from ODE to PDE is called for, and should be done in the future.

A similar problem is the following: what is the cheapest reduction in emissions $E(\mathbf{y_i})$ such that a certain environmental or health objective is attained. Here we need to attach costs to emission levels and the constraints are put on concentration values.

Another application to be explored in the future is uncertainty analysis. To fix the ideas, consider a chemical kinetic model whose output depends on a number of parameters. Among them are the rate coefficients; these coefficients are measured in smog chambers and their numerical value is understood as being the mean value of a certain confidence interval. In other words the value k is accompanied by an uncertitude level ϵ, such that the real value lies with a high probability within $(k-\epsilon, k+\epsilon)$. The question is, what is the uncertitude level of the result induced by the uncertitude level ϵ. A first order approximation gives

$$\mathbf{c} \in (\mathbf{c}-\eta, \mathbf{c}+\eta), \quad \text{where}$$
$$\eta \approx \frac{\partial \mathbf{c}}{\partial k}\epsilon$$

Again, to estimate the uncertainty η one needs the derivative of the solution with respect to the parameter k, and automatic differentiation can be directly applied.

In the context of three-dimensional models, the impacts of uncertainties in the meteorological data (temperature and wind fields), in the initial concentrations, in the radiation levels, etc on the output can be quantified. Using this information more effort can be put into improving the acccuracy of the critical quantities, thus obtaining the maximum benefit for the accuracy of the whole model.

Automatic differentiation in general, and Adifor2.0 in particular are very powerful techniques for sensitivity analysis; they can be used in black box mode, and under reasonable conditions the resulting code will correctly compute the desired derivatives; however, understanding AD, differentiating modules separately and

then assembling them together inteligently results in a more efficient sensitivity code; and this efficiency is crucial for large scale models, e.g. the ones used in air quality modeling.

APPENDIX A
THE STEM (LLOYD- ATKINSON- LURMANN) CHEMICAL MECHANISM

```
{1} NO2 + hv = NO + O3 :                          9.236E-3*SUN ;
{2} NO + O3 = NO2 + O2 :                          ARR(2.2E-12,-1430) ;
{3} NO2 + O3 = NO3 + O2 :                         ARR(1.2E-13,-2450) ;
{4} NO + NO3 = 2NO2 :                             ARR(1.7E-11,150) ;
{5} NO2 + NO3 = N2O5 :                            1.327E-12 ;
{6} N2O5 = NO2 + NO3 :                            2.013E-2 ;
{7} NO2 + NO3 = NO + NO2 + O2 :                   ARR(2.5E-14,-1230) ;
{8} NO3 + hv = 0.15NO + 0.85NO2 + 0.85O3 + O2 :  3.039E-2*SUN ;
{9} NO3 + HO2 = HNO3 + O2 :                       0 ;
{10} O3 + H2O + hv = 2OH :                        9.34E-22*RC(1) ;
{11} NO + OH = HONO :                             7.35E-12 ;
{12} HONO + hv = NO + OH :                        1.893E-3*SUN ;
{13} NO2 + OH = HNO3 :                            1.2794E-11 ;
{14} HNO3 + hv = NO2 + OH :                       6.684E-7*SUN ;
{15} HNO3 + OH = NO3 + H2O :                      ARR(9.4E-15, 778) ;
{16} N2O5 + H2O = 2HNO3 :                         0 ;
{17} CO + OH = HO2 + CO2 :                        2.22E-13 ;
{18} O3 + OH = HO2 + O2 :                         ARR(1.6E-12,-1000) ;
{19} NO + HO2 = NO2 + OH :                        ARR(3.7E-12,240) ;
{20} NO2 + HO2 = HNO4 :                           1.333E-12;
{21} HNO4 = NO2 + HO2 :                           0.02313;
{22} O3 + HO2 = OH + 2O2 :                        ARR(1.4E-14,-600) ;
{23} HO2 + HO2 = H2O2 + O2 :                      5.816E-12 ;
{24} H2O2 + hv = 2OH :                            7.829E-6*SUN ;
{25} H2O2 + OH = HO2 + H2O :                      1.57E-12 ;
{26} NO2 + H2O = HONO + HNO3 - NO2 :              4.00E-24 ;
{27} HNO4 + hv = NO2 + HO2 :                      1.11E-5*SUN ;
{28} HNO4 + OH = NO2 + H2O + O2 :                 ARR(1.3E-12,380) ;
{29} SO2 + OH = SO4 + HO2 :                       1.E-20 ;
{30} HCHO + hv = 2HO2 + CO :                      1.569E-5*SUN ;
{31} HCHO + hv = CO + H2 :                        6.063E-5*SUN ;
{32} HCHO + OH = HO2 + CO + H2O :                 1.00E-11 ;
{33} HCHO + HO2 = AHO2 :                          1.00E-14 ;
```

```
{34} AHO2 + NO = ACO2 + HO2 + NO2 :          ARR(4.2E-12,180) ;
{35} AHO2 + HO2 = ACO2 + H2O + O2 :          2.00E-12 ;
{36} 2AHO2 = ACO2 + 2HO2 + 2O2 :             1.00E-13 ;
{37} ACO2 + OH = HO2 + H2O + CO2 :           3.20E-13 ;
{38} NO3 + HCHO = HNO3 + HO2 +CO :           6.00E-16 ;
{39} ALD2 + OH = MCO3 + H2O :                ARR(6.9E-12,250) ;
{40} ALD2 + NO3 = HNO3 + MCO3 :              2.70E-15 ;
{41} ALD2 + hv = MO2 + HO2 + CO :            2.383E-6*SUN ;
{42} ALD2 + hv = CH4 + CO :                  6.063E-5*SUN ;
{43} MCO3 + NO2 = PAN :                      4.70E-12 ;
{44} PAN = MCO3 + NO2 :                      ARR(2.2E+16,-13435) ;
{45} MCO3 + NO = MO2 + NO2 + CO2 :           ARR(4.2E-12,180) ;
{46} MO2 + NO = HCHO + NO2 + HO2 :           ARR(4.2E-12,180);
{47} CH4 + OH = MO2 + H2O :                  ARR(2.4E-12,1710) ;
{48} C2H6 + OH = ETO2 + H2O :                ARR(1.7E-11,-1232) ;
{49} ETO2 + NO = ALD2 + HO2 + NO2 :          ARR(4.2E-12,180) ;
{50} C3H8 + OH = R3O2 :                      ARR(1.18E-11,-679) ;
{51} R3O2 + NO = 0.03R3N2 + 0.46ALD2 + 0.97NO2 +
           0.97HO2 + 0.49KET :               ARR(4.2E-12,180) ;
{52} ALKA + OH = RAO2 :                      ARR(2E-11,-500) ;
{---- IN THE NEXT REACTION
  BETA 1...9 WHERE REPLACED BY NUMBERS,
  AS READ FROM LILING STEM CODE ----}
{53} RAO2 + NO = 0.9261NO2 - 0.1892NO + 0.263RAN2 +
            1.0482ALD2 + 0.3KET +
            0.1879ETO2 + 0.1116MO2 +
            0.28HO2 + 0.1057R3O2 + 0.06RAO2 : ARR(4.2E-12,180) ;
{54} ALKA + NO3 = HNO3 + RAO2 :              4.00E-17 ;
{55} RAN2 + OH = RAN1 + H2O :                2.00E-12 ;
{56} RAN1 + NO = 2.5NO2  - 0.5NO + 0.8HCHO
          + 2.1ALD2 :                        ARR(4.2E-12,180) ;
{57} MO2 + MO2 = 1.4HCHO + 0.8HO2 + O2 :     ARR(1.5E-13,220) ;
{58} 2ETO2 = 1.6ALD2 + 1.2HO2 :              5.00E-14 ;
{59} R3O2 + R3O2 = 1.9ALD2 + 0.28KET + 0.37HO2 : 5.00E-14 ;
{60} HO2 + MO2 = ROOH + O2 :                 3.00E-12 ;
{61} HO2 + ETO2 = ROOH + O2 :                3.00E-12 ;
{62} HO2 + R3O2 = ROOH + O2 :                3.00E-12 ;
{63} HO2 + RAO2 = ROOH + O2 :                3.00E-12 ;
{64} HO2 + MCO3 = ROOH + O2 :                3.00E-12 ;
{65} KET + OH = KO2 :                        ARR(1.2E-11,-890) ;
{66} KO2 + NO = 0.05RAN2 + 0.95NO2 + 0.94ALD2 +
           0.94MCO3 :                        ARR(4.2E-12,180) ;
{67} KET + hv = MCO3 + ETO2 + H2O :          2.401E-6*SUN ;
{68} KET + NO3 = HNO3 + KO2 :                7.00E-16 ;
```

```
{69} KO2 + HO2 = MGLY + MO2 + H2O :          3.00E-12 ;
{70} ETHE + OH = EO2 :                       ARR(1.66E-12,474) ;
{71} EO2 + NO = NO2 + 2.0HCHO + HO2 :        ARR(4.2E-12,180) ;
{72} ALKE + OH = PO2 :                       ARR(4.1E-12,537) ;
{73} PO2 + NO = NO2 + ALD2 + HCHO + HO2 :    ARR(4.2E-12,180) ;
{74} ETHE + O3 = HCHO + 0.4CHO2 + 0.12HO2 + 0.42CO +
           0.06CH4 :                         ARR(1.2E-14,-2633) ;
{75} ALKE + O3 = 0.525HCHO + 0.5ALD2 + 0.2CHO2 +
           0.2CRO2 + 0.23HO2 + 0.215MO2 :    ARR(7.8E-14,-2105) ;
{76} CHO2 + NO = HCHO + NO2 :                7.00E-12 ;
{77} CHO2 + NO2 = HCHO+ NO3 :                7.00E-13 ;
{78} CHO2 + H2O = ACO2 :                     4.00E-18 ;
{79} CRO2 + NO = ALD2 + NO2 :                7.00E-12 ;
{80} CRO2 + NO2 = ALD2 + NO3 :               7.00E-13 ;
{81} CRO2 + H2O = ACTA :                     4.00E-18 ;
{82} EO2 + EO2 = 2.4HCHO + 1.2HO2 + 0.4ALD2 :    5.00E-14 ;
{83} PO2 + PO2 = 2.2ALD2 + 1.2HO2 :          5.00E-14 ;
{84} HO2 + EO2 = ROOH + O2 :                 3.00E-12 ;
{85} HO2 + PO2 = ROOH + O2 :                 3.00E-12 ;
{86} SO2 + CHO2 = SO4 + HCHO :               {0 ;} 7.00E-14 ;
{87} SO2 + CRO2 = SO4 + ALD2 :               {0 ;} 7.00E-14 ;
{88} ALKE + NO3 = PRN1 :                     1.26E-13 ;
{89} PRN1 + NO2 = PRN2 :                     6.80E-12 ;
{90} PRN1 + HO2 = PRPN + O2 :                3.00E-12 ;
{91} PRN1 + NO = 2NO2 + HCHO + ALD2 :        ARR(4.2E-12,180) ;
{92} CHO2 + HCHO = OZID :                    1.36E-14 ;
{93} CHO2 + ALD2 = OZID :                    1.36E-14 ;
{94} CRO2 + HCHO = OZID :                    1.36E-14 ;
{95} CRO2 + ALD2 = OZID :                    1.36E-14 ;
{96} AROM + OH = 0.84TO2 + 0.16CRES + 0.16HO2 :  1.52E-11 ;
{97} TO2 + NO = NO2 + HO2 + 0.72MGLY
      + 0.18GLYX + DIAL :                    ARR(4.2E-12,180) ;
{98} GLYX + hv = PROD :                      7.389E-5*SUN ;
{99} GLYX + OH = HO2 + 2CO + H2O :           1.15E-11 ;
{100} MGLY + hv = MCO3 + HO2 + CO :          1.755E-4*SUN ;
{101} MGLY + OH = MCO3 + CO + H2O :          1.73E-11 ;
{---- IN THE NEXT REACTION
  BETA 12, 13 WHERE REPLACED BY NUMBERS
  AS GIVEN IN LILING'S STEM CODE ----}
{102} CRES + OH = 0.83HO2 + 0.9ZO2 + 0.9TCO3
           -0.9OH - 0.0315NO2 :              4.25E-11 ;
{103} NO3 + CRES = HNO3 :                    1.00E-11 ;
{104} OH + DIAL = TCO3 :                     2.80E-11 ;
{105} TCO3 + NO2 = TPAN :                    4.70E-12 ;
```

```
{106} TPAN = TCO3 + NO2:                    ARR(2.2E+16,-13435) ;
{107} TCO3 + NO = NO2 + 0.92HO2 + 0.89GLYX + 0.11MGLY +
           0.05MCO3 :                       ARR(4.2E-12,180) ;
{108} ZO2 + NO = NO2 :                      ARR(4.2E-12,180) ;
{109} DIAL + hv = 0.98HO2 + 0.02MCO3 + TCO3 :    9.236E-5*SUN ;
{110} HO2 + TO2 = ROOH + O2 :               4.00E-12 ;
{111} HO2 + TCO3 = ROOH + O2 :              4.00E-12 ;
{112} HO2 + ZO2 = ROOH + O2 :               1.00E-12 ;
{---- HERE COMES AEROSOL PART ----}
{113} AHO2 = HCHO + HO2 :                   0 ;
{114} ISOP + OH = RIO2 :                    ARR(2.5E-11, 409) ;
{115} RIO2 + NO = 0.9NO2 + 0.45MVK + 0.45MACR +
           0.9HO2 + 0.9HCHO :               ARR(4.2E-12,180) ;
{116} RIO2 + HO2 = ROOH :                   3.00E-12 ;
{117} MVK + OH = VRO2 :                     ARR(3.0E-12,500) ;
{118} VRO2 + NO = 0.9NO2 + 0.6MCO3 + 0.6HAC +
           0.3HO2 + 0.3HCHO + 0.3MGGY :     ARR(4.2E-12,180) ;
{119} VRO2 + HO2 = ROOH :                   3.0E-12 ;
{120} OH + MACR = MAO3 :                    1.02E-11 ;
{121} MAO3 + NO2 = MPAN :                   4.7E-12 ;
{122} MPAN = MAO3 + NO2 :                   ARR(2.2E+16,-13435) ;
{123} MAO3 + NO = 3NO3 - 2NO + HO2 + MGGY : ARR(4.2E-12,180) ;
{124} MAO3 + HO2 = ROOH :                   3.00E-12 ;
{125} MACR + OH = MRO2 :                    ARR(3.86E-12,500) ;
{126} MRO2 + NO = 0.9NO2 + 0.9HO2 +
           0.9HCHO + 0.9MGGY :              ARR(4.2E-12,180) ;
{127} MRO2 + HO2 = ROOH :                   3.00E-12 ;
{128} HAC + OH = HACO :                     1.5E-11 ;
{129} HACO + NO2 = IPAN :                   4.7E-12 ;
{130} IPAN = HACO + NO2 :                   ARR(2.2E+16,-13435) ;
{131} HACO + NO = NO2 + HO2 + HCHO :        ARR(4.2E-12,180) ;
{132} HACO + HO2 = ROOH :                   3.00E-12 ;
{133} ISOP + O3 = 0.5HCHO + 0.2MVK + 0.3MACR + 0.2CHO2 +
       0.06HO2 + 0.2MVKO + 0.3MAOO : ARR(7.0E-15,-1900) ;
{134} MVK + O3  = 0.5HCHO + 0.2CHO2 + 0.21HO2 + 0.2MCRG +
           0.15ALD2 + 0.5MGGY + 0.15MCO3 : ARR(4.0E-15,-2000) ;
{135} MACR + O3 = 0.65HCHO + 0.2CHO2 + 0.36HO2
           + 0.15NO2 - 0.15NO + 0.5MGGY
           + 0.2MCRG :                      ARR(4.4E-15,-2500) ;
{136} MVKO + NO = MVK + NO2 :               ARR(4.2E-12,180) ;
{137} MVKO + NO2 = MVK + NO3 :              ARR(4.2E-13,180) ;
{138} MVKO + H2O = PROD :                   3.4E-18 ;
{139} MAOO + NO = MACR + NO2 :              ARR(4.2E-12,180) ;
{140} MAOO + NO2 = MACR + NO3 :             ARR(4.2E-13,180) ;
```

```
{141} MAOO + H2O = MACA :                    3.4E-18 ;
{142} MCRG + NO = MGGY + NO2 :               ARR(4.2E-12,180) ;
{143} MCRG + NO2 = MGGY + NO3 :              ARR(4.2E-13,180) ;
{144} MCRG + H2O = PYVA :                    3.4E-18 ;
{145} HAC + hv = HCHO + 2HO2 :               4.618E-6*SUN ;
{146} MGGY + hv = MCO3 + HO2 :               1.385E-3*SUN ;
{147} MGGY + OH = MCO3 :                     1.7E-11 ;
{148} ISOP + NO3 = INO2 :                    ARR(3.00E-12,-450) ;
{149} INO2 + NO = 2NO2 + HCHO + 0.5MVK + 0.5MACR : ARR(4.2E-12,180) ;
{150} INO2 + NO2 = PROD :                    ARR(4.2E-13,180) ;
{151} INO2 + HO2 = PROD :                    3.00E-12 ;
{152} MVK + NO3 = MVN2 :                     6.00E-14 ;
{153} MVN2 + NO = 2NO2 + HCHO + 0.5MCO3 +
           0.5MGGY + 0.5HO2 :                ARR(4.2E-12,180) ;
{154} MVN2 + HO2 = PROD :                    3.0E-12 ;
{155} MACR + NO3 = MAO3 + HNO3 :             3.3E-15 ;
{156} MACR + NO3 = MAN2 :                    6.7E-15 ;
{157} MAN2 + NO = 2NO2 + HCHO + MGGY :       ARR(4.2E-12,180) ;
{157} MAN2 + HO2 = PROD :                    3.00E-12 ;
{159} HAC + NO3 = HNO3 + HACO :              5.2E-16 ;
{160} MAOO + HO2 = ROOH :                    3.00E-12 ;
{161} MVKO + HO2 = ROOH :                    3.00E-12 ;
{162} MVKO + SO2 = SO4 + MVK :               {0 ;} 7.00E-14 ;
{163} MAOO + SO2 = SO4 + MACR :              {0 ;} 7.00E-14 ;
{164} MCRG + SO2 = SO4 + MVK :               {0 ;} 7.00E-14 ;
{165} MACA + OH = PROD :                     ARR(1.2E-11,500) ;
{166} PYVA + OH = PROD :                     5.00E-14 ;
{167} DOL6 + O3 = 0.11SUCA :                 5.44E-17 ;
{168} DOL7 + O3 = 0.19GLUA :                 3.46E-17 ;
{169} DOL8 + O3 = 0.15ADIA :                 2.21E-17 ;
{170} CPET + O3 = 0.39GLUA :                 1.03E-15 ;
{171} CHEX + O3 = 0.15ADIA :                 2.16E-16 ;
{172} OH + HO2 = H2O + O2 :                  ARR(4.6E-11,230) ;
{173} ROOH + hv = HCHO + OH + HO2 :          4.618E-6*SUN ;
{174} ROOH + OH = 0.5MO2 + 0.5OH + 0.5HCHO :1.00E-11;
{175} DMS + OH = SO2 + MSA :                 8.3E-12 ;
{176} NO3 + NO3 = 2NO2 + O2 :                ARR(8.5E-13,-2450) ;
{177} OH + PAN = PROD :                      ARR(1.23E-12,-651) ;
{178} NO3 + HO2 = 0.6OH + 0.6NO2 + 0.4HNO3 :ARR(2.3E-12,170) ;
```

APPENDIX B
CODES

B.1 Brusselator example

```
 subroutine brus(t,y,f)
 double precision t,y(2),f(2)
 f(1) = 1.d0 + y(1)**2*y(2) - 4.d0*y(1)
 f(2) = 3.d0*y(1) - y(1)**2*y(2)
 return
 end

 subroutine jac(t,y,jy)
 real*8 t,y(2),jy(2,2)
 common /FLOP/ iflop, ifa, iso, ifun, ijac
 jy(1,1) = 2.d0*y(1)*y(2) - 4.d0
 jy(1,2) = y(1)**2
 jy(2,1) = 3.d0 - 2.d0*y(1)*y(2)
 jy(2,2) = - y(1)**2
 ijac = ijac + 6
 return
 end

 subroutine direct(t,y,f)
 real*8 t,y(2),f(6)
 f(1) = 1.d0 + y(1)**2*y(2) - 4.d0*y(1)
 f(2) = 3.d0*y(1) - y(1)**2*y(2)
 f(3) = (2.d0*y(1)*y(2) - 4.d0)*y(3)
*        +  y(1)**2*y(5)
 f(4) = (2.d0*y(1)*y(2) - 4.d0)*y(4)
*        +  y(1)**2*y(6)
 f(5) = (3.d0 - 2.d0*y(1)*y(2))*y(3) - y(1)**2*y(5)
 f(6) = (3.d0 - 2.d0*y(1)*y(2))*y(4) - y(1)**2*y(6)
 return
 end
```

B.2 Merson integrator

```
      subroutine merson(ts,te,nstep,y,fun,atol,rtol)
      implicit double precision (a-h,o-z)
      parameter (n=2)
      common /count/ ic_a, ic_r
      double precision ts,te,h,y(n),dy(n),ynew(n)
      double precision k1(n),k2(n),k3(n),k4(n),k5(n)
      double precision sol,solh,err,rtol,atol
      external fun
      double precision UT,TO,US
      parameter (UT = 0.33333333333333333333d0)
      parameter (TO = 2.66666666666666666666d0)
      parameter (US = 0.16666666666666666666d0)

      step = (te-ts)/dble(nstep)
      iprint = 0

      t = ts

      do i=1,nstep
      t1 = ts + (i-1)*step
      t2 = ts + i*step
      t = t1
      h = 1.d-9
      write(10,10) t,(y(k),k=1,n)

      do while (t.lt.t2)

      if (t+h.gt.t2) then
        h = t2 - t
      end if


C --- Stage 1 -----
      call fun(t,y,k1)
C --- Stage 2 -----
      do j=1,n
        dy(j) = y(j)+h*(ut*k1(j))
      end do
      call fun(t+ut*h,dy,k2)
C --- Stage 3 -----
      do j=1,n
        dy(j) = y(j)+h*us*(k1(j)+k2(j))
      end do
```

```
      call fun(t+ut*h,dy,k3)
C --- Stage 4 -----
      do j=1,n
        dy(j) = y(j)+h*(1.25d-1*k1(j)+to*k3(j))
      end do
      call fun(t+5.d-1*h,dy,k4)
C --- Stage 5 -----
      do j=1,n
        dy(j) = y(j)+h*(5.d-1*k1(j)-1.5d0*k3(j)+2.d0*k4(j))
      end do
      call fun(t+h,dy,k5)

      err = 0.d0
      do j=1,n
        sol  = h*(us*k1(j)+2.d0/3.d0*k4(j)+us*k5(j))
        solh = h*(1.d-1*k1(j)+3.d-1*k3(j)+
     *               4.d-1*k4(j)+2.d-1*k5(j))
        ynew(j) = y(j) + sol
        err  = err + ( (sol-solh)/(atol+rtol*dabs(ynew(j))) )**2
      end do
      err = dsqrt(err/dble(n))

      if (err.lt.0.9)  then
        do j=1,n
          y(j) = ynew(j)
        end do
        t = t+h
        ic_a = ic_a + 1
      else
        ic_r = ic_r + 1
      end if
      h = dmin1(10.d0,9.d-1*h/err**(2.5d-1))
      if (t+h.eq.t) then
        print *,'Abort. Stepsize too small = ',h
        stop
      end if


      end do

      end do
      write(10,10) te,(y(k),k=1,n)
 10   format(F6.2,20(X,E24.16))
```

```
      return
      end
```

B.3 Adifor generated - Merson integrator

```
      program test
        double precision ts, te, y(2)
        double precision g_y(2, 2)
C
        ts = 0.d0
        te = 1.d+1
        nstep = 200000
        g_y(1,1) = 1.d0
        g_y(1,2) = 0.d0
        g_y(2,1) = 0.d0
        g_y(2,2) = 1.d0
        y(1) = 1.d0
        y(2) = 1.d0
C
        call g_test_brus(2, ts, te, nstep, y, g_y, 2)
        stop
      end
C
      subroutine g_test_brus(g_p_, ts, te, nstep, y, g_y, ldg_y)
        double precision y(2), ts, te
        external brus
        integer g_pmax_
        parameter (g_pmax_ = 2)
        integer g_i_, g_p_, ldg_y
        double precision g_y(ldg_y, 2)
        integer g_ehfid
        external g_brus
        external g_merson
        data g_ehfid /0/
C
c3        call ehsfid(g_ehfid, 'test_brus','g_brus.f')
C
        if (g_p_ .gt. g_pmax_) then
          print *, 'Parameter g_p_ is greater than g_pmax_'
          stop
        endif
        open (unit = 10, file = 'Adifor.m')
```

```
        call g_merson(g_p_, ts, te, nstep, 2, y, g_y, ldg_y, g_brus)
        stop
      end
C
C
      subroutine g_brus(g_p_, t, y, g_y, ldg_y, f, g_f, ldg_f)
        double precision t, y(2), f(2)
        integer g_pmax_
        parameter (g_pmax_ = 2)
        integer g_i_, g_p_, ldg_f, ldg_y
        double precision d1_p, d3_v, d6_b, d2_v, d4_b, g_f(ldg_f, 2), g_
     *y(ldg_y, 2)
        integer g_ehfid
        data g_ehfid /0/
C
c3        call ehsfid(g_ehfid, 'brus','g_brus.f')
C
        if (g_p_ .gt. g_pmax_) then
          print *, 'Parameter g_p_ is greater than g_pmax_'
          stop
        endif
        d2_v = y(1) * y(1)
        d1_p = 2.0d0 * y(1)
        d4_b = -4.d0 + y(2) * d1_p
        do g_i_ = 1, g_p_
          g_f(g_i_, 1) = d2_v * g_y(g_i_, 2) + d4_b * g_y(g_i_, 1)
        enddo
        f(1) = 1.d0 + d2_v * y(2) - 4.d0 * y(1)
C--------
        d3_v = y(1) * y(1)
        d1_p = 2.0d0 * y(1)
        d6_b = (-y(2)) * d1_p + 3.d0
        do g_i_ = 1, g_p_
          g_f(g_i_, 2) = (-d3_v) * g_y(g_i_, 2) + d6_b * g_y(g_i_, 1)
        enddo
        f(2) = 3.d0 * y(1) - d3_v * y(2)
C--------
        return
      end
C
C
      subroutine g_merson(g_p_, ts, te, nstep, n1, y, g_y, ldg_y, g_fun)
        parameter (n = 2)
        double precision ts, te, h, y(n), dy(n)
```

```
        double precision k1(n), k2(n), k3(n), k4(n), k5(n)
        double precision ut, to, us
        parameter (ut = 0.33333333333333333333d0)
        parameter (to = 2.66666666666666666666d0)
        parameter (us = 0.16666666666666666666d0)
C
        double precision d1
        integer g_pmax_
        parameter (g_pmax_ = 2)
        integer g_i_, g_p_, ldg_y
        double precision d9_b, d6_b, d7_b, g_dy(2,2), g_y(2,2),
     * g_k2(2,2), g_k3(2,2), g_k4(2,2), g_k5(2,2), g_k1(2,2)
        integer g_ehfid
        save g_dy, g_k2, g_k3, g_k4, g_k5
        external g_fun
        data g_ehfid /0/
C
c3        call ehsfid(g_ehfid, 'merson','g_brus.f')
C
        if (g_p_ .gt. g_pmax_) then
          print *, 'Parameter g_p_ is greater than g_pmax_'
          stop
        endif
        h = (te - ts) / dble(nstep)
        iprint = 0
C
        t = ts
        write(10,10) t,(y(k),k=1,n),((g_y(i1,i2),i2=1,n),i1=1,n)

C
        do i = 1, nstep
C
C
C --- Stage 1 -----
          call g_fun(g_p_, t, y, g_y, ldg_y, k1, g_k1, g_pmax_)
C --- Stage 2 -----
          do j = 1, n
            do g_i_ = 1, g_p_
              g_dy(g_i_, j) = g_y(g_i_, j)+ h * (ut * g_k1(g_i_,j))
            enddo
            dy(j) = y(j) + h * (ut * k1(j))
C--------
          enddo
          d1 = t + ut * h
```

```
          call g_fun(g_p_, d1, dy, g_dy, g_pmax_, k2, g_k2, g_pmax_)
C --- Stage 3 -----
          do j = 1, n
            d6_b = h * us
            do g_i_ = 1, g_p_
              g_dy(g_i_, j) = d6_b * g_k2(g_i_, j) + g_y(g_i_, j)
            enddo
            dy(j) = y(j) + h * (us * k1(j) + us * k2(j))
C--------
          enddo
          d1 = t + ut * h
          call g_fun(g_p_, d1, dy, g_dy, g_pmax_, k3, g_k3, g_pmax_)
C --- Stage 4 -----
          do j = 1, n
            d6_b = h * to
            do g_i_ = 1, g_p_
              g_dy(g_i_, j) = d6_b * g_k3(g_i_, j) + g_y(g_i_, j)
            enddo
            dy(j) = y(j) + h * (1.25d-1 * k1(j) + to * k3(j))
C--------
          enddo
          d1 = t + 5.d-1 * h
          call g_fun(g_p_, d1, dy, g_dy, g_pmax_, k4, g_k4, g_pmax_)
C --- Stage 5 -----
          do j = 1, n
            d7_b = h * 2.d0
            d9_b = (-h) * 1.5d0
            do g_i_ = 1, g_p_
              g_dy(g_i_, j) = d7_b * g_k4(g_i_, j) + d9_b * g_k3(g_i_, j
     *) + g_y(g_i_, j)
            enddo
            dy(j) = y(j) + h * (5.d-1 * k1(j) - 1.5d0 * k3(j) + 2.d0 * k
     *4(j))
C--------
          enddo
          d1 = t + h
          call g_fun(g_p_, d1, dy, g_dy, g_pmax_, k5, g_k5, g_pmax_)
C
          do j = 1, n
            d7_b = h * us
            d9_b = h * (2.d0 / 3.d0)
            do g_i_ = 1, g_p_
              g_y(g_i_, j) = d7_b * g_k5(g_i_, j) + d9_b * g_k4(g_i_, j)
     * + g_y(g_i_, j) + d7_b * g_k1(g_i_, j)
```

```
            enddo
            y(j) = y(j) + h * (us * k1(j) + 2.d0 / 3.d0 * k4(j) + us * k
     *5(j))
C--------
          enddo
C
          t = t + h
C
          iprint = iprint + 1
          if (iprint .eq. (nstep / 200)) then
            write(10,10) t,(y(k),k=1,n),((g_y(i1,i2),i2=1,n),i1=1,n)
10          format (f6.2,20(x,e24.16))
            iprint = 0
          endif
C
        enddo
C
        return
      end
```

B.4 Implicit Euler integrator

```
      subroutine euler(ts,te,nstep,n1,y,fun,jac)
      implicit double precision (a-h, o-z)
      common /FLOP/ iflop, ifa, iso, ifun, ijac
      parameter (n=2)
      real*8 ts,te,h,y(n),f(n),j(n,n),z(n)
      real*8 x(n),er
      external fun, jac

      iprint = 0
      h = (te-ts)/dble(nstep)
      write(10,10) ts,(y(k),k=1,n)

C --- Begin Time Loop ---
      do i=1,nstep
      t = ts + (i-1)*h

      do k=1,n
        z(k) = y(k)
      end do
```

```
C --- Compute and factorize the Jacobian ---
      call jac(t,y,j)
      do i1=1,n
        do i2 = 1,n
          j(i1,i2) = -h*j(i1,i2)
        end do
        j(i1,i1) = 1.d0 + j(i1,i1)
        iflop = iflop + 1
      end do
      call fact(j)

C ---- Begin Quasi Newton -----
      er = 1.d0
      icont = 0
      do while ((er.gt.1.d-2).and.(icont.le.10))
      icont = icont + 1
        call fun(t,y,f)
        do k=1,n
          x(k) = y(k) - z(k) - h*f(k)
        end do
        call solve(j,x)
        er = 0.d0
        do k=1,n
          y(k) = y(k) - x(k)
          er = er + x(k)**2
        end do
        er = dsqrt(er/n)
        iflop = iflop + 1 + n
      end do
C ---- End Quasi Newton -----


      iprint = iprint+1
      if (iprint.eq.(nstep/100)) then
       write(10,10) t+h,(y(k),k=1,n)
 10    format(F6.2,20(X,E24.16))
       iprint = 0
      end if

      end do
C --- End Time Loop ---
      return
```

```
      end

      subroutine fact(a)
      double precision a(2,2),x1
      common /FLOP/ iflop, ifa, iso, ifun, ijac
      x1 = a(2,1)
      if ( dabs(a(1,1)).lt.1.d-8 ) then
        print *,'Singular matrix'
        stop
      end if
      a(2,1) = a(2,1)/a(1,1)
      a(2,2) = a(2,2) - a(1,2)*x1/a(1,1)
      ifa = ifa + 3
      return
      end

      subroutine solve(a,x)
      common /FLOP/ iflop, ifa, iso, ifun, ijac
      double precision a(2,2),x(2)
      x(2) = (-a(2,1)*x(1)+x(2))/a(2,2)
      x(1) = (x(1)-a(2,1)*x(2))/a(1,1)
      iso = iso + 4
      return
      end
```

B.5 DDM - Implicit Euler

```
      subroutine euler_ddm(ts,te,nstep,n1,y,g_y,fun,jac)
      implicit double precision (a-h, o-z)
      common /FLOP/ iflop, ifa, iso, ifun, ijac
      parameter (n=2)
      real*8 ts,te,h,y(n),f(n),j(n,n),z(n)
      real*8 x(n),er,g_y(n,n),p(n*n+n),r(n*n+n)
      external fun, jac

      iprint = 0
      h = (te-ts)/dble(nstep)

C --- Compute and factorize the Jacobian ---
      call jac(ts,y,j)
      do i1=1,n
        do i2 = 1,n
```

```
          j(i1,i2) = -h*j(i1,i2)
        end do
        j(i1,i1) = 1.d0 + j(i1,i1)
      end do
      call fact(j)

      write (10, 10) ts, (y(k), k = 1, n),
     *       g_y(1,1), g_y(1,2),g_y(2,1),g_y(2,2)

C --- Begin Time Loop ---
      do i=1,nstep
      t = ts + (i-1)*h

      do k=1,n
        z(k) = y(k)
      end do

C ---- Begin Quasi Newton -----
      er = 1.d0
      icont = 0
      do while ((er.gt.1.d-2).and.(icont.le.10))
        icont = icont + 1
        er = 0.d0
        call brus(t,y,f)
        do k=1,n
          x(k) = y(k) - z(k) - h*f(k)
        end do
        call solve(j,x)
        do k=1,n
          y(k) = y(k) - x(k)
          er = er + x(k)**2
        end do
        er = dsqrt(er/(n))
        iflop = iflop + 1 + n
      end do
C ---- End Quasi Newton -----

C ----- Sensitivities ---
        r(1) = y(1)
        r(2) = y(2)
        r(3) = g_y(1,1)
        r(4) = g_y(1,2)
        r(5) = g_y(2,1)
        r(6) = g_y(2,2)
```

```
        call direct(t,r,p)
        x(1) =  - h*p(3)
        x(2) =  - h*p(5)
        call solve(j,x)
        g_y(1,1) = r(3) - x(1)
        g_y(2,1) = r(5) - x(2)
        x(1) =  - h*p(4)
        x(2) =  - h*p(6)
        call solve(j,x)
        g_y(1,2) = r(4) - x(1)
        g_y(2,2) = r(6) - x(2)
        iflop = iflop + 4


C --- Compute and factorize the Jacobian ---
      call jac(t,y,j)
      do i1=1,n
        do i2 = 1,n
          j(i1,i2) = -h*j(i1,i2)
        end do
        j(i1,i1) = 1.d0 + j(i1,i1)
      end do
      call fact(j)


      iprint = iprint+1
      if (iprint.eq.(nstep/100)) then
       write(10,10) t+h,(y(k),k=1,n),
     *         g_y(1,1), g_y(1,2),g_y(2,1),g_y(2,2)
 10    format(F6.2,20(X,E24.16))
       iprint = 0
      end if

      end do
C --- End Time Loop ---
      return
      end
```

B.6 Adifor - generated Implicit Euler

```
C                         DISCLAIMER
```

```
C
C   This file was generated on 06/30/97 by the version of
C   ADIFOR compiled on Apr 11 1997.
C
C   ADIFOR was prepared as an account of work sponsored by an
C   agency of the United States Government, Rice University, and
C   the University of Chicago.  NEITHER THE AUTHOR(S), THE UNITED
C   STATES GOVERNMENT NOR ANY AGENCY THEREOF, NOR RICE UNIVERSITY,
C   NOR THE UNIVERSITY OF CHICAGO, INCLUDING ANY OF THEIR EMPLOYEES
C   OR OFFICERS, MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR ASSUMES
C   ANY LEGAL LIABILITY OR RESPONSIBILITY FOR THE ACCURACY, COMPLETE-
C   NESS, OR USEFULNESS OF ANY INFORMATION OR PROCESS DISCLOSED, OR
C   REPRESENTS THAT ITS USE WOULD NOT INFRINGE PRIVATELY OWNED RIGHTS.
C
C
C
      program test
        common /FLOP/ iflop, ifa, iso, ifun, ijac
        real*8 ts, te, y(2)
        external g_brus, g_jac
        double precision g_y(2,2)
        open (10, file = 'Euler.m')
        iflop = 0
        ifa = 0
        iso = 0
        ifun = 0
        ijac = 0
        ts = 0.d0
        te = 1.d+1
        nstep = 1000000
        y(1) = 1.d0
        y(2) = 1.d0
        g_y(1,1) = 1.d0
        g_y(1,2) = 0.d0
        g_y(2,1) = 0.d0
        g_y(2,2) = 1.d0
        call g_euler(2, ts, te, nstep, n1, y, g_y,
     *               2, g_brus, g_jac)
        write(6,*) '   Iflop  Ifa    Iso    Ifun   Ijac'
        write(6,12) iflop, ifa, iso, ifun, ijac
 12     format(10(2X,I9))
       write(6,*) 'Total flops = ', iflop+ifa+iso+ifun+ijac
        stop
      end
```

```
      subroutine g_brus(g_p_, t, y, g_y, ldg_y, f, g_f, ldg_f)
        common /FLOP/ iflop, ifa, iso, ifun, ijac
        real*8 t, y(2), f(2)
        integer g_pmax_
        parameter (g_pmax_ = 2)
        integer g_i_, g_p_, ldg_f, ldg_y
        double precision d1_p, d3_v, d6_b, d2_v, d4_b, g_f(ldg_f, 2), g_
     *y(ldg_y, 2)
        integer g_ehfid
        data g_ehfid /0/
C
        !3 call ehsfid(g_ehfid, 'brus','g_euler.f')
C
        if (g_p_ .gt. g_pmax_) then
          print *, 'Parameter g_p_ is greater than g_pmax_'
          stop
        endif
        d2_v = y(1) * y(1)
        d1_p = 2.0d0 * y(1)
        d4_b = -4.d0 + y(2) * d1_p
        ifun = ifun + 3
        do g_i_ = 1, g_p_
          g_f(g_i_, 1) = d2_v * g_y(g_i_, 2) + d4_b * g_y(g_i_, 1)
          ifun = ifun + 2
        enddo
        f(1) = 1.d0 + d2_v * y(2) - 4.d0 * y(1)
          ifun = ifun + 2
C--------
        d3_v = y(1) * y(1)
        d1_p = 2.0d0 * y(1)
        d6_b = (-y(2)) * d1_p + 3.d0
          ifun = ifun + 3
        do g_i_ = 1, g_p_
          g_f(g_i_, 2) = (-d3_v) * g_y(g_i_, 2) + d6_b * g_y(g_i_, 1)
          ifun = ifun + 2
        enddo
        f(2) = 3.d0 * y(1) - d3_v * y(2)
          ifun = ifun + 2
C--------
        return
      end
C
      subroutine g_jac(g_p_, t, y, g_y, ldg_y, jy, g_jy, ldg_jy)
```

```
      common /FLOP/ iflop, ifa, iso, ifun, ijac
      real*8 t, y(2), jy(2, 2)
      integer g_pmax_
      parameter (g_pmax_ = 2)
      integer g_i_, g_p_, ldg_jy, ldg_y
      double precision d1_p, d5_b, d2_v, g_jy(ldg_jy, 2, 2), g_y(ldg_y
     *, 2)
      integer g_ehfid
      data g_ehfid /0/
C
      !3 call ehsfid(g_ehfid, 'jac','g_euler.f')
C
      if (g_p_ .gt. g_pmax_) then
        print *, 'Parameter g_p_ is greater than g_pmax_'
        stop
      endif
      d2_v = 2.d0 * y(1)
      d5_b = y(2) * 2.d0
      do g_i_ = 1, g_p_
        g_jy(g_i_, 1, 1) = d2_v * g_y(g_i_, 2) + d5_b * g_y(g_i_, 1)
      enddo
      jy(1, 1) = d2_v * y(2) - 4.d0
C--------
      d2_v = y(1) * y(1)
      d1_p = 2.0d0 * y(1)
      do g_i_ = 1, g_p_
        g_jy(g_i_, 1, 2) = d1_p * g_y(g_i_, 1)
      enddo
      jy(1, 2) = d2_v
C--------
      d2_v = 2.d0 * y(1)
      d5_b = (-y(2)) * 2.d0
      do g_i_ = 1, g_p_
        g_jy(g_i_, 2, 1) = (-d2_v) * g_y(g_i_, 2) + d5_b * g_y(g_i_, 1
     *)
      enddo
      jy(2, 1) = 3.d0 - d2_v * y(2)
C--------
      d2_v = y(1) * y(1)
      d1_p = 2.0d0 * y(1)
      do g_i_ = 1, g_p_
        g_jy(g_i_, 2, 2) = (-d1_p) * g_y(g_i_, 1)
      enddo
      jy(2, 2) = -d2_v
```

```
C--------
        ijac = ijac + 22
        return
       end
C
C
       subroutine g_euler(g_p_, ts, te, nstep, n1, y, g_y, ldg_y, g_fun,
     *g_jac)
        implicit double precision (a-h, o-z)
        common /FLOP/ iflop, ifa, iso, ifun, ijac
        parameter (n = 2)
        real*8 ts, te, h, y(n), f(n), j(n, n), z(n)
        real*8 x(n), er
C
        integer g_pmax_
        parameter (g_pmax_ = 2)
        integer g_i_, g_p_, ldg_y
        double precision g_z(g_pmax_, n), g_y(ldg_y, n), g_j(g_pmax_, n,
     * n), g_x(g_pmax_, n), g_f(g_pmax_, n)
        integer g_ehfid
        save g_z, g_j, g_x, g_f
        external g_solve
        external g_fun
        external g_fact
        external g_jac
        data g_ehfid /0/
C
        !3 call ehsfid(g_ehfid, 'euler','g_euler.f')
C
        if (g_p_ .gt. g_pmax_) then
          print *, 'Parameter g_p_ is greater than g_pmax_'
          stop
        endif
        iprint = 0
        h =  (te-ts)/dble(nstep)! 5.d-3
        write (10, 10) ts, (y(k), k = 1, n),
     *         g_y(1,1), g_y(2,1),g_y(1,2),g_y(2,2)
C
C --- Begin Time Loop ---
        do i = 1, nstep
          t = ts + (i - 1) * h
C
          do k = 1, n
            do g_i_ = 1, g_p_
```

```
              g_z(g_i_, k) = g_y(g_i_, k)
            enddo
            z(k) = y(k)
C--------
          enddo
C
C
C --- Compute and factorize the Jacobian ---
          call g_jac(g_p_, t, y, g_y, ldg_y, j, g_j, g_pmax_)
          do i1 = 1, n
            do i2 = 1, n
              do g_i_ = 1, g_p_
                g_j(g_i_, i1, i2) = (-h) * g_j(g_i_, i1, i2)
                iflop = iflop + 1
              enddo
              j(i1, i2) = (-h) * j(i1, i2)
                iflop = iflop + 1
C--------
            enddo
            do g_i_ = 1, g_p_
              g_j(g_i_, i1, i1) = g_j(g_i_, i1, i1)
            enddo
            j(i1, i1) = 1.d0 + j(i1, i1)
C--------
          enddo
          call g_fact(g_p_, j, g_j, g_pmax_)
C
C ---- Begin Quasi Newton -----
          er = 1.d0
          icont = 0
          do while ((er .gt. 1.d-2) .and. (icont .le. 10))
            icont = icont + 1
            call g_fun(g_p_, t, y, g_y, ldg_y, f, g_f, g_pmax_)
            do k = 1, n
              do g_i_ = 1, g_p_
                g_x(g_i_, k) = (-h) * g_f(g_i_, k) + (-g_z(g_i_, k)) + g
     *_y(g_i_, k)
                iflop = iflop + 2
              enddo
              x(k) = y(k) - z(k) - h * f(k)
              iflop = iflop + 1
C--------
            enddo
            call g_solve(g_p_, j, g_j, g_pmax_, x, g_x, g_pmax_)
```

```
            er = 0.d0
            do k = 1, n
              do g_i_ = 1, g_p_
                g_y(g_i_, k) = -g_x(g_i_, k) + g_y(g_i_, k)
              enddo
              y(k) = y(k) - x(k)
C--------
              er = er + x(k) ** 2
              iflop = iflop + 6
            enddo
            er = dsqrt(er / n)
            iflop = iflop + 2
          enddo
C ---- End Quasi Newton -----
C
C
          iprint = iprint + 1
          if (iprint .eq. (nstep / 100)) then
            write (10, 10) t + h, (y(k), k = 1, n),
     *         g_y(1,1), g_y(2,1),g_y(1,2),g_y(2,2)
10          format (f6.2,20(x,e24.16))
            iprint = 0
          endif
C
        enddo
C --- End Time Loop ---
        return
      end
C
      subroutine g_fact(g_p_, a, g_a, ldg_a)
        common /FLOP/ iflop, ifa, iso, ifun, ijac
        double precision a(2, 2), x1
        integer g_pmax_
        parameter (g_pmax_ = 2)
        integer g_i_, g_p_, ldg_a
        double precision d7_b, d6_b, d5_b, d4_b, d3_b, d6_v, d3_v, d2_b,
     * g_x1(g_pmax_), g_a(ldg_a, 2, 2)
        integer g_ehfid
        save g_x1
        data g_ehfid /0/
C
        !3 call ehsfid(g_ehfid, 'fact','g_euler.f')
C
        if (g_p_ .gt. g_pmax_) then
```

```
          print *, 'Parameter g_p_ is greater than g_pmax_'
          stop
        endif
        do g_i_ = 1, g_p_
          g_x1(g_i_) = g_a(g_i_, 2, 1)
        enddo
        x1 = a(2, 1)
C--------
        if (dabs(a(1, 1)) .lt. 1.d-8) then
          print *, 'Singular matrix'
          stop
        endif
        d3_v = a(2, 1) / a(1, 1)
        d2_b = 1.0d0 / a(1, 1)
        d3_b = (-d3_v) / a(1, 1)
        do g_i_ = 1, g_p_
          g_a(g_i_, 2, 1) = d3_b * g_a(g_i_, 1, 1) + d2_b * g_a(g_i_, 2,
     * 1)
        enddo
        a(2, 1) = d3_v
C--------
        d6_v = a(1, 2) * x1 / a(1, 1)
        d4_b = -(1.0d0 / a(1, 1))
        d5_b = -((-d6_v) / a(1, 1))
        d6_b = d4_b * x1
        d7_b = d4_b * a(1, 2)
        do g_i_ = 1, g_p_
          g_a(g_i_, 2, 2) = d5_b * g_a(g_i_, 1, 1) + d7_b * g_x1(g_i_) +
     * d6_b * g_a(g_i_, 1, 2) + g_a(g_i_, 2, 2)
        enddo
        a(2, 2) = a(2, 2) - d6_v
C--------
        ifa = ifa+19
        return
      end
C
      subroutine g_solve(g_p_, a, g_a, ldg_a, x, g_x, ldg_x)
        common /FLOP/ iflop, ifa, iso, ifun, ijac
        double precision a(2, 2), x(2)
        integer g_pmax_
        parameter (g_pmax_ = 2)
        integer g_i_, g_p_, ldg_x, ldg_a
        double precision d8_b, d7_b, d6_b, d7_v, d8_v, d3_b, d2_b, g_x(l
     *dg_x, 2), g_a(ldg_a, 2, 2)
```

```
        integer g_ehfid
        data g_ehfid /0/
C
        !3 call ehsfid(g_ehfid, 'solve','g_euler.f')
C
        if (g_p_ .gt. g_pmax_) then
          print *, 'Parameter g_p_ is greater than g_pmax_'
          stop
        endif
        d8_v = ((-a(2, 1)) * x(1) + x(2)) / a(2, 2)
        d2_b = 1.0d0 / a(2, 2)
        d3_b = (-d8_v) / a(2, 2)
        d7_b = d2_b * (-a(2, 1))
        d8_b = -(d2_b * x(1))
        do g_i_ = 1, g_p_
          g_x(g_i_, 2) = d3_b * g_a(g_i_, 2, 2) + d2_b * g_x(g_i_, 2) +
     *d7_b * g_x(g_i_, 1) + d8_b * g_a(g_i_, 2, 1)
        enddo
        x(2) = d8_v
C--------
        d7_v = (x(1) - a(2, 1) * x(2)) / a(1, 1)
        d2_b = 1.0d0 / a(1, 1)
        d3_b = (-d7_v) / a(1, 1)
        d6_b = (-d2_b) * x(2)
        d7_b = (-d2_b) * a(2, 1)
        do g_i_ = 1, g_p_
          g_x(g_i_, 1) = d3_b * g_a(g_i_, 1, 1) + d7_b * g_x(g_i_, 2) +
     *d6_b * g_a(g_i_, 2, 1) + d2_b * g_x(g_i_, 1)
        enddo
        x(1) = d7_v
C--------
        iso = iso+28
        return
      end
C
```

REFERENCES

[1] T. Alishenas and Ö. Ólafsson. Modeling and velocity stabilization of constrained mechanical systems with comparative study of two test problems. Preprint, NADA, Royal Institute of Technology, Stockholm, 1993.

[2] H. Amann. *Ordinary Differential Equations: An Introduction to Nonlinear Analysis.* Walter de Gruyter, 1990.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. LAPACK User's Guide, second edition. Technical report, SIAM, Philadelphia, PA, 1995.

[4] U. M. Ascher, S. J. Ruuth, and B. T. R. Wetton. Implicit-Explicit methods for time dependent PDE's. *Technical Report 93-15*, 1993.

[5] R.D. Atkinson, D.L. Baulch, R.A. Cox, R.F.JR. Hampson, J.A. Kerr, and J. Troe. Evaluated kinetic and photochemical data for atmospheric chemistry. *International Journal of Chemical Kinetics*, 21:115–190, 1989.

[6] G. Bader and P. Deuflhard. A semi-implicit mid-point rule for stiff systems of ordinary differential equations. *Numer. Math.*, 41:373–398, 1983.

[7] Ch. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland. ADIFOR generating derivative codes from FORTRAN programs. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1992.

[8] Ch. Bischof, A. Carle, P. Khademi, and A. Mauer. The ADIFOR2.0 system for the automatic differentiation of FORTRAN77 programs. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, 1994.

[9] H.G. Bock. Numerical treatment of inverse problems in chemical reaction kinetics. *Modelling of Chemical Reaction Systems, K.H. Ebert, P. Deuflhard and W. Jaeger editors, Springer Series in Chem. Phys.*, 18:102–125, 1981.

[10] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* Elsevier Science Publishers, 1989.

[11] P.N. Brown, G.D. Byrne, and A.C. Hindmarsh. VODE: A Variable Step ODE Solver. *SIAM J. Sci. Stat. Comput.*, 10:1038–1051, 1989.

[12] G.D. Byrne and A.M. Dean. The numerical solution of some chemical kinetics models with VODE and CHEMKIN II. *Computers Chem.*, 17:297–302, 1993.

[13] D. G. Cacuci. Sensitivity theory for nonlinear systems. I. Nonlinear functional analysis approach. II. Extensions to additional classes of responses. *J. Math. Phys.*, 22:2794–2812, 1981.

[14] G.R. Carmichael, L.K. Peters, and T. Kitada. A second generation model for regional-scale transport/ chemistry/ deposition. *Atmospheric environment*, 20:173–188, 1986.

[15] G.R. Carmichael, A. Sandu, and F.A. Potra. Sensitivity Analysis for Atmospheric Chemistry models via Automatic Differentiation. *Atmospheric Environment*, 31:475 – 489, 1997.

[16] B.W. Char, K.O. Geddes, G.H. Gonnet, M.B. Monagan, and S.M. Watt. *Maple V Language Reference Manual.* Springer-Verlag, New York, 1991.

[17] M. Chin, D. Jacob, J. Munger, D. Parrish, and B. Doddridge. Relationship of ozone and carbon monoxide over north america. *J. Geophys. Res.*, 99:14,565–14,573, 1994.

[18] Y. S. Cho. Ph.d. thesis. *The University of Iowa*, 1986.

[19] Y. S. Cho and G.R. Carmichael. Evaluation of liquid phase chemical production of sulfate using sensitivity analysis. *Atmospheric Environment*, 20:l959–l988, 1986.

[20] Y. S. Cho, G.R. Carmichael, and H. Rabitz. Sensitivity analysis of the advection-diffusion equation. *Atmospheric Environment*, 21:2589–2598, 1987.

[21] Y. S. Cho, G.R. Carmichael, and H. Rabitz. The relationship between primary emissions and acid deposition in eulerian models determined by sensitivity analysis. *Water, Air and Soil Pollution*, 40:9–31, 1988.

[22] D. P. Chock and S. L. Winkler. A comparison of advection algorithms coupled with chemistry. *Atmospheric Environment*, 28(16):2659–2675, 1994.

[23] D. Dabdub and J.H. Seinfeld. Extrapolation techniques used in the solution of stiff odes associated with chemical kinetics of air quality models. *Atmospheric Environment*, 29:403–410, 1995.

[24] V. Damian-Iordache. KPP - a chemical development environment. Technical report, The University of Iowa, Iowa City, IA 52246, 1996.

[25] V. Damian-Iordache, A. Sandu, M. Damian-Iordache, G. R. carmichael, and F. A. Potra. KPP - A symbolic preprocessor for chemistry kinetics - User's guide. Technical report, The University of Iowa, Iowa City, IA 52246, 1995.

[26] J. J. B. de Swart and J. G. Blom. Experiences with sparse matrix solvers in parallel ODE software. Technical report, Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, 1995.

[27] J. E. Dennis. On the kanrovitch hypothesis for Newton's method. *SIAM Journal on Numerical Analysis*, 6:493–507, 1969.

[28] J. E. Dennis and R. B. Schnabel. *Numerical Methods for unconstrained optimization and nonlinear equations.* Prentice Hall Inc, Englewood Cliffs, New Jersey 07632, 1985.

[29] R. Dentener and P. Crutzen. Reaction of N_2O_5 on tropospheric aerosols : impact of the global distributions of NO_x, O_3 and OH. *Journal of Geophysical Research*, 98:7149–7163, 1993.

[30] P. Deuflhard. Recent progress in extrapolation methods for ordinary differential equations. *SIAM Review*, 27:505–535, 1985.

[31] J. J. Dongarra, J. R. Bunch, C. B. Moller, and G. W. Stewart. LINPACK User's Guide. Technical report, SIAM, Philadelphia, PA, 1979.

[32] J.J. Dongarra and E. Grosse. Distribution of software via electronic mail. *Communications ACM*, pages 403–407, 1987.

[33] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Oxford Science Publications, Clarendon Press Oxford, 1986.

[34] A. M. Dunker. The decoupled direct method for calculating sensitivity coefficients in chemical kinetics. *J. Chemical Physics*, 81:2385, 1984.

[35] S.C. Eisenstat, M.C. Gursky, M.H. Schultz, and A.H. Sherman. Yale Sparse Matrix Package. ii. The nonsymmetric codes. Research Report 114, Department of Computer Science, Yale University, 1977.

[36] S.C. Eisenstat, M.C. Gursky, M.H. Schultz, and A.H. Sherman. Yale Sparse Matrix Package. i. The symmetric codes. *Int. J. Num. Meth. Eng.*, 18:1145–1151, 1982.

[37] A.S. El-Bakry, R.A. Tapia, T. Tsuchia, and Y. Zhang. On the Formulation of the Primal-Dual Newton Interior-Point Method for Nonlinear Programming. *To appear in Journal of Optimization Theory and Applications*, 1996.

[38] S. Elliot, R.P. Turco, and M.Z. Jacobson. Tests on combined projection/forward differencing integration for stiff photochemical family systems at long time step. *Computers Chem*, 17:91–102, 1993.

[39] M. W. Gery, G.Z. Whitten, J.P. Killus, and M.C. Dodge. A photochemical kinetics mechanism for urban and regional scale computer modelling. *Journal of Geophysical Research*, 94:12925–12956, 1989.

[40] G. Golub and C. F. van Loan. *Matrix computaions.* Johns Hopkins University Press, Baltimore and London, 1983.

[41] W. Gong and H.R. Cho. A numerical scheme for the integration of the gas phase chemical rate equations in 3D atmospheric models. *Atmospheric Environment*, 27A:2147–2160, 1993.

[42] A. Griewank, C. Bischof, G. Corliss, A. Carle, and K. Williamson. Derivative convergence for iterative equation solvers. *Optimization methods and software*, 2:321–355, 1993.

[43] A. Griewank and G. Corliss. Automatic differentiation of algorithms: Theory, implementation, and application. *SIAM, Philadelphia, Pennsylvania*, 1991.

[44] E. Hairer, Ch. Lubich, and M. Roche. *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods.* Springer-Verlag, Berlin, New-York, 1989.

[45] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems.* Springer-Verlag, Berlin, 1993.

[46] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems.* Springer-Verlag, Berlin, 1991.

[47] O. Hertel, R. Berkowicz, J. Christensen, and O. Hov. Test of two numerical schemes for use in atmospheric transport-chemistry models. *Atmospheric Environment*, 27A:2591–2611, 1993.

[48] E. Hesstvedt, O. Hov, and I. Isaacsen. A numerical method to predict secondary air pollutants with an application on oxidant generation in an urban atmosphere. *WMO publication*, 510:219–226, 1978.

[49] E. Hesstvedt, O. Hov, and I. Isaacsen. Quasi-steady-state-approximation in air pollution modelling: comparison of two numerical schemes for oxidant prediction. *Int. J. Chem. Kinet.*, 10:971–994, 1978.

[50] A. Hindmarsch. *ODEPACK: A systematized collection of ODE solvers.* Ed. North Holland, Amsterdam, 1983.

[51] M. Hochbruck, C. Lubich, and H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM Journal of Scientific Computing*, to appear, 1997.

[52] D.J. Jacob, J.A. Logan, G.M. Gardner, C.M. Spivakovsky R.M. Yevich, S.C. Wofsy, S. Sillman, and M.J. Prather. Factors regulating ozone over the United States and its export to the global atmosphere. *J. Geophys. Res.*, 98:14,817–14,826, 1993.

[53] M.Z. Jacobson and R.P. Turco. SMVGEAR: a sparse-matrix, vectorized Gear code for atmospheric models. *Atmospheric Environment*, 17:273–284, 1994.

[54] L. O. Jay. Structure-Preserving Integrators. *University of Minnesota AHPCRC, Preprint 95-038*, 1995.

[55] L.O. Jay, A. Sandu, F.A. Potra, and G.R. Carmichael. Improved QSSA methods for atmospheric chemistry integration. *SIAM Journal on Scientififc Computing*, 18:182–202, 1997.

[56] R. J. Kee, F. M. Rupley, and J. A. Miller. CHEMKIN II: A FORTRAN package for the analysis of gas phase chemical kinetics. Technical report, Sandia National Laboratory, Livermore, CA, 1989.

[57] X. Lin, M. Trainer, and S. Liu. On the nonlinearity of tropospheric ozone production. *Journal of Geophysical Research*, 93:15,879–15,888, 1988.

[58] F.W. Lurmann, A.C. Loyd, and R. Atkinson. A chemical mechanism for use in long-range transport/acid deposition computer modeling. *Journal of Geophysical Research*, 91:10,905–10,936, 1986.

[59] J. Matthijsen. Private Communication. 1995.

[60] G.J. McRae, W.R. Goodin, and J.H. Seinfeld. Numerical solution of the atmospheric diffusion equation for chemically reacting flows. *Journal of Computational Physics*, 45:1–42, 1982.

[61] I.M. Navon and U. Muller. FESW - A finite element FORTRAN IV program for solving the shallow water equations. *Advances in ehngineering software*, 1:77–84, 1970.

[62] Hoa D. Nguyen and Seungho Paik. Solution Domain decomposition with Finite Difference Methods for PDE. *Numerical methods for PDE*, 11:453–466, 1995.

[63] Jorge Nocedal. Theory of Algorithms for Unconstrained Optimization. *Acta Numerica*, pages 1–37, 1991.

[64] U. Nowak. A short user's guide to LARKIN. Technical report, Konrad-Zuse-Zentrum fuer, Informationstechnik Berlin, 1982.

[65] J. Olson, M. Prather, T. Berntsen, G. R. Carmichael, R. Chatfield, P. Connell, R. Derwent, L. Horowitz, S. Jin, M. Kanakidou, P. Kasibhatla, R. Kotomarthi, M. Kuhn, K. Law, S. Sillman, J. Penner, L. Perliski, F. Stordal, A. Thompson, and O. Wild. Results from the IPCC Photochemical Model Intercomparison (PhotoComp): Some Insights into Tropospheric Chemistry. *submitted to Journal of Geophysical Research*, March 1996.

[66] K. Olszyna, E. Bailey, R. Simonaites, and J. Meagher. O_3 and NO_y relationships at a rural site. *Journal of Geophysical Research*, 99:14,557–14,563, 1994.

[67] D. Parrish, J. Holloway, M. Trainer, P. Murphy, G. Forbes, and F. Fehsenfeld. Export of north american ozone pollution to the north atlantic ocean. *Science*, 259:1436–1439, 1993.

[68] F.A. Potra, K. Kortanek, and Y. Ye. On some efficient interior point methods for nonlinear convex programming. *Linear Algebra and its Applications*, 152:191–222, 1991.

[69] M.J.D. Powell. Convergence properties of algorithms for nonlinear optimization. Report DAMTP 1985/NA1, University of Cambridge, Department of Applied Mathematics and Theoretical Physics, Cambridge, October 1985.

[70] M. Prather. Intercomparison of tropospheric chemistry/ transport models. *Scientific assesment of ozone depletion, World meteorological organization*, 1995.

[71] A. Prothero and A. Robinson. On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations. *Math. of Comput.*, 28:145–162, 1974.

[72] H. Rabitz, M. Hramer, and D. Dacol. Sensitivity analysis in chemical kinetics. *Anual review of physical chemistry*, 34, 1983.

[73] D. Ralph and S. Wright. Superlinear convergence of an interior point method for monotone variational inequalities. *Preprint MCS-P556-0196*, Argonne National Laboratory, 1996.

[74] A. Sandu, J. G. Blom, E. Spee, J. G. Verwer, F.A. Potra, and G.R. Carmichael. Benchmarking stiff ODE solvers for atmospheric chemistry equations II - Rosenbrock Solvers. Report on Computational Mathematics 90, The University of Iowa, Department of Mathematics, Iowa City, July 1996.

[75] A. Sandu, F.A. Potra, V. Damian, and G.R. Carmichael. Efficient implementation of fully implicit methods for atmospheric chemistry. *Journal of Computational Physics*, 129:101 – 110, 1996.

[76] A. Sandu, M. van Loon, F.A. Potra, G.R. Carmichael, and J. G. Verwer. Benchmarking stiff ODE solvers for atmospheric chemistry equations I - Implicit vs. Explicit. Report on Computational Mathematics 85, The University of Iowa, Department of Mathematics, Iowa City, January 1996.

[77] R. D. Saylor and G. D. Ford. On the comparison of numerical methods for the integration of kinetic equations in atmospheric chemistry and transport models. *Atmospheric Environment*, 29:2585–2593, 1995.

[78] A.H. Sherman and A.C. Hindmarsh. GEARS: a package for the solution of sparse, stiff ordinary differential equations. *Lawrence Livermore Laboratory Report*, UCRL-84102.

[79] D. Shyan-Shu Shieh, Y. Chang, and G.R. Carmichael. The evaluation of numerical techniques for solution of stiff ODE arising from chemical kinetic problems. *Environmental Software*, 3, 1988.

[80] S. Sillman. A numerical solution for the equations of tropospheric chemistry based on an analysis of sources and sinks of odd hydrogen. *Journal of Geophysical Research*, 96:20735–20744, 1991.

[81] D. Simpson. Biogenic VOC in Europe. Part II: implications for ozone control strategies. *EMEP MSC-W*, 1994.

[82] D. Simpson, Y. Andersson-Skold, and M.E. Jenkin. Updating the chemical scheme for the EMEP MSC-W oxidant model: current status. *EMEP MSC-W*, Technical Report 2/93, 1993.

[83] S. Skelboe and Z. Zlatev. Exploiting the natural partitioning in the numerical solution of ODE systems arising from atmospheric chemistry. Report, University of Copenhagen, Department of Computer Science, Copenhagen, Denmark, 1996.

[84] D. Stoffer. Variable steps for reversible integration methods. *Computing*, 55:1–22, 1995.

[85] M. van Loon. Numerical smog prediction I: the physical and chemical model. *CWI Report NM-R9411*, 1995.

[86] M. van Loon. Numerical smog prediction II: Grid refinement and its application to the Dutch smog prediction model. *CWI Report NM-R95xx*, 1995.

[87] J. Verwer. Gauss-Seidel iterations for stiff ODEs from chemical kinetics. *SIAM Journal of Scientific Computing*, 15:1243–1250, 1994.

[88] J. Verwer, J. G. Blom, and W. Hunsdorfer. An Implicit-Explicit Approach for Atmospheric Transport-Chemistry Problems. *Applied Numerical Mathematics*, 20:191–209, 1996.

[89] J. Verwer, J. G. Blom, M. van Loon, and E. J. Spee. A comparison of stiff ODE solvers for atmospheric chemistry problems. *Atmospheric Environment*, 30:49–58, 1996.

[90] J. Verwer and W. Hunsdorfer. A note on Splitting Errors for Advection-Reaction Equations. *CWI Report NM-R9424*.

[91] J. Verwer and D. Simpson. Explicit Methods for Stiff Odes from Atmospheric Chemistry. *Applied Numerical Mathematics*, 18:413–430, 1995.

[92] J. Verwer and M. van Loon. An evaluation of explicit Pseudo-Steady-State Approximation achemes for stiff ODE systems from chemical kinetics. *Journal of Computational Physics*, 113:347–352, 1994.

[93] P. Werbos. Applications of advances in nonlinear sensitivity analysis. *System modelling and optimization*, Springer-Verlag:762–777, 1982.

[94] R. Yamartino, J. Scire, G.R. Carmichael, and Y.S. Chang. The CALGRID mesoscale photochemical grid model. *Atmospheric Environment*, 26 A:1493–1512, 1992.

[95] N. N. Yanenko. *The method of fractional steps.* Springer-Verlag, New-York, Heidelberg, Berlin, 1971.

[96] T. R. Young and J. P. Boris. A numerical technique for solving stiff ODE associated with the chemical kinetics of reactive flow problems. *Journal of Physical Chemistry*, 81:2424–2427, 1977.

[97] Z. Zlatev. *Computer Treatment of Large Air Pollution Models.* Kluwer Academic Publishers, 1995.