Tangent linear and adjoint models for variational data assimilation

Marcin Chrust

with contributions from:

Sebastien Massart, **Angela Benedetti,** Marta Janisková, Philippe Lopez, Lars Isaksen, Gabor Radnoti and Yannick Tremolet

Introduction

- 4D-Var is based on minimization of a cost function which measures a weighted distance between the model with respect to the observations and with respect to the background state
- In order to minimize the cost function, an initial estimate of the solution and the gradient of the cost function are needed.
- The gradient is computed using a direct model integration and a backward time integration of a suitably forced version of the linearized adjoint model (Le Dimet and Talagrand, 1986)
- Overview:
  - Brief overview of 4D-Var with focus on TL/AD aspects
  - General definitions of Tangent Linear and Adjoint models and why they are extremely useful in variational assimilation
  - Writing TL and AD models and testing them
  - Brief mention of automatic differentiation software

4D-Var

In strong constraint 4D-Var the cost function can be expressed as follows:

$$J = \frac{1}{2}(x_0 - x_b)^T B^{-1}(x_0 - x_b) + \frac{1}{2}\sum_{i=0}^{n}(H_i M_i[x_0] - y_i)^T R_i^{-1}(H_i M_i[x_0] - y_i)$$

B covariance matrix of errors in the background state,
R_i covariance matrices of errors in $(H_i M_i[x_0] - y_i)$ (instrumental + interpolation + observation operator error),
y_i observations
x_o initial model state
x_b background state
M forward nonlinear forecast model (time evolution of the model state, index i),
H observation operator (model space → observation space).

$$\min J \Leftrightarrow \nabla_{x_0} J = B^{-1}(x_0 - x_b) + \sum_{i=0}^{n} M'^T[t_i, t_0] H_i'^T R_i^{-1}(H_i M_i[x_0] - y_i)$$

$H_i'^T$ - adjoint of a linearised observation operator and M'^T - adjoint of a linearized forecast model.

Linearization of a function

Principle: the linearization of a function is the first order term of its Taylor expansion around the linearization state. Consider a system:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{F}(\boldsymbol{x}, t)$$

The linearized system can be written as

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{F}(\boldsymbol{x_0}, t) + D\boldsymbol{F}(\boldsymbol{x_0}, t) \cdot (\mathrm{x} - \boldsymbol{x_0})$$

Where $\boldsymbol{x_0}$ is the linearization state and $D\boldsymbol{F}(\boldsymbol{x_0}, t)$ is the Jacobian of $\boldsymbol{F}(\boldsymbol{x})$ evaluated at $\boldsymbol{x_0}$.

$$D\boldsymbol{F}(\boldsymbol{x}, t) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Incremental 4D-Var at ECMWF

• In incremental 4D-Var, the cost function is minimized in terms of increments:

$$x_0 = x_b + \delta x_0;$$
$$M(x_0) = M(x_b + \delta x_0) = M(x_b) + M'(\delta x_0);$$
$$H(x_0) = H(x_b + \delta x_0) = H(x_b) + H'(\delta x_0);$$

• The 4D-Var cost function can then be approximated to the first order by:

$$J(\delta x_0) = \frac{1}{2}\delta x_0{}^T B^{-1}\delta x_0 + \frac{1}{2}\sum_{i=0}^{n}(H_i' M'[t_0, t_i]\delta x_0 - d_i)^T R_i^{-1}(H_i' M'[t_0, t_i]\delta x_0 - d_i)$$

where $d_i = y_i - H_i(x_{ti})$ is the so-called departure computed using the nonlinear model and observation operator.

•The gradient of the cost function to be minimized is:

$$\min J \Leftrightarrow \nabla_{\delta x_0} J = B^{-1}\delta x_0 + \sum_{i=0}^{n} M'^T[t_i, t_0] H_i'^T R_i^{-1}(H_i' M'[t_0, t_i]\delta x_0 - d_i)$$

M' and H_i' are the tangent linear models which are used in the computations of incremental updates during the minimization (iterative procedure).

Details of the linearisation (cnt.)

The gradient of the cost function with respect to δx_0 is given by:

$$\nabla_{\delta x_0} J = B^{-1}\delta x_0 + \sum_{i=0}^{n} (H_i' M_i' M_{i-1}' \ldots M_0')^T R^{-1} (H_i' M_i' M_{i-1}' \ldots M_0' \delta x_0 - d_i) =$$

remembering that $(AB)^T = B^T A^T$

$$= B^{-1}\delta x_0 + \sum_{i=0}^{n} M_0'^T \ldots M_{i-1}'^T M_i'^T H_i'^T R^{-1} (H_i' M_i' M_{i-1}' \ldots M_0' \delta x_0 - d_i) =$$

$$= B^{-1}\delta x_0 + \sum_{i=0}^{n} M'^T [t_i, t_0] H_i'^T R^{-1} (H_i' \delta x_i - d_i)$$

The optimal initial perturbation is obtained by finding the value
Of δx_0 for which:

$$\nabla_{\delta x_0} J = 0$$

The gradient of the cost function with respect to the initial condition is provided by the **adjoint solution** at time t=0. Let's see how…

Definition of adjoint operator

For any linear operator M' there exist an *adjoint* operator M^* such as:

$$\langle x, My \rangle = \langle M^* x, y \rangle$$

Where $\langle , \rangle$ is an inner scalar product and x, y are vectors (or functions) of the space where this product is defined.

It can be shown that for the inner product defined in the Euclidean space :

$$M^* = M^T$$

We will now show that the gradient of the cost function at time t=0 is provided by the solution of the adjoint equations at the same time:

$$\nabla_{\delta x_0} J = -x_0^*$$

Adjoint solution

Usually the initial guess x_0 is chosen to be equal to the background x_b so that the initial perturbation $\delta x_0 = 0$.

The gradient of the cost function is hence simplified as:

$$\nabla_{\delta x_0} J = -\sum_{i=0}^{n} M^{'T}[t_i, t_0] H_i^{'T} R^{-1} d_i$$

We choose the solution of the adjoint system as follows:

$$x_{n+1}^* = 0 \qquad \text{(“final” condition)}$$

$$x_i^* = M_{i+1}^{'T} x_{i+1}^* + M_i^{'T} H_i^{'T} R^{-1} d_i, i = 0, ..., n \qquad \text{(iterative solution)}$$

We then substitute progressively the solution x_i^* into the expression for x_0^*

Adjoint solution (continuation)

$$x_0^* = M_1^{'T}(M_2^{'T}x_2^*) + M_1^{'T}H_1^{'T}R^{-1}d_1 + M_0^{'T}H_0^{'T}R^{-1}d_0 =$$
$$= M_1^{'T}(M_2^{'T}(M_3^{'T}x_3^*)) + M_1^{'T}M_2^{'T}H_2^{'T}R^{-1}d_2 + M_1^{'T}H_1^{'T}R^{-1}d_1 + M_0^{'T}H_0^{'T}R^{-1}d_0 =$$
$$= \ldots$$

Finally, regrouping and remembering that $x_{n+1}^* = 0$ and that $M^* = M^T$ and $H^* = H^T$ we obtain the following equality:

$$x_0^* = \sum_{i=0}^{n} M^{'T}[t_i, t_0]H_i^{'T}R^{-1}d_i \quad \Rightarrow x_0^* = -\nabla_{\delta x_0} J$$

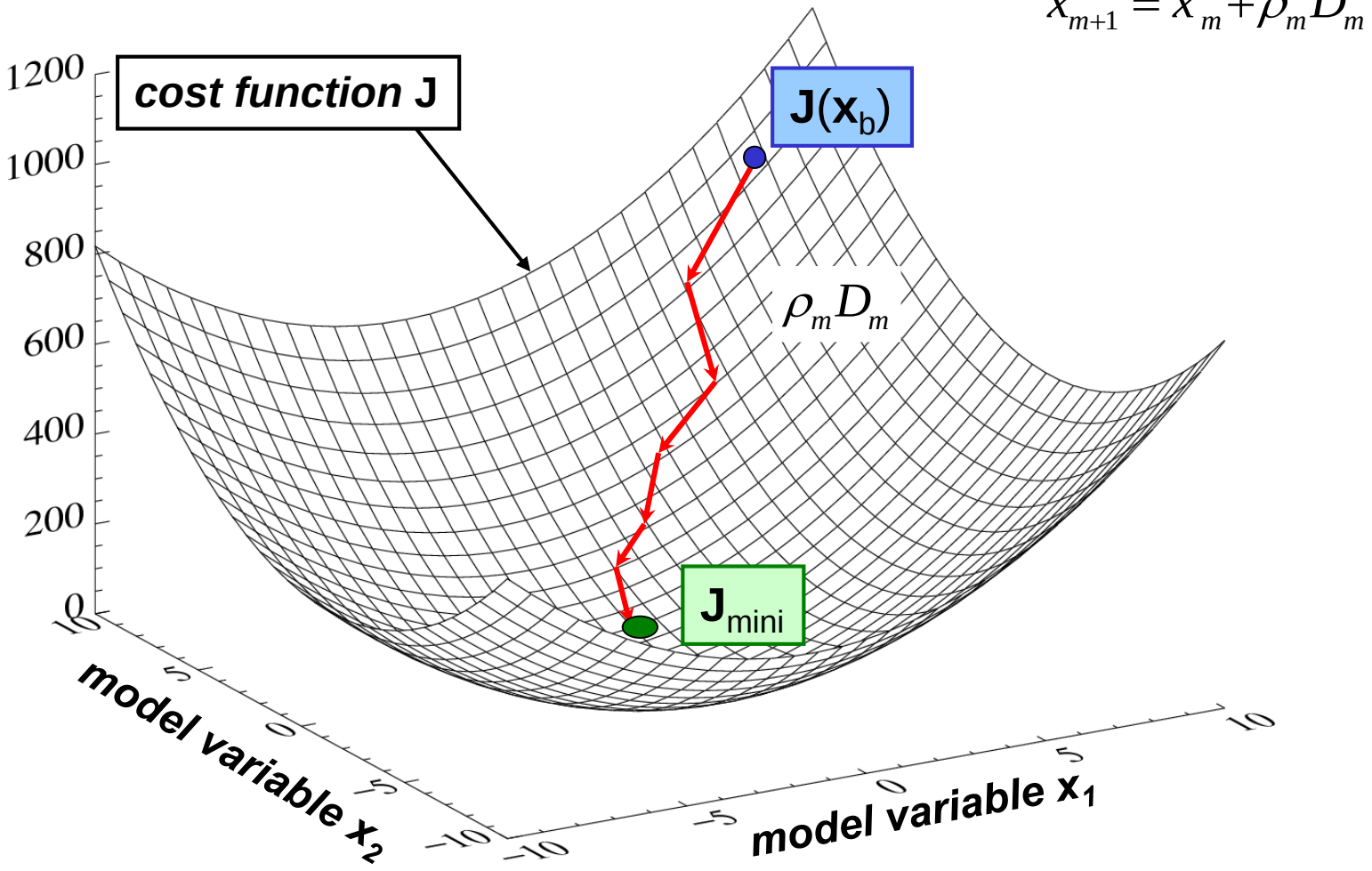
The **gradient** of **the cost function** with respect to the control variable (initial condition) is obtained by a **backward** integration of the suitably forced **adjoint model**.

Iterative steps in the 4D-Var Algorithm

1. Integrate forward model gives J.
2. Integrate forced adjoint model backwards gives ∇J .
3. If $\|\nabla J(x_m)\| \leq \varepsilon$ then stop.
4. Compute descent direction D_m (recall Sebastien's class).
5. Compute step size ρ_m: $J(x_m + \rho_m D_m) = \min_{\rho} J(x_m + \rho D_m)$.
6. Update initial condition: $x_{m+1} = x_m + \rho_m D_m$.

Finding the minimum of cost function J is an iterative minimization procedure

$$x_{m+1} = x_m + \rho_m D_m$$



Recap on TL and AD models

- **TANGENT LINEAR MODEL**

  If M is a model such as:

  $$\mathbf{x}(t_{i+1}) = M[\mathbf{x}(t_i)]$$

  then the tangent linear model of M, called M', is:

  $$\delta\mathbf{x}(t_{i+1}) = M'[\mathbf{x}(t_i)]\delta\mathbf{x}(t_i) = \frac{\partial M[\mathbf{x}(t_i)]}{\partial \mathbf{x}}\delta\mathbf{x}(t_i)$$

- **ADJOINT MODEL**

  The adjoint of a linear operator M' is the linear operator $\mathbf{M}^*$ such that, for the inner product $<,>$,

  $$\forall \mathbf{x}, \forall \mathbf{y} \qquad < M'\mathbf{x}, \mathbf{y} >=< \mathbf{x}, \mathbf{M}^*\mathbf{y} >$$

  Remarks:

  - with the euclidian inner product, $\mathbf{M}^* = {M'}^T$.
  - in variational assimilation, $\nabla_x \mathcal{J} = \mathbf{M}^* \nabla_y \mathcal{J}$, where $\mathcal{J}$ is the cost function.

Simple example of adjoint writing

- **non-linear statement**

$$
\begin{aligned}
x &= y + z^2 \\
\\
z &= z \\
y &= y \\
x &= y + z^2
\end{aligned}
$$

- **tangent linear statement**

$$
\begin{aligned}
\delta z &= \delta z \\
\delta y &= \delta y \\
\delta x &= \delta y + 2z\delta z
\end{aligned}
$$

or in a matrix form:

$$
\begin{pmatrix} \delta z \\ \delta y \\ \delta x \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2z & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \delta z \\ \delta y \\ \delta x \end{pmatrix}
$$

Simple example of adjoint writing (cnt.)

Often the adjoint variables in mathematical formulations are indicated with an asterisk

- **adjoint statement**

  - transpose matrix

$$\begin{pmatrix} \delta z^* \\ \delta y^* \\ \delta x^* \end{pmatrix} = \begin{pmatrix} 1 & 0 & 2z \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \delta z^* \\ \delta y^* \\ \delta x^* \end{pmatrix}$$

or in the form of equation set:

$$\begin{aligned} \delta z^* &= \delta z^* + 2z\delta x^* \\ \delta y^* &= \delta y^* + \delta x^* \\ \delta x^* &= 0 \end{aligned}$$

Do not forget the last equation!!! That too is part of the adjoint!

As an alternative to the matrix method, adjoint coding can be carried out using a **line-by-line** approach.

More practical examples on adjont coding: the Lorenz model

$$\frac{dx_1}{dt} = -px_1 + px_2 \quad (1)$$

$$\frac{dx_2}{dt} = (r - x_3)x_1 - x_2 \quad (2)$$

$$\frac{dx_3}{dt} = x_1 x_2 - bx_3 \quad (3)$$

where p is the Prandtl number, r the Rayleigh number, and b the aspect ratio.

x_1 is the intensity of convection,

x_2 is the maximum temperature difference

x_3 is the stratification change due to convection.

Details on the Lorenz model can be found in the references.

The linear code in Fortran

Linearize each line of the code one by one, and set dx/dt=y for simplicity
(and do not focus on the fact that y is a time derivative, just consider it a dependent variable):

```
   y(1)    = -p*x(1)    +p*x(2)        :Nonlinear statement
(1)yd(1)   = -p*xd(1)   +p*xd(2)       :Tangent linear

   y(2)    =  x(1)*(r-x(3)) -x(2)     :Nonlinear statement
(2)yd(2)   = xd(1)*(r-x(3))
               -x(1)*xd(3)   -xd(2)    :Tangent linear

   …etc
```

Remember that `p, r, b` are constants;
`x(1), x(2)` and `x(3)` are the independent variables;
`y(1), y(2)` and `y(3)` are the dependent variables.

We chose the suffix "d" for the tangent linear variable for consistency with the automatic differentation software TAPENADE. Adjoint variables are indicated with the suffix "b". This is just a convention.

Note that in the ECMWF Integrated Forecast System (IFS) the tangent linear and adjoint variables are indicated without any subscripts and the nonlinear trajectory (x) is indicated with the suffix "5" (x5).

Adjoint of one instruction

We start from the tangent linear code:

```
yd(1)=-p*xd(1)+p*xd(2)
```

In matrix form, it can be written as:

$$\begin{pmatrix} \delta x_1 \\ \delta x_2 \\ \delta y_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -p & p & 0 \end{pmatrix} \begin{pmatrix} \delta x_1 \\ \delta x_2 \\ \delta y_1 \end{pmatrix}$$

which can easily be transposed (asterisk indicates adjoint variables):

$$\begin{pmatrix} \delta x_1^* \\ \delta x_2^* \\ \delta y_1^* \end{pmatrix} = \begin{pmatrix} 1 & 0 & -p \\ 0 & 1 & p \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta x_1^* \\ \delta x_2^* \\ \delta y_1^* \end{pmatrix}$$

The corresponding adjoint code in FORTRAN is:

```
xb(1)=xb(1)-p*yb(1)
xb(2)=xb(2)+p*yb(1)
yb(1)=0
```

Adjoint of one instruction (II)

We start again from the tangent linear code:

```
yd(2)= xd(1)*(r-x(3))-xd(2)-x(1)*xd(3)
```

In matrix form, it can be written as:

$$\begin{pmatrix} \delta x_1 \\ \delta x_2 \\ \delta x_3 \\ \delta y_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ (r - x_3) & -1 & -x_1 & 0 \end{pmatrix} \begin{pmatrix} \delta x_1 \\ \delta x_2 \\ \delta x_3 \\ \delta y_2 \end{pmatrix}$$

These terms come from the trajectory! Needs to be stored in memory or recomputed

which can easily be transposed (asterisk indicates transposition):

$$\begin{pmatrix} \delta x_1^* \\ \delta x_2^* \\ \delta x_3^* \\ \delta y_2^* \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & (r - x_3) \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -x_1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta x_1^* \\ \delta x_2^* \\ \delta x_3^* \\ \delta y_2^* \end{pmatrix}$$

The corresponding adjoint code in FORTRAN is:

```
xb(1)=xb(1)+(r-x(3))*yb(2)
xb(2)=xb(2)- yb(2)
xb(3)=xb(3)- x(1)*yb(2)
yb(2)=0
```

Trajectory

The trajectory has to be available. It can be:

- saved which costs memory,
- recomputed which costs CPU time.

Depending on the complexity of the code, one option or the other is adopted (or both options at the same time).

- The space cost of storing trajectory is linear in time (double the timesteps, double the storage), the time cost is constant (no extra re-computation needed),
- Storing nothing and recomputing everything when it becomes necessary, is quadratic in time and constant in space.

The Adjoint Code

Property of adjoints (transposition):

$$(L_n L_{n-1} ... L_2 L_1)^* = L_1^* L_2^* ... L_{n-1}^* L_n^*$$

Application: $M_i = L_n L_{n-1} ... L_2 L_1$ where L_i represents the line i of the tangent linear model.

The **adjoint code** is made of the **transpose** of each line of the tangent linear code in **reverse order**.

Adjoint of loops

In the TL code for the Lorenz model we have:

```
DO i=1,3
  xd(i)=xd(i)+dt*yd(i)
ENDDO
```

`dt` is a constant for our purposes. This loop can be written explicitly:

```
xd(1)=xd(1)+dt*yd(1)
xd(2)=xd(2)+dt*yd(2)
xd(3)=xd(3)+dt*yd(3)
```

We can now transpose and reverse the lines to get the adjoint:

```
yb(3)=yb(3)+dt*xb(3)
yb(2)=yb(2)+dt*xb(2)
yb(1)=yb(1)+dt*xb(1)
```

which is equivalent to

```
DO i=3,1,-1     !Reverse order of indeces!
  yb(i)=yb(i)+dt*xb(i)
ENDDO
```

Conditional statements (“IF” statements)

- What we want is the adjoint of the statements which were actually executed in the direct model.
- We need to know which “branch” of the IF statement was executed
- The result of the conditional statement has to be stored: **it is part of the trajectory** !!!

Summary of basic rules for line-by-line adjoint coding (1)

Adjoint statements are derived from tangent linear ones in a reversed order

Tangent linear code	Adjoint code
$\delta x = 0$	$\delta x^* = 0$
$\delta x = A\,\delta y + B\,\delta z$	$\delta y^* = \delta y^* + A\,\delta x^*$ $\delta z^* = \delta z^* + B\,\delta x^*$ $\delta x^* = 0$
$\delta x = A\,\delta x + B\,\delta z$	$\delta z^* = \delta z^* + B\,\delta x^*$ $\delta x^* = A\,\delta x^*$
do k = 1, N $\delta x(k) = A\,\delta x(k-1) + B\,\delta y(k)$ end do	do k = N, 1, – 1 (Reverse the loop!) $\delta x^*(k-1) = \delta x^*(k-1) + A\,\delta x^*(k)$ $\delta y^*(k) = \delta y^*(k) + B\,\delta x^*(k)$ $\delta x^*(k) = 0$ end do
if (*condition*) *tangent linear code*	if (*condition*) *adjoint code*

Order of operations is important when variable is updated!

And do not forget to initialize local adjoint variables to zero !

Summary of basic rules for line-by-line adjoint coding (2)

To save memory, the trajectory can be recomputed just before the adjoint calculations (again it depends on the complexity of the model).

Tangent linear code	Trajectory and adjoint code

if ($x > x0$) then
$\delta x = A\, \delta x / x$
$x = A\, \mathrm{Log}(x)$
end if
	---------- Trajectory ----------
	$x_{store} = x$ (storage for use in adjoint)
	if ($x > x0$) then
	$x = A\, \mathrm{Log}(x)$
	end if
	---------- Adjoint ----------
	if ($x_{store} > x0$) then
	$\delta x^* = A\, \delta x^* / x_{store}$
	end if

The most common sources of error in adjoint coding are:

1) Pure coding errors
2) Forgotten initialization of local adjoint variables to zero
3) Mismatching trajectories in tangent linear and adjoint (even slightly)
4) Bad identification of trajectory updates

Adjoint revisited

Let $\mathcal{F}$ be a Hilbert space, with inner product denoted by $(\,,)$, and $\boldsymbol{v} \to \mathfrak{J}(\boldsymbol{v})$ a differentiable scalar function defined on $\mathcal{F}$. At any point in $\mathcal{F}$, the differential $\delta\mathfrak{J}$ of $\mathfrak{J}$ can be expressed as:

$$\delta\mathfrak{J} = (\nabla_{\boldsymbol{v}}\mathfrak{J}, \boldsymbol{\delta v})$$

Let $\mathcal{E}$ be another Hilbert space, with inner product denoted by $\langle\,,\rangle$, and $\boldsymbol{L}$ a continuous linear operator from $\mathcal{E}$ into $\mathcal{F}$. There exists a unique continuous linear operator $\boldsymbol{L}^*$ from $\mathcal{F}$ into $\mathcal{E}$ such that the following equality between inner products holds for any $\boldsymbol{u}$ belonging to $\mathcal{E}$ and any $\boldsymbol{v}$ belonging to $\mathcal{F}$**.**

$$(\boldsymbol{v}, \boldsymbol{Lu}) = \langle \boldsymbol{L}^*, \boldsymbol{u}\rangle$$

Consider a (not necessarily linear) differentiable function $\boldsymbol{u} \to \boldsymbol{v} = \boldsymbol{M}(\boldsymbol{u})$ of $\mathcal{E}$ into $\mathcal{F}$**.** $\mathfrak{J}$**(v)=** $\mathfrak{J}$[**M**(**u**)] is a compound function of **u**. The differential of **v** is equal to

$$\boldsymbol{\delta v} = \boldsymbol{M'\delta u}$$

The differential of $\mathfrak{J}$ is:

$$\delta\mathfrak{J} = (\nabla_{\boldsymbol{v}}\mathfrak{J}, \boldsymbol{\delta v}) = (\nabla_{\boldsymbol{v}}\mathfrak{J}, \boldsymbol{M'\delta u}) = \langle \boldsymbol{M'^*}\nabla_{\boldsymbol{v}}\mathfrak{J}, \boldsymbol{\delta u}\rangle = \langle \nabla_{\boldsymbol{u}}\mathfrak{J}, \boldsymbol{\delta u}\rangle$$

Adjoint revisited

Consider now that $\boldsymbol{u}$ denote the initial conditions from which a numerical model of atmospheric flow is integrated; $\boldsymbol{u} \rightarrow \boldsymbol{v} = \boldsymbol{M}(\boldsymbol{u})$ will be the integration of the model, $\boldsymbol{v}$ denotes the time sequence of successive model states produced by the integration. $\boldsymbol{y}$ denotes observations distributed in time. For simplicity, we would like to find such initial condition $\boldsymbol{u}$ that minimizes the misfit between the model and observations (assume for simplicity **H'** is a linear mapping from the model space to observation space and **R** is a covariance matrix of observation errors) measured by $\mathfrak{J}(\boldsymbol{v})$:

$$\mathfrak{J}(\boldsymbol{v}) = (\mathbf{H}'(\boldsymbol{v}) - \mathbf{y})^T \boldsymbol{R}^{-1}(\boldsymbol{H}'(\boldsymbol{v}) - \boldsymbol{y})$$

We need an expression for $\nabla_{\boldsymbol{u}}\mathfrak{J}$:

$$\nabla_{\boldsymbol{u}}\mathfrak{J} = \boldsymbol{M}'^T \nabla_v \mathfrak{J}$$

We find the familiar expression:

$$\nabla_{\boldsymbol{u}}\mathfrak{J} = \boldsymbol{M}'^T \boldsymbol{H}'^T \boldsymbol{R}^{-1}(\boldsymbol{H}'(\boldsymbol{v}) - \boldsymbol{y})$$

More facts about adjoints

- The adjoint always exists and it is unique, assuming spaces of finite dimension. Hence, coding the adjoint does not raise questions about its existence, only questions related to the technical implementation.

- In the meteorological literature, the term *adjoint* is often improperly used to denote the adjoint of the tangent linear of a non-linear operator. In reality, the adjoint can be defined for any linear operator. One must be aware that discussions about the *existence of the adjoint* usually should address the existence of the tangent linear model.

- Without re-computation, the cost of the TL is usually about 1.5 times that of the non-linear code, the cost of the adjoint between 2 and 3 times.

- The tangent linear model is not strictly necessary to run a 4D-Var system (but it is needed in the incremental 4D-Var formulation in use operationally at ECMWF). It is also needed as an intermediate step to write and test the adjoint.

Test for tangent linear model

- **Taylor formula:**

$$\lim_{\lambda \to 0} \frac{M(\mathbf{x} + \lambda \delta \mathbf{x}) - M(\mathbf{x})}{M'(\lambda \delta \mathbf{x})} = 1$$

Perturbation scaling factor

λ	RATIO
0.1E-09	0.9994875881543574E+00
0.1E-08	0.9999477148855701E+00
0.1E-07	0.9999949234236705E+00
0.1E-06	0.9999993501022509E+00
0.1E-05	0.9999999496119013E+00
0.1E-04	0.9999996111338369E+00
0.1E-03	0.9999953179193711E+00
0.1E-02	0.9999724488345042E+00
0.1E-01	0.9993727842790062E+00
0.1E+00	0.9978007454264978E+00
0.1E+01	0.9583066504549524E+00

machine precision reached

Test for adjoint model

- **adjoint identity:**

$$\forall \mathbf{x}, \forall \mathbf{y} \quad < M'.\mathbf{x}, \mathbf{y} >=< \mathbf{x}, \mathbf{M}^*.\mathbf{y} >$$

$< \mathrm{F(X)} , \mathrm{Y} > =$ -.137651026252516400000E-01
$< \mathrm{X} , \mathrm{F^*(Y)} > =$ -.137651026252516800000E-01

ratio of norms = 1.00000000000000005

THE DIFFERENCE IS 11.351 TIMES THE ZERO OF THE MACHINE

The adjoint test is truly unforgiving. If you do not have a ratio of the norm close to 1 within the precision of the machine, you know there is a bug in your adjoint. At the end of your debugging you will have a perfect adjoint. If properly tested, the adjoint is the only piece of code on Earth to be entirely bug-free (although you may still have an imperfect tangent linear)!

Test of adjoint in practice…

- Consider the following linear equation:

$$\delta y = A\delta x_1 + B\delta x_2$$

- and its adjoint:

$$\begin{bmatrix}\delta x_1\\ \delta x_2\\ \delta y\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ A & B & 0\end{bmatrix}\begin{bmatrix}\delta x_1\\ \delta x_2\\ \delta y\end{bmatrix} = \widetilde{\boldsymbol{F}}\begin{bmatrix}\delta x_1\\ \delta x_2\\ \delta y\end{bmatrix} \longrightarrow \begin{bmatrix}\delta x_1^*\\ \delta x_2^*\\ \delta y^*\end{bmatrix} = \widetilde{\boldsymbol{F}}^T\begin{bmatrix}\delta x_1^*\\ \delta x_2^*\\ \delta y^*\end{bmatrix} \rightarrow \begin{cases}\delta x_1^* = \delta x_1^* + A\delta y\\ \delta x_2^* = \delta x_2^* + B\delta y\\ \delta y^* = 0\end{cases}$$

- Construct an adjoint test:

$$\left\langle \begin{bmatrix}\delta x_1\\ \delta x_2\\ \delta y\end{bmatrix}, \widetilde{\boldsymbol{F}}\begin{bmatrix}\delta x_1\\ \delta x_2\\ \delta y\end{bmatrix} \right\rangle \leftrightarrow \left\langle \widetilde{\boldsymbol{F}}^T\begin{bmatrix}\delta x_1\\ \delta x_2\\ \delta y\end{bmatrix}, \begin{bmatrix}\delta x_1\\ \delta x_2\\ \delta y\end{bmatrix} \right\rangle$$

$$\delta x_1\delta x_1 + \delta x_2\delta x_2 + \delta y(A\delta x_1 + B\delta x_2) \leftrightarrow \delta x_1\delta x_1 + \delta x_2\delta x_2 + A\delta y\delta x_1 + B\delta y\delta x_2$$

$$\delta y\delta y \leftrightarrow \delta x_1^*\,\delta x_1 + \delta x_2^*\,\delta x_2$$

Test of adjoint in practice…

- Alternatively we can derive the result above as follows:

$$\delta y = A\delta x_1 + B\delta x_2 = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \delta x_2 \end{bmatrix} = \boldsymbol{F\delta x}$$

- Construct an adjoint test as usual:

$$\langle \delta y, \boldsymbol{F\delta x} \rangle \leftrightarrow < \boldsymbol{F}^T \delta y, \boldsymbol{\delta x} >$$

$$\delta y(A\delta x_1 + B\delta x_2) \leftrightarrow A\delta y\delta x_1 + B\delta y\delta x_2$$

$$\delta y\delta y \leftrightarrow \delta x_1^* \, \delta x_1 + \delta x_2^* \, \delta x_2$$

Automatic differentiation

- Because of the strict rules of tangent linear and adjoint coding, automatic differentiation is possible.

- Existing tools: TAF (TAMC), TAPENADE (Odyssée), ...
  - Reverse the order of instructions,
  - Transpose instructions instantly without typos !!!
  - Especially good in deriving tangent linear codes!

- There are still unresolved issues:
  - It is **NOT** a black box tool,
  - Cannot handle non-differentiable instructions (TL is wrong),
  - Can create huge arrays to store the trajectory,
  - The codes often need to be cleaned-up and optimised.

- Look in the "Supplementary material" for more information!

Useful References

- **Variational data assimilation:**

  Lorenc, A., 1986, *Quarterly Journal of the Royal Meteorological Society*, **112**, 1177-1194.
  Courtier, P. *et al.*, 1994, *Quarterly Journal of the Royal Meteorological Society*, **120**, 1367-1387.
  Rabier, F. *et al.*, 2000, *Quarterly Journal of the Royal Meteorological Society*, **126**, 1143-1170.

- **The adjoint technique:**

  Errico, R.M., 1997, *Bulletin of the American Meteorological Society*, **78**, 2577-2591.

- **Tangent-linear approximation:**

  Errico, R.M. *et al.*, 1993, *Tellus*, **45A**, 462-477.
  Errico, R.M., and K. Reader, 1999, *Quarterly Journal of the Royal Meteorological Society*, **125**, 169-195.
  Janisková, M. *et al.*, 1999, *Monthly Weather Review*, **127**, 26-45.
  Mahfouf, J.-F., 1999, Tellus, **51A**, 147-166.

- **Lorenz model:**

  X. Y. Huang and X. Yang. Variational data assimilation with the Lorenz model. Technical Report 26, HIRLAM, April 1996. Available on ftp site (see notes for practical session).
  E. Lorenz. Deterministic nonperiodic flow. J. Atmos. Sci., 20:130-141, 1963.

- **Automatic differentiation:**

  Giering R., Tangent Linear and Adjoint Model Compiler, Users Manual Center for Global Change Sciences, Department of Earth, Atmospheric, and PlanetaryScience,MIT,1997
  Giering R. and T. Kaminski, Recipes for Adjoint Code Construction, *ACM Transactions on Mathematical Software*, 1998
  TAMC: http://www.autodiff.org/
  TAPENADE: http://www-sop.inria.fr/tropics/tapenade.html

- **Sensitivity studies using the adjoint technique**

  Janiskova, M. and J.-J. Morcrette., 2005. Investigation of the sensitivity of the ECMWF radiation scheme to input parameters using adjoint technique. *Quart. J. Roy. Meteor. Soc.*, **131**,1975-1996.