RWTH Aachen

Department of Computer Science

Technical Report

Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler

Uwe Naumann and Jan Riehme

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2008-6

RWTH Aachen · Department of Computer Science · March 2008

Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler

Uwe Naumann[1] and Jan Riehme[2]

[1] LuFG Informatik 12, RWTH Aachen University, Germany
[2] Department of Computer Science, University of Hertfordshire, Hatfield, UK

Abstract. We present the final report for a collaborative research project with the Federal Waterways Engineering and Research Institute, Karsruhle. A numerical model of a dune implemented in Sisyphe is considered. Sensitivity analysis is performed using a tangent-linear model that is generated automatically by the differentiation-enabled NAGWare Fortran compiler.

1 Motivation

We consider a simplified numerical model of a (sand) dune on the ground of a channel whose shape and position changes due to flowing water. The given experiment is implemented in Sisyphe[3] [8] using the programming language Fortran 90. Our aim is to determine the sensitivity of the dune's shape with respect to the roughness of the material (sand etc.). The grid consists of 891 mesh points. Values for roughness and height (of the dune) are associated with each mesh point. Two tasks need to be solved:

1. Computation of the full Jacobian $F' \in I\!R^{891\times 891}$.
2. Computation of the sum over the columns of F', that is, $F' \cdot \mathbf{e}$, where $\mathbf{e} = (1, \ldots, 1)^T \in I\!R^{891}$.

The following three approaches are used:

1. A hand-written tangent-linear code available at the Federal Waterways Engineering and Research Institute;
2. finite difference quotients;
3. a tangent-linear code that is automatically generated by a prototype of the differentiation-enabled NAGWare Fortran compiler [12].

2 Introduction to Automatic Differentiation

Automatic Differentiation(AD) is a method for computing derivatives of functions implemented as numerical simulation programs automatically. Refer to [1–4] for an impressive collection of successful applications of AD to a wide variety of real-world application. Information on tools, publications, and applications can also be obtained from the communities web portal

`www.autodiff.org`.

[3] See website `www.telemacsystem.com`.

We consider an implementation of a nonlinear multivariate vector function

$$\mathbf{y} = F(\mathbf{x}, \mathbf{z}), \quad F : I\!R^{n+\tilde{n}} \to I\!R^m \tag{1}$$

as a (part of a) Fortran code. We are interested in the computation of directional derivatives that are products of the Jacobian matrix

$$F' = F'(\mathbf{x}, \mathbf{z}) \equiv \left(\frac{\partial y_j}{\partial x_i}\right)_{i=1,\ldots,n}^{j=1,\ldots,m}$$

containing the sensitivities (partial derivatives) of all *active* outputs (or *dependent* variables) $\mathbf{y} = (y_1, \ldots, y_m)^T$ with respect to the active inputs $\mathbf{x} = (x_1, \ldots, x_n)^T$ (or *independent* variables) with a vector $\dot{\mathbf{x}} \in I\!R^n$. The vector $\mathbf{z} \in I\!R^{\tilde{n}}$ contains all passive inputs, i.e. variables that $\mathbf{y}$ depends on but whose impact on the sensitivity of $\mathbf{y}$ is of no interest in the given context. W.o.w., we are looking for ways to evaluate the *tangent-linear model (TLM)* of $\mathbf{y} = F(\mathbf{x})$ defined as

$$\dot{\mathbf{y}} = \dot{F}(\mathbf{x}, \dot{\mathbf{x}}, \mathbf{z}) \equiv F'(\mathbf{x}, \mathbf{z}) \cdot \dot{\mathbf{x}} \quad .$$

In the following we discus a number of example codes, that can be obtained from our website.[4]

2.1 Approximation of the TLM by Finite Difference Quotients

Finite difference quotients (FD) can be used to compute an approximation of the TLM as follows:

$$\dot{\mathbf{y}} \approx \frac{F(\mathbf{x} + h \cdot \dot{\mathbf{x}}, \mathbf{z}) - F(\mathbf{x}, \mathbf{z})}{h} \quad . \tag{2}$$

The quality of this approximation depends to a large extend on the user's ability to pick the "right" value for h. For complex simulations this search amounts to trial and error. Moreover the objective remains unclear since the quality of a given approximation cannot be evaluated without knowledge of exact values. **Trusting FD approximations of derivatives is dangerous.** Refer to [9] for further discussion of this method. An impressive illustration of the harm inflicted by approximate derivatives in the context of a matrix-free Truncated Newton algorithm [6] for unconstrained nonlinear optimization can be found in [10].

Example: Jacobian by FD Consider the code in Listing 1.1. The example illustrates the use of FD for accumulating an approximation of the Jacobian matrix consisting of the partial derivatives of x(1) and x(2) on the left-hand[5] side of the assignments in lines 56 and 57 with respect to x(1) and x(2) on the right-hand side of both assignments. The code of interest is preceeded and succeeded by further computations. We choose a perturbation parameter h = 10^{-6} (line 32). Three function evaluations (line 42) are required when using forward differences as in Equation (2). The code is run at the original point (i=0) and at two perturbed (line 52) points (i=1,2). The original point needs to be stored (line 44) as x is overwritten in lines 56 and 57. The sensitivities are written into the text file `sensiFD.dat` (line 65). The unperturbed function value is returned to the caller of subroutine foo (lines 75 and 83).

[4] `http://wiki.stce.rwth-aachen.de/bin/view/Projects/ADExamples/WebHome`

[5] x at the left-hand side replaces y in Equation (1).

The main program sets the original point x (lines 8 and 9) followed by calling foo (line 13) and it prints the sensitivities (lines 17–23).

The same example is used throughout this section. The setup is intentional. With this example we aim to show the application of differentiation to a selected section of a numerical simulation code. A similar approach is followed when tangent-linear code is generated by the NAGWare Fortran compiler [11].

A compilation and execution log of Listing 1.1 is given in Figure 1.

Listing 1.1: Jacobian by FD (`jacFD.f95`)

```
1  PROGRAM jacFD
2
3    IMPLICIT NONE
4    DOUBLE PRECISION :: x(2)
5    DOUBLE PRECISION :: sens(2)
6    INTEGER                          :: i
7
8    x(1) =  2.D0
9    x(2) =  3.D0
10
11   PRINT *,'INPUT              ', x
12
13   CALL foo ( x )
14
15   PRINT *,'FUNCTION VALUES    ', x
16
17   OPEN( UNIT=111, FILE='sensiFD.dat' )
18   PRINT *,'JACOBIAN'
19   DO i=1,2
20      READ(111,*) sens
21      PRINT *, sens
22   END DO
23   CLOSE( 111 )
24
25 CONTAINS
26
27   SUBROUTINE  foo ( x )
28     IMPLICIT NONE
29     DOUBLE PRECISION, INTENT(INOUT) :: x(2)
30     DOUBLE PRECISION                :: t,  xorig(2)
31     DOUBLE PRECISION                :: Forig(2), ForigFD(2)
32     DOUBLE PRECISION                :: h = 1.D-6
33     INTEGER                         :: i
34
35     !save inputs for multiple runs
36     xorig = x
37
38     ! open output file
39     OPEN( UNIT=111, FILE='sensiFD.dat' )
40
41     ! loop to get all columns of the Jacobian
42     DO i = 0, 2
43        !restore  input values
44        x = xorig
45
46
47        ! initial computations, sensitivities not wanted
```

```
      t = x(1) + x(2)

      ! perturbed x(i) by h in second and third iteration
      IF ( i .GT. 0 ) THEN
         x(i) = x(i) + h
      END IF

      ! computations that senstivities are wanted for
      x(1) = SIN( t * x(1) / x(2) )
      x(2) = COS( t * x(2) / x(1) )

      ! HARVESTING I    sensitivities
      IF ( i .EQ. 0 ) THEN
         ! store results of unperturbed inputs for FD
         ForigFD = x
      ELSE
         ! output approximation of sensitivities
         WRITE( 111,* )  (x - ForigFD ) / h
      END IF

      ! final computations,  sensitivities not wanted
      x(1) = COS(x(1))
      x(2) = SIN(x(2))

      ! HARVESTING II  function values
      IF ( i .EQ. 0 ) THEN
         ! store final results of unperturbed inputs
         Forig = x
      END IF
   END DO

   ! closing output file
   CLOSE( 111 )

   ! restore unperturbed results
   x = Forig
  END SUBROUTINE foo

END PROGRAM jacFD
```

```
examples> F95 -fpp -O4 -o jacFD jacFD.f95
examples> ./jacFD

 INPUT                  2.0000000000000000   3.0000000000000000
 FUNCTION VALUES        0.9818968118936291  -0.8333826108478083
 JACOBIAN
  -1.6361230767425461  -1.1560945738597894E+02
   1.0907486486244622  81.8333640053570122
```

Fig. 1: Compilation and execution trace of Listing 1.1

Example: Sum over Columns of Jacobian by FD The example in Listing 1.2 illustrates the use of FD for computing an approximation of the sum over all columns of the Jacobian matrix.[6] Two function evaluations are required when using forward differences as in Equation (2) (line 38). The code is run at the original point (i=0) and at a perturbed (line 48) point (i=1). This time the perturbation is applied to both entries of x simultaneously (vector operation). Everything else is similar to the previous example.

A compilation and execution log of Listing 1.2 is given in Figure 2.

Listing 1.2: Sum over Columns of Jacobian by FD (`jacvecFD.f95`)

```
1  PROGRAM jacvecFD
2  
3    IMPLICIT NONE
4    DOUBLE PRECISION :: x(2)
5    DOUBLE PRECISION :: sens(2)
6  
7    x(1) =  2.D0
8    x(2) =  3.D0
9  
10   PRINT *,'INPUT               ', x
11 
12   CALL foo ( x )
13 
14   PRINT *,'FUNCTION VALUES     ', x
15 
16   OPEN( UNIT=111, FILE='sensiFD.dat' )
17   READ(111,*) sens
18   PRINT *,'SENSITIVITIES       ', sens
19   CLOSE( 111 )
20 
21 CONTAINS
22 
23   SUBROUTINE  foo ( x )
24     IMPLICIT NONE
25     DOUBLE PRECISION, INTENT(INOUT) :: x(2)
26     DOUBLE PRECISION                :: t,  xorig(2)
27     DOUBLE PRECISION                :: Forig(2), ForigFD(2)
28     DOUBLE PRECISION                :: h = 1.D-8
29     INTEGER                         :: i
30 
31     !save inputs for multiple runs
32     xorig = x
33 
34     ! open output file
35     OPEN( UNIT=111, FILE='sensiFD.dat' )
36 
37     ! loop to get a single Jacobian vector product
38     DO i = 0, 1
39        !restore  input values
40        x = xorig
41 
42 
43        ! initial computations, sensitivities not wanted
44        t = x(1) + x(2)
45 
46        ! perturb ALL inputs x(i) by h in second loop interation
47        IF ( i .GT. 0 ) THEN
```

[6] Also called row-sum, since for any row of the Jacobian all elements of the row-vectors are added.

```fortran
        x = x + h
     END IF

     ! computations that sensitivities are wanted for
     x(1) = SIN( t * x(1) / x(2) )
     x(2) = COS( t * x(2) / x(1) )

     ! HARVESTING I   sensitivities
     IF ( i .EQ. 0 ) THEN
        ! store results of unperturbed inputs for FD
        ForigFD = x
     ELSE
        ! output approximation of sensitivities
        WRITE( 111,* )  (x - ForigFD ) / h
     END IF

     ! final computations,  sensitivities not wanted
     x(1) = COS(x(1))
     x(2) = SIN(x(2))

     ! HARVESTING II  function values
     IF ( i .EQ. 0 ) THEN
        ! store final results of unperturbed inputs
        Forig = x
     END IF
  END DO

  ! closing output file
  CLOSE( 111 )

  ! restore unperturbed results
  x = Forig
END SUBROUTINE foo

END PROGRAM jacvecFD
```

```
examples>F95 -fpp -O4 -o jacvecFD jacvecFD.f95
examples>./jacvecFD

 INPUT                  2.0000000000000000   3.0000000000000000
 FUNCTION VALUES        0.9818968118936291  -0.8333826108478083
 SENSITIVITIES         -0.5453744039707331 -34.1142907589464528
```

Fig. 2: Compilation and execution trace of Listing 1.2

2.2 Exact TLM by Forward Mode AD

Forward mode AD [7] yields tangent-linear codes for the computation of directional derivatives with machine accuracy. During a semantic modification process the code for F is transformed into code for $\dot{F}$. The Jacobian F' is accumulated column-wise by letting $\dot{\mathbf{x}}$ range over the Cartesian basis vectors in $I\!R^n$. Sparsity in F' can be exploited by merging structurally orthogonal columns as described in [5].

Example: Jacobian by Forward Mode AD The example in Listing 1.3 illustrates the use of the tangent-linear code for accumulating the Jacobian matrix. Two evaluations of the tangent-linear code are required (line 45). The derivative components of x are initialized to the two Cartesian basis vectors in $\mathbb{R}^2$ (lines 53 and 55). Their values are written into a text file after running the relevant code (line 62). The tangent-linear code is generated internally by the compiler. Runtime support routines are defined in `compad_scalar_module.mod` (line 6). Access to the function and derivative components of active program variables is guided by special macros (lines 16,20,25,26, and 62) that are defined in the special preprocessor file `compad_component_handler.fpp` (line 1). Further details go beyond the scope of this document.

A compilation and execution log of Listing 1.3 is given in Figure 3.

Listing 1.3: Jacobian by forward mode AD (`jacAD.f95`)

```
1  #include "compad_component_handler.fpp"
2
3
4  PROGRAM jacAD
5
6    USE compad_scalar_module
7
8    IMPLICIT NONE
9    DOUBLE PRECISION :: x(2)
10   DOUBLE PRECISION :: sens(2)
11   INTEGER                        :: i
12
13   x(1) =  2.D0
14   x(2) =  3.D0
15
16   PRINT *,'INPUT               ', COMPADVAL( x )
17
18   CALL foo ( x )
19
20   PRINT *,'FUNCTION VALUES     ', COMPADVAL( x )
21
22   OPEN( UNIT=111, FILE='sensi.dat' )
23   PRINT *,'JACOBIAN'
24   DO i=1,2
25      READ(111,*) COMPADDRVCOMP(sens)
26      PRINT *, COMPADDRV( sens )
27   END DO
28   CLOSE( 111 )
29
30 CONTAINS
31
32   SUBROUTINE  foo ( x )
33     IMPLICIT NONE
34     DOUBLE PRECISION, INTENT(INOUT) :: x(2)
35     DOUBLE PRECISION                :: t,  xorig(2)
36     INTEGER                         :: i
37
38     !save inputs for multiple runs
39     xorig = x
40
41     ! open output file
42     OPEN( UNIT=111, FILE='sensi.dat' )
```

```
    ! loop to get all columns of the Jacobian
    DO i = 1, 2
       !restore  input values
       x = xorig

       ! initial computations, sensitivities not wanted
       t = x(1) + x(2)

       ! SEEDING I  :  reset all sensitivities of x to ZERO
       CALL SEED( x , 0.D0 )
       ! SEEDING II :  set initial sensitive of i-th input
       CALL SEED( x(i) , 1.D0 )

       ! computations that sensitivities are wanted for
       x(1) = SIN( t * x(1) / x(2) )
       x(2) = COS( t * x(2) / x(1) )

       ! HARVESTING
       WRITE( 111,* )  COMPADDRV(x)

       ! final computations,  sensitivities not wanted
       x(1) = COS(x(1))
       x(2) = SIN(x(2))

    END DO

    ! closing output file
    CLOSE( 111 )
  END SUBROUTINE foo

END PROGRAM jacAD
```

```
examples>F95 -fpp -ad -ad_ovl -ad_scalar \
     -DCOMPADII_ACCESS_COMPADTYPE_COMPONENTS  \
     -O4 -o jacAD jacAD.f95  -lcompadII_scalar -L.
examples>./jacAD

 INPUT                  2.0000000000000000   3.0000000000000000
 FUNCTION VALUES        0.9818968118936291  -0.8333826108478083
 JACOBIAN
  -1.6361233411851317  -1.1583543760732589E+02
   1.0907488941234211  81.7209476588213590
```

Fig. 3: Compilation and execution trace of Listing 1.3

Example: Sum over Columns of Jacobian by Forward Mode AD The example in Listing 1.4 illustrates the use of the tangent-linear code for computing the sum over all columns of the Jacobian matrix. A single evaluation of the tangent-linear code suffices. Both derivative components of x are initialized to one. Their values contain the sum over the Jacobian entries in the corresponding rows after running the relevant code section. They are written into a text file (line 48) and printed by the main program (line 24).

A compilation and execution log of Listing 1.4 is given in Figure 4.

Listing 1.4: Sum over Columns of Jacobian by forward mode AD (`jacvecAD.f95`)

```
#include "compad_component_handler.fpp"

PROGRAM jacvecAD

  USE compad_scalar_module

  IMPLICIT NONE
  DOUBLE PRECISION :: x(2)
  DOUBLE PRECISION :: sens(2)

  x(1) =  2.D0
  x(2) =  3.D0

  PRINT *,'INPUT               ', COMPADVAL( x )

  CALL foo ( x )

  PRINT *,'FUNCTION VALUES     ', COMPADVAL( x )

  OPEN( UNIT=111, FILE='sensi.dat' )
  READ(111,*) COMPADDRVCOMP(sens)
  CLOSE( 111 )
  PRINT *,'SENSITIVITIES       ', COMPADDRV( sens )

CONTAINS

  SUBROUTINE  foo ( x )
    IMPLICIT NONE
    DOUBLE PRECISION, INTENT(INOUT) :: x(2)
    DOUBLE PRECISION                :: t

    ! initial computations, sensitivities not wanted
    t = x(1) + x(2)

    ! SEEDING
    CALL SEED( x(1) , 1.D0 )
    CALL SEED( x(2) , 1.D0 )

    ! computations that sensitivities are wanted for
    x(1) = SIN( t * x(1) / x(2) )
    x(2) = COS( t * x(2) / x(1) )

    ! HARVESTING
    OPEN( UNIT=111, FILE='sensi.dat' )
    WRITE( 111,* )  COMPADDRV(x)
    CLOSE( 111 )

    ! final computations,  sensitivities not wanted
    x(1) = COS(x(1))
    x(2) = SIN(x(2))

  END SUBROUTINE foo

END PROGRAM jacvecAD
```

```
examples>F95 -fpp -ad -ad_ovl -ad_scalar \
     -DCOMPADII_ACCESS_COMPADTYPE_COMPONENTS  \
     -O4 -o jacvecAD jacvecAD.f95  -lcompadII_scalar -L.
examples>./jacvecAD

 INPUT                   2.0000000000000000   3.0000000000000000
 FUNCTION VALUES         0.9818968118936291  -0.8333826108478083
 SENSITIVITIES          -0.5453744470617106 -34.1144899485045130
```

Fig. 4: Compilation and execution trace of Listing 1.4

3 Forward Sensitivities for Sisyphe

The situation in Sisyphe is mostly similar to that described in the previous section. Following the description of the general approach to get a compiler-generated tangent-linear version of the code, we discuss the necessary changes to the subroutine **sisyphe**[7] required to compute and access the sensitivities within Sisyphe.

3.1 Compiler-Generated Tangent-Linear Code

Applying the AD-enabled compiler to Sisyphe in order to generate a tangent-linear version of the code is a much more complex task than that of getting the simple example from Section 1 differentiated. The complete source code is divided into five libraries (`bief`, `special`, `damocles`, `paravoid`, and `sisyphe`). The AD-enabled compiler is applied to all of them[8] to generate the corresponding tangent-linear libraries.

The compilation with the AD-enabled compiler is referred as *active compilation*, whereas *passive compilation* denotes compilation without differentiation of the code. To generate tangent-linear code of Sisyphe with the AD-enabled NAGWare Fortran compiler some modifications of the source code are required: For instance, the code generated by the AD-compiler replaces all calls of intrinsic functions (**sin**, **dim**, **sum**, ...) by overloaded calls of runtime support routines defined in `compad_scalar_module.mod`. Therefore these specifiers cannot be specified no longer as **intrinsic** operations. See Section A.2 for details.

Active compilation of tangent-linear code transforms all floating-point variables into variables of data type **compad_type** (also called *active* type) defined by the AD-enabled compiler, and inserts code operating on the active variables that computes sensitivities in sync with the values of the dependent variables (function values). The **compad_type** for tangent-linear code consists of two double precision components **val** and **drv** representing the value and the sensitivity (derivative), repectively:

Listing 1.5: Compiler defined active data type **compad_type**

```
type compad_type
   sequence
   double precision :: val
   double precision :: drv
end type compad_type
```

[7] contained in the file `sisyphe/sisyphe_v5p4/sources/sisyphe.f` of Sisyphe version 5.4

[8] That is all source code files are compiled into tangent-linear code with only one exception: File `lit.f` of from library `bief` must be compiled without differentiation. See Section A.3 for details and how to adapt `lit.f` manually.

This transformation of data types has a significant impact on any `I/O`-operation that deals with variables of *real* data type (see Sections 3.1 and A.2).

Activated `I/O` in the Compiler-Generated Tangent Linear Code Every `I/O`-statement that reads or writes a real variable in the original source code reads or writes two reals in the tangent-linear code, since the variable referenced by the `I/O`-statement is now of the active **`compad_type`**. Thus the original data files are not a valid input to the tangent-linear code since they do not contain data for the sensitivity components. In order to get the tangent-linear code working with the original data files every **`read`**-operation of active variables is restricted to their value component.

Moreover in the tangent-linear code every `I/O`-statement with format specifications that contains *real*, *engineering*, or *scientific* edit descriptors becomes invalid: Since active variables consist of two real components the format specifications in the differentiated program needs to specify two real edit descriptors per active variable.

In the case of Sisyphe the restriction of all active arguments in `I/O`-statements to their value component keeps all format specifications valid. This restriction can be imposed easily by a small set of Fortran-preprocessor (fpp) macros used to encapsulate all active variables in `I/O`-statements. Moreover the encapsulation by the component access macros avoids the creation of a separate AD-enabled source branch.

Listing 1.6 shows the file `compad_component_handler.fpp` defining the preprocessor macros **`COMPADVALCOMP`** (line 3) and **`COMPADVAL`** (line 5) that encapsulate the value component for active compilation[9] and leave the variable unchanged for passive compilation (lines 12 and 13):

Listing 1.6: code/compad_component_handler.fpp

```
 1 #ifdef COMPADII_ACCESS_COMPADTYPE_COMPONENTS
 2        ! access VALUE component of a COMPAD_TYPE object  (l-value)
 3 #define  COMPADVALCOMP(a) a%val
 4        ! return VALUE of COMPAD_TYPE object   (r-value)
 5 #define  COMPADVAL(a)     VALUE(a)
 6        ! access DERIV component of a COMPAD_TYPE object  (l-value)
 7 #define  COMPADDRVCOMP(a) a%drv
 8        ! return DERIV of COMPAD_TYPE object   (r-value)
 9 #define  COMPADDRV(a)     DERIV(a)
10 #else
11        ! define dummies for passive compilation
12 #define  COMPADVALCOMP(a) a
13 #define  COMPADVAL(a)     a
14 #define  COMPADDRVCOMP(a) NoDerivativeInPassiveMode
15 #define  COMPADDRV(a)     NoDerivativeInPassiveMode
16 #endif
```

The macro **`COMPADVALCOMP`** must be used wherever an "l-value"[10] is required (**`READ`** - statements, assignments). If only a "r-value"[11] is needed, then **`COMPADVAL`** should be used.

[9] only if the conditional define **COMPADII_ACCESS_COMPADTYPE_COMPONENTS** was defined at compile time.

[10] A left value denotes something that is valid on the left-hand side of an assignment, i.e. the value are written by the statement.

[11] Right values cannot be written (i.e. changed) by a statement. They cannot be left values. However left values can be used as right values too.

Note that the macros **COMPADDRVCOMP** (line 14) and **COMPADDRV** (line 15) for harvesting (i.e. accessing) the derivative component of an active variable will cause an compilation error in passive compilation mode, since there is simply no derivative component available. Therefore every source code with accesses to derivative components will not compile in passive mode. However this is not a general restriction that disables the compilation of passive and active variants from one and the same source code branch: Access to derivative components usually occur only in driver programs[12] that call differentiated subroutines. A clean interface to the differentiated code is required. It should be simple enough to get the required declarations and initialization of all active parameters done correctly by hand. The driver is compiled in passive mode.

For Sisyphe it is more appropriate to develop the driver from the subroutine **sisyphe**[13], and compile the driver routine with the AD-enabled compiler avoiding any manual activation. Moreover, since the complete package is compiled actively, there is no need to examine and understand the complex calling hierarchy of the system. The major drawback of this convenient approach is the loss of efficency that arises from the automatic activation of all floating-point variables.[14] As a minor implication the resulting subroutine **sisyphe** can no longer be compiled in passive mode.[15]

For programs that can *restart* an interrupted program execution by restoring the required values from previously stored snapshots, the restriction of all `I/O`-statements to value components yields an error: If derivatives are not contained in the snapshots, then sensitivities will not be propagated across restarts. Hence, the calculated sensitivities are correct if the program runs uninterrupted and wrong after a restart. To get correct sensitivities even after restarts the snapshots must be written and read with sensitivities.

The usage of the component access macros is illustrated by their application to the driver subroutine **sisyphe**. Original line numbers denote lines in the unchanged original source code. See Section A.3 for more applications of the component access macros.

Include the macro definitions to make them available (insert after original line *90*):

```
#include "compad_component_handler.fpp"
```

Change original lines *1573* and *1574* to encapsulate the floating-point variable **XMAX** which is written with explicit format specifications (labels 371 and 372):

```
371      FORMAT(1X,'EVOLUTION MAXIMUM  : ',G16.7,' NOEUD : ',I6)
372      FORMAT(1X,'MAXIMAL EVOLUTION  : ',G16.7,' NODE  : ',I6)
         IF (LNG.EQ.1) WRITE(LU,371) COMPADVAL(XMAX),IMAX
         IF (LNG.EQ.2) WRITE(LU,372) COMPADVAL(XMAX),IMAX
```

Similar changes are required in lines *1578*, *1579*, *1613*, *1614*, *1617*, *1618*, and *1725* to *1736* in the original code.

[12] A driver program is a passively compiled program or subroutine, that calls the differentiated code. Seeding and harvesting of derivatives is done in the driver. In the examples from Section 2.2 the driver program and the code to be differentiated were mixed for the sake of simplicity.

[13] Source file `sisyphe/sisyphe_v5p4/sources/sisyphe.f` of the Sisyphe version 5.4.

[14] This is due to the lack of activity analysis within the AD-enabled compiler, which is currently under development.

[15] Use conditional compilation **#ifdef COMPADII_ACCESS_COMPADTYPE_COMPONENTS** and **#endif** to bypass this.

3.2 Sum over Columns of Jacobian by Compiler-Generated TLM

To compute the sum over all columns of the Jacobian only one execution of the tangent-linear code is required. The derivative components of all independent variables **CHESTR%R** $\in I\!R^{891}$ are initialized to one.

This can be done by calling the routine **SEED** from the runtime library of the AD-enabled compiler (insert after original line *294*):

```
#ifdef COMPADII    ! do that only for active compilation
C     Seeding with vector of ONES to sum over the columns of the Jacobian
      CALL SEED(CHESTR%R, 1.D0) ! vector operation
#endif
```

The conditional define **COMPADII** is used to compile any AD-related code only during active compilation. Thus passive code can be obtained from the activated source code tree by passive compilation if `COMPADII` is not defined.

The computed derivatives $\dot{\mathbf{y}} = F' \cdot \dot{\mathbf{x}} = \frac{\partial F}{\partial \mathbf{x}} \cdot \dot{\mathbf{x}}$ of the dependent variables $\mathbf{y} =$ **ZF%R** $\in I\!R^{891}$ with respect to the independents $\mathbf{x} =$ **CHESTR%R** are stored in the Sisyphe internal variable **PRIVE** by extracting the derivatives[16] with the fpp-macro **COMPAD_DRV** (insert after original line *1622*):

```
#ifdef COMPADII    ! do that only for active compilation
C  special output of the derivatives to TECPLOT
       PRIVE%ADR(1)%P%R = COMPADDRV(ZF%R)
#endif
```

If the letter '**A**' is specified in `VARIABLES FOR GRAPHIC PRINTOUTS` in the Sisyphe input file (*cas_le.txt* in the case of *BOSSE* example), then the sensitivities of any printed time step appear in the result file. To visualize the derivatives with *TecPlot* one can choose the field **PRIVE** as color on the surface of the dune.

These two modifications of `sisyphe.f` and the changes from section 3.1 transform the subroutine **sysiphe** into a driver program that computes the sum over the Jacobian's columns.

3.3 Jacobian by Compiler-Generated TLM

A tangent-linear code needs to propagate n Cartesian basis vectors to accumulate the complete Jacobian

$$F' = F'(\mathbf{x}) \equiv \left(\frac{\partial y_j}{\partial x_i}\right)_{i=1,\ldots,n}^{j=1,\ldots,m}$$

of a multivariate function F. The driver program for Sisyphe needs to execute $n = 891$ times the routine **sisyphe**. Therefore a loop (*AD-loop)* containing the relevant computational parts of the subroutine **sisyphe** is introduced into the subroutine itself. The ith execution (variable **COMPAD_COUNTER** is used as loop counter) of the loop body propagates the ith Cartesian basis

[16] To be more exact: **PRIVE** is also an active variable, thus the assignment **PRIVE%ADR%(1)%P%R** = **COMPAD_DRV(ZF%R)** is an overloaded assignment of a double precision array to an array of **compad_type**, where the **val** component of **PRIVE%ADR(1)%P%R** is set to the **drv** component **ZF%R**. The **drv** component of **PRIVE%ADR(1)%P%R** is set to zero. Since all `I/O` statements of Sisyphe are restricted to the **val** components (see Section A.3), only the values of variables are written by Sisyphe into the result file. Since **PRIVE** holds the derivatives in their **val** components Sisyphe finally places the derivatives into the output file.

vector and stores the computed ith column of the Jacobian in the derivative components of the local matrix **COMPAD_DEP**. The dependent values are stored in the value components. Note again that the matrix **COMPAD_DEP** is declared as a floating-point variable, while active compilation turns **COMPAD_DEP** into a variable of type **COMPAD_TYPE** containing the components **val** and **drv**.

After collecting all columns of the Jacobian the derivatives are printed to the screen by a nested loop using the fpp-macro **COMPADDRV**. Finally, the values of the dependent variables are printed. Note that even for repeated evaluations within the subroutine **SISYPHE** the result file created by Sisyphe contains the data of the last evaluation only. Any new run of the loop body recreates the result file.

The following additional modifications turn `sysiphe.f` into a driver program that computes the complete Jacobian of **ZF%R** with respect to **CHESTR%R** along with the function values for **ZF%R**.

Insert the following variable declarations after original line *154*:

```
      DOUBLE PRECISION  :: COMPAD_DEP(NPOIN,NPOIN)  !store val and drv
      DOUBLE PRECISION  :: COMPAD_SUM
      INTEGER           :: COMPAD_AD_COUNTER, COMPAD_I, COMPAD_J
```

The head of the loop over the $n =$ **NPOIN** required directions is placed after the original line *184*:

```
C  Loop over columns of the Jacobian
      DO 711 COMPAD_AD_COUNTER = 1, NPOIN
```

Initialization of the derivatives of the independent variables **CHESTR%R** (seeding with a Cartesian basis vector) has to be located after original line *294*:

```
C     First:  reset derivative components of all independents to zero
      CALL SEED(CHESTR%R, 0.D0)
C     Seed with the i-th Cartesian basis vector
      CALL SEED(CHESTR%R( COMPAD_AD_COUNTER ), 1.D0)
#endif
```

The computed values and derivatives are stored in a column of the matrix **COMPAD_DEP** (insert after original line *1622*). Note that this is an assignment between active variables:

```
C     store column of the Jacobian  (value and derivative)
       DO COMPAD_I = 1 , NPOIN
          COMPAD_DEP(COMPAD_I,COMPAD_AD_COUNTER) =  ZF%R(COMPAD_I)
       END DO
```

The AD-loop is terminated at the end of the original code (original line *1760*):

```
C  end of loop over independents
711   CONTINUE
```

Right after the end of the AD-loop place output code for the Jacobian:

```
#ifdef COMPADII    ! do that only for active compilation
      PRINT*, '# COMPAD JACOBIAN AD  d ZF%R / d CHESTR%R(i),  i=1... ',
     $     NPOIN
      DO COMPAD_I=1,NPOIN
         DO COMPAD_J=1,NPOIN
            WRITE(*,'(2X,E18.10e3)',ADVANCE='NO' )
     $           COMPADDRV(COMPAD_DEP(COMPAD_I,COMPAD_J))
         END DO
```

```
      PRINT *
   ENDDO
   PRINT *, '# COMPAD JACOBIAN AD  END'
#endif
```

Finally, the values are printed:

```
   PRINT*, '# COMPAD VALUE AD  ZF%R i=1 .. ', NPOIN
   DO COMPAD_I=1,NPOIN
      DO COMPAD_J=1,NPOIN
         WRITE(*,'(2X,E18.10e3)',ADVANCE='NO' )
  $          COMPADVAL(COMPAD_DEP(COMPAD_I,COMPAD_J))
      END DO
      PRINT *
   ENDDO
   PRINT*, '# COMPAD VALUE  AD  END'
```

3.4 Handwritten Tangent-Linear Code

A tangent-linear version of the parts of Sisyphe that are relevant to the given experiment has been written by hand at the Federal Waterways Engineering and Research Institute some time ago. All sides acknowledge that this process has been "painful," tedious, and error-prone. Moreover, a different experiment would require a rewrite of the tangent-linear code possibly involving substantial changes. There is a large number of different experiments that one might think of... The availability of a compiler to generate tangent-linear Fortran code automatically improves this situation dramatically. Nevertheless, the availabiliy of knowingly correct tangent-linear code turned out to be useful for the verifcation of the correctness of the automatically generated version.

3.5 Sum over Columns of Jacobian by FD

The sum over all `NPOIN` columns of the Jacobian can be approximated by FD (see Section 2), where the subroutine `sisyphe` is evaluated once at the unperturbed data `CHESTR%R` and once more with all elements of `CHESTR%R` perturbed by adding the perturbation parameter $h =$ `COMPAD_H`. Again a loop (*FD-loop* in short) is introduced into the code of the subroutine `SISYPHE` to simulate repeated calls of the routine itself. After the second execution of the FD-loop body the FD approximation is computed and stored within the Sisyphe internal variable `PRIVE%ADR(1)%P%R`.

The following modifications compute an approximation of the sum over the columns of the Jacobian by FD for the very last time step only, since only the dependent values of the last step of the undisturbed evaluation are stored. For all other time steps `PRIVE%ADR(1)` is set to zero. Note that the FD codes need to be compiled passively.

Declare the following variables after the original line *154*:

```
   DOUBLE PRECISION :: COMPAD_H    = 1.0D-6    ! pertubation
   DOUBLE PRECISION :: COMPAD_DEP(NPOIN)       ! store values
   DOUBLE PRECISION :: COMPAD_FD               ! temporary
   INTEGER          :: COMPAD_FD_COUNTER, COMPAD_I
```

The two values required for the FD are computed by the following loop (insert after original line *184*):

```
C       FD - Loop with two evaluations
      DO 711 COMPAD_FD_COUNTER = 0,1
```

For the second run the independents **CHESTR%R** need to be modified by adding the perturbation parameter **COMPAD_H** (insert after original line *294*):

```
        IF (COMPAD_FD_COUNTER .GT. 0 ) THEN
           CHESTR%R = CHESTR%R + COMPAD_H
        END IF
```

In original line *1622* the unperturbed dependent values need to be stored during the last time step[17] of the first evaluation, whereas in the last time step of the second run the FD are computed and stored within field **PRIVE%ADR**(1)%**P**%**R**:

```
C        special output of the derivatives to TECPLOT
       IF ( COMPAD_FD_COUNTER .EQ. 0 .AND. MN .EQ. NCALCU) THEN
          COMPAD_DEP(:) =  ZF%R
       ELSE IF ( COMPAD_FD_COUNTER .EQ. 1 ) THEN
C         only directional derivative of last step is stored
          IF ( MN .EQ. NCALCU ) THEN
             DO COMPAD_I=1,NPOIN
                COMPAD_FD =
     +               (ZF%R(COMPAD_I) - COMPAD_DEP(COMPAD_I)) / COMPAD_H
                PRIVE%ADR(1)%P%R(COMPAD_I) = COMPAD_FD
             ENDDO
          ELSE
             PRIVE%ADR(1)%P%R = 0.D0
          END IF
       ENDIF
```

The loop is terminated after original line *1760*:

```
C       FD loop: end of loop over independents
711   CONTINUE
```

Add the following code after original line *1761* to print the derivatives to the screen:

```
      PRINT *, '# COMPAD JACOBIAN FD   d ZF%R / d CHESTR%R(i),',
     $     '  Sum of Cols,   H =', COMPAD_H
      DO COMPAD_I=1,NPOIN
         WRITE(*,'(2X,E18.10e3)' ) PRIVE%ADR(1)%P%R(COMPAD_I)
      ENDDO
      PRINT*, '# COMPAD JACOBIAN FD END'
```

3.6 Jacobian by FD

The complete Jacobian $F' \in I\!R^{891 \times 891}$ containing the partial derivatives of the dependent variables $\mathbf{y} =$ **ZF%R** $\in I\!R^{891}$ with respect to the independent variables $\mathbf{x} =$ **CHESTR%R** $\in I\!R^{891}$ is assembled columnwise. 891 evaluations of the original routine **sisyphe** are required, where only one element of the vector **CHESTR%R** is perturbed at a time with the perturbation parameter **COMPAD_H** (plus one additional evaluation with unperturbed data).

The values of the dependent variables of all 892 evaluations are stored in **COMPAD_DEP**, and finally the FD approximation of the Jacobian is computed and printed. For every evaluation the variable **COMPAD_INDEP** is used to store the unperturbed value of the independent variable to be restored before the next evaluation.

Declaration of required variables (insert after original line *154*):

[17] The condition is **MP .EQ. NCALCU** ...

```
      DOUBLE PRECISION  :: COMPAD_H  =  1.0D-6 ! pertubation
      DOUBLE PRECISION  :: COMPAD_INDEP        ! unperturbed value
      DOUBLE PRECISION  :: COMPAD_DEP(NPOIN,0:NPOIN) ! store values
      DOUBLE PRECISION  :: COMPAD_FD
      INTEGER           :: COMPAD_FD_COUNTER, COMPAD_I, COMPAD_J
```

The loop for the repeated evaluations is inserted after original line *184*:

```
C     Loop over all NPOIN columns of the Jacobian
C       * needs 1 + NPOIN cycles to get all function values required
C         for the full Jacobian
C       * where the first cycle with 0 == COMPAD_FD_COUNTER computes
C         the function value for original (unperturbed) data.

      DO 711 COMPAD_FD_COUNTER = 0,NPOIN
```

Save and perturb the current independent variable[18] by **COMPAD_H**, if the FD-loop counter **COMPAD_FD_COUNTER** is greater than zero (insert after original line *295*):

```
        IF (COMPAD_FD_COUNTER .GT. 0 ) THEN
C            store original value
C            needs to be restored at the end of the loop body
           COMPAD_INDEP=CHESTR%R(COMPAD_FD_COUNTER)
C            add perturbation COMPAD_H to value
           CHESTR%R(COMPAD_FD_COUNTER) = COMPAD_INDEP + COMPAD_H
        END IF
```

Values of the dependent variables are stored in **COMPAD_DEP**, and the original value of the current independent variable is restored (insert after original line *1622*):

```
C     store function value
       COMPAD_DEP(:,COMPAD_FD_COUNTER) = ZF%R
C     restore original value of current independent for next iteration
       IF (COMPAD_FD_COUNTER .GT. 0 )
     $      CHESTR%R(COMPAD_FD_COUNTER) = COMPAD_INDEP
```

The FD-loop is terminated after original line *1760*:

```
711   CONTINUE
```

Finally, the code that computes and prints the FD approximation is inserted after original line *1761*:

```
C      FD approximation of the Jacobian
      PRINT*, '# COMPAD JACOBIAN FD  d ZF%R / d CHESTR%R(i),  i=1... ',
     $     NPOIN,'    H =', COMPAD_H
      DO COMPAD_I=1,NPOIN
         DO COMPAD_J=1,NPOIN
           COMPAD_ FD = (COMPAD_DEP(COMPAD_I,COMPAD_J)
     $                         - COMPAD_DEP(COMPAD_I,0))  /  COMPAD_H
            WRITE(*,'(2X,E18.10)',ADVANCE='NO' )  COMPAD_FD
         END DO
         PRINT *
      ENDDO
      PRINT*, '# COMPAD JACOBIAN FD  END'
```

[18] That is **CHESTR%R**(**COMPAD_FD_COUNTER**).

4 Numerical results

Figure 5 shows the dune after 144 time steps (4 hours), and sensitivities (sum over the columns of the Jacobian) obtained by the compiler-generated TLM are visualized as colors on the surface. A visual comparison of the sensitivities of the compiler-generated TLM with the derivatives computed with hand-written tangent-linear code verifies the correctness of the compiler-generated code. The FD approximation of the sensitivities shown in Figure 6 matches also the AD-compiler based sensitivities.

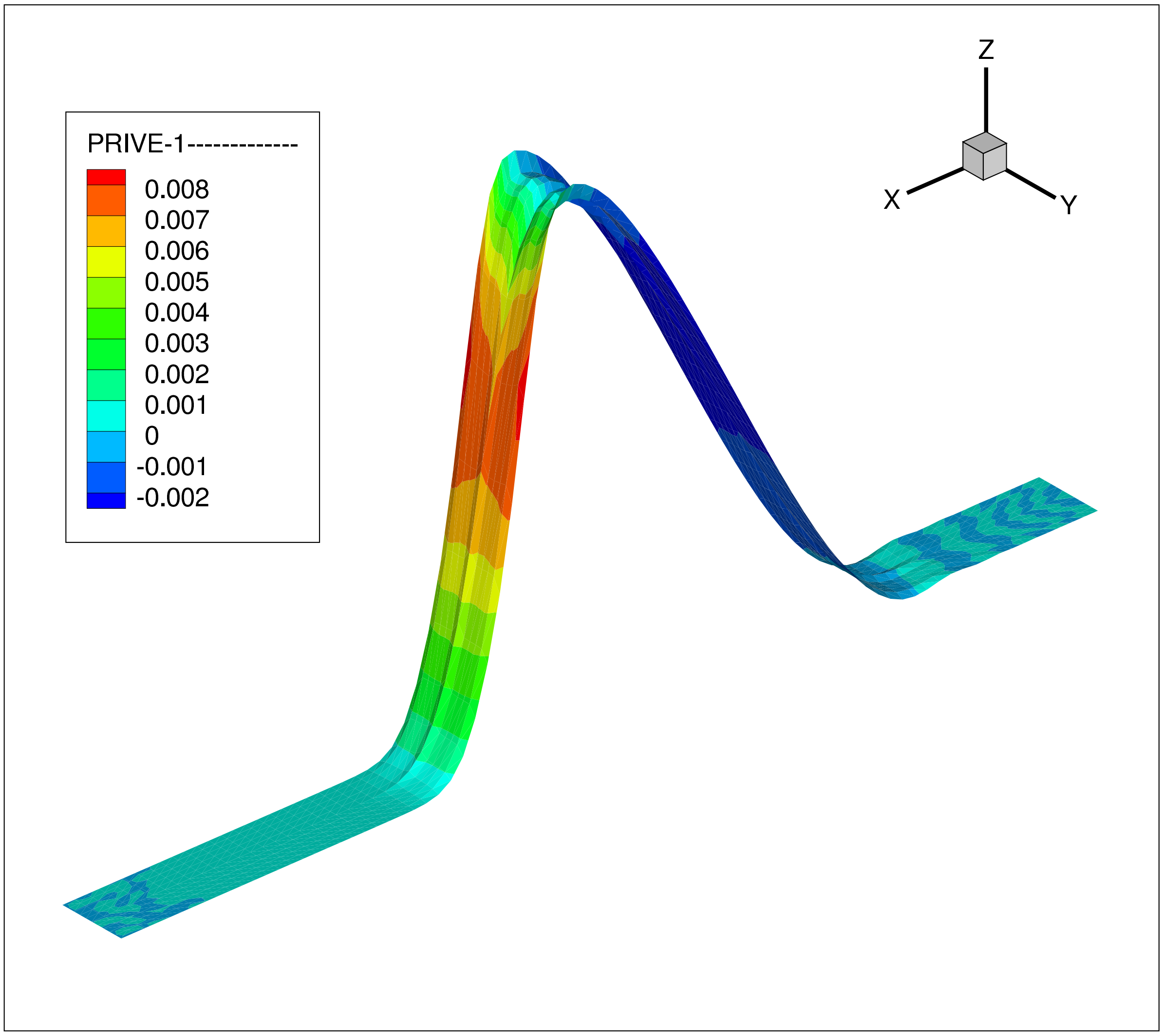

Fig. 5: Sensitivity plot at the surface of the dune after 4 hours (144 time steps), obtained by the compiler-generated tangent-linear code.

Figures 7-10 give a rough impression of the development of both the dune and its sensitivities over time.

Figures 11-13 visualize the sum over all columns of the Jacobian obtained by the compiler-generated tangent-linear code (red line) and by the FD approximation (blue marks). Fig-

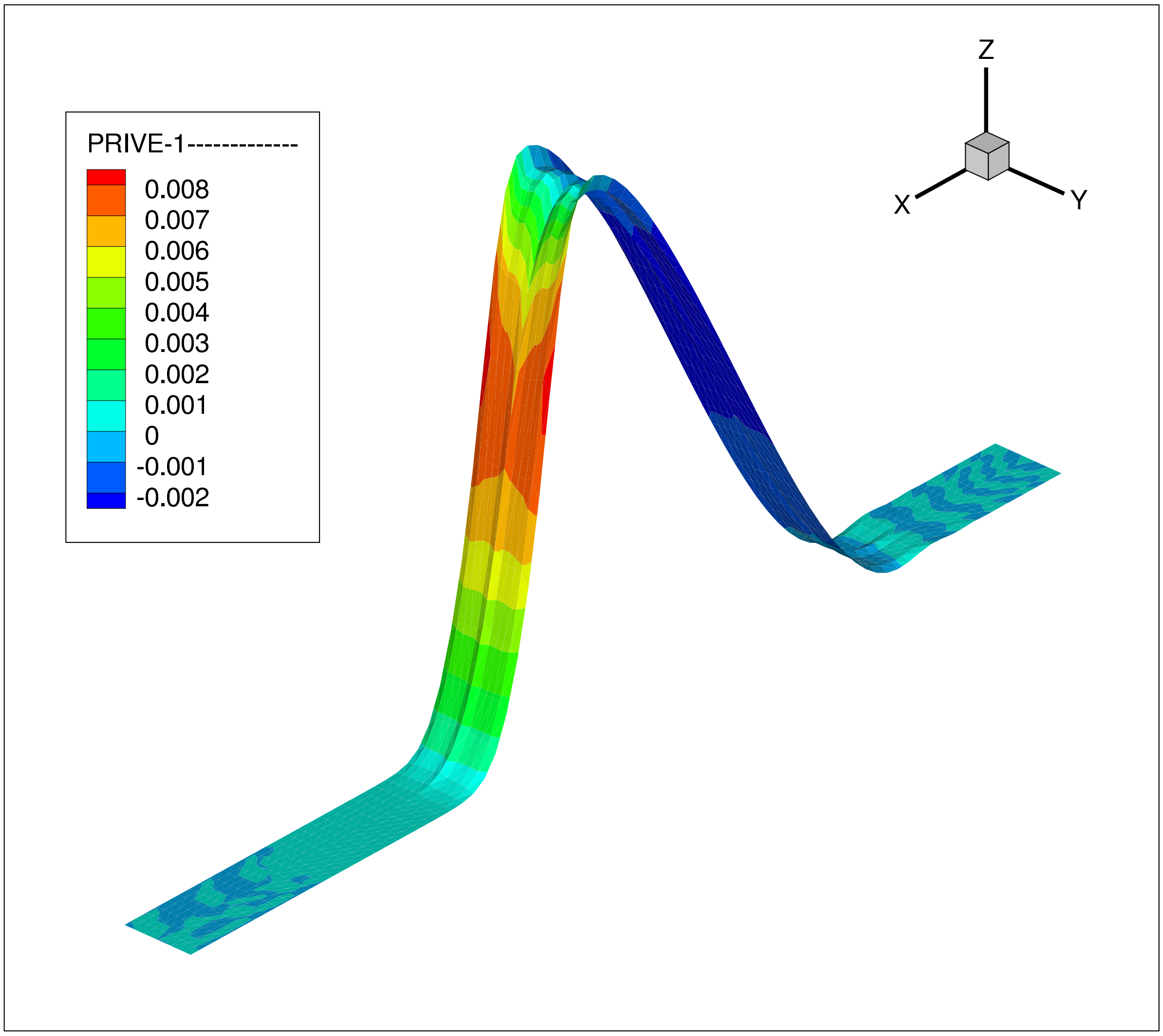

Fig. 6: Sensitivity plot at the surface of the dune after 4 hours (144 time steps), obtained by the FD.

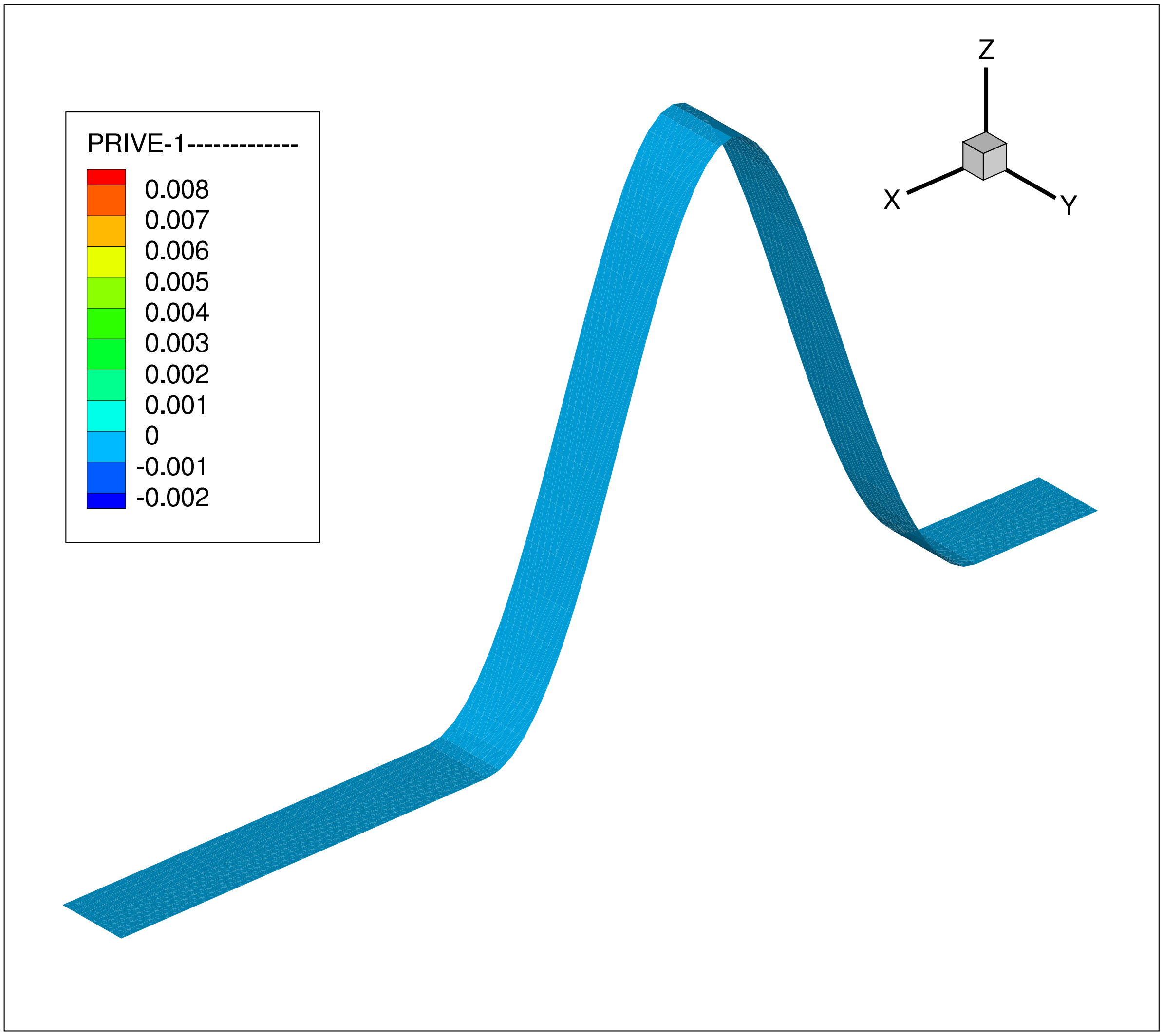

Fig. 7: Dune and sensitivities at the start configuration (compiler-generated tangent-linear code).

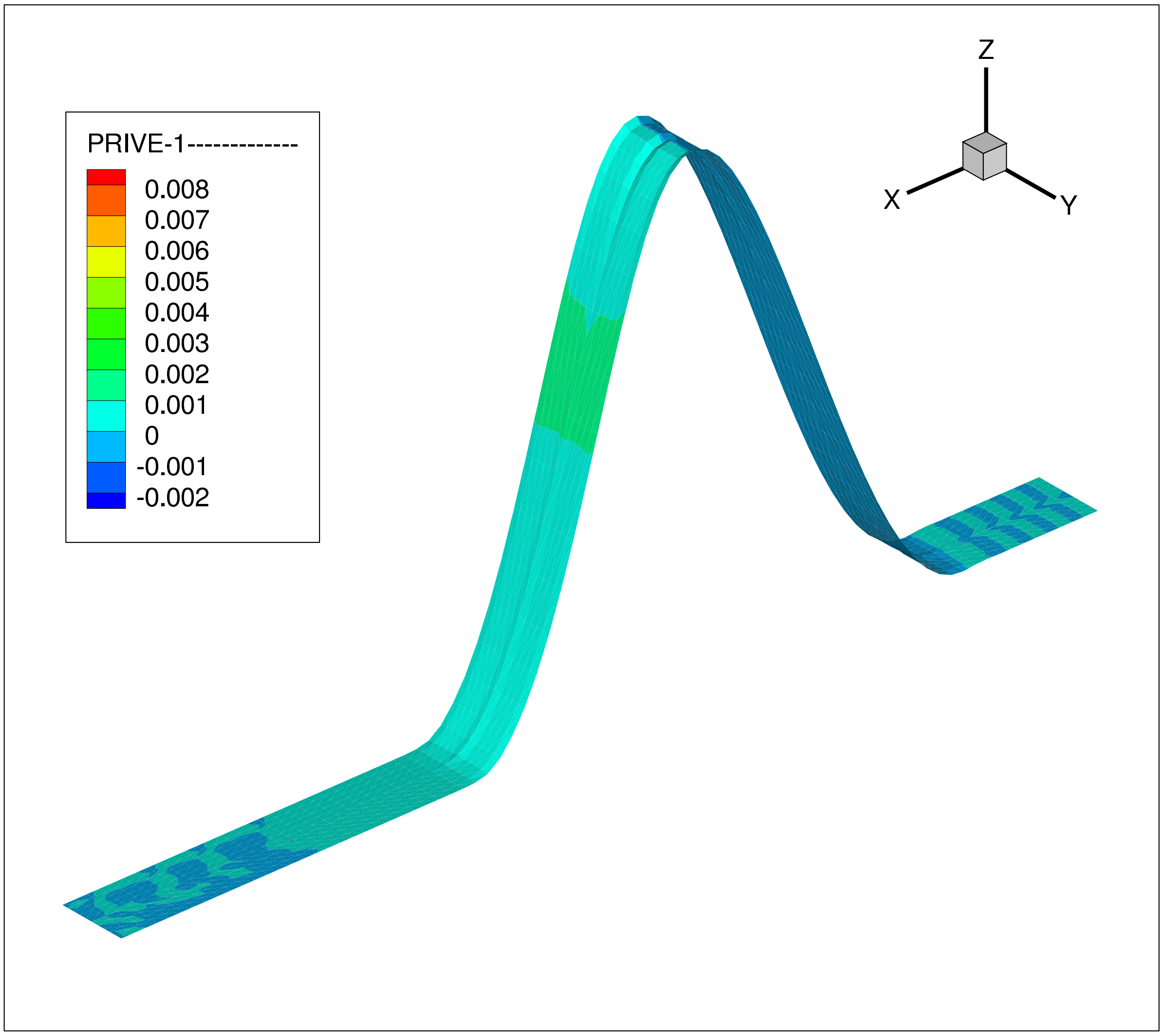

Fig. 8: Dune and sensitivities after 1 hour (compiler-generated tangent-linear code).

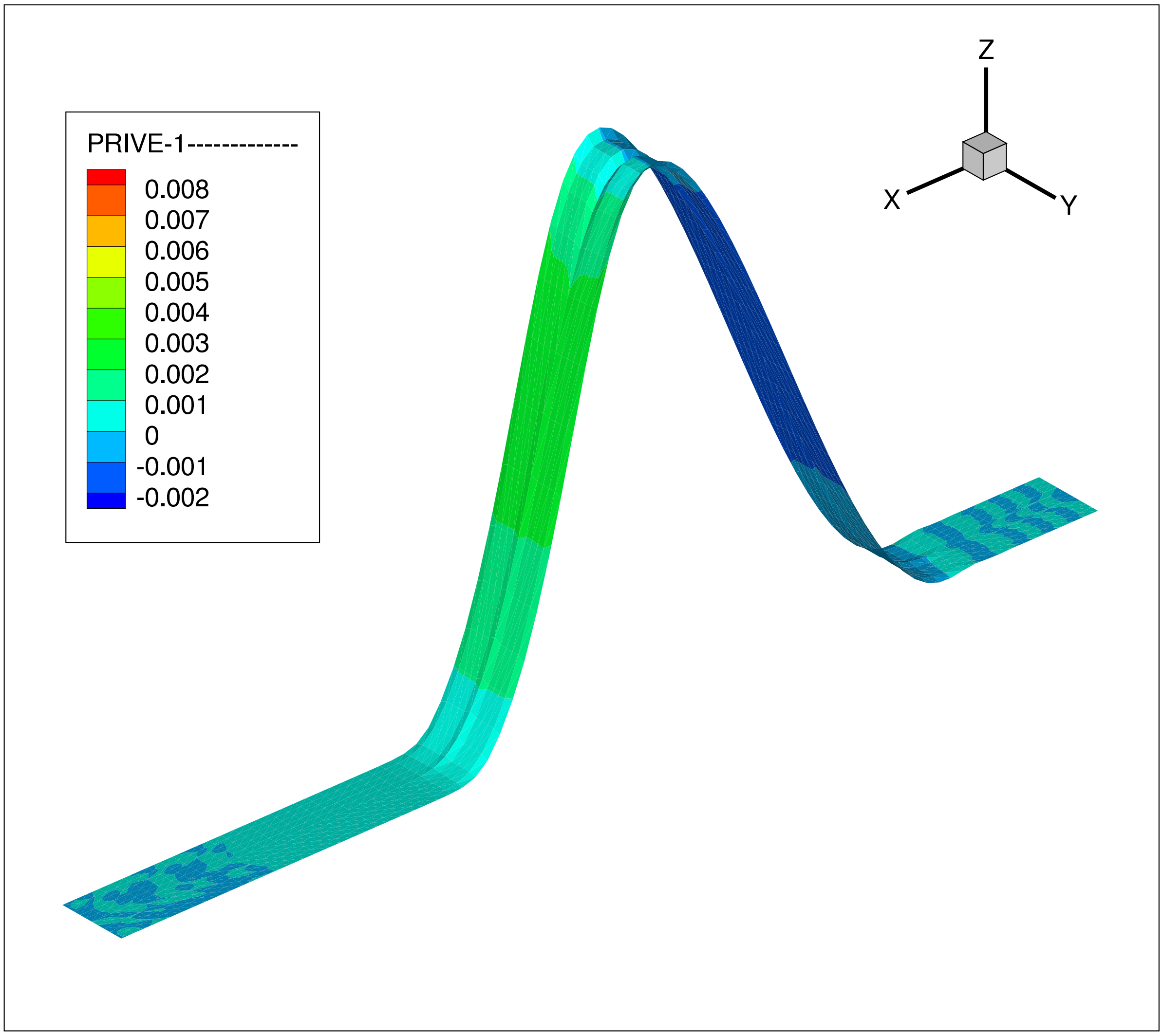

Fig. 9: Dune and sensitivities after 2 hours (compiler-generated tangent-linear code).

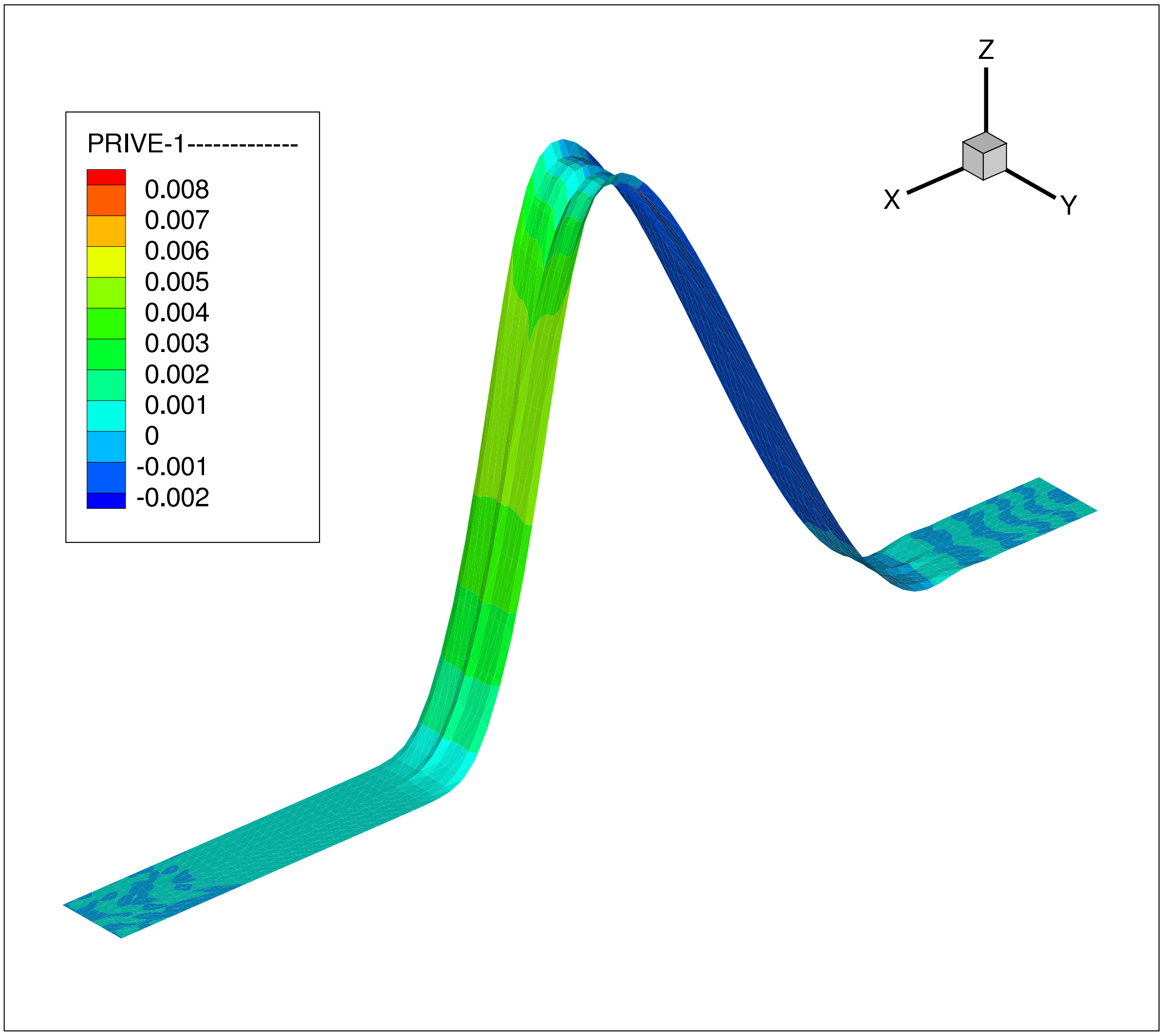

Fig. 10: Dune and sensitivities after 3 hours (compiler-generated tangent-linear code).

ure 11 shows the sum of the sensitivities of all dependent variables (number on the x-axis), Figure 12 the first 81 dependents only. Note that there is no noticeable difference between the compiler-generated and the approximated sensitivities. Variances seem to be introduced by numerical instabilities only as shown in Figure 13, where only dependents are given with vanishing sensitivities.

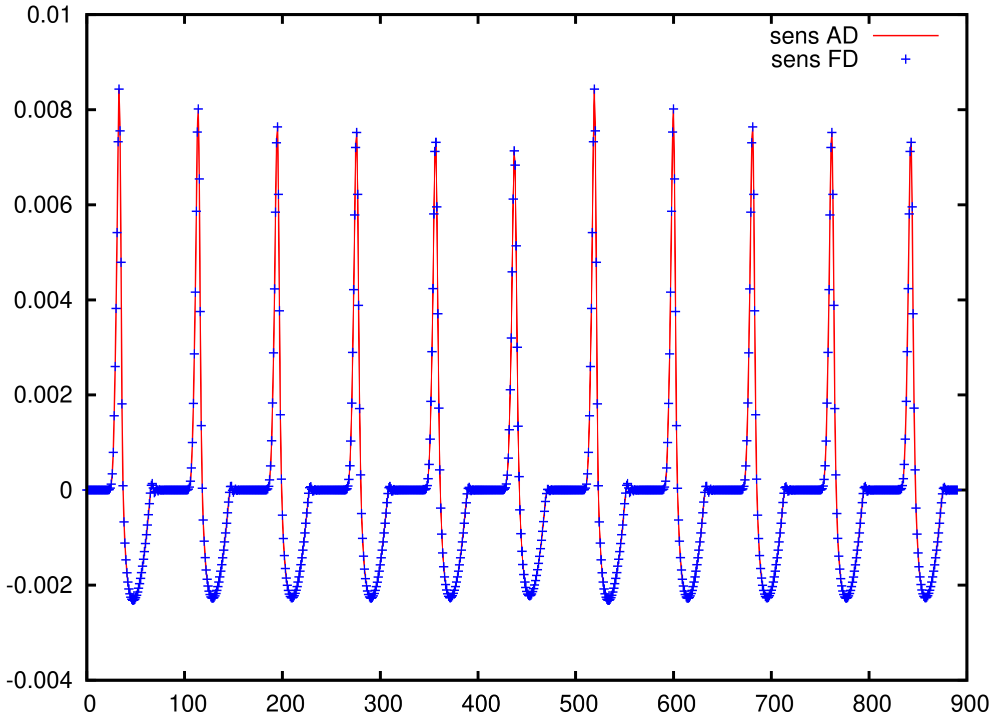

Fig. 11: Comparison of (sum of the) sensitivities for all 891 dependent variables obtained by the compiler-generated TLM (red line) and FD (blue marks).

5 Outlook

5.1 Backward Sensitivities — Adjoints

The computational complexity of the accumulation of a full row of the Jacobian[19] by either FD approximation or tangent-linear code is $O(n)$ times that of a single function evaluation. Every single input needs to be perturbed or, similarly, the derivative in the direction of each Cartesian basis vector needs to be computed. It is well known that for large n neither of the two approaches is feasible. The solution comes in the form of reverse mode AD [7, 11]. It builds the basis for semantic source transformation algorithms that generate adjoint code $\bar{F}$ for the computation of

$$\bar{\mathbf{x}} = \bar{F}(\mathbf{x}, \mathbf{z}, \bar{\mathbf{y}}) = \bar{\mathbf{y}} \cdot F'(\mathbf{x}, \mathbf{z})$$

[19] a single gradient, or in the context of Sisyphe the vector of sensitivities of an dependent variable $y_i \in \mathbf{y}$ with respect to all independents $\mathbf{x}$.

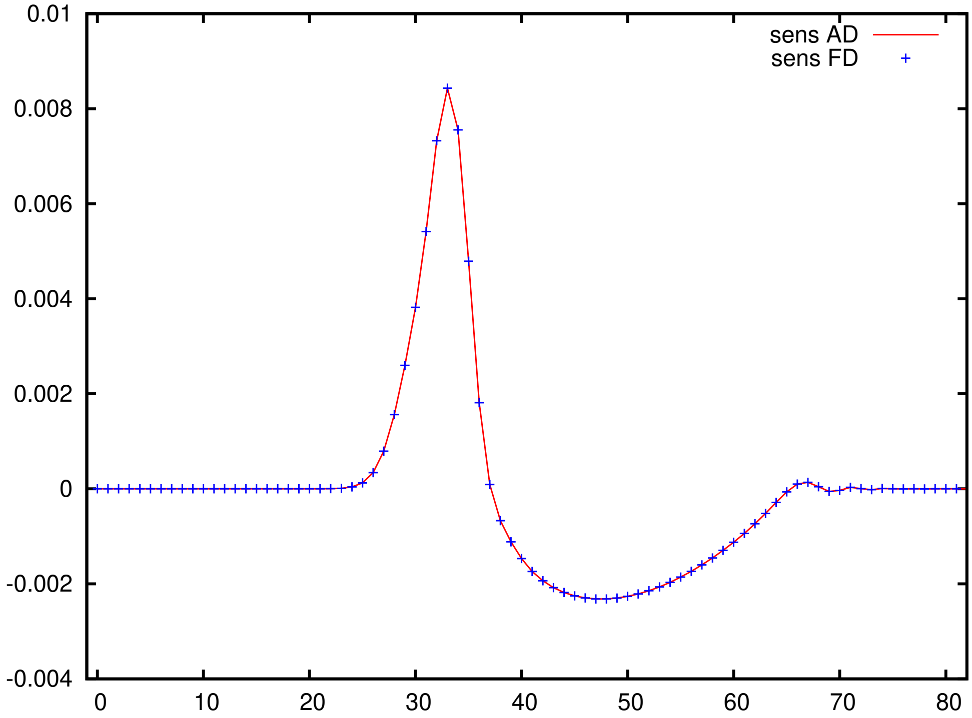

Fig. 12: Comparison of (sum of the) sensitivities for the first 81 dependent variables obtained by the compiler-generated TLM (red line) and FD (blue marks).

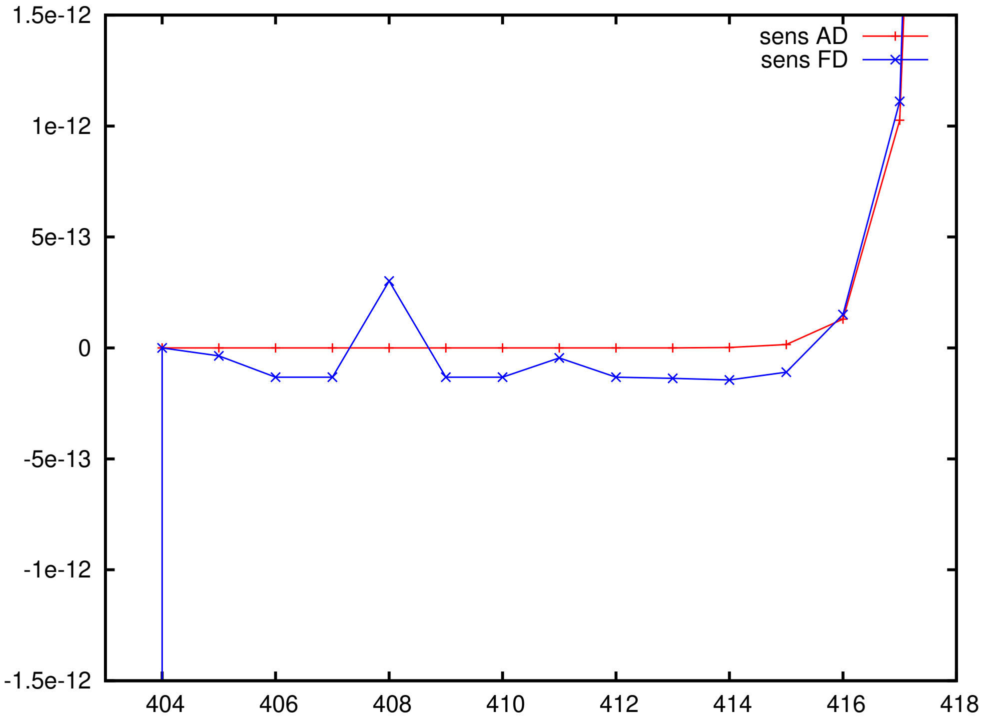

Fig. 13: Comparison of (sum of the) sensitivities, detail zoom.

automatically. The adjoint input vector $\bar{\mathbf{x}}$ becomes equal to the i-th row of $F'(\mathbf{x}, \mathbf{z})$ when setting $\bar{y}_i = 1$ and and the remaining elements of $\mathbf{y}$ equal to zero. A single evaluation of the adjoint code suffices (compared with n evaluations of the tangent-linear code).

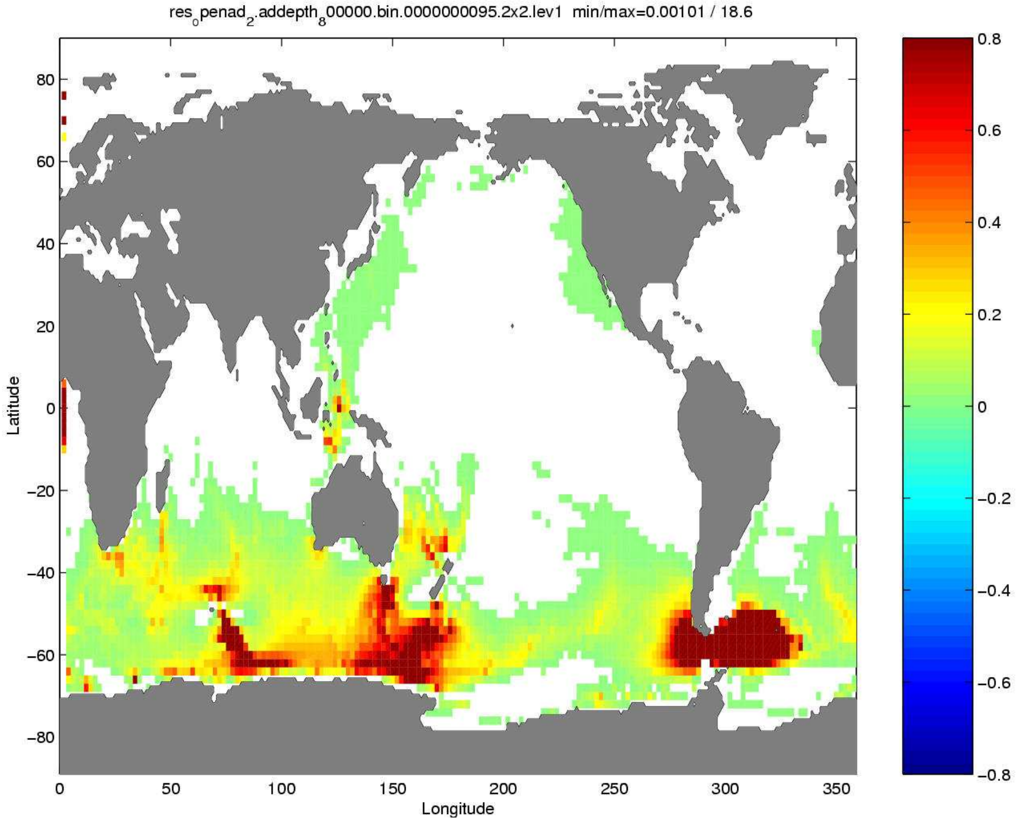

Fig. 14: Sensitivity plot obtained in collaboration with colleagues at MIT, Argonne National Lab, Rice University, and Colorado State University [13].

Certain sensitivity analyses can simply not be run as forward computations with todays computer systems. An example from oceanography is illustrated in Figure 14. Its shows a sensitivity plot of the zonal volume transport through the Drake Passage (between South America and the Antarctis) as a function of bottom topography in earth's oceans [13]. Forward sensitivity analysis would require each point of the shallow water 2D discretization to be perturbed at least once. Any physically meaningful resolution would lead to runtimes at the order of several years on the latest high-performance computer architectures. Obtaining the same result through experiments is obviously impossible. Such results can be used to support decisions about placement of sensors or direction of satellites in order to collect measurement data to evaluate the numerical prediction and hopefully to improve the underlying mathematical models.

5.2 Example: Curve Fitting

Gradients play a crucial role in nonlinear optimization. Neither FD nor TLM represent feasible alternatives for the computation of very large gradients. As an example we consider a simple curve-fitting problem as a non-linear least-squares optimization problem. Consider a numerical simulation program for a mathematical model $\mathbf{y} = F(\mathbf{x}, \mathbf{p})$ where $\mathbf{x} \in I\!R^{n_1}$, $\mathbf{p} \in I\!R^{n_2}$, and $\mathbf{y} \in I\!R^{m}$. For given measurements $(\mathbf{x}^j, \mathbf{y}^j), \quad j = 1, \ldots, k$ we define the residual function $r(\mathbf{p}) : I\!R^{n_2} \to I\!R^k$ as

$$r_i(\mathbf{p}) \equiv \mathbf{y}^i - F(\mathbf{x}^i, \mathbf{p}), \quad i = 1, \ldots, k \quad .$$

Informally, our objective is to adapt the parameters $\mathbf{p}$ such that the data is represented by the model in the best possible way. Curve fitting can be performed by solving the nonlinear least-squares problem

$$\text{minimize} \quad g(\mathbf{p}) \equiv \frac{1}{2}\, r^T(\mathbf{p})\; r(\mathbf{p}) \quad ,$$

for example, by using steepest descent

$$\mathbf{p}^{j+1} = \mathbf{p}^j - \alpha_j \nabla g(\mathbf{p}^j) \quad \text{for } j = 0, 1, \ldots$$

to minimize some norm of the residual iteratively. Note, that

$$\nabla g(\mathbf{p}^j) = (r'(\mathbf{p}^j))^T r(\mathbf{p}^j)$$

can be computed by an adjoint model at small constant multiple of the cost for evaluating r itself. A line search for α_j, that is

$$\text{minimize} \quad g(\alpha_j) \quad \bigl(\text{for given } \mathbf{p}^j \text{ and } \nabla g(\mathbf{p}^j)\bigr)$$

is performed at each step j. For example, one might follow a simple bisection approach where starting from $\alpha_j = 1$ its value is divided by two $\alpha_j = 0.5, 0.25, \ldots$ for as long as there is (sufficient) decrease in the value $g(\alpha_j)$ of the objective function.

The above example is kept very simple in order to illustrate the need for adjoint models in large-scale nonlinear optimization. Similar problems are expected to occur in waterway analysis and optimization. Work toward an adjoint version of Sisyphe should seriously be taken into account.

References

1. M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors. *Computational Differentiation: Techniques, Applications, and Tools*, Proceedings Series. SIAM, 1996.
2. M. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors. *Automatic Differentiation: Applications, Theory, and Tools*, number 50 in Lecture Notes in Computational Science and Engineering, Berlin, 2005. Springer.
3. G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, editors. *Automatic Differentiation of Algorithms – From Simulation to Optimization*, New York, 2002. Springer.
4. G. Corliss and A. Griewank, editors. *Automatic Differentiation: Theory, Implementation, and Application*, Proceedings Series. SIAM, 1991.
5. A. Curtis, M. Powell, and J. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119, 1974.
6. R. Dembo and T. Steihaug. Truncated-Newton algorithms for large-scale optimization. *Math. Prog.*, 26:190–212, 1982.
7. A. Griewank. *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation*. SIAM, Apr. 2000.
8. J.-M. Hervouet and P. Bates. The telemac modelling system special issue. *Hydrological Processes*, 14(13):2207–2208, 2000.
9. C. T. Kelley. *Solving Nonlinear Equations with Newton's Methods*. SIAM, Philadelphia, 2003.

10. U. Naumann, M. Maier, J. Riehme, and B. Christianson. Automatic first- and second-order adjoints for Truncated Newton. In M. Ganzha et al., editor, *Proceedings of IMCSIT'07: Workshop on Computer Aspects of Numerical Algorithms (CANA'07)*, pages 541–555. PTI, 2007.
11. U. Naumann and J. Riehme. Computing adjoints with the NAGWare Fortran 95 compiler. In H. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, editors, *Automatic Differentiation: Applications, Theory, and Tools*, number 50 in Lecture Notes in Computational Science and Engineering, pages 159–170. Springer, 2005.
12. U. Naumann and J. Riehme. A differentiation-enabled Fortran 95 compiler. *ACM Transactions on Mathematical Software*, 31(4):458–474, 2005.
13. J. Utke, U. Naumann, M. Fagan, N. Tallent, M. Strout, P. Heimbach, C. Hill, and C. Wunsch. OpenAD/F: A modular, open-source tool for automatic differentiation of Fortran codes. *ACM Transactions on Mathematical Software*, 34(4), 2008.

A Programming conventions

A.1 AD-aware code development

The following guidelines should be respected by any developer of codes that might be differentiated by AD-tools. Some of them are specific to Fortran and the AD-enabled NAGWare compiler, but become general rules after abstraction from the specific programming language Fortran.

– learn about AD (what it can do for you and what not)
– take (file) `I/O` out of active section, if not possible see Section 3.1
– take dynamic memory allocation (deallocation) out of active section
– avoid side effects of subroutines (write reentrant subroutines; use **`pure`** whenever possible)
– do not use work arrays
– try to avoid pointers (because of aliasing)
– avoid use of black-box routines (derivatives have to be approximated with FD if source code is not available)
– do not use intrinsic names as variable names (**`sum`**, **`dim`**, etc.)
– keep in mind that nondifferentiable submodels cannot be differentiated
– keep in mind that not chain rule differentiable submodels cannot be differentiated, e.g.

$$y = \texttt{sqrt}(x**4)$$

at $x = 0$ we would generate

$$\begin{aligned} dv &= 4*x**3*dx \\ v &= x**4 \\ dy &= -1/(2*\texttt{sqrt}(v))*dv \\ y &= \texttt{sqrt}(v) \end{aligned}$$

Evaluation of 1/(2*sqrt(v)) at v=0 yields and arithmetic exception. If we were to exploit y=sqrt(x**4)=x**2, then everything would be fine.
– avoid common blocks (use modules instead)
– avoid use of CRAY pointers
– avoid aliasing
– avoid implicit declarations (use **`implicit none`**)
– use the compiler hints provided by Fortran (**`intent`**, **`pure`**, etc.)

A.2 NAG - specific changes

This section describes the modifications required to compile and run the Sisyphe system in passive mode.

```
FILE  bief/bief_v5p4/sources/bief.f
139c139
<           TYPE(BIEF_MESH) , INTENT(IN)    :: MESH
---
>           TYPE(BIEF_MESH) , INTENT(INOUT)    :: MESH
251c251
<           TYPE(BIEF_MESH), INTENT(IN) :: MESH
---
>           TYPE(BIEF_MESH), INTENT(INOUT) :: MESH
359c359
<       TYPE(BIEF_OBJ), INTENT(IN)     :: B
---
>       TYPE(BIEF_OBJ), INTENT(INOUT)     :: B
372c372
<       TYPE(BIEF_OBJ), INTENT(IN)     :: B
---
>       TYPE(BIEF_OBJ), INTENT(INOUT)     :: B
530c530
<           TYPE(BIEF_OBJ), INTENT(IN) :: X
---
>           TYPE(BIEF_OBJ)             :: X
537,538c537,538
<           TYPE(BIEF_MESH), INTENT(IN) :: MESH
<           TYPE(BIEF_OBJ), INTENT(IN) :: X,Y
---
>           TYPE(BIEF_MESH) :: MESH
>           TYPE(BIEF_OBJ)  :: X,Y
545,546c545,546
<           INTEGER, INTENT(IN) :: NPOIN
<           DOUBLE PRECISION, INTENT(IN) :: X(NPOIN),Y(NPOIN),FAC(NPOIN)
---
>           INTEGER          :: NPOIN
>           DOUBLE PRECISION :: X(NPOIN),Y(NPOIN),FAC(NPOIN)
626c626
<           TYPE(BIEF_OBJ) , INTENT(INOUT)        :: VEC
---
>           TYPE(BIEF_OBJ) , INTENT(OUT)        :: VEC
647c647
<           CHARACTER(LEN=6) :: NOMGEN
---
>           CHARACTER(LEN=6), INTENT(IN) :: NOMGEN
655c655
<           TYPE(BIEF_OBJ), INTENT(INOUT) :: BLO
---
>           TYPE(BIEF_OBJ), INTENT(OUT) :: BLO
662,666c662,666
<           DOUBLE PRECISION, INTENT(IN)    :: C
<           CHARACTER(LEN=8), INTENT(IN)    :: OP
<           TYPE(BIEF_OBJ)  , INTENT(INOUT) :: X
<           TYPE(BIEF_OBJ)  , INTENT(IN)    :: Y,Z
<           TYPE(BIEF_MESH) , INTENT(IN)    :: MESH
```

```
---
>           DOUBLE PRECISION                 :: C
>           CHARACTER(LEN=8)                 :: OP
>           TYPE(BIEF_OBJ)                   :: X
>           TYPE(BIEF_OBJ)                   :: Y,Z
>           TYPE(BIEF_MESH)                  :: MESH
673c673
<         TYPE(BIEF_OBJ), INTENT(INOUT) :: MAT
---
>         TYPE(BIEF_OBJ), INTENT(OUT) :: MAT
684,685c684,687
<           TYPE(BIEF_MESH), INTENT(INOUT) :: MESH
<           INTEGER, INTENT(IN) :: IELM,CFG(2),NFIC
---
>           TYPE(BIEF_MESH), INTENT(OUT) :: MESH
>           INTEGER, INTENT(IN)          :: IELM
>           INTEGER, INTENT(INOUT)       :: CFG(2)
>           INTEGER, INTENT(IN)          :: NFIC
794,800c796,802
<           INTEGER, INTENT(IN) :: NELEM,NELMAX,NDIAG
<           INTEGER, INTENT(IN) :: IKLE(NELMAX,2)
<           CHARACTER(LEN=8), INTENT(IN)     :: OP
<           DOUBLE PRECISION, INTENT(IN)     :: DN(*),D(*),XN(NELMAX,*)
<           DOUBLE PRECISION, INTENT(INOUT)  :: DM(*),XM(NELMAX,*)
<           CHARACTER(LEN=1), INTENT(INOUT)  :: TYPDIM,TYPEXM,TYPDIN,TYPEXN
<           DOUBLE PRECISION, INTENT(IN)     :: C
---
>           INTEGER             :: NELEM,NELMAX,NDIAG
>           INTEGER             :: IKLE(NELMAX,2)
>           CHARACTER(LEN=8)                 :: OP
>           DOUBLE PRECISION                 :: DN(*),D(*),XN(NELMAX,*)
>           DOUBLE PRECISION                 :: DM(*),XM(NELMAX,*)
>           CHARACTER(LEN=1)                 :: TYPDIM,TYPEXM,TYPDIN,TYPEXN
>           DOUBLE PRECISION                 :: C
943c945
<           TYPE(BIEF_MESH), INTENT(IN)    :: MESH
---
>           TYPE(BIEF_MESH), INTENT(INOUT)    :: MESH
955,957c957,959
<           INTEGER        , INTENT(IN)      :: ICOM
<           TYPE(BIEF_MESH), INTENT(IN)      :: MESH
<           TYPE(BIEF_OBJ) , INTENT(INOUT)   :: X
---
>           INTEGER                          :: ICOM
>           TYPE(BIEF_MESH)                  :: MESH
>           TYPE(BIEF_OBJ)                   :: X
973,976c975,978
<           TYPE(BIEF_MESH) , INTENT(IN)     :: MESH
<           TYPE(BIEF_OBJ)  , INTENT(IN)     :: U,V
<           TYPE(BIEF_OBJ)  , INTENT(INOUT)  :: KX,KY
<           DOUBLE PRECISION, INTENT(IN)     :: XMUL
---
>           TYPE(BIEF_MESH)                  :: MESH
>           TYPE(BIEF_OBJ)                   :: U,V
>           TYPE(BIEF_OBJ)                   :: KX,KY
>           DOUBLE PRECISION                 :: XMUL
984,990c986,992
```

```
<           LOGICAL          , INTENT(IN)     :: MSK
<           DOUBLE PRECISION, INTENT(OUT)    :: CFLMAX
<           DOUBLE PRECISION, INTENT(IN)     :: DT,XMUL
<           TYPE(BIEF_OBJ)   , INTENT(INOUT) :: RES,T1,T2
<           TYPE(BIEF_OBJ)   , INTENT(IN)     :: U,V,F,MASKEL
<           TYPE(BIEF_MESH) , INTENT(IN)     :: MESH
<           INTEGER          , INTENT(IN)     :: IELM
---
>           LOGICAL                           :: MSK
>           DOUBLE PRECISION                  :: CFLMAX
>           DOUBLE PRECISION                  :: DT,XMUL
>           TYPE(BIEF_OBJ)                    :: RES,T1,T2
>           TYPE(BIEF_OBJ)                    :: U,V,F,MASKEL
>           TYPE(BIEF_MESH)                   :: MESH
>           INTEGER                           :: IELM
997,999c999,1001
<           TYPE(BIEF_OBJ) , INTENT(IN)     :: Z
<           TYPE(BIEF_OBJ) , INTENT(INOUT) :: COEF
<           TYPE(BIEF_MESH), INTENT(IN)     :: MESH
---
>           TYPE(BIEF_OBJ)                  :: Z
>           TYPE(BIEF_OBJ)                  :: COEF
>           TYPE(BIEF_MESH)                 :: MESH
1194,1198c1196,1200
<           LOGICAL, INTENT(IN)           :: INFOGR
<           TYPE(BIEF_OBJ), INTENT(INOUT) :: X,B
<           TYPE(BIEF_OBJ), INTENT(INOUT) :: TB
<           TYPE(BIEF_OBJ), INTENT(INOUT) :: A,AUX
<           TYPE(BIEF_MESH), INTENT(IN)   :: MESH
---
>           LOGICAL                       :: INFOGR
>           TYPE(BIEF_OBJ)                :: X,B
>           TYPE(BIEF_OBJ)                :: TB
>           TYPE(BIEF_OBJ)                :: A,AUX
>           TYPE(BIEF_MESH)               :: MESH
1276c1278
<       TYPE(BIEF_OBJ), INTENT(IN)     :: B
---
>       TYPE(BIEF_OBJ), INTENT(INOUT)     :: B
1333c1335
<       TYPE(BIEF_OBJ), INTENT(IN)     :: B
---
>       TYPE(BIEF_OBJ), INTENT(INOUT)     :: B
1354,1360c1356,1362
<           INTEGER, INTENT(IN) :: NELEM,NPOIN,IELM
<           INTEGER, INTENT(IN) :: IKLE(NELEM,*),IFABOR(NELEM,*)
<           INTEGER, INTENT(INOUT) :: IFAMAS(NELEM,*)
<           DOUBLE PRECISION, INTENT(IN)     :: MASKEL(NELEM)
<           DOUBLE PRECISION, INTENT(INOUT) :: MASKPT(NPOIN)
<           DOUBLE PRECISION, INTENT(INOUT) :: XNEBOR(NPOIN),YNEBOR(NPOIN)
<           DOUBLE PRECISION, INTENT(IN)     :: X(NPOIN),Y(NPOIN)
---
>           INTEGER             :: NELEM,NPOIN,IELM
>           INTEGER             :: IKLE(NELEM,*),IFABOR(NELEM,*)
>           INTEGER                :: IFAMAS(NELEM,*)
>           DOUBLE PRECISION                 :: MASKEL(NELEM)
>           DOUBLE PRECISION                 :: MASKPT(NPOIN)
```

```
>            DOUBLE PRECISION                 :: XNEBOR(NPOIN),YNEBOR(NPOIN)
>            DOUBLE PRECISION                 :: X(NPOIN),Y(NPOIN)
1367,1371c1369,1373
<            DOUBLE PRECISION, INTENT(IN)     :: C
<            CHARACTER(LEN=8), INTENT(IN)     :: OP
<            TYPE(BIEF_OBJ)  , INTENT(INOUT)  :: X
<            TYPE(BIEF_OBJ)  , INTENT(IN)     :: Y,Z
<            TYPE(BIEF_MESH) , INTENT(IN)     :: MESH
---
>            DOUBLE PRECISION                 :: C
>            CHARACTER(LEN=8)                 :: OP
>            TYPE(BIEF_OBJ)                   :: X
>            TYPE(BIEF_OBJ)                   :: Y,Z
>            TYPE(BIEF_MESH)                  :: MESH
1378,1381c1380,1383
<            DOUBLE PRECISION, INTENT(IN)     :: XMUL
<            TYPE(BIEF_OBJ)  , INTENT(INOUT)  :: DIAG
<            TYPE(BIEF_OBJ)  , INTENT(IN)     :: A
<            TYPE(BIEF_MESH) , INTENT(IN)     :: MESH
---
>            DOUBLE PRECISION                 :: XMUL
>            TYPE(BIEF_OBJ)                   :: DIAG
>            TYPE(BIEF_OBJ)                   :: A
>            TYPE(BIEF_MESH)                  :: MESH
1436,1443c1438,1445
<            TYPE(BIEF_OBJ)    , INTENT(INOUT) :: M
<            INTEGER           , INTENT(IN)    :: IELM1,IELM2
<            DOUBLE PRECISION  , INTENT(IN)    :: XMUL
<            LOGICAL           , INTENT(IN)    :: MSK
<            CHARACTER(LEN=16), INTENT(IN)     :: FORMUL
<            CHARACTER(LEN=8) , INTENT(IN)     :: OP
<            TYPE(BIEF_OBJ)    , INTENT(IN)    :: F,G,H,U,V,W,MASKEL
<            TYPE(BIEF_MESH)   , INTENT(INOUT) :: MESH
---
>            TYPE(BIEF_OBJ)                    :: M
>            INTEGER                           :: IELM1,IELM2
>            DOUBLE PRECISION                  :: XMUL
>            LOGICAL                           :: MSK
>            CHARACTER(LEN=16)                 :: FORMUL
>            CHARACTER(LEN=8)                  :: OP
>            TYPE(BIEF_OBJ)                    :: F,G,H,U,V,W,MASKEL
>            TYPE(BIEF_MESH)                   :: MESH
1459,1461c1461,1463
<            TYPE(BIEF_OBJ) , INTENT(IN)     :: X
<            TYPE(BIEF_OBJ) , INTENT(INOUT)  :: XEL
<            TYPE(BIEF_MESH), INTENT(IN)     :: MESH
---
>            TYPE(BIEF_OBJ)                  :: X
>            TYPE(BIEF_OBJ)                  :: XEL
>            TYPE(BIEF_MESH)                 :: MESH
1478c1480
<            INTEGER, INTENT(INOUT) :: IT1(*),IT2(*),IT3(*),IT4(*)
---
>            INTEGER, INTENT(OUT) :: IT1(*),IT2(*),IT3(*),IT4(*)

FILE  bief/bief_v5p4/sources/omseg.f
549,550c549,552
```

```
< 810        FORMAT(1X,'OMSEG (BIEF) : MATRICE DEJA NON SYMETRIQUE :')
< 811        FORMAT(1X,'OMSEG (BIEF) : MATRIX ALREADY NON SYMMETRICAL:')
---
> 810        FORMAT(1X,
>      $          'OMSEG (BIEF) : MATRICE DEJA NON SYMETRIQUE :',A1)
> 811        FORMAT(1X,
>      $          'OMSEG (BIEF) : MATRIX ALREADY NON SYMMETRICAL:',A1)
```

```
FILE  sisyphe/sisyphe_v5p4/sources/bilan_sisyphe.f
246c246
< 4000  FORMAT(5X,'GRAIN-FEEDING A CET INSTANT : ,' G16.7)
---
> 4000  FORMAT(5X,'GRAIN-FEEDING A CET INSTANT : ', G16.7)

FILE  sisyphe/sisyphe_v5p4/sources/check.f
143c143
< 124     FORMAT(1X,'CHECK : TYPE DE FICHIER INCONNU :')
---
> 124     FORMAT(1X,'CHECK : TYPE DE FICHIER INCONNU :',1I6)
```

```
FILE  special/special_v5p4/sources/plante.f
99c99
<       CALL EXIT(ICODE)
---
>       STOP
```

A.3 AD - specific conventions and changes

This section describes the modifications in Sisyphe required to generate a tangent-linear code with the AD-enabled NAGWare Fortran compiler (scalar mode, AD by overloading).

Passive compilation in project `BIEF` The file `lit.f` of subproject `BIEF` provides the subroutine **`lit`** used to read data of different data types from external files. The parameter **`TYPE`** (a character array) selects the type of data. With **`'R4'`** single precision floating-point data can be read. Compiling this routine in active mode redeclares all variables as of type **`compad_type`** (see Listing 1.5), including the single precision array **`W`** used to read single precision data. By encapsulating **`W`** with the macro **`COMPADVALCOMP`** (see 3.1) defined in the preprocessor input file `compad_component_handler.fpp`, the activated array **`W`** can be restricted to the value component **`val`** of **`compad_type`**. However component **`val`** is declared as double precision (see Listing 1.5). Hence the list-directed **`READ`** operation for single precision data in (original) line 61 in file `lit.f` is turned into an I/O-operation of double precision data. As the data file read by **`lit`** contains single precision data only (as signaled by the value **`'R4'`** of parameter **`TYPE`**), wrong values will be stored in **`W%val`**.

Thus the file `lit.f` of subproject `BIEF` needs to be compiled in passive mode, and must be activated by hand since all calls are made from within the compiler-generated tangent-linear code. The following modifications are required in file **`lit.f`**:

```
FILE  bief/bief_v5p4/sources/lit.f
0a1
> #include "compad_component_handler.fpp"
37a39,41
> #ifdef COMPADII
>       use COMPADMODULE
> #endif
44a49,53
> #ifdef COMPADII
>       TYPE(COMPAD_TYPE) X(NVAL)
>       TYPE(COMPAD_TYPE) W(NVAL)
>       REAL Wpassive(NVAL)
> #else
46a56,57
>       REAL Wpassive(NVAL)
> #endif
50d60
<       INTRINSIC DBLE
61c71,74
<             READ(CANAL,END=100,ERR=101)(W(J),J=1,NVAL)
---
>             READ(CANAL,END=100,ERR=101)(Wpassive(J),J=1,NVAL)
>             DO J=1,NVAL
>                W(J) = Wpassive(J)
>             END DO
66c79
<             READ(CANAL,END=100,ERR=101)(X(J),J=1,NVAL)
---
>             READ(CANAL,END=100,ERR=101)(COMPADVALCOMP(X(J)),J=1,NVAL)
```

Passive compilation can be achieved easily by adding to the makefile in subproject `bief` the following special rule for building `lit.o` in order to remove the AD-related options from the compiler's arguments (the little `sed`-script in line 17):

```
1  #------- PASSIVE COMPILATION RULES  Regles de compilation des sources FORTRAN
2
3  lit.o: lit.f
4          @echo "=>␣PASSIVE␣Compilation␣de␣lit.f␣:"
5          @if [ ! -d ../$(DIRLIB) ]; then \
6          @echo "creation␣du␣repertoire␣$(DIRLIB)";mkdir ../$(DIRLIB); fi
7          @if [ "$(DIRLIB)" = "unicos" -o "$(DIRLIB)" = "fuji" ]; then \
8            DIR="`whoami`@`uname␣-n`:`pwd`" ; \
9            remsh $(NAME_CIBLE) -l $(USR_CIBLE) \
10           "cd␣workdir/compilation␣;\
11 ␣␣␣␣␣␣␣␣␣␣rcp␣$$DIR/$(<F)␣.␣;\
12 ␣␣␣␣␣␣␣␣␣␣$(FC_CIBLE)␣$(FCFLAG_CIBLE)␣$(<F)␣;\
13 ␣␣␣␣␣␣␣␣␣␣rm␣$(*F).f␣;\
14 ␣␣␣␣␣␣␣␣␣␣rcp␣$(*F).o␣$$DIR/.␣";\
15         else \
16           echo "␣COMMAND:␣$(FC)␣`echo␣"$(FFLAGS)"␣|␣sed␣'s/-ad[a-zA-Z\-\_]*//g'␣`
               ␣lit.f␣"; \
17           $(FC) `echo "$(FFLAGS)" | sed 's/-ad[a-zA-Z\-\_]*//g' ` lit.f ; \
18         fi;
19         @if [ -r *.mod ]; then \
```

```
          echo "Transfert␣du␣module␣$(@F:.o=.mod)␣";\
          cp  *.mod $(DEST)/.; /bin/rm *.mod; fi;
         @if [ -r *.MOD ]; then \
          echo "Transfert␣du␣module␣$(@F:.o=.MOD)␣";\
          cp  *.MOD $(DEST)/.; /bin/rm *.MOD; fi;
```

```
FILE  bief/bief_v5p4/sources/asmve1.f
55d54
<        INTRINSIC MIN

FILE  bief/bief_v5p4/sources/assve1.f
57d56
<        INTRINSIC MIN

FILE  bief/bief_v5p4/sources/atanc.f
42d41
<        INTRINSIC ACOS,ATAN

FILE  bief/bief_v5p4/sources/bief.f
528c528
<          DOUBLE PRECISION FUNCTION SUM( X )
---
>          DOUBLE PRECISION FUNCTION SUM_BIEF( X )
647a648
> !           CHARACTER(LEN=6) :: NOMGEN

FILE  bief/bief_v5p4/sources/bilant.f
75d74
<        INTRINSIC ABS,MAX

FILE  bief/bief_v5p4/sources/cflp11.f
64d63
<        INTRINSIC MAX,MIN

FILE  bief/bief_v5p4/sources/cflp12.f
63d62
<        INTRINSIC MAX,MIN

FILE  bief/bief_v5p4/sources/cgsqua.f
136d135
<        INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/cgstab.f
72d71
<        INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/char11.f
99d98
<        INTRINSIC INT,MAX,MIN,SQRT

FILE  bief/bief_v5p4/sources/char41.f
108d107
<        INTRINSIC ABS , INT , MAX , SQRT

FILE  bief/bief_v5p4/sources/couplage.f
0a1,3
```

```
> #include "compad_component_handler.fpp"
>
>
78d80
<       INTRINSIC TRIM,ABS
194,195c196,197
<       WRITE(80,ERR=85) AT
<       WRITE(80,ERR=86) (TAB%R(I),I=1,TAB%DIM1)
---
>       WRITE(80,ERR=85) COMPADVAL(AT)
>       WRITE(80,ERR=86) (COMPADVAL(TAB%R(I)),I=1,TAB%DIM1)
226c228
<          READ(84,ERR=85,IOSTAT=ERR1)  AT1(IVAR)
---
>          READ(84,ERR=85,IOSTAT=ERR1)  COMPADVALCOMP(AT1(IVAR))
243c245,246
<          READ(84,ERR=86,IOSTAT=ERR1) (TAB%R(I),I=1,TAB%DIM1)
---
>          READ(84,ERR=86,IOSTAT=ERR1)
>      $         (COMPADVALCOMP(TAB%R(I)),I=1,TAB%DIM1)
278,279c281,283
<            READ(84,ERR=85,IOSTAT=ERR1)  AT2(IVAR)
<            READ(84,ERR=86,IOSTAT=ERR1) (TAB2%R(I),I=1,TAB%DIM1)
---
>            READ(84,ERR=85,IOSTAT=ERR1)  COMPADVALCOMP(AT2(IVAR))
>            READ(84,ERR=86,IOSTAT=ERR1)
>      $           (COMPADVALCOMP(TAB2%R(I)),I=1,TAB%DIM1)

FILE  bief/bief_v5p4/sources/crsl11.f
54d53
<       INTRINSIC MAX

FILE  bief/bief_v5p4/sources/crsl12.f
57d56
<       INTRINSIC MAX

FILE  bief/bief_v5p4/sources/decv11.f
56d55
<       INTRINSIC MAX,MIN

FILE  bief/bief_v5p4/sources/decv21.f
56d55
<       INTRINSIC MAX,MIN

FILE  bief/bief_v5p4/sources/des11.f
69d68
<       INTRINSIC MIN

FILE  bief/bief_v5p4/sources/des21.f
74d73
<       INTRINSIC MIN

FILE  bief/bief_v5p4/sources/des41.f
89d88
<       INTRINSIC MIN

FILE  bief/bief_v5p4/sources/dmo.f
```

```
40d39
<       INTRINSIC ACOS,MOD

FILE  bief/bief_v5p4/sources/ecri2.f
0a1,3
> #include "compad_component_handler.fpp"
>
>
61d63
<       INTRINSIC REAL
72c74
<             WRITE(CANAL)(REAL(X(J)),J=1,NVAL)
---
>             WRITE(CANAL)(REAL(COMPADVAL(X(J))),J=1,NVAL)
74c76
<             WRITE(CANAL)(X(J),J=1,NVAL)
---
>             WRITE(CANAL)(COMPADVAL(X(J)),J=1,NVAL)

FILE  bief/bief_v5p4/sources/elapse.f
43d42
<       INTRINSIC INT

FILE  bief/bief_v5p4/sources/equnor.f
86d85
<       INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/errmax.f
49d48
<       INTRINSIC ABS

FILE  bief/bief_v5p4/sources/errmin.f
85d84
<       INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/fluxpr.f
98c98
<       II=P_IMIN(NSEG(ISEC))
---
>       II=NSEG(ISEC)

FILE  bief/bief_v5p4/sources/fond.f
0a1
> #include "compad_component_handler.fpp"
74c75,76
<         READ(NFON,*) BID,BID,BID
---
>         READ(NFON,*) COMPADVALCOMP(BID),COMPADVALCOMP(BID),
>      $          COMPADVALCOMP(BID)
115c117,118
<         READ(NFON,*) XRELV(NP) , YRELV(NP) , COTE(NP)
---
>         READ(NFON,*) COMPADVALCOMP(XRELV(NP)), COMPADVALCOMP(YRELV(NP)),
>      $       COMPADVALCOMP(COTE(NP))

FILE  bief/bief_v5p4/sources/front2.f
72d71
```

```
<        INTRINSIC ABS

FILE  bief/bief_v5p4/sources/gmres.f
160d159
<        INTRINSIC SQRT,ABS

FILE  bief/bief_v5p4/sources/gracjg.f
85d84
<        INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/gtshp1.f
55d54
<        INTRINSIC MIN

FILE  bief/bief_v5p4/sources/inclu2.f
38d37
<        INTRINSIC LEN

FILE  bief/bief_v5p4/sources/inclus.f
36d35
<        INTRINSIC LEN

FILE  bief/bief_v5p4/sources/infcel.f
71d70
<        INTRINSIC ABS,SQRT

FILE  bief/bief_v5p4/sources/inpoly.f
73d72
<        INTRINSIC COS,SIN,ABS,MOD

FILE  bief/bief_v5p4/sources/jultim.f
5c5
<       *(YEAR,MONTH,DAY,HOUR,MIN,SEC,AT)
---
>       *(YEAR,MONTH,DAY,HOUR,MINUTES,SEC,AT)
40c40
<        INTEGER YEAR,MONTH,DAY,HOUR,MIN,SEC
---
>        INTEGER YEAR,MONTH,DAY,HOUR,MINUTES,SEC
44d43
<        INTRINSIC INT
70c69
<        JULTIM=(J+(HOUR+(MIN+(SEC+AT)/60.D0)/60.D0)/24.D0)/36525.D0
---
>        JULTIM=(J+(HOUR+(MINUTES+(SEC+AT)/60.D0)/60.D0)/24.D0)/36525.D0

FILE  bief/bief_v5p4/sources/kspg11.f
73d72
<        INTRINSIC MAX,SQRT

FILE  bief/bief_v5p4/sources/latitu.f
49d48
<        INTRINSIC TAN,ATAN,SIN,COS,EXP

FILE  bief/bief_v5p4/sources/maskto.f
68d67
<        INTRINSIC MAX,NINT
```

```

FILE  bief/bief_v5p4/sources/matvct.f
117d116
<        INTRINSIC MIN

FILE  bief/bief_v5p4/sources/mer11.f
64d63
<        INTRINSIC MIN

FILE  bief/bief_v5p4/sources/mer21.f
65d64
<        INTRINSIC MIN

FILE  bief/bief_v5p4/sources/mer41.f
72d71
<        INTRINSIC MIN

FILE  bief/bief_v5p4/sources/mt04pp.f
100c100,101
<        DOUBLE PRECISION PY1Y2,PZX1,PZX2,PZY1,PZY2,INT
---
>        DOUBLE PRECISION PY1Y2,PZX1,PZX2,PZY1,PZY2,INT1
>
377,412c378,413
<          INT= PZX1*FACUW1+PZY1*FACVW1
<          T (IELEM, 1)= T (IELEM, 1) + INT
<          XM(IELEM, 3)= XM(IELEM, 3) - INT
<          INT=PZX1*FACUW2+PZY1*FACVW2
<          XM(IELEM, 1)= XM(IELEM, 1) + INT
<          XM(IELEM, 4)= XM(IELEM, 4) - INT
<          INT=PZX1*FACUW3+PZY1*FACVW3
<          XM(IELEM, 2)= XM(IELEM, 2) + INT
<          XM(IELEM, 5)= XM(IELEM, 5) - INT
<          INT=PZX2*FACUW1+PZY2*FACVW1
<          XM(IELEM,16)= XM(IELEM,16) + INT
<          XM(IELEM, 7)= XM(IELEM, 7) - INT
<          INT=PZX2*FACUW2+PZY2*FACVW2
<          T (IELEM, 2)= T (IELEM, 2) + INT
<          XM(IELEM, 8)= XM(IELEM, 8) - INT
<          INT=PZX2*FACUW3+PZY2*FACVW3
<          XM(IELEM, 6)= XM(IELEM, 6) + INT
<          XM(IELEM, 9)= XM(IELEM, 9) - INT
<          INT=PZX1*FACUW4+PZY1*FACVW4
<          XM(IELEM,18)= XM(IELEM,18) + INT
<          T (IELEM, 4)= T (IELEM, 4) - INT
<          INT=PZX1*FACUW5+PZY1*FACVW5
<          XM(IELEM,22)= XM(IELEM,22) + INT
<          XM(IELEM,13)= XM(IELEM,13) - INT
<          INT=PZX1*FACUW6+PZY1*FACVW6
<          XM(IELEM,25)= XM(IELEM,25) + INT
<          XM(IELEM,14)= XM(IELEM,14) - INT
<          INT=PZX2*FACUW4+PZY2*FACVW4
<          XM(IELEM,19)= XM(IELEM,19) + INT
<          XM(IELEM,28)= XM(IELEM,28) - INT
<          INT=PZX2*FACUW5+PZY2*FACVW5
<          XM(IELEM,23)= XM(IELEM,23) + INT
<          T (IELEM, 5)= T (IELEM, 5) - INT
```

```
<           INT=PZX2*FACUW6+PZY2*FACVW6
<           XM(IELEM,26)= XM(IELEM,26) + INT
<           XM(IELEM,15)= XM(IELEM,15) - INT
---
>           INT1= PZX1*FACUW1+PZY1*FACVW1
>           T (IELEM, 1)= T (IELEM, 1) + INT1
>           XM(IELEM, 3)= XM(IELEM, 3) - INT1
>           INT1=PZX1*FACUW2+PZY1*FACVW2
>           XM(IELEM, 1)= XM(IELEM, 1) + INT1
>           XM(IELEM, 4)= XM(IELEM, 4) - INT1
>           INT1=PZX1*FACUW3+PZY1*FACVW3
>           XM(IELEM, 2)= XM(IELEM, 2) + INT1
>           XM(IELEM, 5)= XM(IELEM, 5) - INT1
>           INT1=PZX2*FACUW1+PZY2*FACVW1
>           XM(IELEM,16)= XM(IELEM,16) + INT1
>           XM(IELEM, 7)= XM(IELEM, 7) - INT1
>           INT1=PZX2*FACUW2+PZY2*FACVW2
>           T (IELEM, 2)= T (IELEM, 2) + INT1
>           XM(IELEM, 8)= XM(IELEM, 8) - INT1
>           INT1=PZX2*FACUW3+PZY2*FACVW3
>           XM(IELEM, 6)= XM(IELEM, 6) + INT1
>           XM(IELEM, 9)= XM(IELEM, 9) - INT1
>           INT1=PZX1*FACUW4+PZY1*FACVW4
>           XM(IELEM,18)= XM(IELEM,18) + INT1
>           T (IELEM, 4)= T (IELEM, 4) - INT1
>           INT1=PZX1*FACUW5+PZY1*FACVW5
>           XM(IELEM,22)= XM(IELEM,22) + INT1
>           XM(IELEM,13)= XM(IELEM,13) - INT1
>           INT1=PZX1*FACUW6+PZY1*FACVW6
>           XM(IELEM,25)= XM(IELEM,25) + INT1
>           XM(IELEM,14)= XM(IELEM,14) - INT1
>           INT1=PZX2*FACUW4+PZY2*FACVW4
>           XM(IELEM,19)= XM(IELEM,19) + INT1
>           XM(IELEM,28)= XM(IELEM,28) - INT1
>           INT1=PZX2*FACUW5+PZY2*FACVW5
>           XM(IELEM,23)= XM(IELEM,23) + INT1
>           T (IELEM, 5)= T (IELEM, 5) - INT1
>           INT1=PZX2*FACUW6+PZY2*FACVW6
>           XM(IELEM,26)= XM(IELEM,26) + INT1
>           XM(IELEM,15)= XM(IELEM,15) - INT1

FILE  bief/bief_v5p4/sources/mt04tt.f
68a69,72
>       IMPLICIT NONE
>
>       DOUBLE PRECISION  U1, U2, U3, U4, V1, V2, V3, V4, W1, W2, W3, W4
>       DOUBLE PRECISION  XJAC, AUX1, AUX2, AUX3, AUX4

FILE  bief/bief_v5p4/sources/mt05aa.f
91d90
<       INTRINSIC MAX,MIN

FILE  bief/bief_v5p4/sources/mt05pp.f
92c92
<       DOUBLE PRECISION INT,X2,X3,Y2,Y3,DEN
---
>       DOUBLE PRECISION INT1,X2,X3,Y2,Y3,DEN
```

```
233,280c233,280
<           INT=PZ1*2*(3*W41+W52+W63)
<           T(IELEM,1)=INT
<           XM(IELEM,3)=-INT
<           INT=PZ1*(2*(W41+W52)+W63)
<           XM(IELEM,1)=INT
<           XM(IELEM,4)=-INT
<           XM(IELEM,16)=INT
<           XM(IELEM,7)=-INT
<           INT=PZ1*(2*(W41+W63)+W52)
<           XM(IELEM,2)=INT
<           XM(IELEM,5)=-INT
<           XM(IELEM,17)=INT
<           XM(IELEM,10)=-INT
<           INT=PZ1*2*(W41+3*W52+W63)
<           T(IELEM,2)=INT
<           XM(IELEM,8)= -INT
<           INT=PZ1*(2*(W52+W63)+W41)
<           XM(IELEM,6)=INT
<           XM(IELEM,9)=-INT
<           XM(IELEM,21)=INT
<           XM(IELEM,11)=-INT
<           INT=PZ1*2*(W41+W52+3*W63)
<           T(IELEM,3)=INT
<           XM(IELEM,12)=-INT
<           INT=PZ1*2*(3*W14+W25+W36)
<           XM(IELEM,18)=INT
<           T(IELEM,4)=-INT
<           INT=PZ1*(2*(W14+W25)+W36)
<           XM(IELEM,22)=INT
<           XM(IELEM,13)=-INT
<           XM(IELEM,19)=INT
<           XM(IELEM,28)=-INT
<           INT=PZ1*(2*(W14+W36)+W25)
<           XM(IELEM,25)=INT
<           XM(IELEM,14)=-INT
<           XM(IELEM,20)=INT
<           XM(IELEM,29)=-INT
<           INT=PZ1*2*(W14+3*W25+W36)
<           XM(IELEM,23)=INT
<           T(IELEM,5)=-INT
<           INT=PZ1*(2*(W25+W36)+W14)
<           XM(IELEM,26)=INT
<           XM(IELEM,15)=-INT
<           XM(IELEM,24)=INT
<           XM(IELEM,30)=-INT
<           INT=PZ1*2*(W14+W25+3*W36)
<           XM(IELEM,27)=INT
<           T(IELEM,6)=-INT
---
>           INT1=PZ1*2*(3*W41+W52+W63)
>           T(IELEM,1)=INT1
>           XM(IELEM,3)=-INT1
>           INT1=PZ1*(2*(W41+W52)+W63)
>           XM(IELEM,1)=INT1
>           XM(IELEM,4)=-INT1
>           XM(IELEM,16)=INT1
```

```
>           XM(IELEM,7)=-INT1
>           INT1=PZ1*(2*(W41+W63)+W52)
>           XM(IELEM,2)=INT1
>           XM(IELEM,5)=-INT1
>           XM(IELEM,17)=INT1
>           XM(IELEM,10)=-INT1
>           INT1=PZ1*2*(W41+3*W52+W63)
>           T(IELEM,2)=INT1
>           XM(IELEM,8)= -INT1
>           INT1=PZ1*(2*(W52+W63)+W41)
>           XM(IELEM,6)=INT1
>           XM(IELEM,9)=-INT1
>           XM(IELEM,21)=INT1
>           XM(IELEM,11)=-INT1
>           INT1=PZ1*2*(W41+W52+3*W63)
>           T(IELEM,3)=INT1
>           XM(IELEM,12)=-INT1
>           INT1=PZ1*2*(3*W14+W25+W36)
>           XM(IELEM,18)=INT1
>           T(IELEM,4)=-INT1
>           INT1=PZ1*(2*(W14+W25)+W36)
>           XM(IELEM,22)=INT1
>           XM(IELEM,13)=-INT1
>           XM(IELEM,19)=INT1
>           XM(IELEM,28)=-INT1
>           INT1=PZ1*(2*(W14+W36)+W25)
>           XM(IELEM,25)=INT1
>           XM(IELEM,14)=-INT1
>           XM(IELEM,20)=INT1
>           XM(IELEM,29)=-INT1
>           INT1=PZ1*2*(W14+3*W25+W36)
>           XM(IELEM,23)=INT1
>           T(IELEM,5)=-INT1
>           INT1=PZ1*(2*(W25+W36)+W14)
>           XM(IELEM,26)=INT1
>           XM(IELEM,15)=-INT1
>           XM(IELEM,24)=INT1
>           XM(IELEM,30)=-INT1
>           INT1=PZ1*2*(W14+W25+3*W36)
>           XM(IELEM,27)=INT1
>           T(IELEM,6)=-INT1

FILE  bief/bief_v5p4/sources/mt06ff.f
88d87
<        INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/mt06ft.f
87d86
<        INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/mt06tt.f
42a43,44
>        IMPLICIT NONE
>        DOUBLE PRECISION JACOB

FILE  bief/bief_v5p4/sources/mt08pp.f
81c81
```

```
<        DOUBLE PRECISION INT,PZ1,XSU360
---
>        DOUBLE PRECISION INT1,PZ1,XSU360
129,176c129,177
<           INT=PZ1*2*(3*W41+W52+W63)
<           T(IELEM,1)=INT
<           XM(IELEM,18)=-INT
<           INT=PZ1*(2*(W41+W52)+W63)
<           XM(IELEM,16)=INT
<           XM(IELEM,19)=-INT
<           XM(IELEM,1)=INT
<           XM(IELEM,22)=-INT
<           INT=PZ1*(2*(W41+W63)+W52)
<           XM(IELEM,2)=INT
<           XM(IELEM,20)=-INT
<           XM(IELEM,17)=INT
<           XM(IELEM,25)=-INT
<           INT=PZ1*2*(W41+3*W52+W63)
<           T(IELEM,2)=INT
<           XM(IELEM,23)= -INT
<           INT=PZ1*(2*(W52+W63)+W41)
<           XM(IELEM,21)=INT
<           XM(IELEM,24)=-INT
<           XM(IELEM,6)=INT
<           XM(IELEM,26)=-INT
<           INT=PZ1*2*(W41+W52+3*W63)
<           T(IELEM,3)=INT
<           XM(IELEM,27)=-INT
<           INT=PZ1*2*(3*W14+W25+W36)
<           XM(IELEM,3)=INT
<           T(IELEM,4)=-INT
<           INT=PZ1*(2*(W14+W25)+W36)
<           XM(IELEM,7)=INT
<           XM(IELEM,28)=-INT
<           XM(IELEM,4)=INT
<           XM(IELEM,13)=-INT
<           INT=PZ1*(2*(W14+W36)+W25)
<           XM(IELEM,10)=INT
<           XM(IELEM,29)=-INT
<           XM(IELEM,5)=INT
<           XM(IELEM,14)=-INT
<           INT=PZ1*2*(W14+3*W25+W36)
<           XM(IELEM,8)=INT
<           T(IELEM,5)=-INT
<           INT=PZ1*(2*(W25+W36)+W14)
<           XM(IELEM,11)=INT
<           XM(IELEM,30)=-INT
<           XM(IELEM,9)=INT
<           XM(IELEM,15)=-INT
<           INT=PZ1*2*(W14+W25+3*W36)
<           XM(IELEM,12)=INT
<           T(IELEM,6)=-INT
---
>           INT1=PZ1*2*(3*W41+W52+W63)
>           T(IELEM,1)=INT1
>           XM(IELEM,18)=-INT1
>           INT1=PZ1*(2*(W41+W52)+W63)
```

```
>           XM(IELEM,16)=INT1
>           XM(IELEM,19)=-INT1
>           XM(IELEM,1)=INT1
>           XM(IELEM,22)=-INT1
>           INT1=PZ1*(2*(W41+W63)+W52)
>           XM(IELEM,2)=INT1
>           XM(IELEM,20)=-INT1
>           XM(IELEM,17)=INT1
>           XM(IELEM,25)=-INT1
>           INT1=PZ1*2*(W41+3*W52+W63)
>           T(IELEM,2)=INT1
>           XM(IELEM,23)= -INT1
>           INT1=PZ1*(2*(W52+W63)+W41)
>           XM(IELEM,21)=INT1
>           XM(IELEM,24)=-INT1
>           XM(IELEM,6)=INT1
>           XM(IELEM,26)=-INT1
>           INT1=PZ1*2*(W41+W52+3*W63)
>           T(IELEM,3)=INT1
>           XM(IELEM,27)=-INT1
>           INT1=PZ1*2*(3*W14+W25+W36)
>           XM(IELEM,3)=INT1
>           T(IELEM,4)=-INT1
>           INT1=PZ1*(2*(W14+W25)+W36)
>           XM(IELEM,7)=INT1
>           XM(IELEM,28)=-INT1
>           XM(IELEM,4)=INT1
>           XM(IELEM,13)=-INT1
>           INT1=PZ1*(2*(W14+W36)+W25)
>           XM(IELEM,10)=INT1
>           XM(IELEM,29)=-INT1
>           XM(IELEM,5)=INT1
>           XM(IELEM,14)=-INT1
>           INT1=PZ1*2*(W14+3*W25+W36)
>           XM(IELEM,8)=INT1
>           T(IELEM,5)=-INT1
>           INT1=PZ1*(2*(W25+W36)+W14)
>           XM(IELEM,11)=INT1
>           XM(IELEM,30)=-INT1
>           XM(IELEM,9)=INT1
>           XM(IELEM,15)=-INT1
>           INT1=PZ1*2*(W14+W25+3*W36)
>           XM(IELEM,12)=INT1
>           T(IELEM,6)=-INT1
>

FILE  bief/bief_v5p4/sources/mvseg.f
84d83
<        INTRINSIC MIN,MAX

FILE  bief/bief_v5p4/sources/normab.f
81d80
<        INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/omseg.f
71d70
<        INTRINSIC MIN
```

```

FILE  bief/bief_v5p4/sources/ov.f
101d100
<         INTRINSIC SQRT,ABS,COS,SIN,ATAN,MAX,MIN

FILE  bief/bief_v5p4/sources/paraco.f
91d90
<         INTRINSIC ABS

FILE  bief/bief_v5p4/sources/proxim.f
42d41
<         INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/rem11.f
61d60
<         INTRINSIC MIN

FILE  bief/bief_v5p4/sources/rem21.f
66d65
<         INTRINSIC MIN

FILE  bief/bief_v5p4/sources/rem41.f
73d72
<         INTRINSIC MIN

FILE  bief/bief_v5p4/sources/rescjg.f
98d97
<         INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/slop10.f
57d56
<         INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/sortie.f
47d46
<         INTRINSIC LEN

FILE  bief/bief_v5p4/sources/sum.f
2c2
<                             DOUBLE PRECISION FUNCTION SUM
---
>                             DOUBLE PRECISION FUNCTION SUM_BIEF
38c38
<         USE BIEF, EX_SUM => SUM
---
>         USE BIEF, EX_SUM_BIEF => SUM_BIEF
56c56
<           SUM = SOMME(X%R,X%DIM1)
---
>           SUM_BIEF = SOMME(X%R,X%DIM1)

FILE  bief/bief_v5p4/sources/tsloc.f
5c5
<        * (YEAR,MONTH,DAY,HOUR,MIN,SEC,AT)
---
>        * (YEAR,MONTH,DAY,HOUR,MINUTES,SEC,AT)
56c56
```

```
<       INTEGER YEAR,MONTH,DAY,HOUR,MIN,SEC
---
>       INTEGER YEAR,MONTH,DAY,HOUR,MINUTES,SEC
61d60
<       INTRINSIC ACOS,INT
65c64
<       ATR = AT + ( HOUR * 60.D0 + MIN ) * 60.D0 + SEC
---
>       ATR = AT + ( HOUR * 60.D0 + MINUTES ) * 60.D0 + SEC

FILE  bief/bief_v5p4/sources/valida.f
0a1
> #include "compad_component_handler.fpp"
66d66
<       INTRINSIC MAX
125,126c125,128
<            IF(LNG.EQ.1) WRITE(LU,60) TEXTRES(IVAR)(1:16),P_DMAX(ERRMAX)
<            IF(LNG.EQ.2) WRITE(LU,61) TEXTRES(IVAR)(1:16),P_DMAX(ERRMAX)
---
>            IF(LNG.EQ.1) WRITE(LU,60) TEXTRES(IVAR)(1:16),
>      $            COMPADVAL( P_DMAX(ERRMAX) )
>            IF(LNG.EQ.2) WRITE(LU,61) TEXTRES(IVAR)(1:16),
>      $            COMPADVAL(  P_DMAX(ERRMAX) )
128,129c130,133
<            IF(LNG.EQ.1) WRITE(LU,60) TEXTRES(IVAR)(1:16),ERRMAX
<            IF(LNG.EQ.2) WRITE(LU,61) TEXTRES(IVAR)(1:16),ERRMAX
---
>            IF(LNG.EQ.1) WRITE(LU,60) TEXTRES(IVAR)(1:16),
>      $            COMPADVAL( ERRMAX )
>            IF(LNG.EQ.2) WRITE(LU,61) TEXTRES(IVAR)(1:16),
>      $            COMPADVAL( ERRMAX )

FILE  bief/bief_v5p4/sources/vc00ff.f
64d63
<       INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/vc00ft.f
66d65
<       INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/vc01ff.f
74d73
<       INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/vc01ft.f
76d75
<       INTRINSIC SQRT

FILE  bief/bief_v5p4/sources/vc08aa.f
75d74
<       INTRINSIC MIN,MAX,SIGN

FILE  bief/bief_v5p4/sources/vc08bb.f
84d83
<       INTRINSIC MAX,MIN,SIGN

FILE  bief/bief_v5p4/sources/vgfpsi.f
```

```
51d50
<       INTRINSIC INT

FILE  bief/bief_v5p4/sources/waitfor.f
61d60
<       INTRINSIC TRIM
```

```
FILE  damocles/damo_v5p4/sources/cmd.f
0a1
> #include "compad_component_handler.fpp"
213c214,215
<      *          MOTCLE(N,I)(1:ISIZE),IAD,MOTATT(N,IAD)(1:L1),MOTREA(IAD)
---
>      *             MOTCLE(N,I)(1:ISIZE),IAD,MOTATT(N,IAD)(1:L1),
>      *             COMPADVAL(MOTREA(IAD))

FILE  damocles/damo_v5p4/sources/intlu.f
82d81
<       INTRINSIC DLOG10,DBLE,INT,CHAR

FILE  damocles/damo_v5p4/sources/realu.f
0a1
> #include "compad_component_handler.fpp"
86d86
<       INTRINSIC DLOG10,DBLE,INT,CHAR
229c229
<       READ  ( LIGNE , FORMA , ERR=995 ) RVAL
---
>       READ  ( LIGNE , FORMA , ERR=995 ) COMPADVALCOMP(RVAL)
```

```
FILE  sisyphe/sisyphe_v5p4/sources/bilan_sisyphe.f
0a1
> #include "compad_component_handler.fpp"
99c100
<         RMASSE = SUM(T1)
---
>         RMASSE = SUM_BIEF(T1)
107c108
<         RCUMU = SUM(T1)
---
>         RCUMU = SUM_BIEF(T1)
116c117
<         FLUDDL = SUM(T1)
---
>         FLUDDL = SUM_BIEF(T1)
124c125
<       FLUDIR = SUM(T1)
---
>       FLUDIR = SUM_BIEF(T1)
140c141
<         RMASCLA(I) = SUM(T1)
---
```

```
>         RMASCLA(I) = SUM_BIEF(T1)
148c149
<         FLUDDLCLA(I) = SUM(T1)
---
>         FLUDDLCLA(I) = SUM_BIEF(T1)
155c156
<        FLUDIRCLA(I) = SUM(T1)
---
>        FLUDIRCLA(I) = SUM_BIEF(T1)
170c171
<          MASST = SUM(T1)
---
>          MASST = SUM_BIEF(T1)
181,185c182,186
<             WRITE(LU,1010) RMASSE
<             IF(ABS(FLUDIR).GT.1.D-10) WRITE(LU,1020) -FLUDIR
<             IF(ABS(FLUDDL).GT.1.D-10) WRITE(LU,1021) -FLUDDL
<             WRITE(LU,1030) RCUMU
<             WRITE(LU,1031) VCUMU
---
>             WRITE(LU,1010) COMPADVAL(RMASSE)
>             IF(ABS(FLUDIR).GT.1.D-10) WRITE(LU,1020) COMPADVAL(-FLUDIR)
>             IF(ABS(FLUDDL).GT.1.D-10) WRITE(LU,1021) COMPADVAL(-FLUDDL)
>             WRITE(LU,1030) COMPADVAL(RCUMU)
>             WRITE(LU,1031) COMPADVAL(VCUMU)
188,192c189,193
<             WRITE(LU,2010) RMASSE
<             IF(ABS(FLUDIR).GT.1.D-10) WRITE(LU,2020) -FLUDIR
<             IF(ABS(FLUDDL).GT.1.D-10) WRITE(LU,2021) -FLUDDL
<             WRITE(LU,2030) RCUMU
<             WRITE(LU,2031) VCUMU
---
>             WRITE(LU,2010) COMPADVAL(RMASSE)
>             IF(ABS(FLUDIR).GT.1.D-10) WRITE(LU,2020) COMPADVAL(-FLUDIR)
>             IF(ABS(FLUDDL).GT.1.D-10) WRITE(LU,2021) COMPADVAL(-FLUDDL)
>             WRITE(LU,2030) COMPADVAL(RCUMU)
>             WRITE(LU,2031) COMPADVAL(VCUMU)
199,200c200,201
<              WRITE(LU,3011) RMASCLA(I)
<              WRITE(LU,3032) VCUMUCLA(I)
---
>              WRITE(LU,3011) COMPADVAL(RMASCLA(I))
>              WRITE(LU,3032) COMPADVAL(VCUMUCLA(I))
204,205c205,206
<              WRITE(LU,3010) RMASCLA(I)
<              WRITE(LU,3031) VCUMUCLA(I)
---
>              WRITE(LU,3010) COMPADVAL(RMASCLA(I))
>              WRITE(LU,3031) COMPADVAL(VCUMUCLA(I))
211,212c212,213
<                WRITE(LU, 4000) MASST
<                WRITE(LU, 4010) MASS_GF
---
>                WRITE(LU, 4000) COMPADVAL(MASST)
>                WRITE(LU, 4010) COMPADVAL(MASS_GF)
215,216c216,217
<                WRITE(LU, 4001) MASST
```

```
<                 WRITE(LU, 4011) MASS_GF
---
>                 WRITE(LU, 4001) COMPADVAL(MASST)
>                 WRITE(LU, 4011) COMPADVAL(MASS_GF)

FILE  sisyphe/sisyphe_v5p4/sources/bilan_susp.f
120d119
<        INTRINSIC ABS

FILE  sisyphe/sisyphe_v5p4/sources/calcuw.f
42d41
<        INTRINSIC SQRT, SINH

FILE  sisyphe/sisyphe_v5p4/sources/cfl.f
57d56
<        INTRINSIC MAX,MIN,ABS

FILE  sisyphe/sisyphe_v5p4/sources/coefro_sisyphe.f
53d52
<        INTRINSIC SQRT,MAX,LOG

FILE  sisyphe/sisyphe_v5p4/sources/disper_susp.f
64d63
<        INTRINSIC  SQRT

FILE  sisyphe/sisyphe_v5p4/sources/effpnt.f
0a1
> #include "compad_component_handler.fpp"
83c84
<        PI = ACOS(-1.D0)
---
>        PI = ACOS(COMPADVAL(-1.D0))

FILE  sisyphe/sisyphe_v5p4/sources/entete_sisyphe.f
44d43
<        INTRINSIC INT
94,95c93,94
<            IF(LNG.EQ.1) WRITE(LU,10) FR(11),LT,FR(9),J,H,M,S,AT
<            IF(LNG.EQ.2) WRITE(LU,11) GB(11),LT,GB(9),J,H,M,S,AT
---
>            IF(LNG.EQ.1) WRITE(LU,*) FR(11),LT,FR(9),J,H,M,S,AT
>            IF(LNG.EQ.2) WRITE(LU,*) GB(11),LT,GB(9),J,H,M,S,AT
97,98c96,97
<            IF(LNG.EQ.1) WRITE(LU,20) FR(11),LT,FR(9),H,M,S,AT
<            IF(LNG.EQ.2) WRITE(LU,20) GB(11),LT,GB(9),H,M,S,AT
---
>            IF(LNG.EQ.1) WRITE(LU,*) FR(11),LT,FR(9),H,M,S,AT
>            IF(LNG.EQ.2) WRITE(LU,*) GB(11),LT,GB(9),H,M,S,AT
100,101c99,100
<            IF(LNG.EQ.1) WRITE(LU,30) FR(11),LT,FR(9),M,S,AT
<            IF(LNG.EQ.2) WRITE(LU,30) GB(11),LT,GB(9),M,S,AT
---
>            IF(LNG.EQ.1) WRITE(LU,*) FR(11),LT,FR(9),M,S,AT
>            IF(LNG.EQ.2) WRITE(LU,*) GB(11),LT,GB(9),M,S,AT
103,104c102,103
<            IF(LNG.EQ.1) WRITE(LU,40) FR(11),LT,FR(9),S
<            IF(LNG.EQ.2) WRITE(LU,40) GB(11),LT,GB(9),S
```

```
---
>             IF(LNG.EQ.1) WRITE(LU,*) FR(11),LT,FR(9),S
>             IF(LNG.EQ.2) WRITE(LU,*) GB(11),LT,GB(9),S

FILE  sisyphe/sisyphe_v5p4/sources/flused.f
70d69
<         INTRINSIC SQRT

FILE  sisyphe/sisyphe_v5p4/sources/flused_susp.f
57d56
<         INTRINSIC SQRT , MAX

FILE  sisyphe/sisyphe_v5p4/sources/krone_part.f
54d53
<         INTRINSIC SQRT , MAX

FILE  sisyphe/sisyphe_v5p4/sources/layer.f
288c288
<            NLAYER%I(J) = NLAYNEW(J)
---
>            NLAYER%I(J) = INT(NLAYNEW(J))

FILE  sisyphe/sisyphe_v5p4/sources/leclis.f
0a1,2
> #include "compad_component_handler.fpp"
>
87,89c89,94
<      &                 bid , bid, bid,
<      &                 bid,
<      &                 liebor(k), eborlu(k), afbor(k), bfbor(k),
---
>      &                 COMPADVALCOMP(bid) , COMPADVALCOMP(bid),
>      &                 COMPADVALCOMP(bid) , COMPADVALCOMP(bid),
>      &                 liebor(k),
>      &                 COMPADVALCOMP(eborlu(k)),
>      &                 COMPADVALCOMP(afbor(k)),
>      &                 COMPADVALCOMP(bfbor(k)),
95,99c100,109
<      &                 bid, bid, bid,
<      &                 bid,
<      &                 liebor(k), eborlu(k), afbor(k), bfbor(k),
<      &                 nbor(k), kfich,
<      &                 iseg(k), xseg(k), yseg(k), numliq(k)
---
>      &                 COMPADVALCOMP(bid) , COMPADVALCOMP(bid),
>      &                 COMPADVALCOMP(bid) , COMPADVALCOMP(bid),
>      &                 liebor(k),
>      &                 COMPADVALCOMP(eborlu(k)),
>      &                 COMPADVALCOMP(afbor(k)),
>      &                 COMPADVALCOMP(bfbor(k)),
>      &                 nbor(k), kfich, iseg(k),
>      &                 COMPADVALCOMP(xseg(k)), COMPADVALCOMP(yseg(k)),
>      &                 numliq(k)
>
```

Configuring the compilation with AD-enabled NAGWare Fortran compiler Add the following section to configuration file `config/systel.ini`:

```
#
#------------- Compad II Project,
# Generic Linux + NAGWare f95 v 5.1, TLM AD (scalar) by overloading
#
[compadII-TLM-ovl-scalar-linux-nagf95]
DIRLIB="compadII-TLM-ovl-scalar-linux-nagf95"
FC_NAM="/home/riehme/WORK-COMPADII/PROJECTS/BAW-AD/../../F95"
FC_OPT_OBJEXT="o"
FC_OPT_COMPIL="␣-c␣-ad_scalar␣-ad_ovl␣-DCOMPADII␣-DCOMPADII_ACCESS_COMPADTYPE_COMPONENTS␣"
FC_OPT_DEBUG="␣-O4␣"
FC_OPT_PROFILE="␣"
FC_OPT_INCLUDE="␣-I␣"
FC_OPT_OTHERS="␣-fpp␣-I␣/home/riehme/WORK-COMPADII/PROJECTS/BAW-AD/../../PROJECTS/BAW-AD/␣-DCOMPADMODULE=compad_scalar_module␣"
#
LK_NAM="/home/riehme/WORK-COMPADII/PROJECTS/BAW-AD/../../F95"
LK_OPT_NORMAL=
LK_OPT_OUTNAME="␣-g␣␣-o␣"
LK_OPT_DEBUG="␣"
LK_OPT_PROFILE="␣"
LK_LIB_SPECIAL="␣/home/riehme/WORK-COMPADII/PROJECTS/BAW-AD/../../PROJECTS/BAW-AD//compad_scalar_module.o␣"
#
LIB_NAM=ar
LIB_OPT_LIBEXT="a"
LIB_OPT_OUTNAME="cru"
LIB_OPT_OTHERS=
LIB_RANLIB="ranlib"
#
RUN_DEBUG="␣"
RUN_PROFILE=
#
FC_MPI="${F95}␣-c␣-O1␣"
LK_MPI="${F95}␣-lmpi␣␣-o␣<EXE>␣␣<OBJS>␣␣<LIBS>␣"
LIBS_MPI="-lmpi"
#RUN_MPI="mpirun -np <N> <EXE>"
RUN_MPI="mpirun␣-np␣<N>␣<EXE>"
#RUN_MPI="mpirun -np <N> /sw/sdev/histx_1.2a/bin/lipfpm -e FP_OPS_RETIRED -e LOADS_RETIRED -e STORES_RETIRED -e CPU_CYCLES -f   <EXE>"
#RUN_MPI="mpirun -np <N> /usr/local/bin/profile.pl -s1 -c0-63 <EXE>"
#
```

For passive compilation of the system:

```
#
#------------- Compad II Project,
# Generic Linux + NAGWare f95 v 5.1, passive compilation
#
[compadII-passive-linux-nagf95]
DIRLIB="compadII-passive-linux-nagf95" ### COMPADIIHOSTTYPE
FC_NAM="/home/riehme/WORK-COMPADII/PROJECTS/BAW-AD/../../F95"
FC_OPT_OBJEXT="o"
FC_OPT_COMPIL="␣-O4␣"
FC_OPT_DEBUG="␣-g␣"
FC_OPT_PROFILE="␣"
```

```
FC_OPT_INCLUDE="␣-I␣"
FC_OPT_OTHERS="␣-fpp␣-I␣/home/riehme/WORK-COMPADII/PROJECTS/BAW-AD/../../PROJECTS/BAW-AD/␣"
#
LK_NAM="/home/riehme/WORK-COMPADII/PROJECTS/BAW-AD/../../F95"
LK_OPT_NORMAL=
LK_OPT_OUTNAME="␣-o␣"
LK_OPT_DEBUG="␣"
LK_OPT_PROFILE="␣"
LK_LIB_SPECIAL="␣␣"
#
LIB_NAM=ar
LIB_OPT_LIBEXT="a"
LIB_OPT_OUTNAME="cru"
LIB_OPT_OTHERS=
LIB_RANLIB="ranlib"
#
RUN_DEBUG="␣"
RUN_PROFILE=
#
FC_MPI="${F95}␣-c␣-O1␣"
LK_MPI="${F95}␣-lmpi␣␣-o␣<EXE>␣␣<OBJS>␣␣<LIBS>␣"
LIBS_MPI="-lmpi"
#RUN_MPI="mpirun -np <N> <EXE>"
RUN_MPI="mpirun␣-np␣<N>␣<EXE>"
#RUN_MPI="mpirun -np <N> /sw/sdev/histx_1.2a/bin/lipfpm -e FP_OPS_RETIRED -e LOADS_RETIRED -e STORES_RETIRED -e CPU_CYCLES -f   <EXE>"
#RUN_MPI="mpirun -np <N> /usr/local/bin/profile.pl -s1 -c0-63 <EXE>"
#
```

Note that after changing the configuration a complete re-compile is required after explicitly cleaning all sub-projects.