Automatic Differentiation-based perturbation methods for uncertainties and errors

Anca Belme, Massimiliano Martinelli, Laurent Hascoët, Valérie Pascual, Alain Dervieux
Tropics project
http://www-sop.inria.fr/tropics/
Overview:

1. Progress in Automatic Differentiation,

2. Numerical error reduction,

1. Progress in Automatic Differentiation

The TAPENADE program differentiator: TAPENADE (115 000 lines of JAVA) transforms a FORTRAN program into a FORTRAN program which computes the derivative of the given FORTRAN program. TAPENADE is organised around an intermediate step, the analysis and differentiation of an abstract imperative language.

http://www-sop.inria.fr/tropics/

Overall Architecture of TAPENADE
Differentiability/Differentiation modes

Direct/Tangent mode:
- Differentiated code computes \((c, \delta c) \mapsto j'(c).\delta c\)
- Gâteaux derivative of a functional
- Computational cost factor: \(\alpha_T \approx 4\)
- Chain rule applies in same order as initial code
- Does not store intermediate variables

Backward/Reverse mode:
- Differentiated code computes \((c, \delta j) \mapsto (j'(c))^*.\delta j\)
- Gradient of a functional
- Computational cost factor: \(\alpha_R \approx 5\)
- Chain rule applies in reverse order to initial code.
- Needs storing intermediate variables in a first direct sweep.
Select the input language:

- given by the suffix of the files: Fortran 77, Fortran 95, C

Upload source and include files, repeatedly or copy paste your fortran source program.

Type the file path in, or browse:

and upload it

[ ] as a source [ ] as an include

Name of the top routine:

Dependent output variables (separator: white space, default: all variables):

Independent input variables (separator: white space, default: all variables):

(Optional) For our records, could you please give us your name and the application you have in mind (it can very well be only "test"):

Differentiate in

[ ] Tangent Mode [ ] Tangent Multidirectional Mode [ ] Reverse Mode
Find $W_h$ such that $|\Psi_h(W_h)| \leq \varepsilon$

If the iteration for $W_h$ is stopped by a test with $\varepsilon > 0$, then $W_h$ depends on initial conditions of solution algorithm, in a similar manner to an unsteady system. Also $W_h$ is only piecewise differentiable. It then is possible to apply R-AD to the whole programme, including iterative solution algorithms. Due to dependance to intermediate states they need be stored (or recomputed). For this case and for unsteady problems, efficient strategies of storage or recomputations can be introduced by user by means of “check point” controls.

Assuming $\varepsilon = 0$, state $W_h$ does not depend on initial conditions of solution algorithm and is differentiable with respect to $c_h$. The iteration should be pointed as “independant” or parallel in order to avoid unnecessary storage of intermediate values.
1. Progress in Automatic Differentiation

   The TAPENADE program differentiator, recent advances:
In its Version 3(alpha), TAPENADE takes into account user directives inserted into sources.

Directive II-LOOP allows for a more efficient reverse differentiation of parallel loops by inhibiting storages useless in “parallel” case.

Directive NOCHECKPOINT can be used for optimal checkpointing in reverse differentiation. TAPENADE lets the user specify finely which procedure calls must be checkpointed or not.

TAPENADE V3 has been extended to deal with Fortran95 and with ANSI C. TAPENADE is optimised for successive reverse differentiations.
2. Numerical error reduction

Abstract representation of the Partial Differential Equation:

\[ \Psi(W) = 0. \]

Discretisation of the PDE:

\[ \Psi_h(W_h) = 0 \in \mathbb{R}^N \]
\[ W_h \in \mathbb{R}^N, \ W_h = [(W_h)_i]. \]

Operators between continuous and discrete:

\[ R_h : \mathbb{R}^N \rightarrow V \subset L^2(\Omega) \cap C^0(\overline{\Omega}) \]
\[ T_h : V \rightarrow \mathbb{R}^N \]

\[ W_h(x, y, z) = (R_h W_h)(x, y, z) \quad ; \quad W - W_h = \text{small? corrector?} \]
2. Numerical error reduction

**A priori estimate:**

\[ \Psi_h(W) - \Psi_h(W_h) = \Psi_h(W) \Rightarrow W - W_h \approx \left[ \frac{\partial \Psi_h}{\partial W_h} \right]^{-1} \Psi_h(W), \]

in practice:

\[ \delta W_h = R_h \left[ \frac{\partial \Psi_h}{\partial W_h} \right]^{-1} (\Psi_h - T_h \Psi)(W_h). \]

**A posteriori estimate:**

\[ \Psi(W) - \Psi(W_h) = -\Psi(W_h) \Rightarrow W - W_h \approx -\left[ \frac{\partial \Psi}{\partial W} \right]^{-1} \Psi(W_h), \]

in practice:

\[ \delta W_h = -R_h \left[ \frac{\partial \Psi_h}{\partial W_h} \right]^{-1} T_h \Psi(R_h W_h), \]

quadratures formulas can be used for \( \Psi(R_h W_h). \)
2. Numerical error reduction

Goal-oriented error:

\[ j(W) = (g, W)_{L^2(\Omega)} \text{ s.t. } \Psi(W) = 0 \]
\[ (\frac{\partial \Psi}{\partial W})^* p = g \]

\[ j_h = (g, R_h W_h)_{L^2(\Omega)} \text{ s.t. } \Psi_h(W_h) = 0 \]
\[ g_h = T_h g \]
\[ (\frac{\partial \Psi_h}{\partial W_h})^* p_h = g_h \Leftrightarrow p_h = [\frac{\partial \Psi_h}{\partial W_h}]^{-*} T_h g_h \]
\[ p_h = R_h p_h . \]

A fundamental assumption of the present analysis is that this discrete adjoint is a good enough approximation of continuous adjoint.
2. Numerical error reduction

*A posteriori* goal-oriented errors and correctors

**Field corrector**

\[
\delta W_h = -R_h \left[ \frac{\partial \Psi_h}{\partial W_h} \right]^{-1} T_h \Psi(W_h)
\]

**Direct-linearized goal-oriented corrector**

\[
\delta_1 j = - \left( g , R_h \left[ \frac{\partial \Psi_h}{\partial W_h} \right]^{-1} T_h \Psi(W_h) \right)_{L^2(\Omega)}
\]

**Adjoint-based goal-oriented correctors (Giles-Pierce)**

\[
\delta_1 j = - \left( p_h , T_h \Psi(W_h) \right)_{L^2(\Omega)}
\]
2. Numerical error reduction

An equivalence:

Assuming that we have chosen:

\[ T_h = R^*_h, \]

then direct-linearised and adjoint-based correctors are identical

\[- \left( g , R_h \left[ \frac{\partial \Psi_h}{\partial W_h} \right]^{-1} T_h \Psi(W_h) \right)_{L^2(\Omega)} = - \left( \left[ \frac{\partial \Psi_h}{\partial W_h} \right]^{-*} R^*_h g , T_h \Psi(W_h) \right)_{R^N} \]

\[ = - \left( T^*_h \left[ \frac{\partial \Psi_h}{\partial W_h} \right]^{-*} R^*_h g , \Psi(W_h) \right)_{L^2(\Omega)} \]
2. Numerical error reduction

A preliminary numerical example
3D Euler flow around a wing with NACA0012 section.
1. Direct calculation
2. Direct calculation + Correction
   CPU is 30% larger, error on entropy deviation is 10 times smaller.

Entropy spurious generation for a direct computation of a steady flow and for a corrected one.
2. Numerical error reduction by correction, conclud’d

Which *A posteriori* goal-oriented errors and correctors

Direct-linearized goal-oriented corrector

\[ \delta_{1j} = - \left( g, R_h \left[ \frac{\partial \Psi_h}{\partial W_h} \right]^{-1} T_h \Psi(W_h) \right)_{L^2(\Omega)} \]

Adjoint-based goal-oriented correctors

\[ \delta_{1j} = - (p_h, T_h \Psi(W_h))_{L^2(\Omega)} \]

Adjoints, as compared with linearised perturbations, are *slightly* more complex to compute for steady PDE, and *much* more complex for unsteady PDE’s (despite TAPENADE is well adapted for this exercice). For computing correctors and for several mesh adaptation applications, the “Direct-linearized goal-oriented corrector” can be used for an identical result.
2. *A priori* numerical error reduction, mesh adaption

Minimizing P1-Interpolation error

(Loseille, PhD, Paris VI, 2008)

\[ ||u - \pi_M u||^2 = \int \left( |\frac{\partial^2 u}{\partial \xi^2}|.m_\xi^2 + |\frac{\partial^2 u}{\partial \eta^2}|.m_\eta^2 \right)^2 dxdy \]

where \( \xi \) and \( \eta \) are directions of diagonalization of the Hessian of \( u \).

\[ \min_{\mathcal{M}} \mathcal{E}_M \]

under the constraint \( N_M = N \).

This can be solved analytically.
2. *A priori* numerical error reduction, mesh adaption

**Optimal Metric for interpolation**

\[
\mathcal{M}_{opt} = \frac{C}{N} \mathcal{R}^{-1} \begin{pmatrix}
|\frac{\partial^2 u}{\partial \eta^2}|^{-5/6} & \frac{\partial^2 u}{\partial \xi^2}^{1/6} & 0 \\
0 & |\frac{\partial^2 u}{\partial \xi^2}|^{-5/6} & \frac{\partial^2 u}{\partial \eta^2}^{1/6}
\end{pmatrix} \mathcal{R} . \tag{1}
\]

with: \(C = \int \left( |\frac{\partial^2 u}{\partial \xi^2}| \cdot |\frac{\partial^2 u}{\partial \eta^2}| \right)^2 \, dxdy\).

But this only minimises an interpolation error...
2. *A priori* numerical error reduction, mesh adaption

**Approximation a priori error** (with Alauzet & Loseille)

Steady Euler equations:

\[
W \in V = \left[ H^1(\Omega) \right]^5, \forall \phi \in V,
\]

\[
(\Psi(W), \phi) = \int_{\Omega} \phi \nabla \cdot \mathcal{F}(W) \, d\Omega - \int_{\Gamma} \phi \hat{\mathcal{F}}(W) \cdot \mathbf{n} \, d\Gamma = 0
\]

where \( \Gamma = \partial \Omega \) and \( \hat{\mathcal{F}} \) is B.C. fluxes.

Mixed-Element-Volume appoximation:

\[
\forall \phi_h \in V_h, \quad \int_{\Omega_h} \phi_h \nabla \cdot \Pi_h \mathcal{F}(W_h) \, d\Omega_h - \int_{\Gamma_h} \phi_h \Pi_h \hat{\mathcal{F}}(W_h) \cdot \mathbf{n} \, d\Gamma_h =
\]

\[
- \int_{\Omega_h} \phi_h D_h(W_h) \, d\Omega_h,
\]

where \( \Pi_h \) is the usual elementwise linear interpolation and where \( D_h \) holds for a numerical dissipation term.
2. *A priori* numerical error reduction, mesh adaption

**A priori adjoint-based** error estimate:

\[(g, W - W_h) \approx ((\Psi_h - \Psi)(W), P), \text{ with } \left[ \frac{\partial \Psi}{\partial W} \right]^* P = g,\]

the optimal mesh is obtained after some transformations by solving:

\[
\text{Find } \mathcal{M}_{\text{opt}} = \text{Argmin}_{\mathcal{M}} \int_{\Omega} |\nabla P| |\mathcal{F}(W) - \pi_{\mathcal{M}} \mathcal{F}(W)| d\Omega
+ \int_{\Gamma} |P| (\bar{\mathcal{F}}(W) - \pi_{\mathcal{M}} \bar{\mathcal{F}}(W)).n| d\Gamma
\]

under the constraint \( C(\mathcal{M}) = N. \)

**Solved analytically as interpolation case.**

Remark: The adjoint-based formulation is compulsory.
Application to sonic boom: Hessian-based (Mach L2)
Goal-oriented + Hessian-based ("foot print" funct.)
Pressure track at ground
A differentiation problem:

Given a computer program computing a functional $j(c)$ for $c \in \mathbb{R}^n$

$$j(c) = J(c, W(c)),$$

such that $\Psi(c, W) = 0$

we want, applying a source-to-source programme differentiation software, viz. TAPENADE, to get a computer program computing the second derivatives:

$$\frac{d^2 j}{dc^2} = (H_{ii}).$$

The solution $W(c) \in \mathbb{R}^N$ of state equation $\Psi(c, W) = 0$ is supposed here to be steady-state. It can be obtained by explicit or implicit pseudo-time advancing techniques. Application to an unsteady problem can be treated in a similar manner, but with important extra difficulties.
Why do we need Hessian?

Perturbative methods for uncertainty propagation:
- Method of Moments

\[ \mu_j \approx j(\mu_c) + \frac{1}{2} \sum_i H_{ii} \sigma_i^2 \]
\[ \sigma_j^2 \approx \sum_i G_i^2 \sigma_i^2 + \frac{1}{2} \sum_{i,k} H_{ik} \sigma_i^2 \sigma_k^2 \]

Robust optimization:
- Gradient-based methods for \( j^{\text{robust}} (c) = j(c) + \varepsilon || \frac{dj}{dc} || \)
- Gradient-free methods for \( j^{\text{robust}} (c) = j(c) + \frac{1}{2} \sum_i H_{ii} \sigma_i^2 \)
- Gradient-free methods for \( j^{\text{robust}} (c) = j(c) + k \sigma_j^2 \)

Adjoint-corrected functionals:
- Gradient-based methods for \( j^{\text{corr}} (c) = j(c) - \langle \Psi_{\text{ex}}(c, W), \Pi_0 \rangle \)
Second derivative: Tangent-on-Tangent approach

\[
\frac{d^2 j}{dc_i dc_k} = -\Pi^*_0 D^2_{i,k} \Psi + D^2_{i,k} J
\]

with the adjoint \( \Pi_0 \) and \( \theta_k = \frac{dW}{dc_k} \) solutions of:

\[
\left( \frac{\partial \Psi}{\partial W} \right)^* \Pi = \left( \frac{\partial J}{\partial W} \right)^*
\]

\[
\frac{\partial \Psi}{\partial W} \frac{dW}{dc_k} = -\frac{\partial \Psi}{\partial c} e_k
\]

and:

\[
D^2_{i,k} J = \frac{\partial}{\partial c} \left( \frac{\partial J}{\partial e_i} e_i \right) e_k + \frac{\partial}{\partial W} \left( \frac{\partial J}{\partial e_i} \right) \theta_k + \frac{\partial}{\partial W} \left( \frac{\partial J}{\partial c} e_i \right) \theta_i + \frac{\partial}{\partial W} \left( \frac{\partial J}{\partial W} \theta_i \right) \theta_k
\]

\[
D^2_{i,k} \Psi = \frac{\partial}{\partial c} \left( \frac{\partial \Psi}{\partial e_i} e_i \right) e_k + \frac{\partial}{\partial W} \left( \frac{\partial \Psi}{\partial e_i} \right) \theta_k + \frac{\partial}{\partial W} \left( \frac{\partial \Psi}{\partial c} e_i \right) \theta_i + \frac{\partial}{\partial W} \left( \frac{\partial \Psi}{\partial W} \theta_i \right) \theta_k
\]
Second derivative: Tangent-on-Reverse

\[
\frac{\partial}{\partial c_i} \left( \frac{\partial j}{\partial c} \right)^* = \left( \frac{\partial^2 j}{\partial c^2} \right) e_i = \frac{\partial}{\partial c} \left( \frac{\partial j}{\partial c} \right)^* e_i + \frac{\partial}{\partial W} \left( \frac{\partial j}{\partial c} \right)^* \theta_i + \\
- \frac{\partial}{\partial c} [(\frac{\partial \Psi}{\partial c})^* \Pi_0] e_i - \frac{\partial}{\partial W} [(\frac{\partial \Psi}{\partial c})^* \Pi_0] \theta_i - (\frac{\partial \Psi}{\partial c})^* \lambda_i
\]

with \( \Pi_0 \) solution of the adjoint system

\[
(\frac{\partial \Psi}{\partial W})^* \Pi = (\frac{\partial J}{\partial W})^*
\]

\( \theta_i \) and \( \lambda_i \) solution of the systems

\[
\frac{\partial \Psi}{\partial W} \theta_i = -\frac{\partial \Psi}{\partial c} e_i
\]

\[
(\frac{\partial \Psi}{\partial W})^* \lambda_i = \frac{\partial}{\partial c} \left( \frac{\partial J}{\partial W} \right)^* e_i + \frac{\partial}{\partial W} \left( \frac{\partial J}{\partial W} \right)^* \theta_i \\
- \frac{\partial}{\partial c} [(\frac{\partial \Psi}{\partial W})^* \Pi_0] e_i - \frac{\partial}{\partial W} [(\frac{\partial \Psi}{\partial W})^* \Pi_0] \theta_i
\]
Full Hessian: ToT vs. ToR

Common part:
- Adjoint state \( \left( \frac{\partial \Psi}{\partial W} \right)^* \Pi = \left( \frac{\partial J}{\partial W} \right)^* \)
- \( n \) linear system to solve: \( \frac{\partial \Psi}{\partial W} dW \sum c_i = - \frac{\partial \Psi}{\partial c} e_i \)

Specific parts:
- We assume unitary cost for a single residual evaluation \( \Psi(c, W) \)
- ToT : computation of \( D_{ik}^2 \Psi \) and \( D_{ik}^2 J \) \( \Rightarrow \frac{n(n + 1)}{2} \alpha_T^2 \) \( (1 < \alpha_T < 4) \)
- ToR : \( n \) (adjoint) linear system to solve: \( \left( \frac{\partial \Psi}{\partial W} \right)^* \lambda_i = \dot{J}_W - \dot{\Psi}_W \) \( \Rightarrow \) \( n(n_{\text{iter}} + \alpha_T) \alpha_R \) \( (1 < \alpha_R \leq 5) \)

Use ToT when \( n < \frac{2 \alpha_R(n_{\text{iter}} + \alpha_T)}{\alpha_T^2} \)
Application to reduced order modeling of 3D-Euler flow

Functional to model:
- Inputs: Mach number and angle of attack
- Output: drag coefficient.

Numerical basis:
Upwind vertex-centered finite-volume, implicit pseudo-time advancing with first-order Jacobian
Building the reduced quadratic model

Nonlinear simulations

Taylor 1st order ($\alpha=2.0, M=0.83$)

Taylor 2nd order ($\alpha=2.0, M=0.83$)

Relative Difference: Nonlinear vs. Taylor 2nd order
Comparing the reduced quadratic model with metamodels (*)

<table>
<thead>
<tr>
<th>Metamodel</th>
<th>Points</th>
<th>Expectation</th>
<th>Relative error</th>
<th>Variance</th>
<th>Relative error</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBF</td>
<td>8</td>
<td>6.855 $10^{-3}$</td>
<td>0.029%</td>
<td>1.562 $10^{-7}$</td>
<td>0.579%</td>
</tr>
<tr>
<td>KRG</td>
<td>8</td>
<td>6.853 $10^{-3}$</td>
<td>0.058%</td>
<td>1.551 $10^{-7}$</td>
<td>0.128%</td>
</tr>
<tr>
<td>First-order</td>
<td>8</td>
<td>6.830 $10^{-3}$</td>
<td>0.393%</td>
<td>1.547 $10^{-7}$</td>
<td>0.386%</td>
</tr>
<tr>
<td>Second-order</td>
<td>8</td>
<td>6.860 $10^{-3}$</td>
<td>0.043%</td>
<td>1.558 $10^{-7}$</td>
<td>0.321%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flow solver</th>
<th>First derivatives</th>
<th>Second derivatives</th>
<th>Memory in Mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>first-order Jacobian + Block-diagonal Precond. + work vectors</td>
<td>ILU(1) Precond. + GMRES(200) + differentiated vectors</td>
<td>86+10+35</td>
<td></td>
</tr>
<tr>
<td>ILU(1) Precond. + GMRES(200) + differentiated vectors</td>
<td>170+250+120</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ILU(1) Precond. + GMRES(200) + differentiated vectors</td>
<td>170+250+240</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU time in second</th>
<th>CPU time in second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow solver</td>
<td>403</td>
</tr>
<tr>
<td>Gradient</td>
<td>255</td>
</tr>
<tr>
<td>Hessian</td>
<td>278</td>
</tr>
<tr>
<td>Total</td>
<td>936</td>
</tr>
<tr>
<td></td>
<td>3224</td>
</tr>
</tbody>
</table>

(*) Martinelli-Duvigneau, submitted in a journal.
Concluding remarks

Efficiency oriented AD application

- First and second derivative architecture well identified.
- Optimum use of modern iterative solvers.
- Use reverse mod whenever necessary
- When using reverse mode a better efficiency is obtained by using new TAPENADE functionalities.

Current and future works

- AD current studies concerning MPI efficiency.
- Mesh adaptation, correction and accuracy control for large instationary state systems.