

DERIVATION OF AN ADJOINT CODE:

Principle:

Any code, no matter how complex it is, boils down to a serie of transformation of values through the application of basic operations.

For instance, the computation of a second derivative can be written into the computation of a first derivative repeated twice, then the computation of a first derivative can be broken into differences and divisions, etc.

We can always write a code as a product of elementary functions: sum, multiplication, division.

1. Forward code:

Let's consider the basic statement (addition & multiplication):

$$d = a * b + c$$

As it stands, this statements can be seen as a function f with input variables a, b, c and output variables d , thus:

$$d = f(a, b, c)$$

2. Tangent linear code:

The tangent linear code is obtained by differentiation of f :

$$\delta f = \frac{\partial f}{\partial a} \delta a + \frac{\partial f}{\partial b} \delta b + \frac{\partial f}{\partial c} \delta c$$

ie:

$$\delta d = b * \delta a + a * \delta b + \delta c$$

This assumes f differentiable.

2. Adjoint code:

To derive the adjoint code, we need to express the tangent linear code in a matrix form. In order to find such expression, we need to consider all the input and output variables. In most cases, it is likely that the variables a , b , c and d will be used in other parts of the code. As a consequence, these variables should be considered as input and output. In other words, the complete expression of the tangent linear code is:

$$\delta a = 1 * \delta a$$

$$\delta b = 1 * \delta b$$

$$\delta c = 1 * \delta c$$

$$\delta d = b * \delta a + a * \delta b + 1 * \delta c$$

or:

$$\underbrace{\begin{bmatrix} \delta a \\ \delta b \\ \delta c \\ \delta d \end{bmatrix}}_{X^A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ b & a & 1 & 0 \end{bmatrix}}_P \times \underbrace{\begin{bmatrix} \delta a \\ \delta b \\ \delta c \\ \delta d \end{bmatrix}}_{X^B}$$

The adjoint code is obtained after transposition of the matrix Q

$$\underbrace{\begin{bmatrix} \delta a^* \\ \delta b^* \\ \delta c^* \\ \delta d^* \end{bmatrix}}_{X^{B^*}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & b \\ 0 & 1 & 0 & a \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{P^T} \times \underbrace{\begin{bmatrix} \delta a^* \\ \delta b^* \\ \delta c^* \\ \delta d^* \end{bmatrix}}_{X^{A^*}}$$

the corresponding adjoint statements are:

$$\begin{aligned} \delta a^* &= \delta a^* + b^* \delta d^* \\ \delta b^* &= \delta b^* + a^* \delta d^* \\ \delta c^* &= \delta c^* + \delta d^* \\ \delta d^* &= 0 \end{aligned}$$

Adjoint statements accumulate information, state (global) adjoint variables need to be reset at the beginning of the adjoint program, local local adjoint variables need to be reset at the beginning of the adjoint routines.

The “trajectory” variables that enter in non-linear forward statements, here a and b , need to be stored and pass to the adjoint code.

Forward code:

Subroutine example (a, b, c, d)

$$d = a * b + c$$

return

end

Tangent linear code:

Subroutine lexample (δa , δb , δc , δd , a, b)

$$\delta d = b * \delta a + a * \delta b + \delta c$$

return

end

Adjoint code:

$$\delta a^* = 0$$

$$\delta b^* = 0$$

$$\delta c^* = 0$$

Subroutine aexample (δa , δb , δc , δd , a, b)

$$\delta a^* = \delta a^* + b * \delta d^*$$

$$\delta b^* = \delta b^* + a * \delta d^*$$

$$\delta c^* = \delta c^* + \delta d^*$$

$$\delta d^* = 0$$

return

end

Remark:

For historical and practical reasons, it is usual to rename the tangent linear and adjoint variables with the name of their corresponding forward variables, ie δa will be rename a ; while the “trajectory” variable are indexed with a “9”, ie in adjoint a is renamed $a9$. With thoses conventions, the tangent linear and adjoint code look like:

Subroutine lexample (a, b, c, d, a9, b9)

$$d = b9*a + a9*b + c$$

return

end

Subroutine aexample (a, b, c, d, a9, b9)

$$a = a + b9*d$$

$$b = b + a9*d$$

$$c = c + d$$

$$d = 0$$

return

end

Adjoint of other FORTRAN statements

Loop:

Because the adjoint model is integrated backward, loops are reversed. For instance, the adjoint of the middle of two points will be:

Forward code:

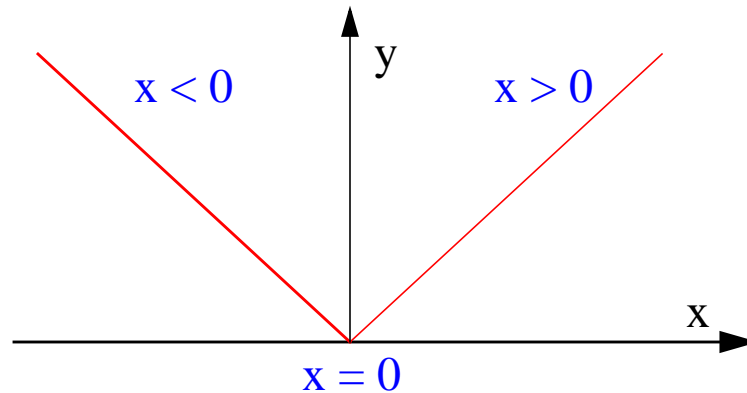
```
Do n = Nmax-1, 1, -1
    m (n) = (x (n) + x (n+1)) / 2.
enddo
```

Adjoint code:

```
Do n = Nmax-1, 1, -1
    x (n+1) = x (n+1) + m (n) / 2.
    x (n) = x (n) + m (n) / 2.
    m (n) = 0.
enddo
```

if:

if statements are the expression of non-differentiable functions. For instance the function $|x|$ can be expressed as:



Forward code:

```

if ( x > 0 ) then
y = x
else
y = -x
endif

```

However, the function is differentiable for $x > 0$ and $x < 0$ and the probability for x to be exactly the test value, like 0 here, during the execution of the program is very small. Thus in most cases, if statements are differentiable. The derivation has to be performed along each possible trajectory. In our example:

adjoint code:

```

if ( x > 0 ) then
 $\delta x^* = \delta x^* + \delta y^*$ 
 $\delta y^* = 0.$ 
else
 $\delta x^* = \delta x^* - \delta y^*$ 
 $\delta y^* = 0.$ 
endif

```

ADJOINT OF AN AGCM

Method:

=> Identify the “active” variables, ie:

- The state vector, usually wind, pressure, temperature, humidity...
- All intermediate variables function of the state vector variables, for instance the vorticity, divergence, etc.

=> Derive each routine with respect to the active variables.

=> Select the amount of storage / re-computation

=> Tests

Storage / Re-computation:

All active variables entering into a non linear statement is needed in the adjoint run. These values must be either stored in memory or in files during the forward run or re-computed in the adjoint run.

In general, for memory saving, only the state vector, which is the minimal information, is saved at each time step. All other intermediate variables are re-computed from the values of the state vector.

Tests:

Tangent linear code:

For each routine, the tangent linear P code is tested against the forward code Q , using the Taylor-Lagrange formula:

$$\frac{Q(z + \alpha h) - Q(z)}{\alpha P h} = 1 + O(\alpha)$$

where z in the input vector, h a perturbation on the input vector and α a scaling coefficient.

Adjoint code:

For each routine, the adjoint code P^T is tested again the tangent linear code P using the adjointness relation:

$$\langle Pz, Pz \rangle = \langle z, P^T Pz \rangle$$

Gradient verification:

The correctness of the Gradient is checked for the whole code according to the relation:

$$J(z + \alpha h) - J(z) = \langle \alpha h, \underbrace{\nabla J}_{\alpha P^T z} \rangle + O(\alpha)$$

Automatic adjoint generators:

- TAMC: Ralf Giering (MIT)

<http://www.mit.edu/~giering/tamc>

- ADIFOR: Argonne Laboratory

<http://www.mcs.anl.gov/autodiff>

- ODYSSEE: INRIA France

References:

Zou X., F. Vandenberghe, M. Pondeva, Y.-H. Kuo, 1997:
Introduction to adjoint techniques and the MM5 adjoint modeling system.

NCAR technical notes. NCAR/TN-435-STR, 110 pp.

Giering, R. and T. Kaminski, 1996:
Recipes for adjoint code construction.

Internal Report 212 from Max-Planck Institute für Meteorologie, Hamburg, Germany. 35pp

Talagrand O., 1991:

The use of adjoint equations in numerical modeling of the atmospheric circulation.

In proceedings of the Workshop on Automatic differentiation Algorithms: Theory, Implementation and Applications. *SIAM*,