Inverse method for non linear ablative thermics and trajectory problems.

Groupe de travail « Modelisation des systemes complexes » CNAM Paris

22 Septembre 2010

Stéphane ALESTRA

EADS INNOVATION WORKS - FRANCE

Modèle M équation EDP / EDO

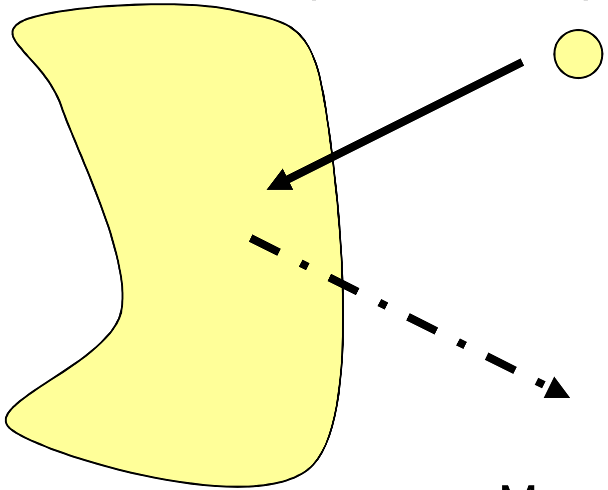
Y=M(p)

Problème Inverse

Trouver p connaissant Yobs et M

➔ problème non linéaire **mal posé**

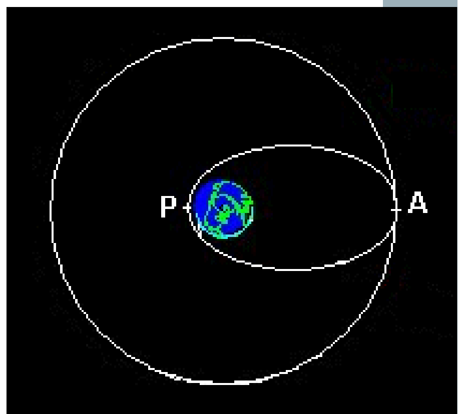
Materiau p ou Source p ?



Mesure

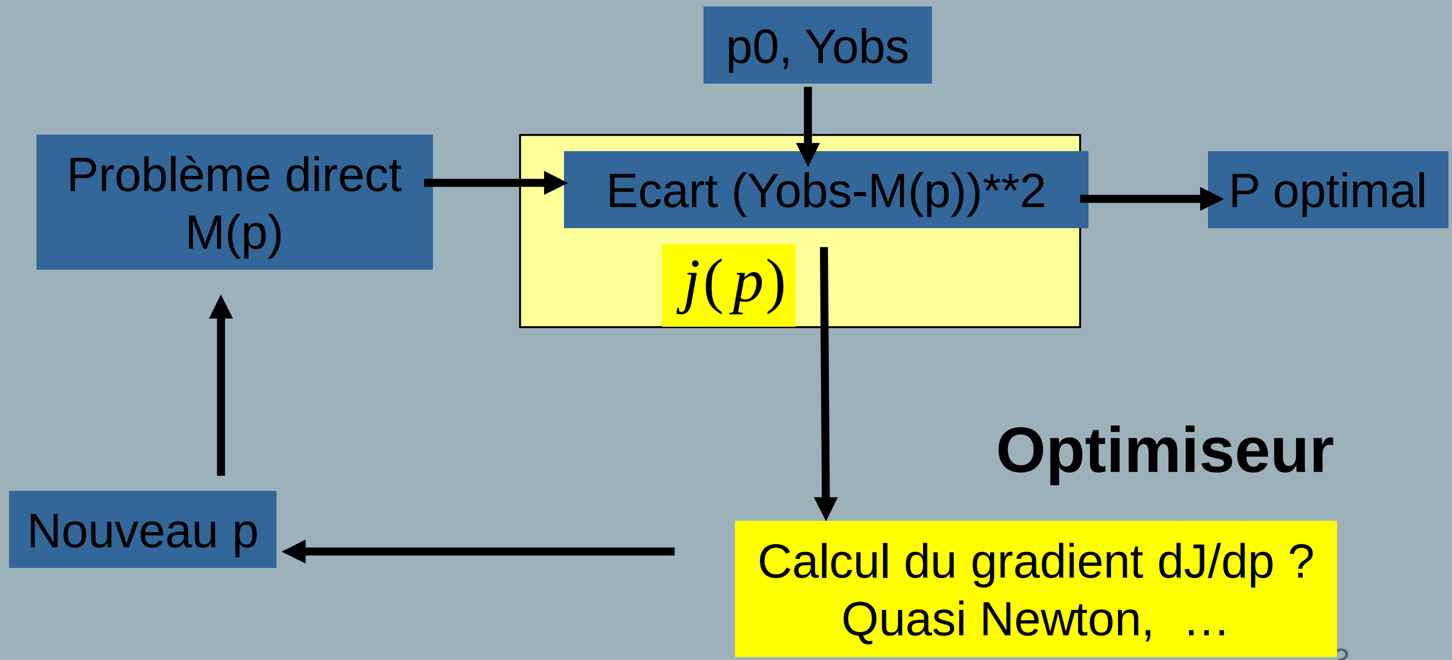
Yobs

Optimisation



Résolution du problème inverse

Minimisation Ecart j(p): EDP, EDO, ….



Méthode Contrôle optimal

Système Dynamique

Optimisation : Calcul des gradients

- **Classique : différences finies**

$$\nabla J_i = \frac{J(p_1, \dots p_i + \varepsilon, \dots, p_n) - J((p_i)_{1 \le i \le n})}{\varepsilon}$$

... coûteux, gestion du paramètre ε

- Plus économique et précis :

- Formulation de contrôle optimal “à la main”
- Génération automatique d'un code adjoint par le logiciel TAPENADE de l'INRIA (Sophia-Antipolis)

➔ **Calcul du gradient précis et rapide**

Contrôle optimal dynamique

Equation d'état

$$\frac{\partial U}{\partial t} = f(U,p) + g(p)$$
$$U(t=0) = 0$$

Equation Directe
Progressive
Condition initiale

Coût Lagrangien

$$j(p) = J(U(p)) = \frac{1}{2}\int \left(U(p,t) - U_{obs}(t)\right)^2 dt$$

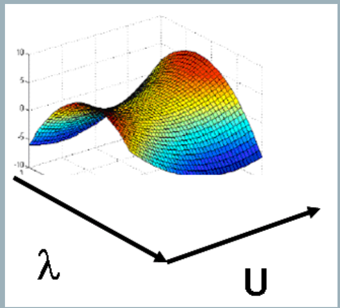
$$L(p,U,\lambda) = J(U) + \int {}^t\lambda \left(f(U,p) + g(p) - \frac{\partial U}{\partial t} \right) dt$$

Variations de L

$$\delta L_{1*1} = \frac{\partial J}{\partial U}\delta U + \int \left({}^t\lambda \frac{\partial f}{\partial U}\delta U - {}^t\lambda \frac{\partial \delta U}{\partial t} \right) dt$$
$$+ \int \left({}^t\lambda \frac{\partial f}{\partial p}\delta p + {}^t\lambda \frac{\partial g}{\partial p}\delta p \right) dt + \int \delta {}^t\lambda \left(f(U,p) + g(p) - \frac{\partial U}{\partial t} \right) dt$$

Contrôle optimal dynamique

En regroupant et après intégration par parties



$$\delta L = \int \left(\left(U(p,t) - U_{obs}(t)\right) + \frac{\partial \lambda}{\partial t} + {}^t\lambda \frac{\partial f}{\partial U} \right) \delta U \, dt \; - \left[{}^t\lambda \delta U \right]_0^T$$

$$+ \left(\int {}^t\lambda \left(\frac{\partial f}{\partial p} + \frac{\partial g}{\partial p} \right) dt \right) \delta p$$

$$+ \int \delta {}^t\lambda \left(-\frac{\partial U}{\partial t} + f(U,p)U + g(p) \right) dt$$

Adjoint f/U

$$\frac{\partial L}{\partial U} = 0$$

$$\frac{\partial \lambda}{\partial t} + \left(\frac{\partial f}{\partial U} \right)^* \lambda = -\left(U(p,t) - U_{obs}(t)\right)$$

$$\lambda(t = T) = 0$$

Equation Adjointe Rétrograde
Condition finale

Gradients f/p

$$\frac{\partial j}{\partial p} = \frac{\partial L}{\partial p} = \int {}^t\lambda \left(\frac{\partial f(U,p)}{\partial p} + \frac{\partial g}{\partial p} \right) dt$$

Complexe à calculer « à la main »

Risques d'erreurs … ➔ DA ....

Contrôle optimal dynamique

Equation Directe
Progressive
Condition Initiale

$$\frac{\partial X}{\partial t} = f(X,p) + g(p)$$
$$X(t=0) = 0$$

X

temps

df / dX

temps

Equation Adjointe
Rétrograde
Condition finale

$$\frac{\partial \lambda}{\partial t} + \left(\frac{\partial f}{\partial X}\right)^{*} \lambda = -\left(X(p,t) - X_{obs}(t)\right)$$
$$\lambda(t=T) = 0$$

Mesures

df / dp

λ

Gradients

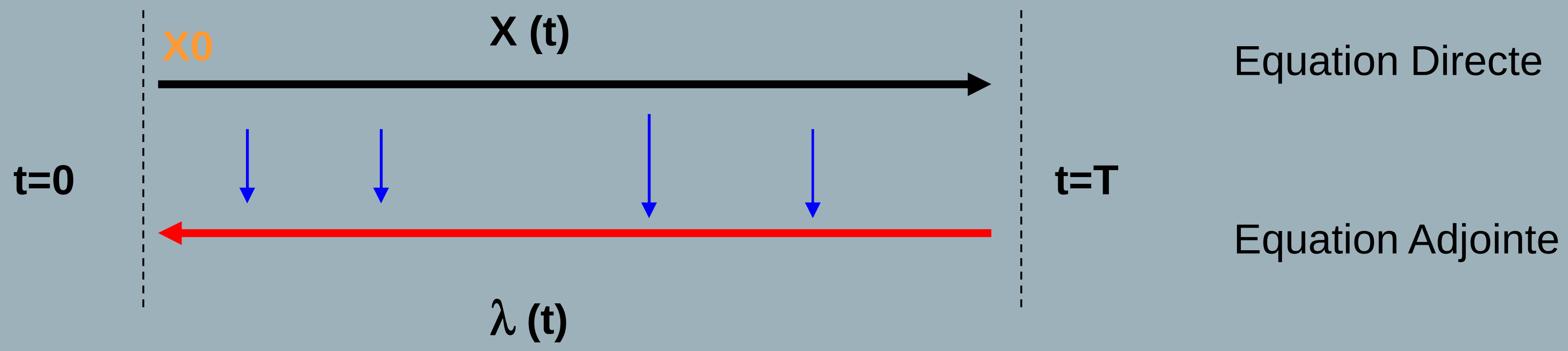
$$\frac{\partial j}{\partial p} = \frac{\partial L}{\partial p} = \int {}^{t}\lambda \left(\frac{\partial f(X,p)}{\partial p} + \frac{\partial g}{\partial p} \right) dt$$

Travailler sur la formulation adjointe discrétisée plus précise !!

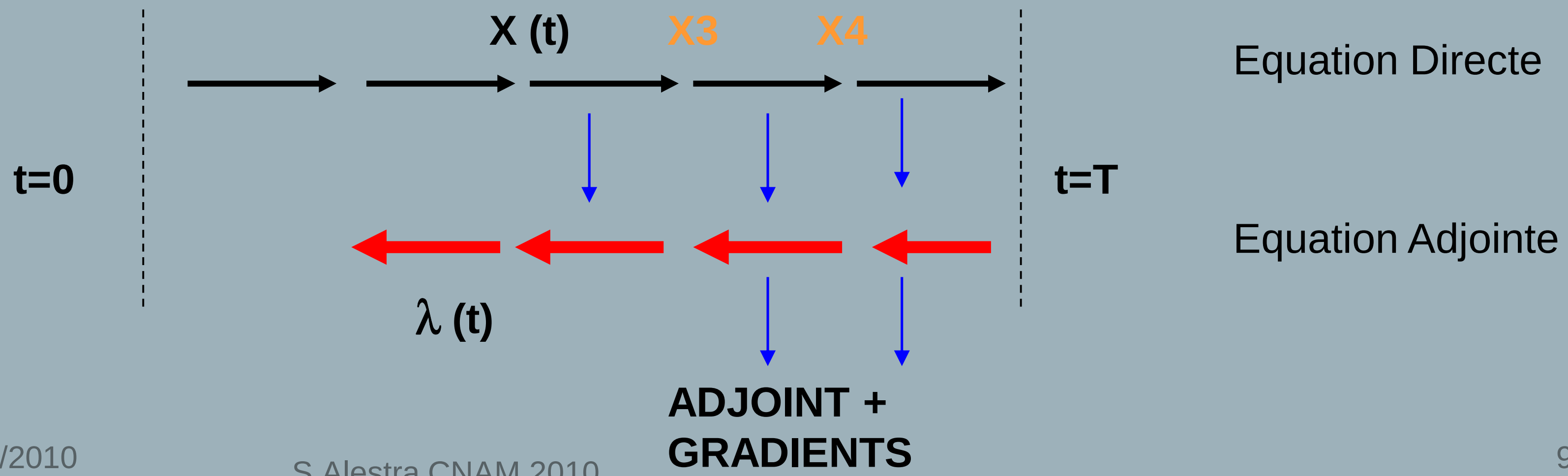
Méthode Contrôle optimal :

techniques de checkpointing

Stockage de tout l'historique de X(t) puis croisement avec λ (t)



Sauvegarde par tranches temporelles avec regénération de X(t)
À sa valeur initiale Xi sauvegardée de chaque tranche



EADS

Differentiation automatique

Utilisation de TAPENADE INRIA Sophia Antipolis

http://tapenade.inria.fr:8080/tapenade/index.jsp

L.Hascoet

Differentiation automatique (DA) d'instructions

Entrée p Parametre

Sortie J cout

Contraintes successives

$$s_1(p) \Rightarrow \quad F^{(1)}(X^{(1)}, p) = 0$$

$$s_2(s_1(p)) \Rightarrow \quad F^{(2)}(X^{(2)}, X^{(1)} p) = 0$$

....

$$J(s_k)$$

$$\mathbf{J(s_k(s_{k-1}(\ldots\ldots s_1(p)))}$$

Le solveur dynamique est vu comme une suite d'instructions (avec flot de dépendances)

➔ **codes Fortran en Recherche**

➔ **techniques de contrôle optimal**

➔ **Éviter de calculer un adjoint à la main**

➔ **Aspect modularité du code et modifications morceaux de codes**

Gradient de J / p ?

Mode Tangent, Direct = mode_d

Entree

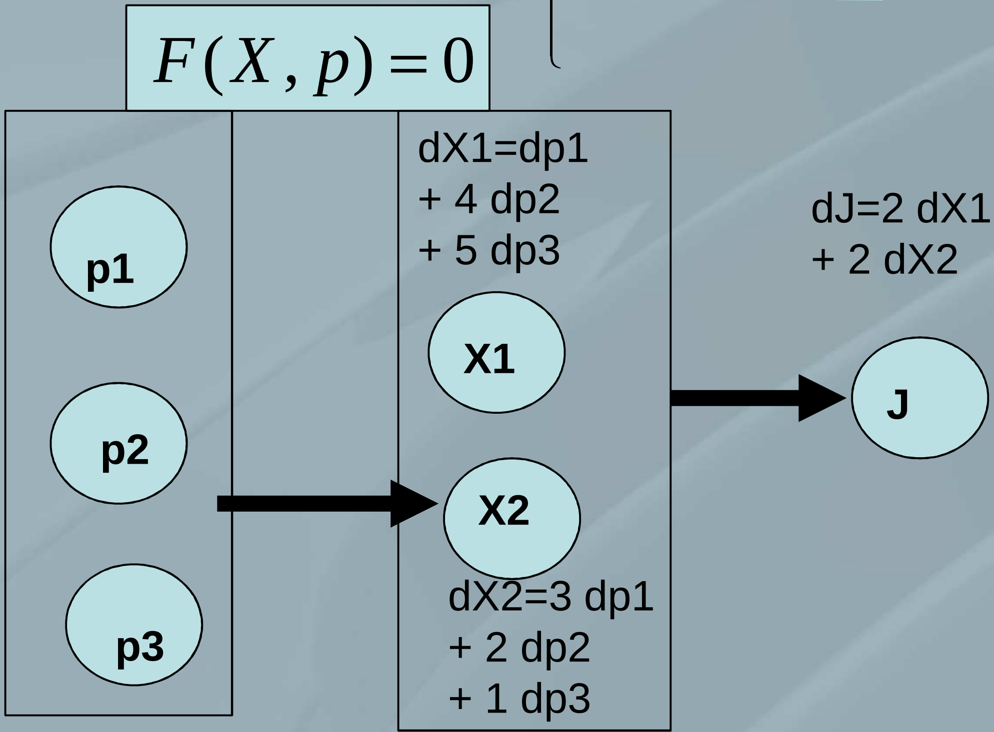
$$p = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

Contraintes X=(X1,X2)

$$\begin{cases} F_1(X) = X_1 - \begin{pmatrix} 1 & 4 & 5 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = 0 \\ F_2(X) = X_2 - \begin{pmatrix} 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = 0 \end{cases}$$

Sortie

$$J(X) = X_1^2 + X_2^2$$



Pour peu de paramètres p

Pour grand nombre de sorties J

➔ Efficace en résolution

Mode Tangent, Direct

Entree

$$p = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

Contraintes d'état X=(X1,X2)

$$\begin{cases} F_1(X) = X_1 - \begin{pmatrix} 1 & 4 & 5 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = 0 \\ F_2(X) = X_2 - \begin{pmatrix} 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = 0 \end{cases}$$

Sortie

$$J(X) = X_1^2 + X_2^2$$

Jacobienne

$$dJ = 2X_1 dX_1 + 2X_2 dX_2 = \left[2X_1 \begin{pmatrix} 1 & 4 & 5 \end{pmatrix}\right] \begin{pmatrix} dp_1 \\ dp_2 \\ dp_3 \end{pmatrix} + \left[2X_2 \begin{pmatrix} 3 & 2 & 1 \end{pmatrix}\right] \begin{pmatrix} dp_1 \\ dp_2 \\ dp_3 \end{pmatrix}$$

$$dp_1 = 1, dp_2 = dp_3 = 0$$

$$\frac{\partial J}{\partial p_1} = 2X_1 + 6X_2$$

$$dp_2 = 1, dp_1 = dp_3 = 0$$

$$\frac{\partial J}{\partial p_2} = 8X_1 + 4X_2$$

$$\left(\frac{\partial J}{\partial p}\right) = \left(\frac{\partial J}{\partial X}\right)\left(\frac{\partial X}{\partial p}\right)$$

Dérivation directe dans le sens du flot

3 RUNS « canoniques » pour avoir le gradient

Mode Tangent, Direct => Jacobienne

Paramètre Entrée	p
Equation des contraintes	$F(X, p) = 0$
Cout / sortie	$J(X)$

On veut calculer la jacobienne de sortie J(X) par rapport à l'entrée p

$$\frac{\partial J}{\partial p} = \frac{\partial J}{\partial X} \boxed{\frac{\partial X}{\partial p}}$$

→ **Sensibilité des contraintes**

$$\frac{\partial F}{\partial X} \boxed{\frac{\partial X}{\partial p}} + \frac{\partial F}{\partial p} = 0$$

$$\frac{\partial X}{\partial p} = -\left(\frac{\partial F}{\partial X}\right)^{-1} \frac{\partial F}{\partial p}$$

Th fct. implicites

Pour peu de paramètres p

Pour grand nombre de sorties J

Efficace en résolution

Mode Reverse, Adjoint = mode_b ➔ Gradient

Entree

$$p = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

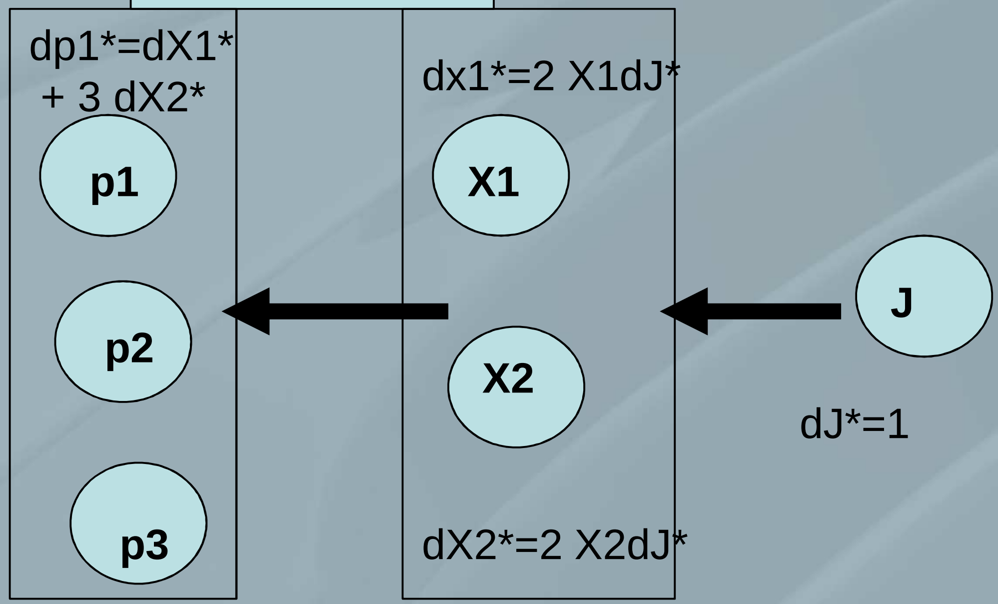
Contraintes

$$\begin{cases} F_1(X) = X_1 - \begin{pmatrix} 1 & 4 & 5 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = 0 \\ F_2(X) = X_2 - \begin{pmatrix} 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = 0 \end{cases}$$

Sortie

$$J(X) = X_1^2 + X_2^2$$

$$F(X, p) = 0$$



Pour p un grand nombre de paramètres

Pour peu de sorties J

➔ Efficace en résolution

Mode Adjoint, Dual

Entree

$$p = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

Contraintes

$$\begin{cases} F_1(X) = X_1 - (1 \quad 4 \quad 5)\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = 0 \\ F_2(X) = X_2 - (3 \quad 2 \quad 1)\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = 0 \end{cases}$$

Sortie

$$J(X) = X_1^2 + X_2^2$$

Gradient

$$\begin{pmatrix} dp_1* \\ dp_2* \\ dp_3* \end{pmatrix} = \begin{bmatrix} 1 & 3 \\ 4 & 2 \\ 5 & 1 \end{bmatrix} \begin{bmatrix} 2X_1 \\ 2X_2 \end{bmatrix} dJ*$$

$$\left(\frac{\partial J}{\partial p}\right)^t = -\left(\frac{\partial F}{\partial p}\right)^t \left(\frac{\partial J}{\partial X}\right)^t$$

$dJ* = 1$

$$\frac{\partial J}{\partial p_1} = dp_1* = 2X_1 + 6X_2$$

$dJ* = 1$

$$\frac{\partial J}{\partial p_2} = dp_2* = 8X_1 + 4X_2$$

Dérivation rétrograde du flot ➔ stocker en mémoire le flot direct

, 1 RUN pour avoir le gradient

Mode adjoint/Reverse Gradients

Cout / sortie $J(X)$

Equation des contraintes / etat intermediaire $F(X,p)=0$

Paramètre Entrée p

Lagrangien $L(p,\lambda,x)=J(x)+\langle\lambda,F(x,p)\rangle$

Posons λ Adjoint de X

$$\left(\frac{\partial F}{\partial X}\right)^t \lambda = \left(\frac{\partial J}{\partial X}\right)^t$$

Pour p un grand nombre de paramètres

Pour peu de sorties J

Efficace en résolution

$$\longrightarrow \left(\frac{\partial J}{\partial p}\right)^t = -\left(\frac{\partial F}{\partial p}\right)^t \lambda$$

MAIS GARDER L'HISTORIQUE

La DA empile → PUSH et dépile POP

Exemples instructions explicites

Instructions

$$x_1 = f_1(x_0, p)$$
$$x_2 = f_2(x_1, p)$$
$$j(p) = J(x_2) = g(x_2)$$

$$x_1 = 3p^2$$
$$x_2 = 2x_1 + 3p$$
$$J(x) = x_2^2$$

TANGENT : Jacobienne dans le sens du flot

lourd produit matrice*matrice

$$\frac{\nabla j}{\nabla p} = \underbrace{\underbrace{\frac{\partial J}{\partial x_2}}_{1*M} \underbrace{\left(\underbrace{\frac{\partial f_2}{\partial x_1}}_{M*M} \underbrace{\left(\frac{\partial f_1}{\partial p}\right)}_{M*P} + \underbrace{\frac{\partial f_2}{\partial p}}_{M*P} \right)}_{M*P}}_{1*P}$$

REVERSE :

Gradient dans le sens inverse du flot

Economique produit matrice*vecteur

$$\left(\frac{\nabla j}{\nabla p}\right)^t = \left(\frac{\nabla L}{\nabla p}\right)^t = -\left(\frac{\partial f_1}{\partial p}\right)^t \lambda_1 - \left(\frac{\partial f_2}{\partial p}\right)^t \lambda_2$$

$$= \underbrace{\underbrace{\underbrace{\left(\frac{\partial f_1}{\partial p}\right)^t}_{P*M} \underbrace{\underbrace{\left(\frac{\partial f_2}{\partial x_1}\right)^t}_{M*M} \underbrace{\left(\frac{\partial J}{\partial x_2}\right)^t}_{M*1}}_{M*1}}_{P*1} + \underbrace{\underbrace{\left(\frac{\partial f_2}{\partial p}\right)^t}_{P*M} \underbrace{\left(\frac{\partial J}{\partial x_2}\right)^t}_{M*1}}_{P*1}}_{P*1}$$

Instruction implicite, Mode tangent

Instructions, F non linéaire

$$F(x,p)=0$$
$$j(p)=J(x)$$

$$x^3 p + x p^2 + 3 = 0$$
$$J(x)=x^2$$

P Entrées, M Etats, 1 Sortie

$$p=\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ \vdots \\ p_P \end{pmatrix} \qquad x=\begin{pmatrix} x_1^1 \\ x_1^2 \\ \vdots \\ \vdots \\ x_1^M \end{pmatrix}$$

$$\frac{\partial F}{\partial x} = 3x^2 p + p^2$$
$$\frac{\partial F}{\partial p} = x^3 p + 2xp$$

x calculé par Newton itératif sur F

$$(x_0, p) \qquad F(x_0,p)\neq 0$$
$$(x_1, p)$$
$$(x_N, p) \qquad F(x_N,p)=0$$

Gradient dans le sens du flot

$$\frac{\nabla j}{\nabla p} = -\frac{\partial J}{\partial x}\underbrace{\left(\frac{\partial F}{\partial x}\right)^{-1}\frac{\partial F}{\partial p}}_{\frac{\partial x}{\partial p}}$$

xd ↗ … ↖ Pd

Différentier au point solution x, apres la boucle Newton

➔ Linéarisation

On ne differentie pas dans la boucle de Newton

Instruction implicite, Mode reverse

Instructions

$$F(x,p)=0$$
$$j(p)=J(x)$$

$$x^3p+xp^2+3=0$$
$$J(x)=x^2$$

Gradient dans le sens inverse du flot

$$\lambda=-\left(\frac{\partial F}{\partial x}\right)^{-t}\left(\frac{\partial J}{\partial x}\right)^{t} \leftarrow \text{xb}$$

$$\frac{\nabla j}{\nabla p}=\frac{\nabla L}{\nabla p}=\left\langle \lambda,\frac{\partial F}{\partial p}\right\rangle \leftarrow \text{Pb}$$

P Entrées, M Etats, 1 Sortie

$$p=\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ \vdots \\ p_P \end{pmatrix} \qquad x=\begin{pmatrix} x_1^1 \\ x_1^2 \\ \vdots \\ \vdots \\ x_1^M \end{pmatrix}$$

- **Différentier au point solution x, apres la boucle Newton**
- **Remontée en backward = Pb d'instructions pas explicite**

➔**Utiliser la relation suivante + dérivées tangentes**

(Pd|Pb) = (xd|xb)

Intervention manuelle "mahématiques' + reste du code en DA

Applications

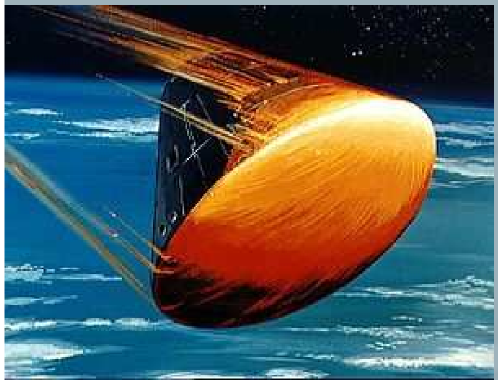
I Identification de coefficients aérodynamiques

II Mise en orbite avec maximisation charge utile

III Identification de flux en thermique

I Identification of aerodynamic coefficients

E.Leibenguth, A.Charpe, V.Srithammavanh, S.Alestra,, E.Clopeau (EADS), F.Dubois (CNAM)



• Atmospheric re-entry probe control trajectory

• Necessity to identify accurately the aerodynamic behaviour of the probe

➔ Trajectory measurements on ground with probe's shot by a cannon

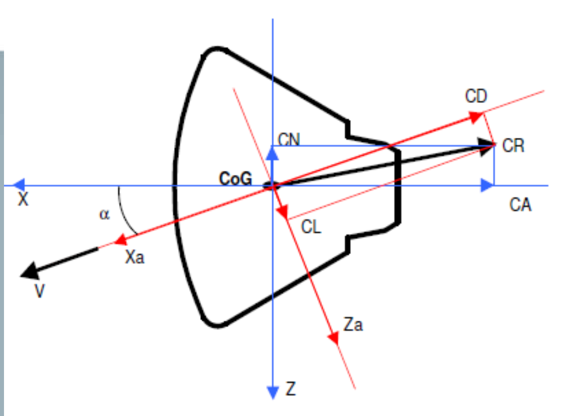
➔ Determine the aerodynamic coefficients

Aerodynamics & 2D kinematics modelling

- **Aerodynamic forces and coefficients**

DRAG $D = C_D * Q * S$ **LIFT** $L = C_L * Q * S$

Q: Dynamic pressure; S: Reference area



Aerodynamics coordinate system (wind)

$$C_D = C_A.\cos\alpha + C_N.\sin\alpha$$

$$C_L = -C_A.\sin\alpha + C_N.\cos\alpha$$

- **Aerodynamic moment**

$$M_{aero} = C_m * Q * S \quad \text{with} \quad C_m = C_{m\alpha}\alpha + C_{mq} * \frac{\omega L}{V}$$

- **Dynamics equation solved by RUNGE-KUTTA (RK4)**

$$X = \begin{bmatrix} x \\ y \\ V_x \\ V_y \\ \theta \\ \omega \end{bmatrix} \qquad \dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{V}_x \\ \dot{V}_y \\ \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} V_x \\ V_y \\ F_{aeroX}/m \\ F_{aeroY}/m - g \\ \omega \\ M_{aero}/I_z \end{bmatrix}$$

depending on

$$C_D, C_L, C_{m\alpha}, C_{mq}$$

➔ **We want to identify coefficients C from measurements of X**

- **Problem :** reconstruct the aerodynamic coefficient from measurements of angle of attack and velocity

- **Parameter:** $p(\alpha_i, M_i) = (C_A, C_N, C_{ma}, Cm_{mq})(\alpha_i, M_i)$ tables →

α(°)		0,00	1,00	2,00	4,00
Mach					
0,0		0,00000	-0,00242	-0,00484	-0,01024
0,4		0,00000	-0,00242	-0,00484	-0,01024
0,6		0,00000	-0,00219	-0,00557	-0,01024
0,8		0,00000	-0,00196	-0,00489	-0,00974

- **Constrained minimization**

$$\min_p J(p) = \int f(X_{mes} - X(p))$$

subject to :

$$\dot{X}(p) = \begin{bmatrix} V_x \\ V_y \\ F_{aeroX}/m \\ F_{aeroY}/m - g \\ \omega \\ M_{aero}/I_z \end{bmatrix}$$

Dimension of parameters = around 1000=4*50*50

➔Use adjoint techniques

➔Manual
➔ AD

- **Gradients**

➔ Compute $\dfrac{\nabla j}{\nabla p}$ **Adjoint techniques + Optimizer**

- **Gradient computation**

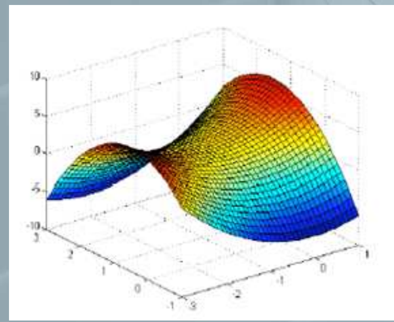
  - Adjoint / Gradients obtained **manually** by Optimal control

$$\text{Lagrangian } L(p, X, \lambda) = J(X) + \left\langle \lambda, \dot{X} - F_p(X) \right\rangle$$

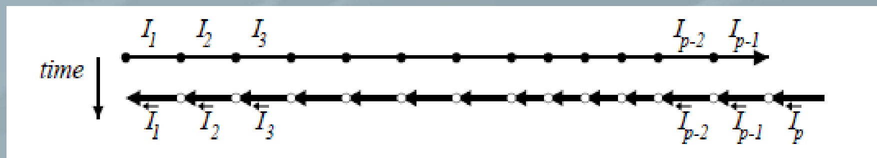
$$\frac{\partial L}{\partial X} = 0 \quad \text{to obtain backward adjoint system in } \lambda$$

$$\dot{\lambda} - \frac{\partial F_p(X)}{\partial X} = -\frac{\partial J(X)}{\partial X}$$

$$\Rightarrow \frac{\nabla j}{\nabla p} = \left\langle \lambda, -\frac{\partial}{\partial p} F_p(X) \right\rangle$$



  - AD : Adjoint of time dependent "instructions of Fortran 77 trajectory code : **automatic**

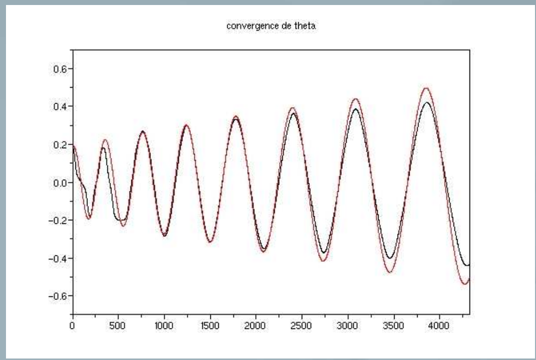


➔ **TAPENADE➔ mode REVERSE**

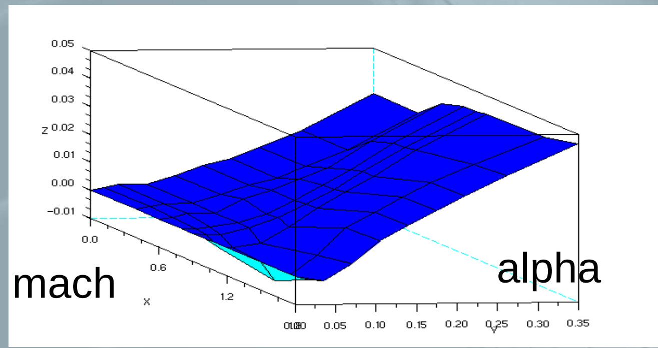
- **OPTIMIZATION**

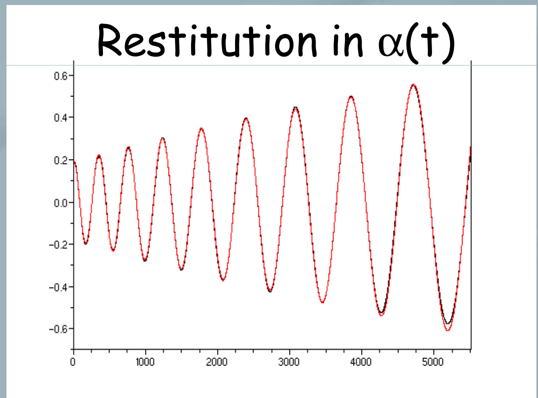
  - **Quasi Newton** / SQP DONLP2: Optimizer from Prof. Spellucci (Darmstadt University),

- **Angle of attack measurements / simulation**
- **Identification of Aerodynamic moment**

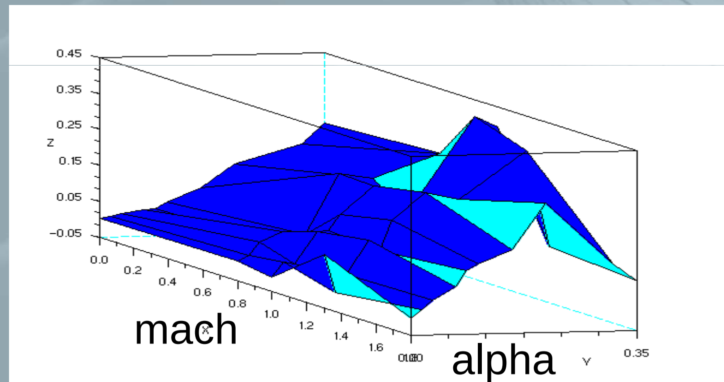


A priori





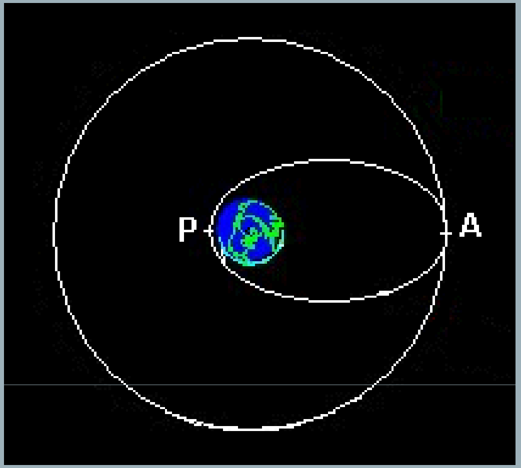
Optimized



- **Good results :**
  - **Aerodynamic coefficients obtained consistent with trajectory physics**
  - **Good strategy for inversion : Time progressive continuation + regularization**
- **Necessity of efficient / powerful large scale optimizers (number parameters > 5000)**

II OPTIMAL COMMAND FOR TRAJECTORY

E.Leibenguth, V.Srithammavanh, S.Alestra, M.Cerf (EADS), F.Dubois (CNAM)



Command Parameter p (t)

Cost j(p)

$$j(p,t_f) = J_f(t_f, U(t_f))$$

State Equation

$$\frac{\partial U}{\partial t} = f(t, U, p)$$
$$U(t=0) = 0$$

State Constraint Equation
+ Bounded parameters

$$C(t, p, U) \leq 0$$

Constraint at final time

$$\psi_f(t_f, U(t_f)) = 0$$

Non Linear Programming (NLP)

Constrained Optimization

Optimal trajectories

Cost j(p)

Maximize the payload given tf

Command Parameter p (t)

Payload m_{cu}

Vertical launch time tv

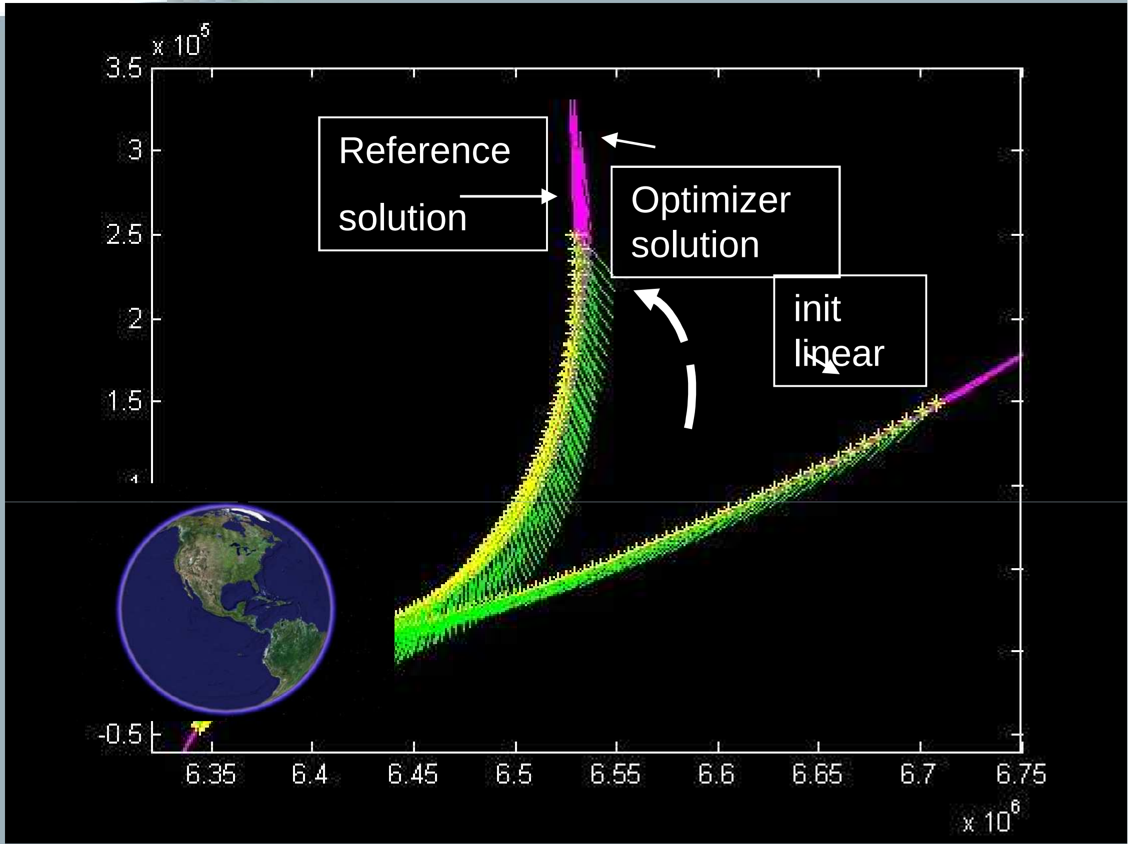
Switching time γ_b

Azimuth Az_0

Dimension input p=20

For each stage i :

- Initial attitude $\theta_{i,1}$
- Final attitude $\theta_{i,2}$
- Balistic time $t_{bal\ i}$



Equality Constraints

Final apogee, perigee

Inequality Constraints

Maximal Dynamic Pressure, Inclination

Use Automatic Differentiation to compute

→ gradients of cost

→ gradient of constraints

NLP optimizer

Check gradient cost function (payload)

```
COUT(0)           -4819.500
COUTT( EPS   )   -4824.320
COUTT( -EPS  )   -4814.681
DIFF  = COUT( EPS  ) - COUT(0)                      -4.819824
DIFF CENTREE = (COUT(EPS) -COUT(-EPS) )/2          -4.819580
PERT LINEAIR = GRAD * EPS                           -4.819500
```

Check Gradient apogee constraint

```
COUT(0)          -4.1942146E-02
COUTT( EPS   ) -4.3294031E-02
COUTT( -EPS  ) -4.0705111E-02
DIFF  = COUT( EPS  ) - COUT(0)                  -1.3518855E-03
DIFF CENTREE = (COUT(EPS) -COUT(-EPS) )/2 -1.2944601E-03
PERT LINEAIR = GRAD * EPS                       -1.3518754E-03
```

Check Gradient perigee constraint

```
COUT(0)           0.1923415
COUTT( EPS   )   0.1969594
COUTT( -EPS  )   0.1876415
DIFF  = COUT( EPS  ) - COUT(0)                      4.6179295E-03
DIFF CENTREE = (COUT(EPS) -COUT(-EPS) )/2          4.6589375E-03
PERT LINEAIR = GRAD * EPS                           4.8509836E-03
```

➔ **OK to include in NLP optimizers**

III Inverse method for non linear ablative thermics

S.Alestra, J.Collinet (EADS), F.Dubois (CNAM)

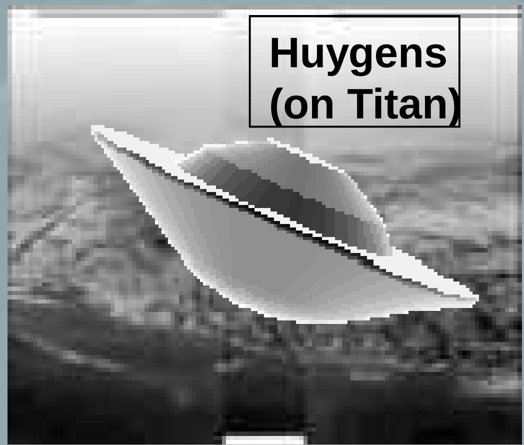
40 Th AIAA Thermophysics, Seattle (June 08)
International Journal of Engineering Systems Modelling and Simulation (IJESMS) 2009

Atmospheric re-entry missions

➔design and sizing of the Thermal Protection System (TPS)

➔ the identification of heat fluxes is of great industrial interest

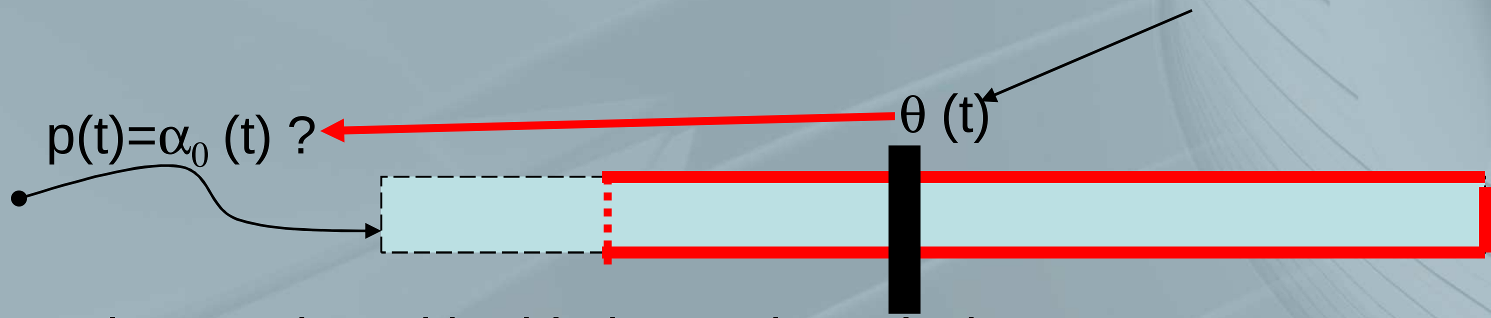
ARD



Huygens (on Titan)

« Monopyro » one dimensional numerical tool

- heat fluxes from temperature measurements ?



on thermal protection with ablation and pyrolysis

- ARD post-flight analysis
- Higher fluxes (fast reentry)
- TPS with higher ablation rates

- In 2007: first tests of AD on MONOPYRO Fortran code

Direct Problem

p Heat Flux

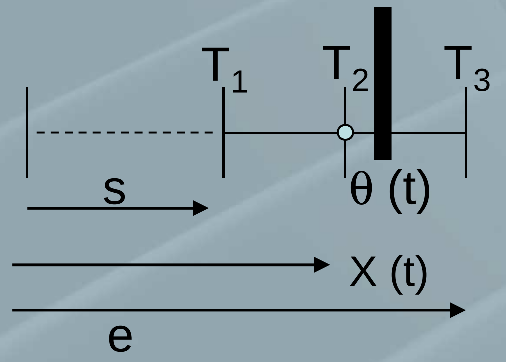
$$W = \begin{pmatrix} T \\ s \end{pmatrix}$$ vector of temperature T and ablation s, functions of time *t* and position *x*.

:

$$\frac{dW}{dt} = F(W, p)$$

$$T(x,0) = T_0 \qquad s(x,0) = 0$$

$$t \in [0, t_f], \; x \in [s(t), e]$$



- System is rewritten in reduced variables (t, ξ) $\qquad \xi \in [0,1]$

$$x = (1 - \xi)s(t) + \xi e$$

Direct Discrete scheme

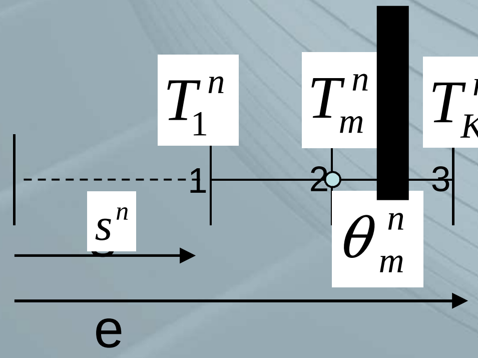
- K grid points, N time iterations in the numerical scheme

- The equation is written at time *(n+1) :*

$$w^n = \left(T_1^n, T_2^n, \cdots, T_K^n, s^n\right)$$

$$\frac{w^{n+1} - w^n}{\Delta t} = f\left(w^{n+1}, p\right)$$

$$w^0 = 0 \qquad 0 \le n \le N$$



- Linearization at time n ➔ forward time discrete linearized Euler scheme, stability

$$\frac{w^{n+1} - w^n}{\Delta t} = f\left(w^n, p\right) + (df)\left(w^n, p\right)\left(w^{n+1} - w^n\right)$$

$$w^0 = 0 \qquad 0 \le n \le N$$

Cost Function

- $p = (p^1, \ldots, p^N)$ time domain unknown heat flux convection coefficient
- Quadratic error or cost function j(p)

$$J(p) = J\underbrace{\left(w^1(p), \ldots, w^N(p)\right)}_{\text{variables } W} = \sum_{n=1}^{N} \left(T_m^n - \theta_m^n\right)^2 \Delta t$$

- Measured temperature θ_m^n
- Computed temperature T_m^n

➔ we need the derivatives of J(p), with respect to p.

➔ p is large scale input parameter = 2000 ➔ Need adjoint reverse mode

Adjoint System

- Adjoint variable $\varphi^{n+1/2}$: dual multiplyer of w^n
- Lagrangian L + calculus of variations

$$L(p,w,\varphi) = L\left(\underbrace{p^1,\dots,p^N}_{parameter\ p},\underbrace{w^1,\dots,w^N}_{variables\ w},\underbrace{\varphi^{1/2},\dots,\varphi^{N+1/2}}_{adjoint\ variables\ \varphi}\right)$$

$$= \sum_{n=1}^{N}\left(T_m^n - \theta^n\right)^2 \Delta t + \sum_{n=0}^{N-1}\left\langle \varphi^{n+1/2}, \frac{w^{n+1}-w^n}{\Delta t} - f\left(w^n,p\right) - (df)\left(w^n,p\right)\left(w^{n+1}-w^n\right)\right\rangle$$

- Cancel the variations of δL with respect to δφ ➔ Direct system, forward in time
- Cancel the variations of δL with respect to δw ➔ Adjoint system, backward in time

$$\frac{\varphi^{n-1/2}-\varphi^{n+1/2}}{\Delta t} = df^t\left(w^{n-1},p\right)\varphi^{n-1/2} + \left[\left(d^2 f\right)\left(w^n,p\right)\left(w^{n+1}-w^n\right)\right]\varphi^{n+1/2} + 2\left(T_m^n - \theta_m^n\right)^2 \Delta t$$

$$\varphi^{N+1/2} = 0 \qquad N \geq n \geq 0$$

Gradient computation (manually)

- With this particular choice of φ, the gradient of the cost function is simply obtained by :

$$\nabla J = \frac{\partial J}{\partial p} = \frac{\partial L}{\partial p}$$

- Variations δL function of δp ➔ discrete gradients

$$\frac{\partial J}{\partial p} = \sum_{n=0}^{N-1} \left\langle \varphi^{n+1/2}, -\frac{\partial f}{\partial p}\left(w^n\right) - \frac{\partial df}{\partial p}\left(w^n\right)\left(w^{n+1} - w^n\right) \right\rangle$$

➔ Test on AD

Gradient : test with Automatic Differentiation Tapenade

- Direct problem instruction

$$w_i^{n+1} = w_i^n + f_i^n(w_i^n, \cdots, w_j^n, w_i^{n-k}, \cdots, w_j^{n-k}, t, p)$$

time →

- Cost Function

$$J = J\underbrace{\left(w^1(p), \ldots, w^N(p)\right)}_{\text{variables } W} = \sum_{n=1}^{N} \left(T_m^n - \theta_m^n\right)^2 \Delta t$$

- Differentiation in reverse mode, with push, pop

$$\varphi_i^n = \varphi_i^{n+1} + \sum_k \sum_j \frac{\partial f_j^k(w_i^n, \cdots, w_j^n, w_i^{n-k}, \cdots, w_j^{n-k}, t, p)}{\partial w_i^{n-k}} \varphi_j^{n+k} - \frac{\partial J}{\partial w_i^{n-k}}$$

← time

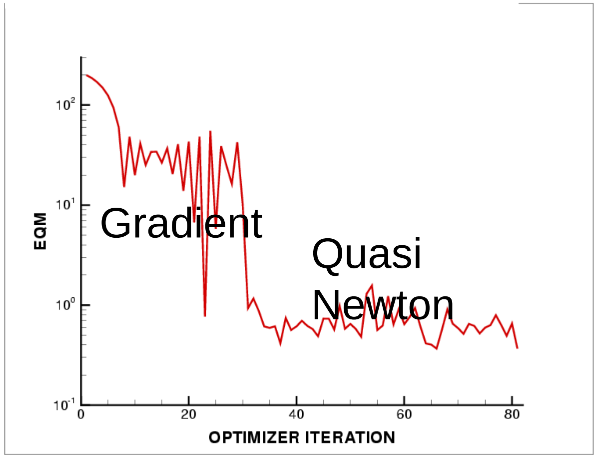
- Gradient computed by reverse mode

$$\frac{\partial J}{\partial p} = \sum_{n=0}^{N-1} \left\langle \varphi^{n+1/2}, -\frac{\partial f}{\partial p}\left(w^n\right) - \frac{\partial df}{\partial p}\left(w^n\right)\left(w^{n+1} - w^n\right) \right\rangle$$

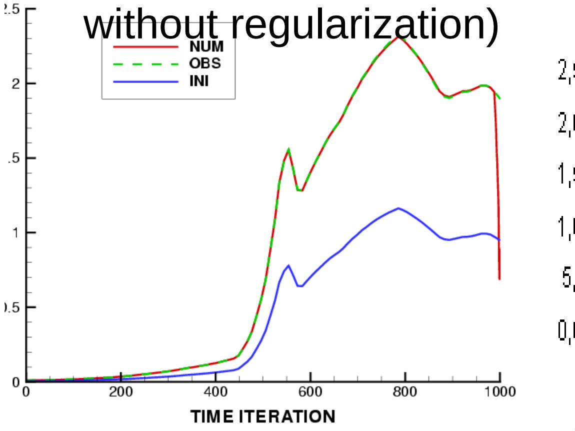
Carbon/Resin with ablation, pyrolysis

- Results OK with pyrolysis and ablation (without and with AD)
- Results OK with 2% noise on pseudo measurement
- Tichonov regularization to stabilize the solution

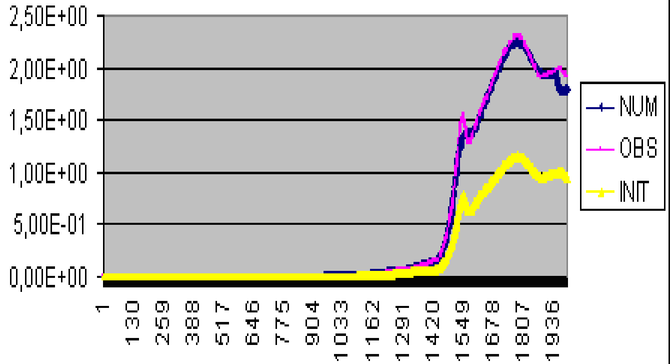
Cost Function



Convection (noise without regularization)

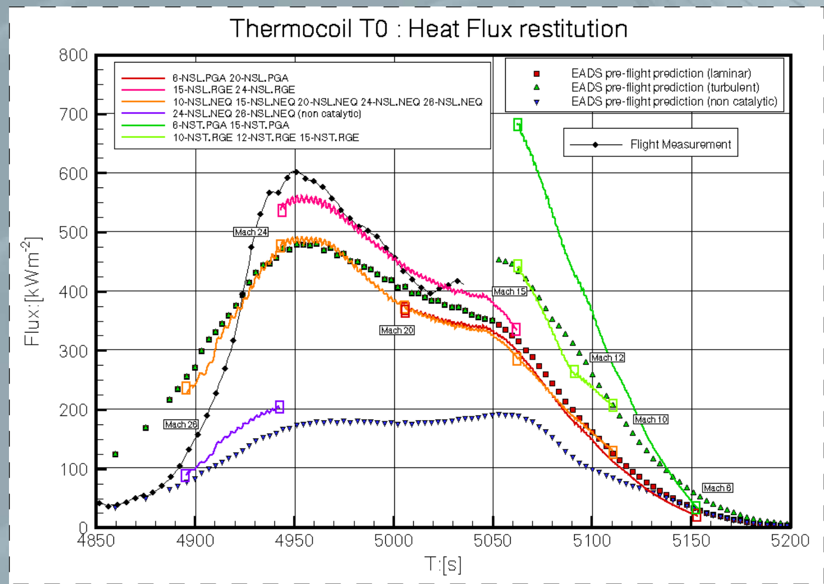


Convection (noise with regularization)



ARD

- First use of the inverse method for « in-flight » rebuilding during ARD post-flight analysis
- Last improvements of the method OK



Recent evolutions of MONOPYRO direct code (2009/2010)

N.Dechampvallins, H.Sacilotto, S.Alestra, V.Srithammavanh (EADS)

- Temperature, ablation, mass flow
- Multi layers
- Multi sensors
- Adapative grid in space & time
- At each time, non linear equation to solve F(U,p,t)=0

  ⇒ Newton method with Fkinsol library
- Complexity of the Fortran code : common, interpolation tables switch, static array declaration, many imbricated routines

➔ Tapenade has been tried for faisability study of inverse problem with gradients to compute

Thermal code : AD tangent mode

EADS

p heat flux **M space, N time**

U temperature + ablation + volumic mass

$$p = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ \vdots \\ p_P \end{pmatrix} \qquad U_1 = \begin{pmatrix} U_1^1 \\ U_1^2 \\ \vdots \\ \vdots \\ U_1^M \end{pmatrix} \quad \cdots \quad U_N = \begin{pmatrix} U_2^1 \\ U_2^2 \\ \vdots \\ \vdots \\ U_2^M \end{pmatrix}$$

implicit solver instructions

$$F_1(U_1, p) = 0$$
$$F_2(U_2, p) = 0$$
$$\vdots$$
$$F_N(U_N, p) = 0$$

$$j(p) = J(U) = \sum_{i=1}^{N} \left\| U_i - U_{obs\,i} \right\|^2$$

Linearization around solution U

Tangent mode

$$\frac{\nabla j}{\nabla p} = -\sum_{i=1}^{N} \frac{\partial J}{\partial U_i} \underbrace{\left(\frac{\partial F_i}{\partial U_i} \right)^{-1} \left(\frac{\partial F_i}{\partial p} \right)}_{-\frac{\partial U_i}{\partial p}}$$

$$A_i = \left(\frac{\partial F_i}{\partial U_i} \right)$$ **Matrix M*M**

$$B_i = \left(\frac{\partial F_i}{\partial p} \right)$$ **Matrix M*P**

$$C_i = \left(\frac{\partial J}{\partial U_i} \right)$$ **Matrix 1*M**

Tapenade on recent MONOPYRO direct code (2009/2010)

- Tangent mode (linearization) is OK, but hard to obtain !
  - Complexity of the code, duplication of arrays with differentiation
  - CPU and memory constraints
  - First results are promising but costly (number of parameters)

- Adjoint mode is under development for multi parameters
  - Use reverse differentiation + mathematic adjoint algorithms
  - Problem of storing and memory stack at backward sweep

➔ Tapenade has given good faisability results but still work to do !!

Conclusion / Perspectives

- Premières applications prometteuses de la DA (Tapenade) à des codes inverses métiers trajectoires et thermiques EADS
- Nombreuses autres applications possibles (calcul sensibilités, optimisation forme)
- Outil précieux si calcul gradients fastidieux à la main
- Travail sur solveurs non linéaires / instructions non explicites en cours
- Calcul du Hessien avec la DA ➔ INRIA
- Declaration des tableaux, allocations, allocation dynamique
- Conseils, bonnes pratiques méthodologiques de la DA ➔ INRIA
- Tapenade Fortran 95, Langage C ?
- Checkpointing ? ➔ INRIA
- Autres outils DA : ADIFOR (US, Fortran), ADOL-C, Matlab