Tape validation and verification

M. Sagebaum

AG Scientific Computing
TU Kaiserslautern

AD Workshop London
Overview

- Motivation
- Interface validation
- Tape validation
Motivation - TRACE CFD Suite

Example: A380 Turbofan: GP7000

Program shares:
30.0 % P&W
30.0 % GE
22.5 % MTU
10.0 % Snecma
7.5 % Techspace Aero

MTU share (22.5 %):
5.0 % HP Turbine
5.0 % Turbine Center Frame
12.5 % LP Turbine

Picture www.mtu.de
Motivation - TRACE CFD Suite

Example: A380 Turbofan: GP7000

Inlet  Fan  Compressor  Combustion chamber  Turbine

**Program shares:**
- 30.0% P&W
- 30.0% GE
- 22.5% MTU
- 10.0% Snecma
- 7.5% Techspace Aero

**MTU share (22.5%):**
- 5.0% HP Turbine
- 5.0% Turbine Center Frame
- 12.5% LP Turbine

Picture www.mtu.de
Motivation - TRACE CFD Suite

- Finite volume solver
- (U)RANS equations and many more
- Structured and unstructured grids
- Multidisciplinary:
  - Aerodynamics, aeroelastics, aeroacoustics, combustion
- Multi stage computations
- Complex geometries
  - Fillets, inlets, cavities, bleeds, etc.

Pictures DLR-AT
Motivation - Validation of interfaces

TRACE: \[ y = G(y, u) \]

- AD creates required derivatives
- Validation of derivatives with finite differences
- Implicit assumption: \( y \) and \( u \) are well defined and known “interfaces”
Motivation - Validation of interfaces

TRACE: \( y = G(y, u) \)

- AD creates required derivatives
- Validation of derivatives with finite differences
- Implicit assumption: \( y \) and \( u \) are well defined and known “interfaces”

Assumption in this talk: The interfaces of \( y \) und \( u \) are not or only partly known
- How to validate and complete the interface definitions of \( y \) and \( u \)?
Validation of interfaces

Idea: Use AD techniques for the validation

Approach:

- All variables in $y$ are tagged (e.g. 1)
- $G$ is run once
  - All variables that depend on $y$ have now the tag (e.g. 1)
- Each variable in $y$ gets a different tag (e.g. 2)
- $G$ is run a second time
  - If a variable is encountered with the old tag (i.e. 1) it needs to be added to the interface $y$
Validation of interfaces

Example:

```c
void iter(double y1, double y2, double& y1_new, double& y2_new) {
    double t = y1 * y2; //
    y1_new = y1 + t;
    y2_new = y2 + t;
}
```
Validation of interfaces

Example:

```c
void iter(double y1, double y2, double& y1_new, double& y2_new) {
    double t = y1 * y2;  //
    y1_new = y1 + t;
    y2_new = y2 + t;
}
```

- Step 1: $y1$ is defined as the interface and gets the tag 1
Validation of interfaces

Example:

```c
void iter(double y1, double y2, double& y1_new, double& y2_new) {
    double t = y1 * y2; //
    y1_new = y1 + t;
    y2_new = y2 + t;
}
```

- Step 1: `y1` is defined as the interface and gets the tag 1
- Step 2: `iter` is called
Validation of interfaces

Example:

```c
void iter(double y1, double y2, double& y1_new, double& y2_new) {
    double t = y1 * y2;  //
    y1_new = y1 + t;
    y2_new = y2 + t;
}
```

- Step 1: $y1$ is defined as the interface and gets the tag 1
- Step 2: $iter$ is called
- Step 3: $y1$ is defined as the interface and get the tag 2
Validation of interfaces

Example:

```c
void iter(double y1, double y2, double& y1_new, double& y2_new) {
    double t = y1 * y2; // Error
    y1_new = y1 + t;
    y2_new = y2 + t;
}
```

- Step 1: $y1$ is defined as the interface and gets the tag 1
- Step 2: `iter` is called
- Step 3: $y1$ is defined as the interface and get the tag 2
- Step 4: `iter` is called
Validation of interfaces

Example:

```c
void iter(double y1, double y2, double& y1_new, double& y2_new) {
    double t = y1 * y2;  // Error
    y1_new = y1 + t;
    y2_new = y2 + t;
}
```

- Step 1: y1 is defined as the interface and gets the tag 1
- Step 2: iter is called
- Step 3: y1 is defined as the interface and gets the tag 2
- Step 4: iter is called
  \Rightarrow y2 is also part of the interface
Validation of interfaces - Implementation

- Via operator overloading
- Define the active structure:

```c
struct TReal {    // Tag Real
    double primal;
    int tag;
};
```

- Overloading of operators/functions

```c
TReal operator + (const TReal& a, const TReal& b) {
    RReal w;
    w.primal = a.primal + b.primal;

    if(a.tag != globalTag || b.tag != globalTag) {
        throwTagError();
    }

    w.tag = globalTag;
    return w;
}
```
Validation of interfaces

Validation of arbitrary interfaces:

```c
void program(Interface& a, Interface& b) {
    firstPart();
    secondPart();
}
```

- Interface \(a\) is given
- Interface \(b\) needs to be detected/extended
- Variables are used indirectly in \(firstPart\) and \(secondPart\)
Validation of interfaces

Validation of arbitrary interfaces:

```c
void program(Interface& a, Interface& b) {
    firstPart();
    secondPart();
}
```

- Interface \( a \) is given
- Interface \( b \) needs to be detected/extended
- Variables are used indirectly in \( \text{firstPart} \) and \( \text{secondPart} \)

Algorithm for the detection:

```c
void program(Interface& a, Interface& b) {
    a.setTag(1);
    firstPart();
    a.setTag(2);
    b.setTag(2);
    secondPart(); // On error add variable to interface \( b \)
}
```
Validation of tapes

Why do we need tape validation?
- Change from Black Box process to checkpoints and recompute
- Is computational path still the same?

Ingredients:
- Validate interfaces (e.g. variables stored in checkpoints)
- Validate tapes (e.g. is the same computational path taken)
Validation of tapes - Motivation

Black box process:

Step 1: Record

\[ x \rightarrow \text{record} \rightarrow z \]

Step 2: Reverse

\[ \bar{x} \leftarrow \text{reverse} \leftarrow \bar{z} \]

Black box with checkpoints process:

Step 1: Create Checkpoints

\[ x \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow z \]

Step 2: Record from s3

Step 3: Reverse

Step 4: Record from s2

Step 5: Reverse

Step 6: Record from s1

Step 7: Reverse

\[ \bar{x} \leftarrow \text{store/load adjoint state} \leftarrow \bar{z} \]
Validation of tapes - Prerequisites

**Definition: Correct interface**

For a tape that is created by the function \( y = F(x) \) with \( x \in \mathbb{R}^n \) and \( y \in \mathbb{R}^m \). The interfaces defined by the input variables \( x \) and the output variables \( y \) are correct if and only if for all \( i \in [1, m] \subset \mathbb{N} \) there exists an index \( j \in [1, n] \subset \mathbb{N} \) such that
\[
\frac{\partial F_i}{\partial x_j} \neq 0.
\]

- Short: Jacobian matrix should contain no row that is zero
Validation of tapes - Prerequisites

**Theorem: Equality of tapes**

Let \( p = F(a) \) be the definition of the master tape (M) with \( a \in \mathbb{R}^{n_M} \) as the input interface and \( p \in \mathbb{R}^{m_M} \) as the output interface.

Let \( y = F(x) \) be the definition of the secondary tape (S) with \( x \in \mathbb{R}^{n_S} \) as the input interface and \( y \in \mathbb{R}^{m_S} \) as the output interface.

Let \( F \) be in both cases the same function that is used with different sets of parameters.

Without loss of generality let \( x = (a, b) \) with \( b \in \mathbb{R}^{n_S-n_M} \) and \( y = (p, q) \) with \( q \in \mathbb{R}^{m_S-m_M} \).

Let the interfaces \( p \) and \( a \) of the master tape be correct.

Then \( \bar{q} \neq 0 \) has no influence on \( \bar{a} \).

\[
\bar{x} = \frac{\partial F^T}{\partial x} \bar{y} \approx \begin{pmatrix} \bar{a} \\ \bar{b} \end{pmatrix} = \frac{\partial F}{\partial (a,b)}^T (a,b) \begin{pmatrix} \bar{p} \\ \bar{q} \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \bar{p} \\ \bar{q} \end{pmatrix}
\]

- Short: Entry \( B \) in the Jacobian matrix is equal to zero.
Validation of tapes - Results

\[
\bar{x} = \frac{\partial F^T}{\partial x} \bar{y} = \begin{pmatrix} \bar{a} \\ \bar{b} \end{pmatrix} = \frac{\partial F}{\partial (a,b)}^T (a,b) \begin{pmatrix} \bar{p} \\ \bar{q} \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} \bar{p} \\ \bar{q} \end{pmatrix}
\]

Results from the theorem:

- Interfaces of master tape need to be valid
- Submatrix A is the essential part of the program
- Submatrix C and D are noise
- A needs to be recorded \(\Rightarrow\)
  - Every statement in the master tape has to occur in the secondary tape
  - Every Jacobian entry in the master tape has to occur in the secondary tape
- C and D need to be ignored \(\Rightarrow\)
  - Every additional statement in the secondary tape can be ignored
  - Every additional Jacobian entry in the secondary tape can be ignored
Validation of tapes - Tape output format

File:

# for each statement that is recorded on the tape
<number of arguments> <Jacobians of all arguments> [<index of lhs> \n <indices of all arguments>] [<code position of the statement>]

Documentation:

# Comments start with a # at the beginning of the line

# number of arguments:
#   The number of arguments on the right hand side of the statement.
# Jacobians of all arguments:
#   Jacobian for d lhs / d rhs_i
#     i = 1 ... <number of arguments>
# index of lhs:
#   The index of the assigned value on the left hand side of the statement.
# indices of all arguments:
#   AD identifier for rhs variables
# code position of the statement:
#   The line and file where the statement is defined
Conclusion

Tagging:
- Used to identify/extend interfaces
- Possible approaches:
  - Iterative methods
  - Separation between two methods
- Makes an online validation of the tape possible

Tape validation:
- Comparison of known/validated tapes to unknown/unvalidated tapes
- Helpful for a change of the derivative algorithm