Derivatives

Minimisation of a cost function $\rightarrow \nabla_x J_0$

$$J_o(x) = \sum_{i=1}^{N} \| H(\underbrace{M(M(\ldots M(x)))}_{k(i)\,times}) - y_i \|_R^{\,2}$$

- R is the observation error correlation matrix ($\|X\|_R^2 = XR^{-1}X$),
- M is the `croco step`,
- x the control vector,
- H an observation operator,
- $y_1, y_2, \ldots y_N$ observations.

Derivatives

Minimisation of a cost function $\rightarrow \nabla_x J_0$

$$J_o(x) = \sum_{i=1}^{N} \| H(\underbrace{M(M(\ldots M(x)))}_{k(i)\,times}) - y_i \|_R^{\,2}$$

- R is the observation error correlation matrix ($\|X\|_R^2 = XR^{-1}X$),
- M is the `croco step`,
- x the control vector,
- H an observation operator,
- $y_1, y_2, \ldots y_N$ observations.

Derivatives

Minimisation of a cost function $\rightarrow \nabla_x J_0$

$$J_o(x) = \sum_{i=1}^{N} \| H(\underbrace{M(M(\ldots M(x)))}_{k(i)\,times}) - y_i \|_R^{\,2}$$

- R is the observation error correlation matrix ($\|X\|_R^2 = XR^{-1}X$),
- M is the `croco step`,
- x the control vector,
- H an observation operator,
- $y_1, y_2, \ldots y_N$ observations.

Derivatives

Minimisation of a cost function $\rightarrow \nabla_x J_0$

$$J_o(x) = \sum_{i=1}^{N} \| H(\underbrace{M(M(\ldots M(x)))}_{k(i)\,times}) - y_i \|_R^{\,2}$$

- R is the observation error correlation matrix ($\|X\|_R^2 = XR^{-1}X$),
- M is the `croco step`,
- x the control vector,
- H an observation operator,
- $y_1, y_2, \ldots y_N$ observations.

Derivatives

Minimisation of a cost function $\rightarrow \nabla_x J_0$

$$J_o(x) = \sum_{i=1}^{N} \| H(\underbrace{M(M(\ldots M(x)))}_{k(i)\,times}) - y_i \|_R^{\,2}$$

- R is the observation error correlation matrix ($\|X\|_R^2 = XR^{-1}X$),
- M is the `croco step`,
- x the control vector,
- H an observation operator,
- $y_1, y_2, \ldots y_N$ observations.

Derivatives

Minimisation of a cost function $\rightarrow \nabla_x J_0$

$$J_o(x) = \sum_{i=1}^{N} \| H(\underbrace{M(M(\ldots M(x)))}_{k(i)\,times}) - y_i \|_R^{\,2}$$

- R is the observation error correlation matrix ($\|X\|_R^2 = XR^{-1}X$),
- M is the `croco step`,
- x the control vector,
- H an observation operator,
- $y_1, y_2, \ldots y_N$ observations.

Automatic differentiation, cf RT Tapenade

A program P of a function F is a sequence of instructions:

$$P: \{I_1; I_2; \ldots; I_p\}$$

Each I_k implements an elementary vector function f_k

$$F = f_p \circ f_{p-1} \cdots \circ f_1$$

So the derivation of F may be computed with the chain rule

$$F'(X) = f'_p(f_{p-1}(f_{p-2}(\ldots))) \times f'_{p-1}(f_{p-2}(\ldots)) \ldots f'_2(f_1(X)) \times f'_1(X)$$

Automatic differentiation, cf RT Tapenade

A program P of a function F is a sequence of instructions:

$$P\colon \{I_1; I_2; \ldots; I_p\}$$

Each I_k implements an elementary vector function f_k

$$F = f_p \circ f_{p-1} \cdots \circ f_1$$

So the derivation of F may be computed with the chain rule

$$F'(X) = f'_p(f_{p-1}(f_{p-2}(\ldots))) \times f'_{p-1}(f_{p-2}(\ldots)) \ldots f'_2(f_1(X)) \times f'_1(X)$$

Automatic differentiation, cf RT Tapenade

A program P of a function F is a sequence of instructions:

$$P\colon \{I_1; I_2; \ldots; I_p\}$$

Each I_k implements an elementary vector function f_k

$$F = f_p \circ f_{p-1} \cdots \circ f_1$$

So the derivation of F may be computed with the chain rule

$$F'(X) = f'_p(f_{p-1}(f_{p-2}(\ldots))) \times f'_{p-1}(f_{p-2}(\ldots)) \ldots f'_2(f_1(X)) \times f'_1(X)$$

Automatic differentiation

$F'(X)$ is the jacobian matrix

$$F'(X) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}|X & \frac{\partial F_1}{\partial x_2}|X & \cdots & \frac{\partial F_1}{\partial x_n}|X \\ \frac{\partial F_2}{\partial x_1}|X & \frac{\partial F_2}{\partial x_2}|X & \cdots & \frac{\partial F_2}{\partial x_n}|X \\ & & \cdots & \\ \frac{\partial F_n}{\partial x_1}|X & \frac{\partial F_n}{\partial x_2}|X & \cdots & \frac{\partial F_n}{\partial x_n}|X \end{pmatrix}$$

With $X_0 = X$ and for $k > 1$, $X_k = f_k(X_{k-1})$

- tangent linear model, forward mode, column access

$$TLM(F, X) : v \to F'(X) \times v = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \cdots \times f'_1(X_0) \times v$$

- adjoint model, reverse mode, line access

$$ADJ(F', X) : u \to F'(X)^t \times u = f'_1(X_0)^t \times f'_2(X_1)^t \cdots \times f'_p(X_{p-1})^t \times u$$

Automatic differentiation

$F'(X)$ is the jacobian matrix

$$F'(X) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}|X & \frac{\partial F_1}{\partial x_2}|X & \cdots & \frac{\partial F_1}{\partial x_n}|X \\ \frac{\partial F_2}{\partial x_1}|X & \frac{\partial F_2}{\partial x_2}|X & \cdots & \frac{\partial F_2}{\partial x_n}|X \\ & & \cdots & \\ \frac{\partial F_n}{\partial x_1}|X & \frac{\partial F_n}{\partial x_2}|X & \cdots & \frac{\partial F_n}{\partial x_n}|X \end{pmatrix}$$

With $X_0 = X$ and for $k > 1$, $X_k = f_k(X_{k-1})$

- tangent linear model, forward mode, column access

$$TLM(F, X) : v \to F'(X) \times v = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \cdots \times f'_1(X_0) \times v$$

- adjoint model, reverse mode, line access

$$ADJ(F', X) : u \to F'(X)^t \times u = f'_1(X_0)^t \times f'_2(X_1)^t \cdots \times f'_p(X_{p-1})^t \times u$$

Automatic differentiation

$F'(X)$ is the jacobian matrix

$$F'(X) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}|X & \frac{\partial F_1}{\partial x_2}|X & \cdots & \frac{\partial F_1}{\partial x_n}|X \\ \frac{\partial F_2}{\partial x_1}|X & \frac{\partial F_2}{\partial x_2}|X & \cdots & \frac{\partial F_2}{\partial x_n}|X \\ & & \cdots & \\ \frac{\partial F_n}{\partial x_1}|X & \frac{\partial F_n}{\partial x_2}|X & \cdots & \frac{\partial F_n}{\partial x_n}|X \end{pmatrix}$$

With $X_0 = X$ and for $k > 1$, $X_k = f_k(X_{k-1})$

- tangent linear model, forward mode, column access

$$TLM(F, X) : v \to F'(X) \times v = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \cdots \times f'_1(X_0) \times v$$

- adjoint model, reverse mode, line access

$$ADJ(F', X) : u \to F'(X)^t \times u = f'_1(X_0)^t \times f'_2(X_1)^t \cdots \times f'_p(X_{p-1})^t \times u$$

Automatic differentiation

$F'(X)$ is the jacobian matrix

$$F'(X) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}|X & \frac{\partial F_1}{\partial x_2}|X & \cdots & \frac{\partial F_1}{\partial x_n}|X \\ \frac{\partial F_2}{\partial x_1}|X & \frac{\partial F_2}{\partial x_2}|X & \cdots & \frac{\partial F_2}{\partial x_n}|X \\ & & \cdots & \\ \frac{\partial F_n}{\partial x_1}|X & \frac{\partial F_n}{\partial x_2}|X & \cdots & \frac{\partial F_n}{\partial x_n}|X \end{pmatrix}$$

With $X_0 = X$ and for $k > 1$, $X_k = f_k(X_{k-1})$

- tangent linear model, forward mode, column access

$$TLM(F, X) : v \to F'(X) \times v = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \cdots \times f'_1(X_0) \times v$$

- adjoint model, reverse mode, line access

$$ADJ(F', X) : u \to F'(X)^t \times u = f'_1(X_0)^t \times f'_2(X_1)^t \cdots \times f'_p(X_{p-1})^t \times u$$

Example: as in a forward-backward scheme

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x*y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x*y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1+y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1+x \end{pmatrix}$$

TLM & ADJ

$$X_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, X_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$TLM : f_2(f_1(X)) \times f_1'(X) \times X_0$$

$$ADJ : f_1'(X) \times f_2'(f_1(X)) \times X_0$$

Tangent

Adjoint

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x * y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x * y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1 + y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1 + x \end{pmatrix}$$

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x * y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x * y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1 + y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1 + x \end{pmatrix}$$

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x * y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x * y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1 + y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1 + x \end{pmatrix}$$

TLM & ADJ

$$X_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix}, X_b = \begin{pmatrix} x_b \\ y_b \end{pmatrix}$$

$$TLM: f_2'(f_1(X)) \times f_1'(X) \times X_d$$

$$ADJ: f_1'^t(X) \times f_2'^t(f_1(X)) \times X_b$$

Tangent

```
xd = xd + xd*y + x*yd
x = x + x*y
yd = yd + xd*y + x*yd
```

Adjoint

```
CALL PUSHREAL4(x)
x = x + x*y
xb = xb + y*yb
yb = (x+1.0)*yb
CALL POPREAL4(x)
yb = yb + x*xb
xb = (y+1.0)*xb
```

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x * y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x * y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1 + y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1 + x \end{pmatrix}$$

TLM & ADJ

$$X_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix}, X_b = \begin{pmatrix} x_b \\ y_b \end{pmatrix}$$

$$TLM: f_2'(f_1(X)) \times f_1'(X) \times X_d$$

$$ADJ: f_1'^t(X) \times f_2'^t(f_1(X)) \times X_b$$

Tangent

```
xd = xd + xd*y + x*yd
x = x + x*y
yd = yd + xd*y + x*yd
```

Adjoint

```
CALL PUSHREAL4(x)
x = x + x*y
xb = xb + y*yb
yb = (x+1.0)*yb
CALL POPREAL4(x)
yb = yb + x*xb
xb = (y+1.0)*xb
```

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x * y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x * y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1 + y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1 + x \end{pmatrix}$$

TLM & ADJ

$$X_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix}, X_b = \begin{pmatrix} x_b \\ y_b \end{pmatrix}$$

$$TLM: f_2'(f_1(X)) \times f_1'(X) \times X_d$$

$$ADJ: f_1'^t(X) \times f_2'^t(f_1(X)) \times X_b$$

Tangent

```
xd = xd + xd*y + x*yd
x = x + x*y
yd = yd + xd*y + x*yd
```

Adjoint

```
CALL PUSHREAL4(x)
x = x + x*y
xb = xb + y*yb
yb = (x+1.0)*yb
CALL POPREAL4(x)
yb = yb + x*xb
xb = (y+1.0)*xb
```

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x * y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x * y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1 + y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1 + x \end{pmatrix}$$

TLM & ADJ

$$X_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix}, X_b = \begin{pmatrix} x_b \\ y_b \end{pmatrix}$$

$$TLM: f_2'(f_1(X)) \times f_1'(X) \times X_d$$

$$ADJ: f_1'^t(X) \times f_2'^t(f_1(X)) \times X_b$$

Tangent

```
xd = xd + xd*y + x*yd
x = x + x*y
yd = yd + xd*y + x*yd
```

Adjoint

```
CALL PUSHREAL4(x)
x = x + x*y
xb = xb + y*yb
yb = (x+1.0)*yb
CALL POPREAL4(x)
yb = yb + x*xb
xb = (y+1.0)*xb
```

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x * y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x * y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1 + y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1 + x \end{pmatrix}$$

TLM & ADJ

$$X_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix}, X_b = \begin{pmatrix} x_b \\ y_b \end{pmatrix}$$

$$TLM: f_2'(f_1(X)) \times f_1'(X) \times X_d$$

$$ADJ: f_1'^t(X) \times f_2'^t(f_1(X)) \times X_b$$

Tangent

```
xd = xd + xd*y + x*yd
x = x + x*y
yd = yd + xd*y + x*yd
```

Adjoint

```
CALL PUSHREAL4(x)
x = x + x*y
xb = xb + y*yb
yb = (x+1.0)*yb
CALL POPREAL4(x)
yb = yb + x*xb
xb = (y+1.0)*xb
```

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x * y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x * y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1 + y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1 + x \end{pmatrix}$$

TLM & ADJ

$$X_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix}, X_b = \begin{pmatrix} x_b \\ y_b \end{pmatrix}$$

$$TLM: f_2'(f_1(X)) \times f_1'(X) \times X_d$$

$$ADJ: f_1'^t(X) \times f_2'^t(f_1(X)) \times X_b$$

Tangent

```
xd = xd + xd*y + x*yd
x = x + x*y
yd = yd + xd*y + x*yd
```

Adjoint

```
CALL PUSHREAL4(x)
x = x + x*y
xb = xb + y*yb
yb = (x+1.0)*yb
CALL POPREAL4(x)
yb = yb + x*xb
xb = (y+1.0)*xb
```

Example: as in a forward-backward scheme

Program

```
subroutine f(x,y)
implicit none
real x,y
x = x + x*y
y = y + x*y
end subroutine f
```

Elementary functions

$$X = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$f_1(X) = \begin{pmatrix} x + x * y \\ y \end{pmatrix}, f_2(X) = \begin{pmatrix} x \\ y + x * y \end{pmatrix}$$

$$f_1'(X) = \begin{pmatrix} 1 + y & x \\ 0 & 1 \end{pmatrix}, f_2'(X) = \begin{pmatrix} 1 & 0 \\ y & 1 + x \end{pmatrix}$$

TLM & ADJ

$$X_d = \begin{pmatrix} x_d \\ y_d \end{pmatrix}, X_b = \begin{pmatrix} x_b \\ y_b \end{pmatrix}$$

$$TLM: f_2'(f_1(X)) \times f_1'(X) \times X_d$$

$$ADJ: f_1'^t(X) \times f_2'^t(f_1(X)) \times X_b$$

Tangent

```
xd = xd + xd*y + x*yd
x = x + x*y
yd = yd + xd*y + x*yd
```

Adjoint

```
CALL PUSHREAL4(x)
x = x + x*y
xb = xb + y*yb
yb = (x+1.0)*yb
CALL POPREAL4(x)
yb = yb + x*xb
xb = (y+1.0)*xb
```

croco AD status (https://gitlab.inria.fr/bremond/croco)

Differentiation of croco 2D and some part of croco 3D

- ad-seq-friction-tide : sequential control of Cb in a configuration similar to DAS & LARDNER
- ad-par-friction-tide : parallel (MPI) control of Cb
- to-tapenade3.14 : North Atlantic config, control of a spatialized z0b (dyneco_rec) (cf Martial Boutet thesis)
- **common-ad-config** : Internal + penalization, Basin (3D, MPI) + all the others 2D

croco AD status (https://gitlab.inria.fr/bremond/croco)

Differentiation of croco 2D and some part of croco 3D

- ad-seq-friction-tide : sequential control of Cb in a configuration similar to DAS & LARDNER
- ad-par-friction-tide : parallel (MPI) control of Cb
- to-tapenade3.14 : North Atlantic config, control of a spatialized z0b (dyneco_rec) (cf Martial Boutet thesis)
- **common-ad-config** : Internal + penalization, Basin (3D, MPI) + all the others 2D

croco AD status (https://gitlab.inria.fr/bremond/croco)

Differentiation of croco 2D and some part of croco 3D

- ad-seq-friction-tide : sequential control of Cb in a configuration similar to DAS & LARDNER
- ad-par-friction-tide : parallel (MPI) control of Cb
- to-tapenade3.14 : North Atlantic config, control of a spatialized z0b (dyneco_rec) (cf Martial Boutet thesis)
- **common-ad-config** : Internal + penalization, Basin (3D, MPI) + all the others 2D

croco AD status (https://gitlab.inria.fr/bremond/croco)

Differentiation of croco 2D and some part of croco 3D

- ad-seq-friction-tide : sequential control of Cb in a configuration similar to DAS & LARDNER
- ad-par-friction-tide : parallel (MPI) control of Cb
- to-tapenade3.14 : North Atlantic config, control of a spatialized z0b (dyneco_rec) (cf Martial Boutet thesis)
- **common-ad-config** : Internal + penalization, Basin (3D, MPI) + all the others 2D

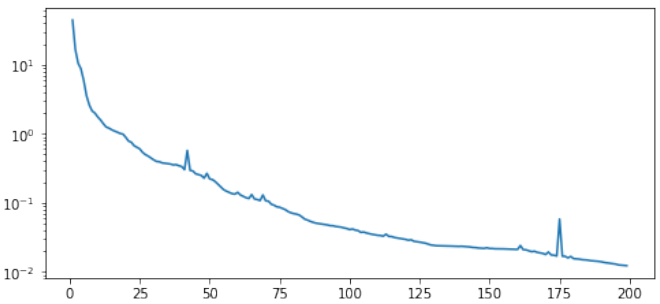
croco AD status (https://gitlab.inria.fr/bremond/croco)

Differentiation of croco 2D and some part of croco 3D

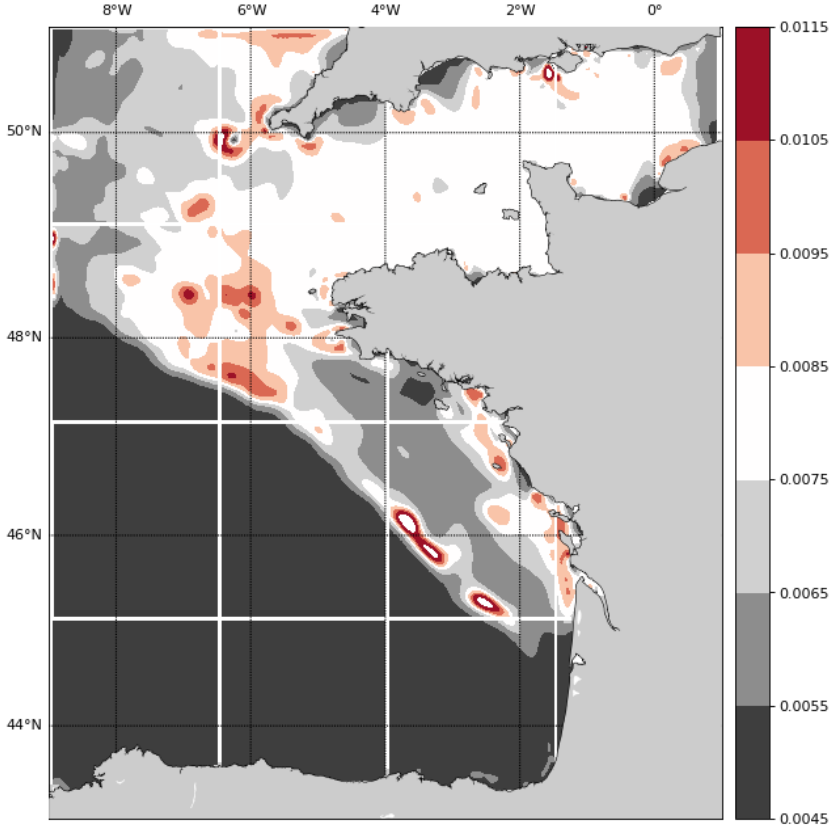
- ad-seq-friction-tide : sequential control of Cb in a configuration similar to DAS & LARDNER
- ad-par-friction-tide : parallel (MPI) control of Cb
- to-tapenade3.14 : North Atlantic config, control of a spatialized z0b (dyneco_rec) (cf Martial Boutet thesis)
- **common-ad-config** : Internal + penalization, Basin (3D, MPI) + all the others 2D

croco AD status (https://gitlab.inria.fr/bremond/croco)

Differentiation of croco 2D



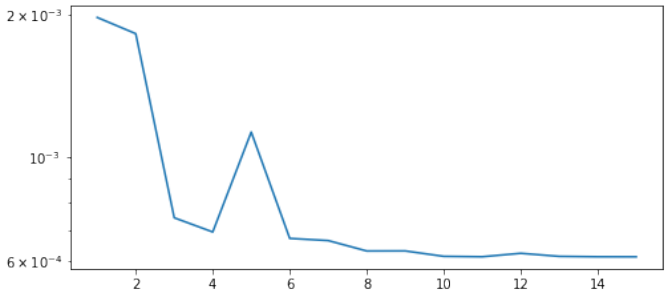
Cost (m^2)



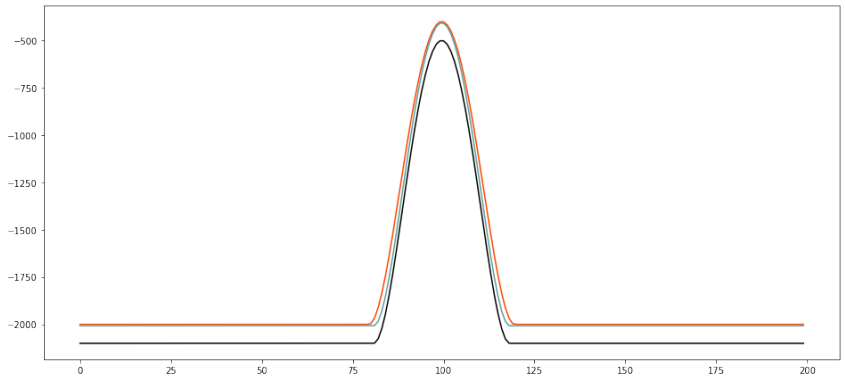
z0b (mm)

croco AD status (https://gitlab.inria.fr/bremond/croco)

Differentiation of croco 3D



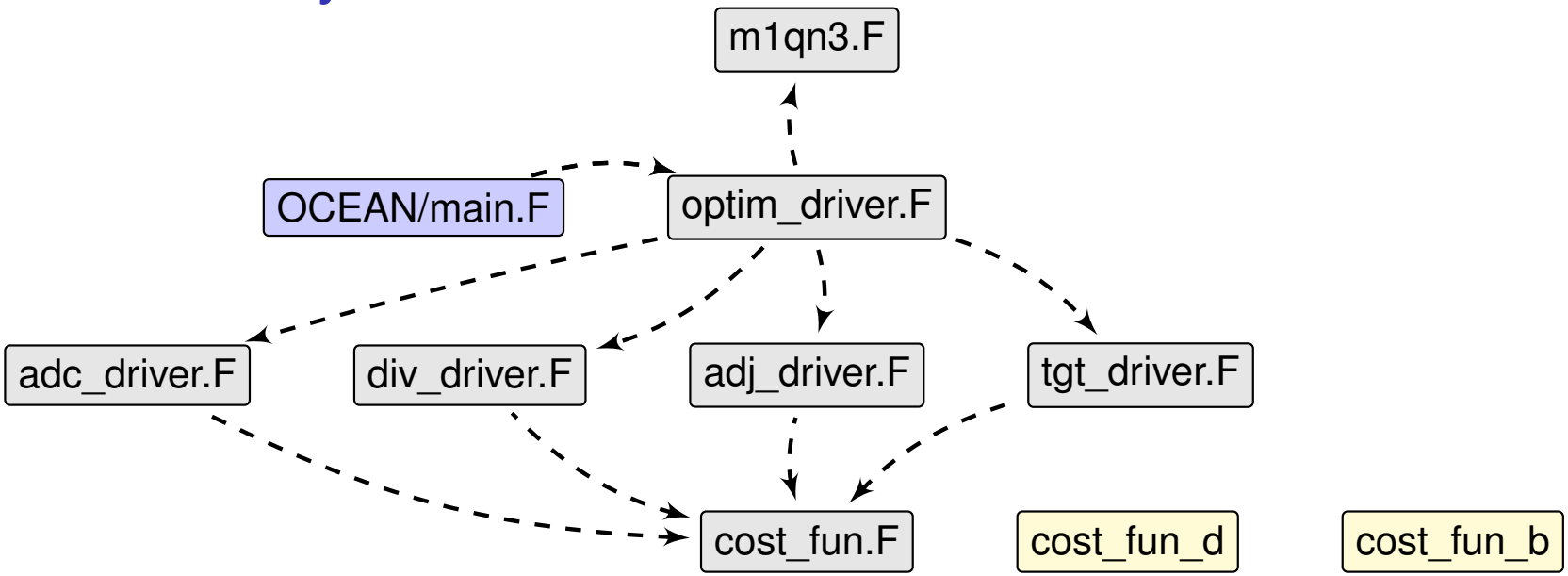
Cost (m^2)



penalization layer

AD code organization

AD directory



Compile directory

Tapenade

Compile/autodiff_d.F

Compile/autodiff_b.F

Makefile targets

croco

croco_adc

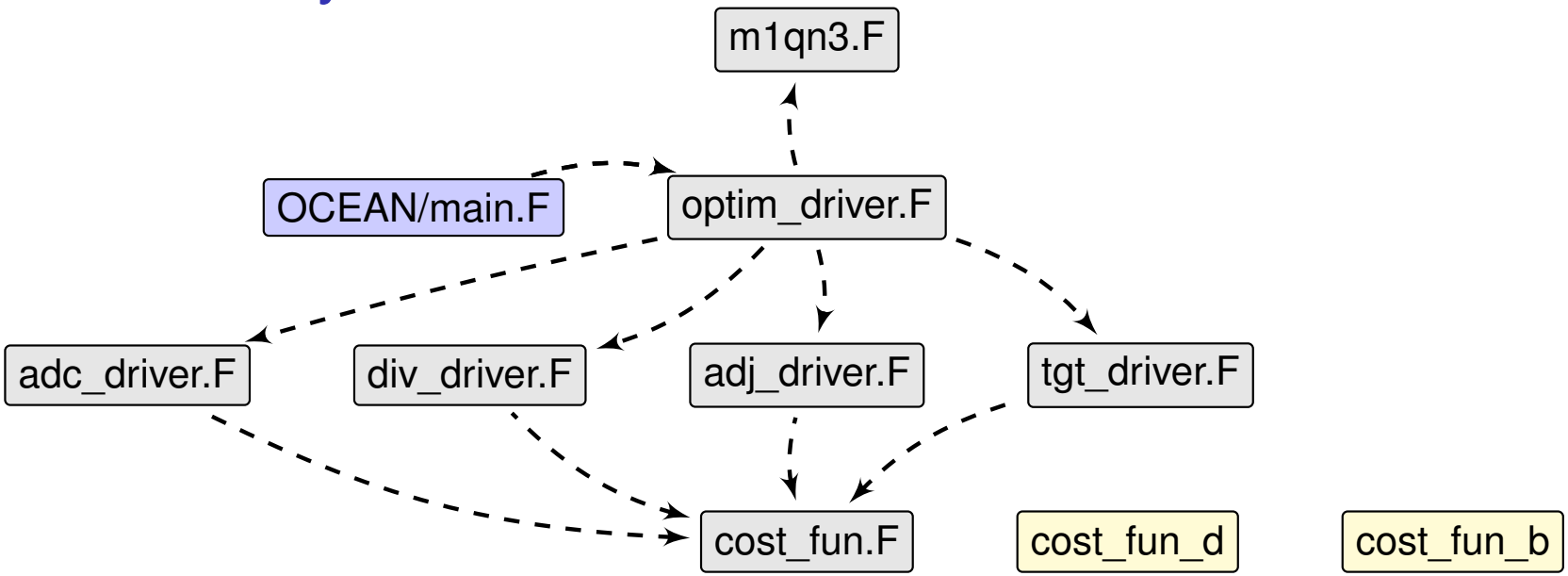
croco_div

croco_tgt

croco_adj

AD code organization

AD directory



Compile directory

Tapenade

Compile/autodiff_d.F

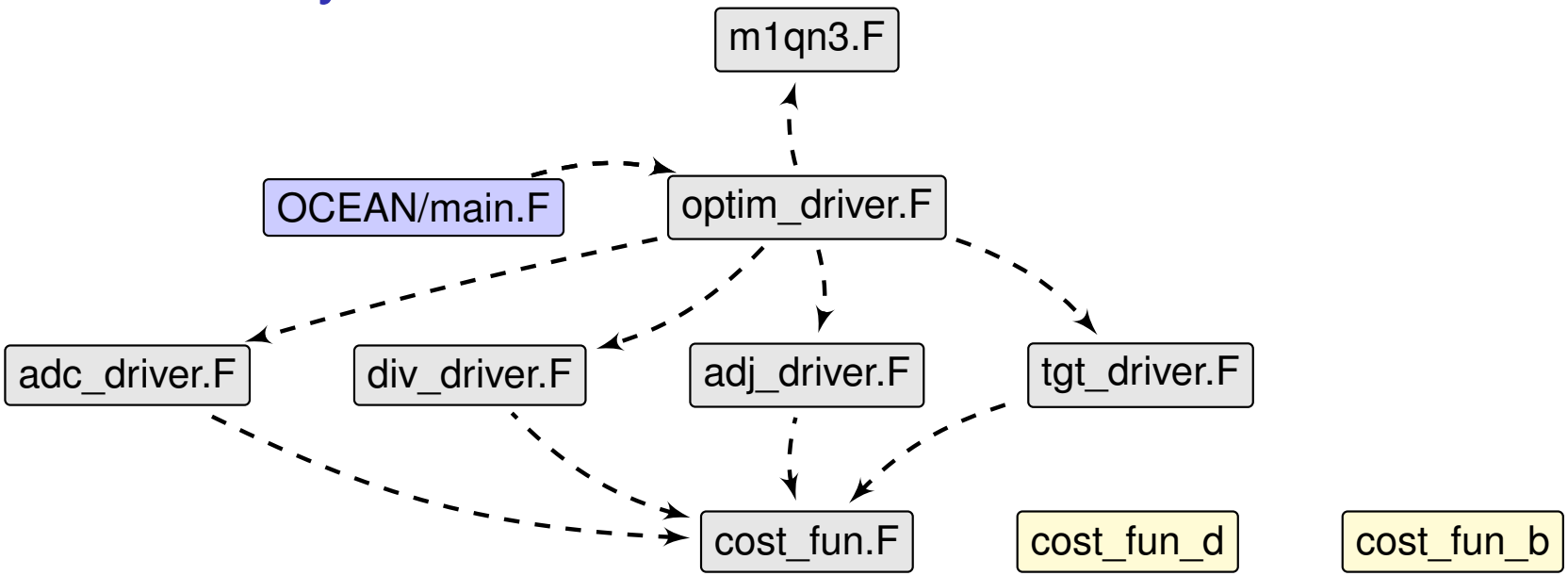
Compile/autodiff_b.F

Makefile targets

croco croco_adc croco_div croco_tgt croco_adj

AD code organization

AD directory



Compile directory

Tapenade

Compile/autodiff_d.F

Compile/autodiff_b.F

Makefile targets

croco

croco_adc

croco_div

croco_tgt

croco_adj

AD code organization

Configuration specific

adparam.h read_obs.F write_obs.F save_restore.F set_state.F

- `adparam.h`: parameters (size of the problem, of assimilation window, etc.),
- `save_restore`: full croco state,
- `set_state`: from control vector (`ad_x`) to croco state.

AD code organization

Configuration specific

adparam.h read_obs.F write_obs.F save_restore.F set_state.F

- `adparam.h`: parameters (size of the problem, of assimilation window, etc.),
- `save_restore`: full croco state,
- `set_state`: from control vector (`ad_x`) to croco state.

AD code organization

Configuration specific

adparam.h read_obs.F write_obs.F save_restore.F set_state.F

- `adparam.h`: parameters (size of the problem, of assimilation window, etc.),
- `save_restore`: full croco state,
- `set_state`: from control vector (`ad_x`) to croco state.

AD code organization

Configuration specific

adparam.h read_obs.F write_obs.F save_restore.F set_state.F

- `adparam.h`: parameters (size of the problem, of assimilation window, etc.),
- `save_restore`: full croco state,
- `set_state`: from control vector (`ad_x`) to croco state.

AD code organization

Configuration specific

adparam.h read_obs.F write_obs.F save_restore.F set_state.F

- `adparam.h`: parameters (size of the problem, of assimilation window, etc.),
- `save_restore`: full croco state,
- `set_state`: from control vector (`ad_x`) to croco state.

AD code organization

Tangent driver

```
do i=1,Sn
    ad_xd(i) = 1
    cost = icost
    idfdx = grad(i)
    call save_croco_state()
    write (*,*) '--------------------------------------'
    write (*,*) 'start tangent:',i
    write (*,*) '--------------------------------------'
    call cost_fun_d(ad_x, ad_xd, cost, idfdx)

    write (*,*) '--------------------------------------'
    write (*,*) 'end tangent:',i
    write (*,*) '--------------------------------------'
    call restore_croco_state()
    ad_xd(i) = 0
    grad(i)  = idfdx
 end do
```

AD code organization

Adjoint driver

```
call save_croco_state()
write (*,*) '--------------------------------------'
write (*,*) 'start cost'
write (*,*) '--------------------------------------'
call cost_fun(ad_x, icost)
cost = icost
write (*,*) '--------------------------------------'
write (*,*) 'end cost:'
write (*,*) '--------------------------------------'
call restore_croco_state()

call save_croco_state()
write (*,*) '--------------------------------------'
write (*,*) 'start adjoint'
write (*,*) '--------------------------------------'
ad_g(:) = 0.
costb = 1
call cost_fun_b(ad_x, ad_g, icost, costb)
write (*,*) '--------------------------------------'
write (*,*) 'end adjoint:'
write (*,*) '--------------------------------------'
call restore_croco_state()
```

Some issues for code writing:

1. non differentiable functions,
2. checkpointing of unused parts of arrays,
3. more checkpoints with temporary variables,
4. independent iterations loop,
5. global control of checkpointing.

Some issues for code writing:

1. non differentiable functions,
2. checkpointing of unused parts of arrays,
3. more checkpoints with temporary variables,
4. independent iterations loop,
5. global control of checkpointing.

Some issues for code writing:

1. non differentiable functions,
2. checkpointing of unused parts of arrays,
3. more checkpoints with temporary variables,
4. independent iterations loop,
5. global control of checkpointing.

Some issues for code writing:

1. non differentiable functions,
2. checkpointing of unused parts of arrays,
3. more checkpoints with temporary variables,
4. independent iterations loop,
5. global control of checkpointing.

Some issues for code writing:

1. non differentiable functions,
2. checkpointing of unused parts of arrays,
3. more checkpoints with temporary variables,
4. independent iterations loop,
5. global control of checkpointing.

Non differentiable functions

Non differentiable functions

Euclidian distance function

```
function f(x,y)

implicit none
real f,x,y

f = sqrt(x*x + y*y)

end function f
```

Tapenade adjoint

```
REAL tempb
IF (x**2 + y**2 .EQ. 0.0) THEN
  tempb = 0.0
ELSE
  tempb = fb/(2.0*SQRT(x**2+y**2))
END IF
xb = xb + 2*x*tempb
yb = yb + 2*y*tempb
```

Non differentiable output

Tapenade tangent

Non differentiable functions

Euclidian distance function

```
function f(x,y)

implicit none
real f,x,y

f = sqrt(x*x + y*y)

end function f
```

Non differentiable functions

Euclidian distance function

```
function f(x,y)

implicit none
real f,x,y

f = sqrt(x*x + y*y)

end function f
```

Non differentiable functions

Euclidian distance function

```
function f(x,y)

implicit none
real f,x,y

f = sqrt(x*x + y*y)

end function f
```

Tapenade adjoint

```
REAL tempb
IF (x**2 + y**2 .EQ. 0.0) THEN
  tempb = 0.0
ELSE
  tempb = fb/(2.0*SQRT(x**2+y**2))
END IF
xb = xb + 2*x*tempb
yb = yb + 2*y*tempb
```

Non differentiable functions

Euclidian distance function

```
function f(x,y)

implicit none
real f,x,y

f = sqrt(x*x + y*y)

end function f
```

Tapenade adjoint

```
REAL tempb
IF (x**2 + y**2 .EQ. 0.0) THEN
  tempb = 0.0
ELSE
  tempb = fb/(2.0*SQRT(x**2+y**2))
END IF
xb = xb + 2*x*tempb
yb = yb + 2*y*tempb
```

Non differentiable output

```
function f(x)
implicit none

real x,f

if (x.eq.1) then
   f = 1
else
   f = x*x
end if

end function f
```

Tapenade tangent

```
      FUNCTION F_D(x, xd, f)
      IMPLICIT NONE

C
      REAL x, f
      REAL xd, f_d

C
      IF (x .EQ. 1) THEN
        f = 1
        f_d = 0.0
      ELSE
        f_d = 2*x*xd
        f = x*x
      END IF
      END
```

derivative is 0 at 1

Non differentiable functions

Euclidian distance function

```
function f(x,y)

implicit none
real f,x,y

f = sqrt(x*x + y*y)

end function f
```

Tapenade adjoint

```
REAL tempb
IF (x**2 + y**2 .EQ. 0.0) THEN
  tempb = 0.0
ELSE
  tempb = fb/(2.0*SQRT(x**2+y**2))
END IF
xb = xb + 2*x*tempb
yb = yb + 2*y*tempb
```

Non differentiable output

```
function f(x)
implicit none

real x,f

if (x.eq.1) then
   f = 1
else
   f = x*x
end if

end function f
```

Tapenade tangent

```
      FUNCTION F_D(x, xd, f)
      IMPLICIT NONE
C
      REAL x, f
      REAL xd, f_d
C
      IF (x .EQ. 1) THEN
        f = 1
        f_d = 0.0
      ELSE
        f_d = 2*x*xd
        f = x*x
      END IF
      END
```

derivative is 0 at 1

Non differentiable functions

Euclidian distance function

```
function f(x,y)

implicit none
real f,x,y

f = sqrt(x*x + y*y)

end function f
```

Non differentiable output

```
function f(x)
implicit none

real x,f

if (x.eq.1) then
   f = 1
else
   f = x*x
end if

end function f
```

Tapenade adjoint

```
REAL tempb
IF (x**2 + y**2 .EQ. 0.0) THEN
  tempb = 0.0
ELSE
  tempb = fb/(2.0*SQRT(x**2+y**2))
END IF
xb = xb + 2*x*tempb
yb = yb + 2*y*tempb
```

Tapenade tangent

```
      FUNCTION F_D(x, xd, f)
      IMPLICIT NONE
C
      REAL x, f
      REAL xd, f_d
C
      IF (x .EQ. 1) THEN
        f = 1
        f_d = 0.0
      ELSE
        f_d = 2*x*xd
        f = x*x
      END IF
      END
```

derivative is 0 at 1

Non differentiable functions

Euclidian distance function

```
function f(x,y)

implicit none
real f,x,y

f = sqrt(x*x + y*y)

end function f
```

Tapenade adjoint

```
REAL tempb
IF (x**2 + y**2 .EQ. 0.0) THEN
  tempb = 0.0
ELSE
  tempb = fb/(2.0*SQRT(x**2+y**2))
END IF
xb = xb + 2*x*tempb
yb = yb + 2*y*tempb
```

Non differentiable output

```
function f(x)
implicit none

real x,f

if (x.eq.1) then
   f = 1
else
   f = x*x
end if

end function f
```

Tapenade tangent

```
      FUNCTION F_D(x, xd, f)
      IMPLICIT NONE
C
      REAL x, f
      REAL xd, f_d
C
      IF (x .EQ. 1) THEN
        f = 1
        f_d = 0.0
      ELSE
        f_d = 2*x*xd
        f = x*x
      END IF
      END
```

derivative is 0 at 1

Non differentiable functions

Euclidian distance function

```
function f(x,y)

implicit none
real f,x,y

f = sqrt(x*x + y*y)

end function f
```

Non differentiable output

```
function f(x)
implicit none

real x,f

if (x.eq.1) then
   f = 1
else
   f = x*x
end if

end function f
```

Tapenade adjoint

```
REAL tempb
IF (x**2 + y**2 .EQ. 0.0) THEN
  tempb = 0.0
ELSE
  tempb = fb/(2.0*SQRT(x**2+y**2))
END IF
xb = xb + 2*x*tempb
yb = yb + 2*y*tempb
```

Tapenade tangent

```
      FUNCTION F_D(x, xd, f)
      IMPLICIT NONE
C
      REAL x, f
      REAL xd, f_d
C
      IF (x .EQ. 1) THEN
        f = 1
        f_d = 0.0
      ELSE
        f_d = 2*x*xd
        f = x*x
      END IF
      END
```

derivative is 0 at 1

Checkpointing of unused parts of arrays

Only a part of x is used

```
subroutine f(x)

implicit none
real x(999,999)
integer i

do i=2,4,1
  call step(x,i)
end do

end subroutine f

subroutine step(x,i)
implicit none
real x(999,999)

integer i

x(i,1) = x(i-1,1) + x(i-1,1)*x(i-1,1)

end subroutine step
```

Tapenade adjoint

```
SUBROUTINE F_B(x, xb)
IMPLICIT NONE
REAL x(999, 999)
REAL xb(999, 999)
INTEGER i

DO i=2,4,1
  CALL PUSHREAL4ARRAY(x, 999**2)
  CALL STEP(x, i)
ENDDO
DO i=4,2,-1
  CALL POPREAL4ARRAY(x, 999**2)
  CALL STEP_B(x, xb, i)
ENDDO
END
```

Solution here

use a C$AD NOCHECKPOINT on step

Checkpointing of unused parts of arrays

Only a part of x is used

```
subroutine f(x)

implicit none
real x(999,999)
integer i

do i=2,4,1
  call step(x,i)
end do

end subroutine f

subroutine step(x,i)
implicit none
real x(999,999)

integer i

x(i,1) = x(i-1,1) + x(i-1,1)*x(i-1,1)

end subroutine step
```

Tapenade adjoint

```
SUBROUTINE F_B(x, xb)
IMPLICIT NONE
REAL x(999, 999)
REAL xb(999, 999)
INTEGER i

DO i=2,4,1
  CALL PUSHREAL4ARRAY(x, 999**2)
  CALL STEP(x, i)
ENDDO
DO i=4,2,-1
  CALL POPREAL4ARRAY(x, 999**2)
  CALL STEP_B(x, xb, i)
ENDDO
END
```

Solution here

use a C$AD NOCHECKPOINT on step

Checkpointing of unused parts of arrays

Only a part of x is used

```
subroutine f(x)

implicit none
real x(999,999)
integer i

do i=2,4,1
  call step(x,i)
end do

end subroutine f

subroutine step(x,i)
implicit none
real x(999,999)

integer i

x(i,1) = x(i-1,1) + x(i-1,1)*x(i-1,1)

end subroutine step
```

Tapenade adjoint

```
SUBROUTINE F_B(x, xb)
IMPLICIT NONE
REAL x(999, 999)
REAL xb(999, 999)
INTEGER i

DO i=2,4,1
  CALL PUSHREAL4ARRAY(x, 999**2)
  CALL STEP(x, i)
ENDDO
DO i=4,2,-1
  CALL POPREAL4ARRAY(x, 999**2)
  CALL STEP_B(x, xb, i)
ENDDO
END
```

Solution here

use a C$AD NOCHECKPOINT on step

Checkpointing of unused parts of arrays

Only a part of x is used

```
subroutine f(x)

implicit none
real x(999,999)
integer i

do i=2,4,1
  call step(x,i)
end do

end subroutine f

subroutine step(x,i)
implicit none
real x(999,999)

integer i

x(i,1) = x(i-1,1) + x(i-1,1)*x(i-1,1)

end subroutine step
```

Tapenade adjoint

```
SUBROUTINE F_B(x, xb)
IMPLICIT NONE
REAL x(999, 999)
REAL xb(999, 999)
INTEGER i

DO i=2,4,1
  CALL PUSHREAL4ARRAY(x, 999**2)
  CALL STEP(x, i)
ENDDO
DO i=4,2,-1
  CALL POPREAL4ARRAY(x, 999**2)
  CALL STEP_B(x, xb, i)
ENDDO
END
```

Solution here

use a C$AD NOCHECKPOINT on step

Checkpointing of unused parts of arrays

Only a part of x is used

```
subroutine f(x)

implicit none
real x(999,999)
integer i

do i=2,4,1
  call step(x,i)
end do

end subroutine f

subroutine step(x,i)
implicit none
real x(999,999)

integer i

x(i,1) = x(i-1,1) + x(i-1,1)*x(i-1,1)

end subroutine step
```

Tapenade adjoint

```
SUBROUTINE F_B(x, xb)
IMPLICIT NONE
REAL x(999, 999)
REAL xb(999, 999)
INTEGER i

DO i=2,4,1
  CALL PUSHREAL4ARRAY(x, 999**2)
  CALL STEP(x, i)
ENDDO
DO i=4,2,-1
  CALL POPREAL4ARRAY(x, 999**2)
  CALL STEP_B(x, xb, i)
ENDDO
END
```

Solution here

use a C$AD NOCHECKPOINT on step

Checkpointing of unused parts of arrays

Only a part of x is used

```
subroutine f(x)

implicit none
real x(999,999)
integer i

do i=2,4,1
  call step(x,i)
end do

end subroutine f

subroutine step(x,i)
implicit none
real x(999,999)

integer i

x(i,1) = x(i-1,1) + x(i-1,1)*x(i-1,1)

end subroutine step
```

Tapenade adjoint

```
SUBROUTINE F_B(x, xb)
IMPLICIT NONE
REAL x(999, 999)
REAL xb(999, 999)
INTEGER i

DO i=2,4,1
  CALL PUSHREAL4ARRAY(x, 999**2)
  CALL STEP(x, i)
ENDDO
DO i=4,2,-1
  CALL POPREAL4ARRAY(x, 999**2)
  CALL STEP_B(x, xb, i)
ENDDO
END
```

Solution here

use a `C$AD NOCHECKPOINT` on step

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(x)
  x = x + x*x
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  xb = (2*x+1.0)*xb
ENDDO
```

With a temp variable

Adjoint

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(x)
  x = x + x*x
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  xb = (2*x+1.0)*xb
ENDDO
```

With a temp variable

Adjoint

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(x)
  x = x + x*x
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  xb = (2*x+1.0)*xb
ENDDO
```

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(x)
  x = x + x*x
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  xb = (2*x+1.0)*xb
ENDDO
```

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(x)
  x = x + x*x
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  xb = (2*x+1.0)*xb
ENDDO
```

With a temp variable

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i
do i=1,2
   cff1=x
   cff2=cff1*x
   x=x+cff2
end do
end subroutine f
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(cff1)
  cff1 = x
  cff2 = cff1*x
  CALL PUSHREAL4(x)
  x = x + cff2
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  cff2b = xb
  cff1b = x*cff2b
  xb = xb + cff1b + cff1*cff2b
  CALL POPREAL4(cff1)
ENDDO
```

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

With a temp variable

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i
do i=1,2
   cff1=x
   cff2=cff1*x
   x=x+cff2
end do
end subroutine f
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(x)
  x = x + x*x
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  xb = (2*x+1.0)*xb
ENDDO
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(cff1)
  cff1 = x
  cff2 = cff1*x
  CALL PUSHREAL4(x)
  x = x + cff2
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  cff2b = xb
  cff1b = x*cff2b
  xb = xb + cff1b + cff1*cff2b
  CALL POPREAL4(cff1)
ENDDO
```

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

With a temp variable

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i
do i=1,2
   cff1=x
   cff2=cff1*x
   x=x+cff2
end do
end subroutine f
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(x)
  x = x + x*x
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  xb = (2*x+1.0)*xb
ENDDO
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(cff1)
  cff1 = x
  cff2 = cff1*x
  CALL PUSHREAL4(x)
  x = x + cff2
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  cff2b = xb
  cff1b = x*cff2b
  xb = xb + cff1b + cff1*cff2b
  CALL POPREAL4(cff1)
ENDDO
```

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

With a temp variable

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i
do i=1,2
   cff1=x
   cff2=cff1*x
   x=x+cff2
end do
end subroutine f
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(x)
  x = x + x*x
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  xb = (2*x+1.0)*xb
ENDDO
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(cff1)
  cff1 = x
  cff2 = cff1*x
  CALL PUSHREAL4(x)
  x = x + cff2
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  cff2b = xb
  cff1b = x*cff2b
  xb = xb + cff1b + cff1*cff2b
  CALL POPREAL4(cff1)
ENDDO
```

More checkpoints with temp variables

Direct update

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i

do i=1,2
   x=x+x*x
end do
```

With a temp variable

```
subroutine f(x)
implicit none
real x,cff1,cff2
integer i
do i=1,2
   cff1=x
   cff2=cff1*x
   x=x+cff2
end do
end subroutine f
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(x)
  x = x + x*x
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  xb = (2*x+1.0)*xb
ENDDO
```

Adjoint

```
DO i=1,2
  CALL PUSHREAL4(cff1)
  cff1 = x
  cff2 = cff1*x
  CALL PUSHREAL4(x)
  x = x + cff2
ENDDO
DO i=2,1,-1
  CALL POPREAL4(x)
  cff2b = xb
  cff1b = x*cff2b
  xb = xb + cff1b + cff1*cff2b
  CALL POPREAL4(cff1)
ENDDO
```

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Tapenade adjoint

```
DO i=2,4,2
  CALL PUSHREAL4(x(i))
  x(i) = x(i-1)*x(i-1)
ENDDO
DO i=4,2,-2
  CALL POPREAL4(x(i))
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Solution here: !$acc loop directive

```
!$acc loop
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
```

Again...

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Tapenade adjoint

```
DO i=2,4,2
  CALL PUSHREAL4(x(i))
  x(i) = x(i-1)*x(i-1)
ENDDO
DO i=4,2,-2
  CALL POPREAL4(x(i))
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Solution here: !$TLOOP directive

Again

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Tapenade adjoint

```
DO i=2,4,2
  CALL PUSHREAL4(x(i))
  x(i) = x(i-1)*x(i-1)
ENDDO
DO i=4,2,-2
  CALL POPREAL4(x(i))
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Solution here: !$acc loop directive

Again

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Tapenade adjoint

```
DO i=2,4,2
  CALL PUSHREAL4(x(i))
  x(i) = x(i-1)*x(i-1)
ENDDO
DO i=4,2,-2
  CALL POPREAL4(x(i))
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Tapenade adjoint

```
DO i=2,4,2
  CALL PUSHREAL4(x(i))
  x(i) = x(i-1)*x(i-1)
ENDDO
DO i=4,2,-2
  CALL POPREAL4(x(i))
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Solution here: II-LOOP directive

```
C$AD   II-LOOP
       do i = 2,4,2
          x(i) = x(i-1)*x(i-1)
       end do
```

Adjoint

```
      SUBROUTINE F_B(x, xb)
      IMPLICIT NONE
      REAL x(4)
      REAL xb(4)
      INTEGER i
C$BWD-OF II-LOOP
      DO i=2,4,2
        xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
        xb(i) = 0.0
      ENDDO
      END
```

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Tapenade adjoint

```
DO i=2,4,2
  CALL PUSHREAL4(x(i))
  x(i) = x(i-1)*x(i-1)
ENDDO
DO i=4,2,-2
  CALL POPREAL4(x(i))
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Solution here: II-LOOP directive

```
C$AD  II-LOOP
      do i = 2,4,2
         x(i) = x(i-1)*x(i-1)
      end do
```

Adjoint

```
      SUBROUTINE F_B(x, xb)
      IMPLICIT NONE
      REAL x(4)
      REAL xb(4)
      INTEGER i
C$BWD-OF II-LOOP
      DO i=2,4,2
        xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
        xb(i) = 0.0
      ENDDO
      END
```

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Solution here: II-LOOP directive

```
C$AD  II-LOOP
      do i = 2,4,2
         x(i) = x(i-1)*x(i-1)
      end do
```

Tapenade adjoint

```
DO i=2,4,2
  CALL PUSHREAL4(x(i))
  x(i) = x(i-1)*x(i-1)
ENDDO
DO i=4,2,-2
  CALL POPREAL4(x(i))
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Adjoint

```
SUBROUTINE F_B(x, xb)
IMPLICIT NONE
REAL x(4)
REAL xb(4)
INTEGER i
C$BWD-OF II-LOOP
DO i=2,4,2
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Solution here: II-LOOP directive

```
C$AD  II-LOOP
      do i = 2,4,2
         x(i) = x(i-1)*x(i-1)
      end do
```

Tapenade adjoint

```
DO i=2,4,2
  CALL PUSHREAL4(x(i))
  x(i) = x(i-1)*x(i-1)
ENDDO
DO i=4,2,-2
  CALL POPREAL4(x(i))
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Adjoint

```
      SUBROUTINE F_B(x, xb)
      IMPLICIT NONE
      REAL x(4)
      REAL xb(4)
      INTEGER i
C$BWD-OF II-LOOP
      DO i=2,4,2
        xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
        xb(i) = 0.0
      ENDDO
      END
```

Independent iterations : simple example

Update at even indices

```
subroutine f(x)
implicit none
real x(4)
integer i
do i = 2,4,2
   x(i) = x(i-1)*x(i-1)
end do
end subroutine f
```

Solution here: II-LOOP directive

```
C$AD  II-LOOP
      do i = 2,4,2
         x(i) = x(i-1)*x(i-1)
      end do
```

Tapenade adjoint

```
DO i=2,4,2
  CALL PUSHREAL4(x(i))
  x(i) = x(i-1)*x(i-1)
ENDDO
DO i=4,2,-2
  CALL POPREAL4(x(i))
  xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
  xb(i) = 0.0
ENDDO
END
```

Adjoint

```
      SUBROUTINE F_B(x, xb)
      IMPLICIT NONE
      REAL x(4)
      REAL xb(4)
      INTEGER i
C$BWD-OF II-LOOP
      DO i=2,4,2
        xb(i-1) = xb(i-1) + 2*x(i-1)*xb(i)
        xb(i) = 0.0
      ENDDO
      END
```

Dependent iterations : example in cost_fun.F

croco step call

```
C$AD BINOMIAL-CKP ad_nt+1 40 1
      do ta=1,ad_nt
         do xk=1,ad_ns
            call step()
         end do
         call cost_fun_step(ad_x_f,cost,ta)
      end do
```

- control of the maximum number q of allowed snapshots (here $q = 40$)
- The time required to compute the adjoint grows like the q-th root of ad_nt.
- config ATLN with 40 allowed snapshots, $ad\_nt = 80$, $ad\_ns = 180$: time of adjoint computation is 3.5 the time of model.

Dependent iterations : example in cost_fun.F

croco step call

```
C$AD BINOMIAL-CKP ad_nt+1 40 1
      do ta=1,ad_nt
         do xk=1,ad_ns
            call step()
         end do
         call cost_fun_step(ad_x_f,cost,ta)
      end do
```

- control of the maximum number q of allowed snapshots (here $q = 40$)
- The time required to compute the adjoint grows like the q-th root of ad_nt.
- config ATLN with 40 allowed snapshots, $ad\_nt = 80$, $ad\_ns = 180$: time of adjoint computation is 3.5 the time of model.

Dependent iterations : example in cost_fun.F

croco step call

```
C$AD BINOMIAL-CKP ad_nt+1 40 1
      do ta=1,ad_nt
         do xk=1,ad_ns
            call step()
         end do
         call cost_fun_step(ad_x_f,cost,ta)
      end do
```

- control of the maximum number q of allowed snapshots (here $q = 40$)
- The time required to compute the adjoint grows like the q-th root of ad_nt.
- config ATLN with 40 allowed snapshots, $ad\_nt = 80$, $ad\_ns = 180$: time of adjoint computation is 3.5 the time of model.

Dependent iterations : example in cost_fun.F

croco step call

```
C$AD BINOMIAL-CKP ad_nt+1 40 1
      do ta=1,ad_nt
         do xk=1,ad_ns
            call step()
         end do
         call cost_fun_step(ad_x_f,cost,ta)
      end do
```

- control of the maximum number q of allowed snapshots (here $q = 40$)
- The time required to compute the adjoint grows like the q-th root of ad_nt.
- config ATLN with 40 allowed snapshots, $ad\_nt = 80$, $ad\_ns = 180$: time of adjoint computation is 3.5 the time of model.

Aliasing

Example pre_step3D

```
            call pre_step3d_tile (Istr,Iend,Jstr,Jend,
     &                A3d(1,1,trd), A3d(1,2,trd), A3d(1,3,trd),
     &                A2d(1,1,trd), A2d(1,2,trd), A2d(1,3,trd),
# ifdef VADV_ADAPT_IMP
     &                                            A2d(1,4,trd),
# endif
     &                A2d(1,1,trd), A2d(1,2,trd), A2d(1,3,trd)
#ifdef OPENACC
     &   ,A3d(1,4,trd),A3d(1,5,trd),A3d(1,6,trd),A3d(1,7,trd)
#endif
     &                                          , A3d(1,8,trd)
     &                                                       )
```

Unaliasing

```
            call pre_step3d_tile (Istr,Iend,Jstr,Jend,
     &                ad_x_ru(:,trd),ad_x_rv(:,trd),ad_x_rw(:,trd),
     &                ad_x_FC(:,trd),ad_x_CF(:,trd),ad_x_DC(:,trd),
# ifdef VADV_ADAPT_IMP
     &                                            ad_x_WC(:,4,trd),
# endif
     &                ad_x_FX(:,trd), ad_x_FE(:,trd), ad_x_WORK(:,trd)
     &                                          , ad_x_Hz_half(:,trd)
     &                                                       )
```

Aliasing

Example pre_step3D

```
            call pre_step3d_tile (Istr,Iend,Jstr,Jend,
     &                A3d(1,1,trd), A3d(1,2,trd), A3d(1,3,trd),
     &                A2d(1,1,trd), A2d(1,2,trd), A2d(1,3,trd),
# ifdef VADV_ADAPT_IMP
     &                                            A2d(1,4,trd),
# endif
     &                A2d(1,1,trd), A2d(1,2,trd), A2d(1,3,trd)
#ifdef OPENACC
     &   ,A3d(1,4,trd),A3d(1,5,trd),A3d(1,6,trd),A3d(1,7,trd)
#endif
     &                                          , A3d(1,8,trd)
     &                                                       )
```

Unaliasing

```
           call pre_step3d_tile (Istr,Iend,Jstr,Jend,
     &                ad_x_ru(:,trd),ad_x_rv(:,trd),ad_x_rw(:,trd),
     &                ad_x_FC(:,trd),ad_x_CF(:,trd),ad_x_DC(:,trd),
# ifdef VADV_ADAPT_IMP
     &                                            ad_x_WC(:,4,trd),
# endif
     &                ad_x_FX(:,trd), ad_x_FE(:,trd), ad_x_WORK(:,trd)
     &                                          , ad_x_Hz_half(:,trd)
     &                                                       )
```