NATIONAL TECHNICAL UNIVERSITY OF ATHENS (NTUA)
SCHOOL OF MECHANICAL ENGINEERING
PARALLEL CFD & OPTIMIZATION UNIT (PCOpt/NTUA)

Gradient-Based Optimisation

Dr. Kyriakos C. Giannakoglou, Professor NTUA

Dr. Varvara Asouti, Adjunct Lecturer NTUA

Contributors:

Dr. E. Papoutsis-Kiachagias

Dr. K. Gkaragkounis

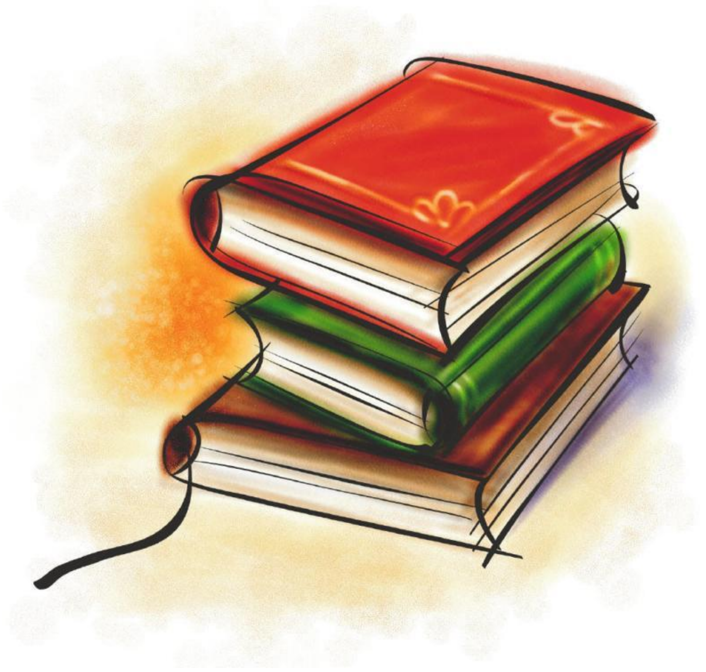
Dr. I. Kavvadias

Dr. A. Zymaris

Dr. K. Tsiakas

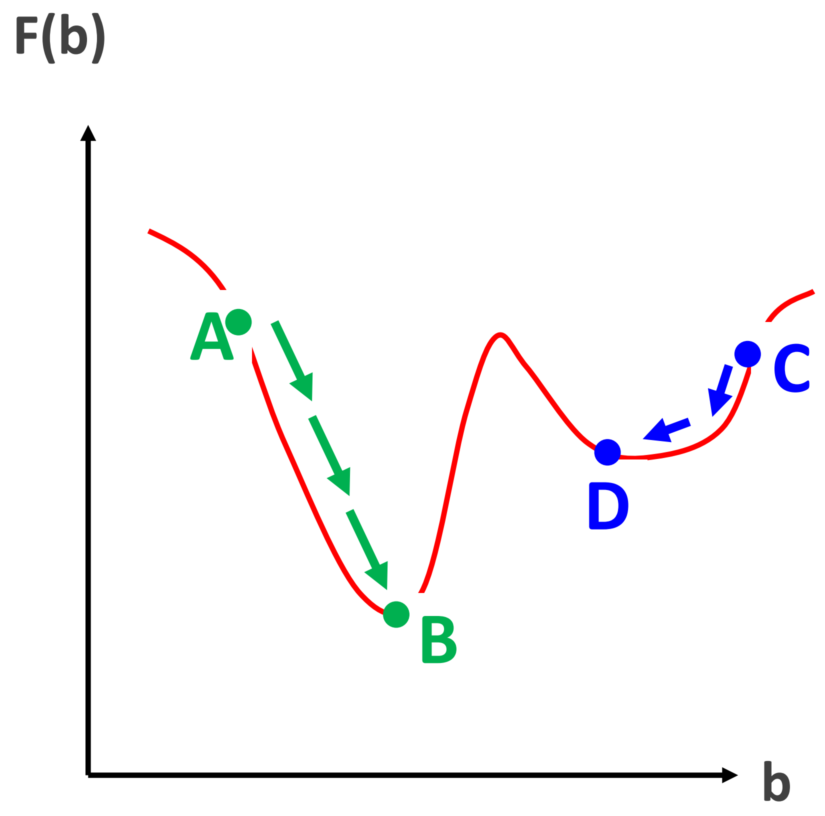
Dr. X. Trompoukis

Dr. M. Kontou



Introduction

Gradient-Based Optimisation Methods (GBMs)

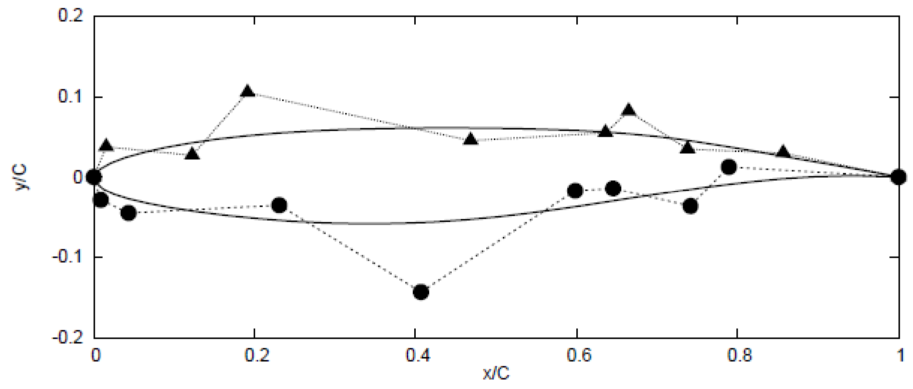


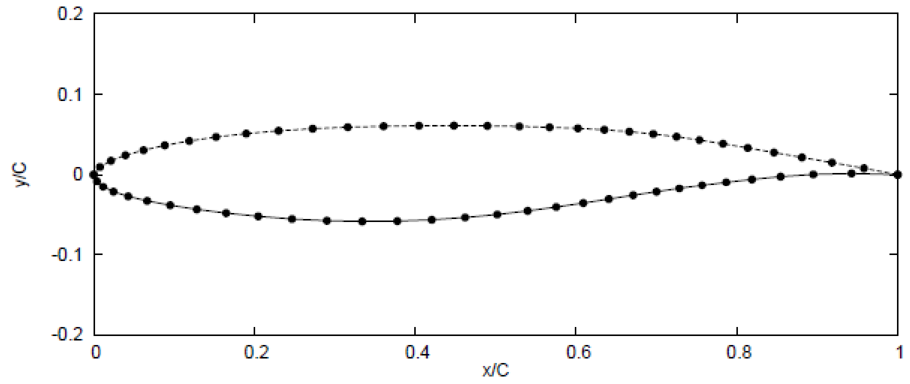
The computation of the gradient of the objective function w.r.t. the design variables is the heart (and the costliest part) of all GBMs!

Gradient-based (GBM) Optimisation – Computing Grad(F)

Aerodynamic Shape Optimisation (ShpO) example

$$\vec{b} = (b_1, b_2, b_3, \ldots, b_N)^T \in R^N$$





$$\nabla F = \left(\frac{\partial F}{\partial b_1}, \frac{\partial F}{\partial b_2}, \cdots, \frac{\partial F}{\partial b_N}\right)$$

$$\nabla F = \left(\frac{\delta F}{\delta b_1}, \frac{\delta F}{\delta b_2}, \cdots, \frac{\delta F}{\delta b_N}\right)$$

$$\nabla F = \left(\frac{dF}{db_1}, \frac{dF}{db_2}, \cdots, \frac{dF}{db_N}\right)$$

Derivatives: *theta, delta or d*?

- **Accuracy of gradients**
- **Computational Cost**
- **Method & S/W Development Time**

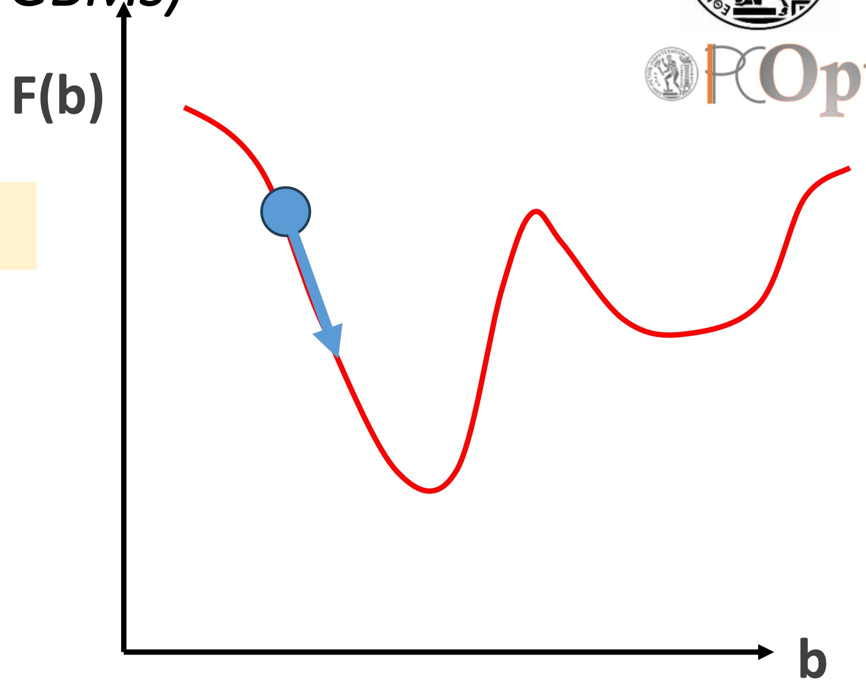
Gradient-based (Deterministic) Optimisation Methods (GBMs)

Steepest Descent

$$\vec{b}^{\,new} = \vec{b}^{\,old} - \eta \nabla F(\vec{b}^{\,old})^T$$

about η

$$b_n^{new} = b_n^{old} - \eta \frac{\delta F}{\delta b_n}^{old} \; , \qquad n = 1, \cdots N$$



Newton's Methods

$$\vec{b}^{\,new} = \vec{b}^{\,old} - \nabla^2 F(\vec{b}^{\,old})^{-1} \nabla F(\vec{b}^{\,old})^T$$

- **Compute (the exact) or approximate grad(F)? Accuracy vs. Computational cost.**
- **Computing grad(F) is the most important part of GBMs.**
- **The computational cost of grad(F) should not scale with N (number of design variables).**

Computing the exact Hessian! Quasi-Newton methods!

Newton's Method

$$F(\overrightarrow{x}^n + \overrightarrow{p}^n) \approx F(\overrightarrow{x}^n) + \overrightarrow{p}^{nT} \nabla F(\overrightarrow{x}^n)$$

Grad $F(\overrightarrow{x}^n + \overrightarrow{p}^n) \rightarrow 0$

$$\nabla F(\overrightarrow{x}^n) + \nabla^2 F(\overrightarrow{x}^n)\ \overrightarrow{p}^n = 0$$

Design variables correction vector: $\vec{x}^{n+1} = \vec{x}^n + \vec{p}^n$

$$\overrightarrow{p}^n = -\left(\nabla^2 F(\overrightarrow{x}^n)\right)^{-1} \nabla F(\overrightarrow{x}^n)$$

$$\overrightarrow{p}^n = -(B^n)^{-1} \nabla F(\overrightarrow{x}^n)$$

Quasi Newton (Approximating Grad(F)) (1/2)

For: $\vec{s}^n = \vec{b}^{n+1} - \vec{b}^n$, $\vec{y}^n = \nabla F\left(\vec{b}^{n+1}\right) - \nabla F\left(\vec{b}^n\right)$

widely used recursive formulas for Hess(F) = B are:

- **SR1 (Symmetric Rank One) Method**

$$B^{n+1} = B^n + \frac{(\vec{y}^n - B^n \vec{s}^n)(\vec{y}^n - B^n \vec{s}^n)^T}{(\vec{y}^n - B^n \vec{s}^n)^T \vec{s}^n}$$

- **BFGS (Broyden-Fletcher-Goldfarb-Shanno) Method**

$$B^{n+1} = B^n - \frac{B^n \vec{s}^n \vec{s}^{nT} B^n}{\vec{s}^{nT} B^n \vec{s}^n} + \frac{\vec{y}^n \vec{y}^{nT}}{\vec{y}^{nT} \vec{s}^n}$$

Attention: the recursive formula should retain the symmetry of the Hessian matrix. Initialising matrix B (B^0; scaled identity matrix?) might be critical.

Quasi Newton (Approximating Grad(F)) (=2/2)

Or, even better, recursively compute the inverse of matrix B (i.e. matrix H):

$$H^n = (B^n)^{-1}$$

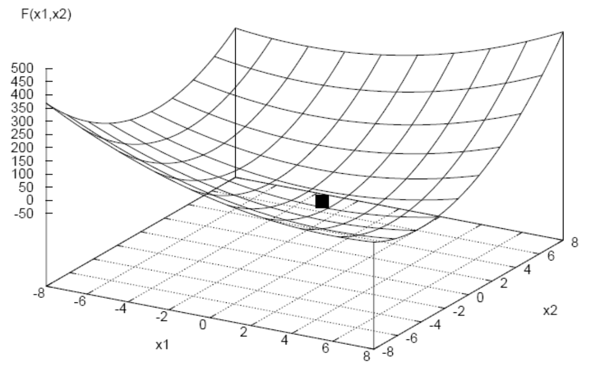
$$H^{n+1} = (I - \rho^n \overrightarrow{s}^n \overrightarrow{y}^{nT}) H^n (I - \rho^n \overrightarrow{y}^n \overrightarrow{s}^{nT}) + \rho^n \overrightarrow{s}^n \overrightarrow{s}^{nT}$$

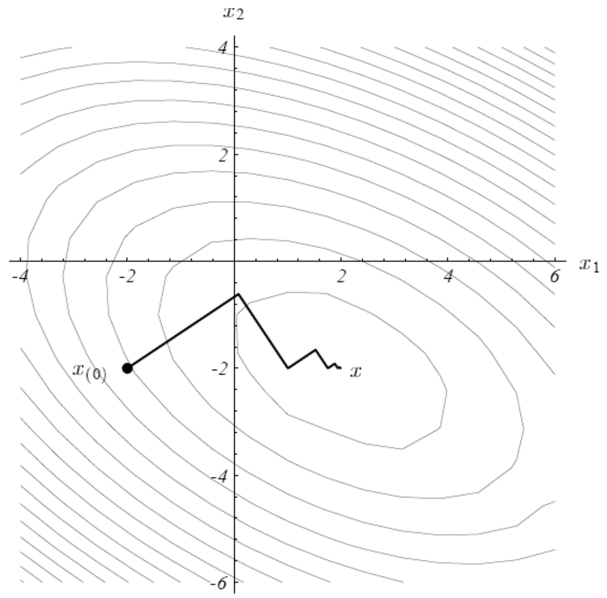
$$\rho^n = \frac{1}{\overrightarrow{y}^{nT} \overrightarrow{s}^n} \qquad \overrightarrow{p}^n = -H^n \nabla F(\overrightarrow{x}^n)$$

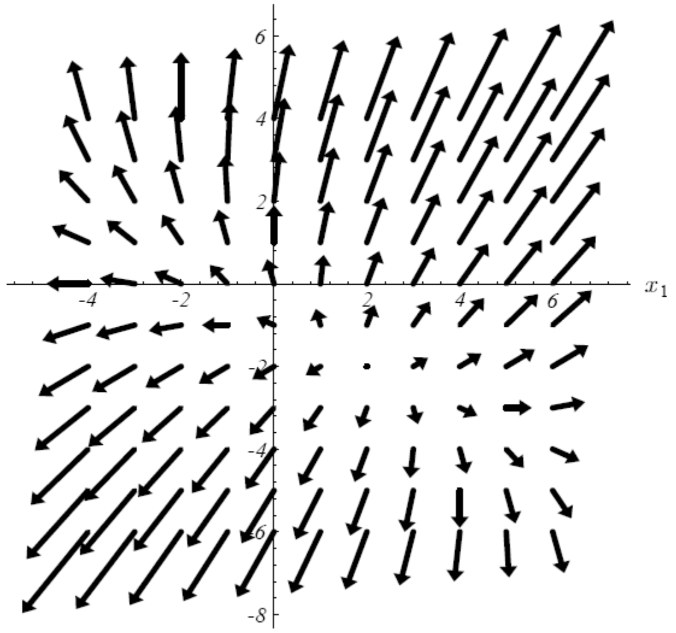
Comments

- ✓ Newton's methods are much faster (in terms of the number of optimisation cycles required to converge) than steepest descent, especially if the starting point is not far from the optimal solution.
- ✓ Requires the Hessian matrix of the objective function F (a NxN symmetric matrix with all second-order derivatives). **Expensive!**
- ✓ Alternatively, second-order derivatives can be approximated, giving rise to **Quasi-Newton methods**.
- ✓ The approximation is based on a recursive formula for the Hessian matrix or (directly) its inverse, which requires grad(F). However, in this case, the exact grad(F) should be available.
- ✓ Close to the optimal solution, the Quasi-Newton method usually becomes quite fast.

Gradient-based Methods (GBMs)







- It is likely to compute a local (rather than the global) optimum.
- Computed solutions heavily depend on the initialisation. Many restarts might help but this increases the cost.
- Expected to be fast, if computing Grad(F) isn't time-consuming.

Important Note

Working with a GBM, the gradients of the objective and constraint functions must be computed.

In contrast:

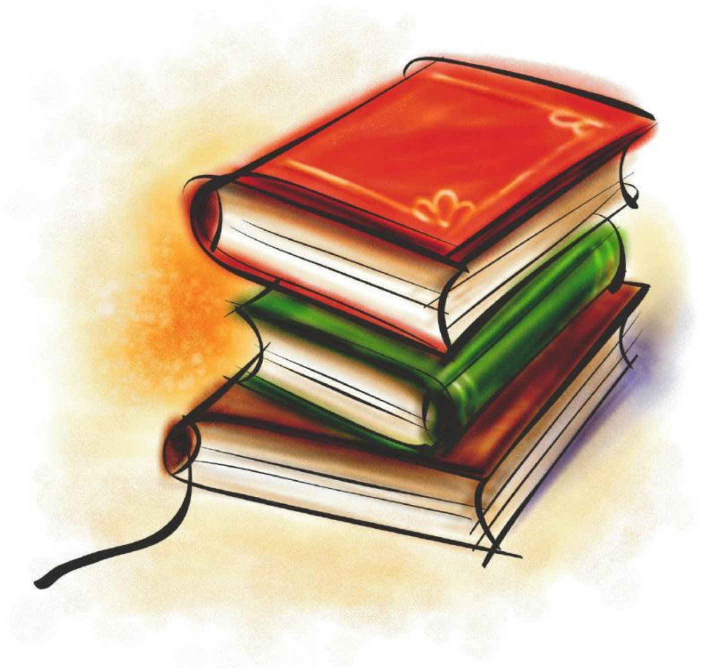
Constraint handling in a GFM (such as an EA) is merely based on (escalated) penalty functions.

Possible Ways to Compute (the exact) Grad(F)

In a problem governed by PDEs (or linear systems of equations)

- **Finite Differences (FD)**
- **Complex Variable (CV) Method**
- **Direct Differentiation (DD)**
- **Automatic or Algorithmic Differentiation (AD)**
- **Adjoint Method (AM) or Adjoint Variable Method (AV)**

The computation of the gradient of the objective and/or constraint functions is the most important and costly part of a GBM. Its cost should be measured in terms of the cost of solving the analysis problem (one time unit, 1 TU or 1 EPS= Equivalent Problem Solution).



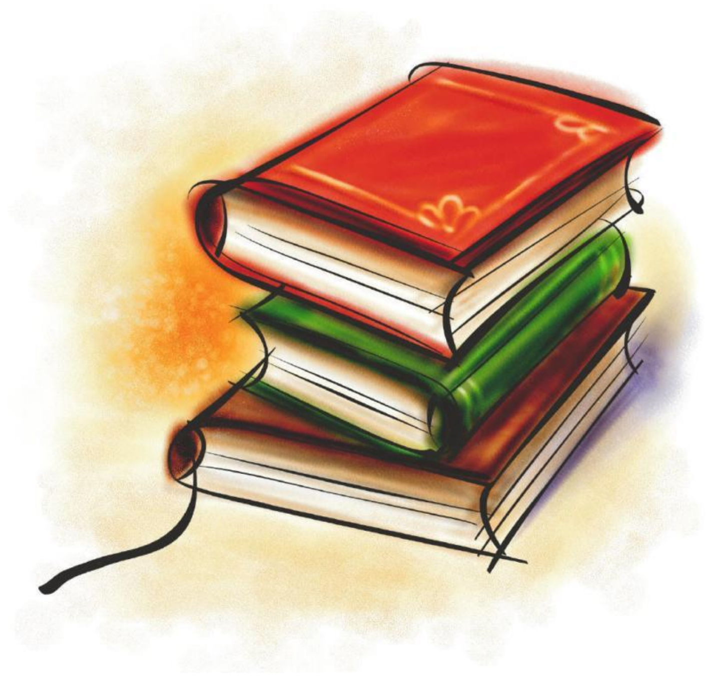
Grad(F) computation using Finite Differences (FD)

Finite Differences (FDs)

$$\frac{\partial F}{\partial b_i} = \frac{F(b_1, b_2, \ldots, b_i + \epsilon, \ldots, b_N) \; - \; F(b_1, b_2, \ldots, b_i, \ldots, b_N)}{\epsilon}$$

$$\frac{\partial F}{\partial b_i} = \frac{F(b_1, b_2, \ldots, b_i + \epsilon, \ldots, b_N) \; - \; F(b_1, b_2, \ldots, b_i - \epsilon, \ldots, b_N)}{2\epsilon}$$

- **The cost for computing grad(F) w.r.t. b_n (n=1,…,N) scales with N. Solution of 2N (for second-order accuracy) or N (for first-order accuracy) systems of PDEs for N sensitivity derivatives, in a PDE-governed application! Expensive!**
- **Amenable to parallelisation! Exploit a multi-processor platform to reduce the wall clock time.**
- **Accuracy and the role of ε.**
- **Although expensive, FDs are often used to compute REFERENCE sensitivities for the assessment of "better" methods, during development phase.**



Grad(F) computation using the Complex Variable (CV) Method

Complex Variable (CV) Method

In the hypothetical case of a single design variable b:

$$F(b+\delta b) = F(b) + \frac{\delta b}{1!}\frac{dF}{db}\bigg|_b + \frac{\delta b^2}{2!}\frac{d^2F}{db^2}\bigg|_b + O(\delta b^3)$$

or:

$$F(b+i\delta b) = F(b) + \frac{i\delta b}{1!}\frac{dF}{db}\bigg|_b + O(\delta b^2)$$

Real part: $real[F(b+i\delta b)] = F(b) + \cdots$

Imaginary part: $imag[F(b+i\delta b)] = \delta b \frac{dF}{db}\bigg|_b + \cdots$

In the most general case:

$$\boxed{\frac{\vartheta F}{\vartheta b_i} = \lim_{\epsilon \to 0} \frac{imag[F(b_i + i\epsilon)]}{\epsilon}}$$

- **Investment/software programming needed!**
- **N evaluations per gradient computation.**
- **Parallelisation!**
- **Accuracy and the role of ϵ .**

Complex Variable (CV) Method

Demonstration in a simple example:

$$F(b_1,b_2) = b_1^2 + 3b_2^4 - 4b_1b_2 \quad \Rightarrow \quad \begin{cases} \dfrac{\partial F}{\partial b_1} = 2b_1 - 4b_2 \\ \dfrac{\partial F}{\partial b_2} = 12b_2^3 - 4b_1 \end{cases}$$

$$(b_1,b_2) = (1,2) \quad \Rightarrow \quad \begin{cases} \dfrac{\partial F}{\partial b_1}(1,2) = -6 \\ \dfrac{\partial F}{\partial b_2}(1,2) = 92 \end{cases}$$

Complex Variable (CV) Method - Coding

```
program demo_complex_variables
implicit double precision (a-h,o-z)
double complex F,x,y
F(x,y) = x**2 + 3.d0*y**4 -4.d0*x*y
epsilon = 1.d-30
b1 = 1.d0
b2 = 2.d0
x=b1*(1.d0,0.d0)+epsilon*(0.d0,1.d0)
y=b2*(1.d0,0.d0)
dFdb1 = imag(F(x,y))/epsilon
x=b1*(1.d0,0.d0)
y=b2*(1.d0,0.d0)+epsilon*(0.d0,1.d0)
dFdb2 = imag(F(x,y))/epsilon
write(*,*)' Computed   dF/db1 = ',dFdb1
write(*,*)' Computed   dF/db2 = ',dFdb2
end
```

```
#include <iostream>
#include <complex>
std::complex<double> F(std::complex<double> x,
                       std::complex<double> y){
   return pow(x,2)+3.0*pow(y,4)-4.0*x*y;
}
int main()
{
   const double epsilon = 1.e-30;
   const double b1 = 1.0;
   const double b2 = 2.0;
   std::complex<double> x(b1, epsilon);
   std::complex<double> y(b2, 0.0);
   double dFdb1 = imag(F(x,y))/epsilon;
   x = std::complex<double> (b1, 0.0);
   y = std::complex<double> (b2, epsilon);
   double dFdb2 = imag(F(x,y))/epsilon;
   std::cout<<"Computed dF/db1 ="<<dFdb1<<std::endl;
   std::cout<<"Computed dF/db2 ="<<dFdb2<<std::endl;
   return 0;
}
```

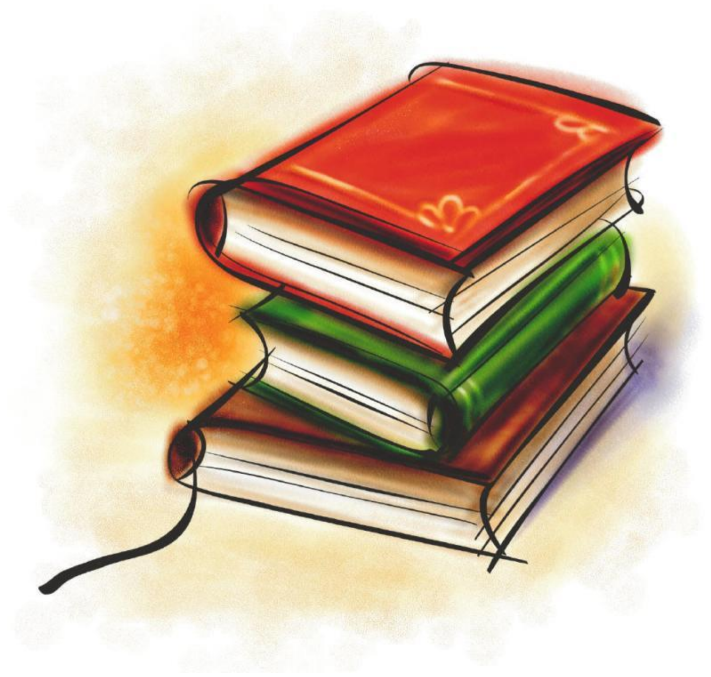
Complex Variable (CV) Method – On the Accuracy of the Computed Gradient

CV

ϵ	$\partial F/\partial b_1$	$\partial F/\partial b_2$
10^{-30}	-6.000000000000000	92.000000000000000
10^{-20}	-6.000000000000000	92.000000000000000
10^{-15}	-6.000000000000000	91.999999999999986
10^{-10}	-6.000000000000000	92.000000000000000
10^{-5}	-6.000000000000000	91.999999997600000

FD

ϵ	$\partial F/\partial b_1$	$\partial F/\partial b_2$
10^{-30}	0.000000000000000	0.000000000000000
10^{-20}	0.000000000000000	0.000000000000000
10^{-15}	-7.105427357601001	92.370555648813010
10^{-10}	-5.999964969305438	92.000007612114132
10^{-5}	-6.000000000128124	92.000000003267232



Grad(F) computation using Automatic Differentiation (AD)

Algorithmic or Automatic Differentiation (AD)

AD computes the derivative of a code/function by transforming the underlying code that computes the (numerical values of the) function.

$$\begin{tikzcd} \overrightarrow{b} \arrow[r] & \boxed{\text{The analysis source code written in Fortran 77/90 or ANSI C++}} \arrow[r] & F(\overrightarrow{b}) \end{tikzcd}$$

Two modes:

- **Tangent or forward → equivalent to Direct Differentiation, DD**
- **Reverse → equivalent to the Adjoint Method, AV.**

- **Cost?**
- **Memory Footprint?**

AD - Modes

Possible Implementations of AD:

Forward AD or **Reverse AD**

$$\dot{F}\left(b,\dot{b}\right)=\frac{dF}{db}(b)\dot{b}$$

Hybrid AD

$$\overline{b_i}=\bar{F}\left(b_i,\overline{b_j}\right)=\frac{dF}{db_i}^T(b_i)\overline{b_j}$$

$\dot{b}$	**Tangent direction of input b**
$\dot{F}$	**Tangent of F (generated by the forward AD)**

$\overline{b_i}$	**Adjoint of input b_i**
$\overline{b_j}$	**Adjoint of output b_j**
$\bar{F}$	**Adjoint of F (generated by the reverse AD)**

AD – Introduction of Intermediate Variables

The evaluation of F is represented by a sequence of elementary (unary or binary) functions, by introducing the necessary intermediate variables (indices: N+1,…,m).

$$\text{intermediate variables}\left\{\begin{array}{l} b_{N+1} = f_{N+1}\left(\overbrace{b_1,\ldots,b_N}^{\text{independent variables}}\right) \\ b_{N+2} = f_{N+2}\left(b_1,\ldots,b_{N+1}\right) \\ \quad\vdots \\ b_m = f_m\left(b_1,\ldots,b_{m-1}\right) \end{array}\right.$$

$$\boxed{F = b_m}$$

Forward AD

$$
\begin{aligned}
b_{N+1} &= f_{N+1}(b_1, \ldots, b_N) \\
\nabla b_{N+1} &= \sum_{i=1}^{N} \frac{\partial f_{N+1}}{\partial b_i} \vec{e}_i \\
b_{N+2} &= f_{N+2}(b_1, \ldots, b_{N+1}) \\
\nabla b_{N+2} &= \sum_{i=1}^{N} \frac{\partial f_{N+2}}{\partial b_i} \vec{e}_i + \frac{\partial f_{N+2}}{\partial b_{N+1}} \nabla b_{N+1} \\
&\vdots \\
b_m &= f_m(b_1, \ldots, b_{m-1}) \\
\nabla b_m &= \sum_{i=1}^{N} \frac{\partial f_m}{\partial b_i} \vec{e}_i + \sum_{i=N+1}^{m-1} \frac{\partial f_m}{\partial b_i} \nabla b_i
\end{aligned}
$$

$$
\boxed{\begin{aligned} F &\equiv b_m \\ \nabla F &\equiv \nabla b_m \end{aligned}}
$$

Apply the chain rule of differentiation of composite functions, by associating its statement with an extra statement computing its derivatives.

Forward AD represents how an infinitesimally small change in the input/independent variables propagates and, finally, affect the output.

At the expense of memory!!!

Forward AD - Example

$$\boxed{F = b_1 b_2 b_3 b_4}$$

$$\Re^4 \rightarrow \Re$$

Intermediate variables:

$$\begin{aligned}
b_5 &= f_5(b_1, b_2, b_3, b_4) = b_1 \\
b_6 &= f_6(b_1, b_2, b_3, b_4, b_5) = b_2 b_5 \\
b_7 &= f_7(b_1, b_2, b_3, b_4, b_5, b_6) = b_3 b_6 \\
b_8 &= f_7(b_1, b_2, b_3, b_4, b_5, b_6, b_7) = b_4 b_7 \\
F &= b_8
\end{aligned}$$

$$\begin{aligned}
b_5 &= b_1 \\
\nabla b_5 &= \overrightarrow{e}_1 \\
b_6 &= b_2 b_5 \\
\nabla b_6 &= b_5 \overrightarrow{e}_2 + b_2 \nabla b_5 \\
b_7 &= b_3 b_6 \\
\nabla b_7 &= b_6 \overrightarrow{e}_3 + b_3 \nabla b_6 \\
b_8 &= b_4 b_7 \\
\nabla b_8 &= b_7 \overrightarrow{e}_4 + b_4 \nabla b_7 \\
F &= b_8 \\
\nabla F &= \nabla b_8
\end{aligned}$$

Reverse AD (1/2)

Compute the sensitivity of the output to changes in the input/independent variables.

Define: $$\bar{b}_i = \frac{\partial b_m}{\partial b_i} \qquad (i = N+1, \ldots, m) \qquad \text{with} \qquad \bar{b}_m \equiv 1$$

Compute:

$$\bar{b}_i = \sum_{j=i+1}^{m} \frac{\partial b_m}{\partial b_j}\frac{\partial f_j}{\partial b_i} = \sum_{j=i+1}^{m} \bar{b}_j \frac{\partial f_j}{\partial b_i} \; , \qquad \underbrace{i = N+1, \ldots, m-1}_{\textbf{Intermediate variables}}$$

$$\bar{b}_i = \sum_{j=N+1}^{m} \frac{\partial b_m}{\partial b_j}\frac{\partial f_j}{\partial b_i} = \sum_{j=N+1}^{m} \bar{b}_j \frac{\partial f_j}{\partial b_i} \; , \qquad \underbrace{i = 1, \ldots, N}_{\textbf{Independent variables}}$$

Reverse AD (2/2)

Forward loop:

$$b_i = f_i\left(b_1, \ldots, b_{i-1}\right) \ , \quad i = N+1, \ldots, m$$

$$\bar{b}_i = 0 \ , \quad i = 1, \ldots, m-1$$

$$\bar{b}_m = 1$$

Backward (reverse) loop:

```
$do\ j = m, N+1, \boxed{-1}$
    $do\ i = 1, j-1$
        $\bar{b}_i = \bar{b}_i + \bar{b}_j \dfrac{\partial f_j}{\partial b_i}$
    $enddo$
$enddo$
```

Reverse AD – Example (1/3)

$$\boxed{F = b_1 b_2 b_3}$$

$$\Re^3 \longrightarrow \Re$$

$$
\begin{aligned}
\bar{b}_6 &= \frac{\partial f_6}{\partial b_6} \equiv 1\\
\bar{b}_5 &= \frac{\partial f_6}{\partial b_6}\frac{\partial f_6}{\partial b_5} = \bar{b}_6 \frac{\partial f_6}{\partial b_5}\\
\bar{b}_4 &= \frac{\partial f_6}{\partial b_6}\frac{\partial f_6}{\partial b_4} + \frac{\partial f_6}{\partial b_5}\frac{\partial f_5}{\partial b_4} = \bar{b}_6 \frac{\partial f_6}{\partial b_4} + \bar{b}_5 \frac{\partial f_5}{\partial b_4}\\
\bar{b}_3 &= \frac{\partial f_6}{\partial b_6}\frac{\partial f_6}{\partial b_3} + \frac{\partial f_6}{\partial b_5}\frac{\partial f_5}{\partial b_3} + \frac{\partial f_6}{\partial b_4}\frac{\partial f_4}{\partial b_3} = \bar{b}_6 \frac{\partial f_6}{\partial b_3} + \bar{b}_5 \frac{\partial f_5}{\partial b_3} + \bar{b}_4 \frac{\partial f_4}{\partial b_3}\\
\bar{b}_2 &= \frac{\partial f_6}{\partial b_6}\frac{\partial f_6}{\partial b_2} + \frac{\partial f_6}{\partial b_5}\frac{\partial f_5}{\partial b_2} + \frac{\partial f_6}{\partial b_4}\frac{\partial f_4}{\partial b_2} = \bar{b}_6 \frac{\partial f_6}{\partial b_2} + \bar{b}_5 \frac{\partial f_5}{\partial b_2} + \bar{b}_4 \frac{\partial f_4}{\partial b_2}\\
\bar{b}_1 &= \frac{\partial f_6}{\partial b_6}\frac{\partial f_6}{\partial b_1} + \frac{\partial f_6}{\partial b_5}\frac{\partial f_5}{\partial b_1} + \frac{\partial f_6}{\partial b_4}\frac{\partial f_4}{\partial b_1} = \bar{b}_6 \frac{\partial f_6}{\partial b_1} + \bar{b}_5 \frac{\partial f_5}{\partial b_1} + \bar{b}_4 \frac{\partial f_4}{\partial b_1}
\end{aligned}
$$

Intermediate variables

$$
\begin{aligned}
b_4 &= f_4(b_1, b_2, b_3) = b_1\\
b_5 &= f_5(b_1, b_2, b_3, b_4) = b_2 b_4\\
b_6 &= f_6(b_1, b_2, b_3, b_4, b_5) = b_3 b_5\\
F &= b_6
\end{aligned}
$$

$$\boxed{\frac{\partial f_3}{\partial b_1} = \frac{\partial f_3}{\partial b_2} = \frac{\partial f_2}{\partial b_1}}$$

Reverse AD – Example (2/3)

1 $\bar{b}_6 \equiv 1 \;,\quad \bar{b}_5 = 0 \;,\quad \bar{b}_4 = 0 \;,\quad \bar{b}_3 = 0 \;,\quad \bar{b}_2 = 0 \;,\quad \bar{b}_1 = 0$

2 $\boxed{f_6 = b_3 b_5}$ $\dfrac{\partial f_6}{\partial b_5} = b_3 \;,\quad \dfrac{\partial f_6}{\partial b_4} = 0 \;,\quad \dfrac{\partial f_6}{\partial b_3} = b_5 \;,\quad \dfrac{\partial f_6}{\partial b_2} = 0 \;,\quad \dfrac{\partial f_6}{\partial b_1} = 0$

Σ: $\bar{b}_5 = b_3 \;,\quad \bar{b}_4 = 0 \;,\quad \bar{b}_3 = b_5 \;,\quad \bar{b}_2 = 0 \;,\quad \bar{b}_1 = 0$

3 $\boxed{f_5 = b_2 b_4}$ $\dfrac{\partial f_5}{\partial b_4} = b_2 \;,\quad \dfrac{\partial f_5}{\partial b_3} = 0 \;,\quad \dfrac{\partial f_5}{\partial b_2} = b_4 \;,\quad \dfrac{\partial f_5}{\partial b_1} = 0$

Σ: $\bar{b}_4 = b_2 b_3 \;,\quad \bar{b}_3 = b_5 \;,\quad \bar{b}_2 = b_3 b_4 \;,\quad \bar{b}_1 = 0$

4 $\boxed{f_4 = b_1}$ $\dfrac{\partial f_4}{\partial b_3} = 0 \;,\quad \dfrac{\partial f_4}{\partial b_2} = 0 \;,\quad \dfrac{\partial f_4}{\partial b_1} = 1$

Σ: $\bar{b}_3 = b_5 \;,\quad \bar{b}_2 = b_3 b_4 \;,\quad \bar{b}_1 = b_2 b_3$

Reverse AD – Example (3/3)

Final (forward) summation:

$$
\begin{aligned}
b_4 &= b_1 \\
\nabla b_4 &= \overrightarrow{e}_1 \\
b_5 &= b_2 b_4 \\
\nabla b_5 &= b_4 \overrightarrow{e}_2 + b_2 \nabla b_4 = b_4 \overrightarrow{e}_2 + b_2 \overrightarrow{e}_1 \\
b_6 &= b_3 b_5 \\
\nabla b_6 &= b_5 \overrightarrow{e}_3 + b_3 \nabla b_5 = b_5 \overrightarrow{e}_3 + b_3 b_4 \overrightarrow{e}_2 + b_3 b_2 \overrightarrow{e}_1
\end{aligned}
$$

AD Software

ADIFOR (Automatic Differentiation of Fortran)
AMC (Tangent linear and Adjoint Model Compiler)
TAF (Transformation of Algorithms in Fortran)
DAFOR (Differential Algebraic Extension of Fortran)
GRESS (Gradient--Enhanced Software System)
Odyssee
TAPENADE
AD01
ADOL-F (Automatic Differentiation of FORTRAN Codes)
IMAS (Integrated Modeling and Analysis System)
OPTIMA90
ADIC (Automatic Differentiation of C Programs)
ADOL-C (Automatic Differentiation of Algorithms written in C/C++)
etc…..

AD using TAPENADE (Forward Mode)

```
void ff_d(double x1, double x1d, double x2, double x2d, double x3, double x3d,  double *f1,
double *f1d, double *f2, double *f2d)      //function produced by TAPENADE - forward
(tangent) mode
{
/*
  Differentiation of ff in forward (tangent) mode:
  variations  of useful results: *f1 *f2
  with respect to varying inputs: x1 x2 x3
  RW status of diff variables: f1:(loc) *f1:out f2:(loc) *f2:out
       x1:in x2:in x3:in
  Plus diff mem management of: f1:in f2:in
*/

   *f1d = x1d + x2d + x3d;
   *f1 = x1 + x2 + x3;
   *f2d = x3*(x2*x1d+x1*x2d) + x1*x2*x3d;
   *f2 = x1*x2*x3;
}
```

```
#include <stdio.h>

void ff(double x1, double x2, double x3, double* f1,
double* f2)              //functions to be differentiated
{
   *f1 = x1 + x2 + x3;    //1st function
   *f2 = x1 * x2 * x3;    //2nd function
}
```

The ff_d function must be called as many times as there are design variables (x1, x2, x3), with (x1d,x2d,x3d)=(1,0,0),(0,1,0) and (0,0,1).
Each call computes the derivatives of all objectives (both f1 & f2) w.r.t. the design variable for which xkd equals 1.

AD using TAPENADE (Reverse Mode)

```
void ff_b(double x1, double *x1b, double x2, double *x2b, double x3, double *x3b, double
*f1, double *f1b, double *f2, double *f2b)     //function produced by TAPENADE - reverse
(adjoint) mode
{
/*
  Differentiation of ff in reverse (adjoint) mode:
   gradient     of useful results: *f1 *f2
   with respect to varying inputs: *f1 *f2 x1 x2 x3
   RW status of diff variables: f1:(loc) *f1:in-out f2:(loc) *f2:in-out
               x1:out x2:out x3:out
   Plus diff mem management of: f1:in f2:in
*/
    *x1b = x2*x3*(*f2b) + *f1b;
    *x2b = x1*x3*(*f2b) + *f1b;
    *x3b = x1*x2*(*f2b) + *f1b;
    *f2b = 0.0;
    *f1b = 0.0;
}
```

```
#include <stdio.h>

void ff(double x1, double x2, double x3, double* f1,
double* f2)               //functions to be differentiated
{
   *f1 = x1 + x2 + x3;     //1st function
   *f2 = x1 * x2 * x3;     //2nd function
}
```

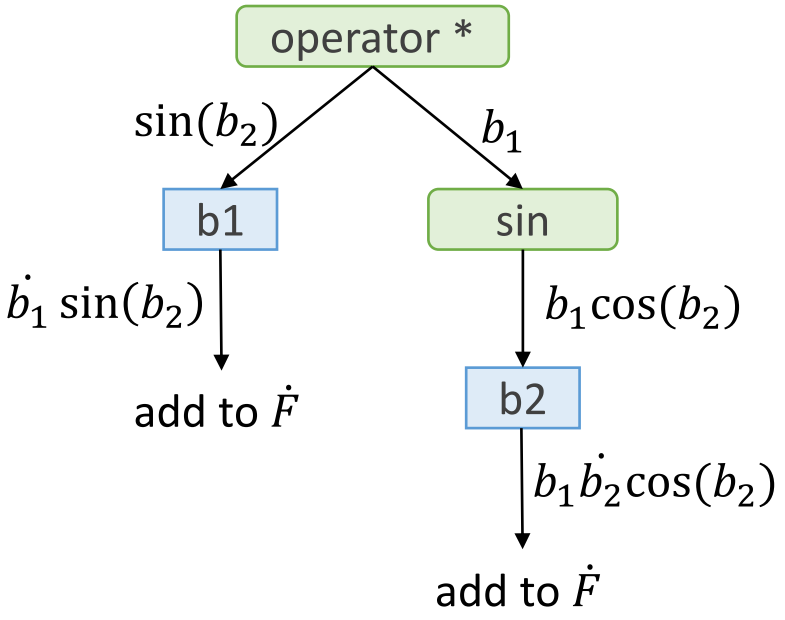
The ff_b function will be called as many times as there are objective functions (f1, f2), with (f1b,f2b)=(1,0),(0,1). Each call computes the derivatives of fkb for which k equals 1 w.r.t. all design variables (x1,x2,x3)

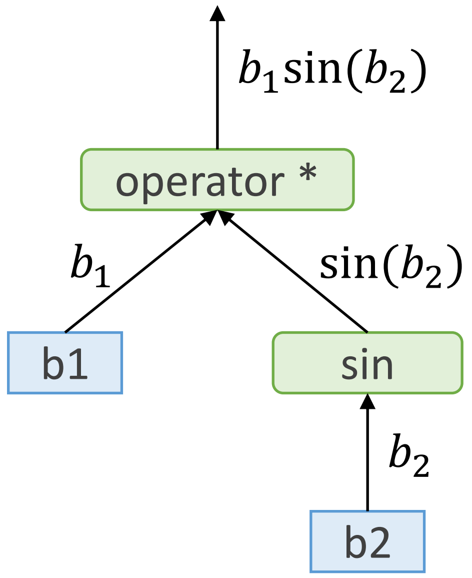
AD – An Example

$$\boxed{F(b_1, b_2) = b_1 \sin(b_2)}$$

Forward AD:

$$\dot{F} = \dot{b_1} \sin(b_2) + b_1 \dot{b_2} \cos(b_2)$$

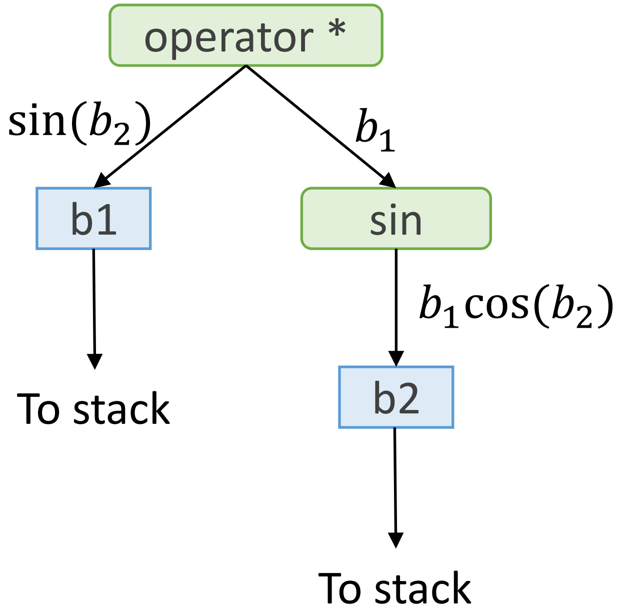




Reverse AD:

$$\bar{b}_1 \mathrel{+}= sin(b_2)\bar{F}$$

$$\bar{b}_2 \mathrel{+}= b_1 cos(b_2)\bar{F}$$



AD - An Example using TAPENADE

Assume a (piece of) code computing the values of 2 functions $f_1(b_1,b_2,b_3)$ and $f_2(b_1,b_2,b_2)$, written in any language that can be processed by an AD tool (herein Tapenade by INRIA). AD will be used to compute six derivatives, in total.

$$f_1 = b_1^2 + b_2\sqrt{\frac{b_3}{b_1}} + sin(b_1b_2)exp(b_2+b_3)$$

$$f_2 = log_{10}(b_1b_2b_3) - 5(b_2-b_1)\tan(b_3)$$

```
#include <iostream>
#include <cmath>
void my(double b1, double b2, double b3, double& f1,
double& f2) {
   f1 = b1 * b1 + b2 * std::sqrt(b3 / b1) + std::sin(b1 * b2) *
std::exp(b2 + b3);
   f2 = std::log10(b1 * b2 * b3) - 5.0 * (b2 - b1) *
std::tan(b3);
}
```

```
      subroutine my(b1,b2,b3,f1,f2)
      f1=b1*b1+b2*sqrt(b3/b1)+sin(b1*b2)*exp(b2+b3)
      f2=log10(b1*b2*b3)-5.*(b2-b1)*tan(b3)
      return
      end
```

AD - An Example using TAPENADE (in tangent/forward mode)

```
C       Generated by TAPENADE    (INRIA, Ecuador team)  in forward (tangent) mode:
      SUBROUTINE MY_D(b1, b1d, b2, b2d, b3, b3d, f1, f1d, f2, f2d)
      arg1d = (b3d-b3*b1d/b1)/b1
      arg1 = b3/b1
      temp = DSQRT(arg1)
      result1d = arg1d/(2.D0*DSQRT(arg1))
      result1 = temp
      temp = DEXP(b2 + b3)
      temp0 = DSIN(b1*b2)
      f1d = 2*b1*b1d + result1*b2d + b2*result1d + temp*DCOS(b1*b2)*(b2*
     +  b1d+b1*b2d) + temp0*DEXP(b2+b3)*(b2d+b3d)
      f1 = b1*b1 + b2*result1 + temp0*temp
      temp1 = b1*b2*b3
      temp0 = DTAN(b3)
      f2d = (b3*(b2*b1d+b1*b2d)+b1*b2*b3d)/(temp1*DLOG(10.D0)) - 5.*(
     +  temp0*(b2d-b1d)+(b2-b1)*(1.D0+DTAN(b3)**2)*b3d)
      f2 = DLOG10(temp1) - 5.*((b2-b1)*temp0)
      RETURN
      END
```

…d

Call this routine as many times as there are design variables (3 calls, for: b1, b2, b3)., with: (b1d,b2d,b3d) = (1,0,0),(0,1,0) and (0,0,1). In each call, derivatives of all functions (f1 and f2) w.r.t. the design variable marked with «1» are computed.

AD - An Example using TAPENADE (in tangent/forward mode)

$$f_1 = b_1^2 + b_2\sqrt{\frac{b_3}{b_1}} + sin(b_1 b_2)exp(b_2+b_3)$$

$b_4 = \frac{b_3}{b_1}$ ◆ $b_{4d} = \frac{b_{3d} - b_3 b_{1d}/b_1}{b_1}$ ◆ $b_5 = \sqrt{b_4}$ ◆ $b_{5d} = \frac{b_{4d}}{2\sqrt{b_4}}$

$b_6 = sin(b_1 b_2)$ ◆ $b_7 = exp(b_2+b_3)$ ◆ $f_1 = b_1 b_1 + b_2 b_5 + b_6 b_7$

$$f_{1d} = 2b_1 b_{1d} + b_{2d} b_5 + b_2 b_{5d} + b_7 cos(b_1 b_2)(b_{2d} b_1 + b_2 b_{1d}) + b_6 exp(b_2+b_3)(b_{2d} + b_{3d})$$

$$f_2 = log_{10}(b_1 b_2 b_3) - 5(b_2 - b_1)\tan(b_3)$$

$b_8 = b_1 b_2 b_3$ ◆ $b_9 = \tan(b_3)$ ◆ $f_2 = log_{10}(b_8) - 5(b_2 - b_1) b_9$

$$f_{2d} = \frac{[b_3(b_{2d} b_1 + b_2 b_{1d}) + b_1 b_2 b_{3d}]}{b_1 b_2 b_3 ln(10)} - 5\left[b_9(b_{2d} - b_{1d}) - (b_2 - b_1)\left(1 + tan^2(b_3)\right)\right] b_{3d}$$

AD - An Example using TAPENADE (in reverse/adjoint mode)

```
C       Generated by TAPENADE    (INRIA, Ecuador team)  in reverse (adjoint) mode:
      SUBROUTINE MY_B(b1, b1b, b2, b2b, b3, b3b, f1, f1b, f2, f2b)
      tempb1 = f2b/(b1*b2*b3*DLOG(10.D0))
      tempb0 = -(DTAN(b3)*5.*f2b)
      b3b = b1*b2*tempb1 - (1.D0+DTAN(b3)**2)*(b2-b1)*5.*f2b
      b2b = tempb0 + b1*b3*tempb1
      b1b = -tempb0
      temp = b3/b1
      temp0 = DSQRT(temp)
      tempb = b2*f1b/(b1*2.D0*DSQRT(temp))
      tempb0 = DCOS(b1*b2)*DEXP(b2+b3)*f1b
      b1b = b1b + b2*b3*tempb1 + 2*b1*f1b + b2*tempb0 - temp*tempb
      tempb1 = DEXP(b2+b3)*DSIN(b1*b2)*f1b
      b2b = b2b + temp0*f1b + tempb1 + b1*tempb0
      b3b = b3b + tempb1 + tempb
      f1b = 0.D0
      f2b = 0.D0
      END
```

...b

**Call this routine as many times as there are functions to be differentiated (2 calls, for: f1, f2), with: (f1b,f2b) = (1,0) and (0,1).
In each call, derivatives of the function (f1 or f2) marked with «1»,w.r.t. all design variables are computed.**

An AD Example using TAPENADE (in reverse/adjoint mode)

$$f_2 = log_{10}(b_1 b_2 b_3) - 5(b_2 - b_1)\tan(b_3)$$

$$t_{1b} = \frac{f_{2b}}{b_1 b_2 b_3 ln(10)} \quad ◆ \quad t_{0b} = -5 f_{2b} \tan(b_3) \quad ◆ \quad b_{3b} = b_1 b_2 t_{1b} - 5(b_2 - b_1)\left(1 + tan^2(b_3)\right) f_{2b}$$

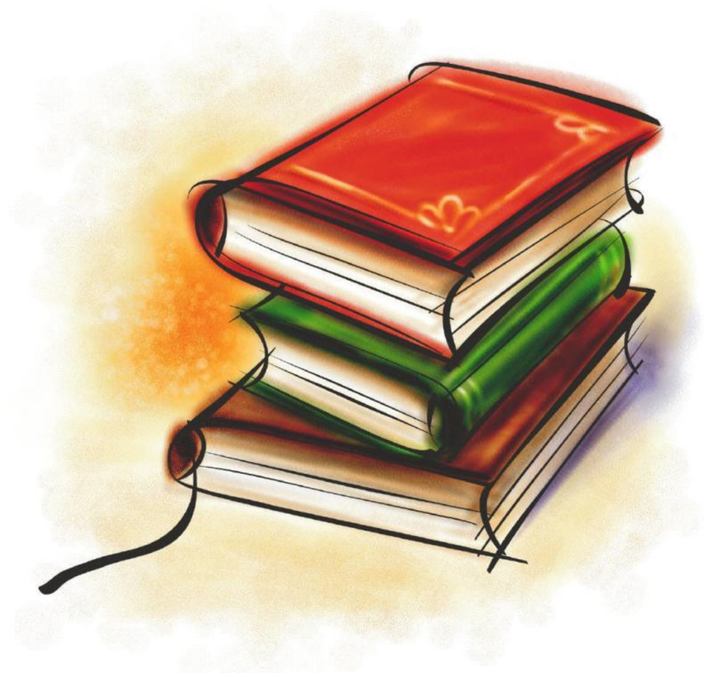
$$b_{2b} = t_{0b} + b_1 b_3 t_{1b} \quad ◆ \quad b_{1b} = -t_{0b}$$

$$f_1 = b_1^2 + b_2 \sqrt{\frac{b_3}{b_1}} + sin(b_1 b_2) exp(b_2 + b_3)$$

$$t = \frac{b_3}{b_1} \quad ◆ \quad t_b = \frac{b_2 f_{1b}}{2 b_1 \sqrt{t}} \quad ◆ \quad t_0 = \sqrt{t} \quad ◆ \quad t_{0b} = \cos(b_1 b_2) exp(b_2 + b_3) f_{1b}$$

$$b_{1b} = b_{1b} + b_2 b_3 t_{1b} + 2 b_1 f_{1b} + b_2 t_{0b} - t t_b \quad ◆ \quad t_{1b} = \sin(b_1 b_2) exp(b_2 + b_3) f_{1b}$$

$$b_{2b} = b_{2b} + t_0 f_{1b} + t_{1b} + b_1 t_{0b} \quad ◆ \quad b_{3b} = b_{3b} + t_{1b} + t_b$$



Grad(F) computation using Direct Differentiation (DD)

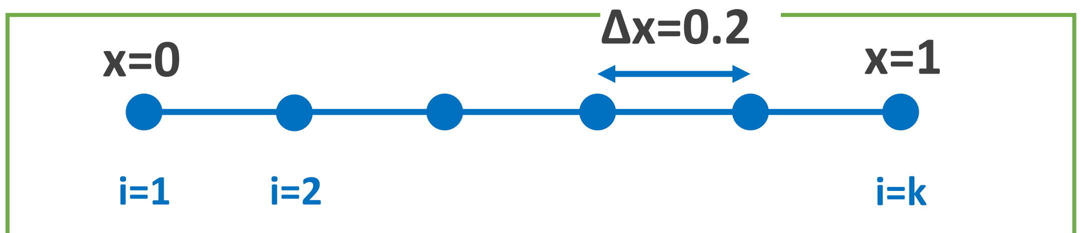
Example (ODE) – The Primal Problem in Discrete Form (1/3)

Example: (Primal) ODE:

$$\frac{d^2v}{dx^2} + b_1 v + b_2 v^2 - 5 = 0$$

$$0 \le x \le 1$$

Boundary conditions: $\begin{cases} v(0) = 1 \\ v(1) = 3 \end{cases}$



Spatial discretisation using equidistant nodes.

Linearisation / Delta formulation: ($v_i^{new} = v_i^{old} + \Delta v_i$)

$$\frac{d^2(v^{old} + \Delta v)}{dx^2} + b_1(v^{old} + \Delta v) + b_2((v^{old})^2 + 2v^{old}\Delta v) - 5 = 0 \qquad 2 \le i \le k-1$$

or

$$\frac{d^2\Delta v}{dx^2} + b_1\Delta v + 2b_2 v^{old}\Delta v = -\frac{d^2 v^{old}}{dx^2} - b_1 v^{old} - b_2(v^{old})^2 + 5$$

$i = 1$	$\Delta v_1 = 0$
$i = k$	$\Delta v_k = 0$

Example (ODE) – The Primal Problem in Discrete Form (2/3)

$$\frac{d^2\Delta\upsilon}{dx^2}+b_1\Delta\upsilon+2b_2\upsilon^{old}\Delta\upsilon=-\frac{d^2\upsilon^{old}}{dx^2}-b_1\upsilon^{old}-b_2(\upsilon^{old})^2+5$$

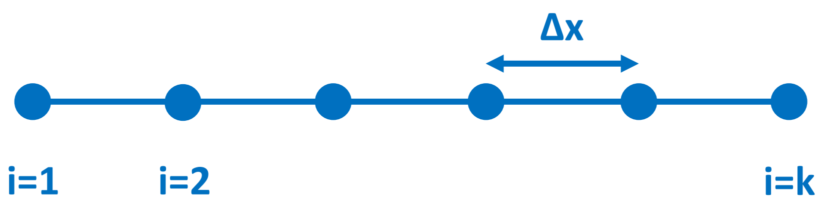
$$\left[\frac{1}{\Delta x^2}\right]\Delta\upsilon_{i-1}+\left[b_1+2b_2\upsilon_i^{old}-\frac{2}{\Delta x^2}\right]\Delta\upsilon_i+\left[\frac{1}{\Delta x^2}\right]\Delta\upsilon_{i+1}=-\frac{\upsilon_{i+1}^{old}-2\upsilon_i^{old}+\upsilon_{i-1}^{old}}{\Delta x^2}-b_1\upsilon_i^{old}-b_2(\upsilon^{old})^2+5$$

If: $\alpha=\dfrac{1}{\Delta x^2}$ $\qquad\gamma_i=b_1+2b_2\upsilon_i^{old}-2\alpha$

$$\boxed{\begin{bmatrix}1&&&&&\\ \alpha&\gamma_2&\alpha&&&\\ &\alpha&\gamma_3&\alpha&&\\ &&\alpha&\gamma_4&\alpha&\\ &&&\alpha&\gamma_5&\alpha\\ &&&&&1\end{bmatrix}\begin{bmatrix}\Delta\upsilon_1\\ \Delta\upsilon_2\\ \Delta\upsilon_3\\ \Delta\upsilon_4\\ \Delta\upsilon_5\\ \Delta\upsilon_6\end{bmatrix}=-\begin{bmatrix}0\\ R_2^{old}\\ R_3^{old}\\ R_4^{old}\\ R_5^{old}\\ 0\end{bmatrix}}$$

$$R_1=\upsilon_1-1=0$$

$$R_6=\upsilon_6-3=0$$



(for k=6 nodes)

Example (ODE) – The Primal Problem in Discrete Form (3/3)

$$\left.\frac{d\vec{R}(\vec{v})}{d\vec{v}}\right|^{old}\Delta\vec{v} = -\vec{R}\left(\vec{v}^{old}\right) \qquad \vec{v}^{new} = \vec{v}^{old} + \Delta\vec{v}$$

Jacobian! (Could be replaced by an easily invertible approximation to the real Jacobian, if needed)!

$$\boxed{\begin{bmatrix} 1 & & & & & \\ \alpha & \gamma_2 & \alpha & & & \\ & \alpha & \gamma_3 & \alpha & & \\ & & \alpha & \gamma_4 & \alpha & \\ & & & \alpha & \gamma_5 & \alpha \\ & & & & & 1 \end{bmatrix}\begin{bmatrix}\Delta v_1\\ \Delta v_2\\ \Delta v_3\\ \Delta v_4\\ \Delta v_5\\ \Delta v_6\end{bmatrix} = -\begin{bmatrix}0\\ R_2^{old}\\ R_3^{old}\\ R_4^{old}\\ R_5^{old}\\ 0\end{bmatrix}}$$

Comments on the Use of the Jacobian

Solve the state/primal equations $\vec{R}(\vec{U}) = 0$

through an iterative method (linearisation, delta formulation ...)

$$\vec{R}(\vec{U}^{n+1}) = 0$$

$$\vec{R}(\vec{U}^{n+1}) = \vec{R}(\vec{U}^{n}) + \left.\frac{\partial \vec{R}}{\partial \vec{U}}\right|_{\vec{U}^n} (\vec{U}^{n+1} - \vec{U}^{n}) = \vec{R}(\vec{U}^{n}) + A|_{\vec{U}^n} \Delta \vec{U}$$

or $\underbrace{A|_{\vec{U}^n} \Delta \vec{U}}_{\text{Numerics}} = \underbrace{-\vec{R}(\vec{U}^{n})}_{\text{Physics}} \Rightarrow \vec{U}^{n+1} = \vec{U}^{n} + \Delta \vec{U}$

Direct Differentiation (DD)

Minimise (objective function / trapezoidal rule of integration):

$$F=\int_0^1 \upsilon(x)dx=\frac{\Delta x}{2}\upsilon_1+\sum_{i=2}^{5}\upsilon_i\Delta x+\frac{\Delta x}{2}\upsilon_6 \quad \Rightarrow \quad \frac{\delta F}{\delta b_i}=\int_0^1\frac{\delta \upsilon}{\delta b_i}(x)dx\cong\sum_{i=1}^{k}\left.\frac{\delta \upsilon}{\delta b_i}\right|_k\Delta x\ (or\frac{\Delta x}{2})$$

Optimisation (steepest descent):

$$b_i^{new}=b_i^{old}-\eta\frac{\delta F}{\delta b_i}\left(b_1^{old},b_2^{old}\right),\quad \text{i=1,…,N}$$

Continuous DD

$$\frac{d^2v}{dx^2} + b_1 v + b_2 v^2 - 5 = 0 \qquad v(0) = 1 \quad ◆ \quad v(1) = 3$$

First DD-ODE:

$$\frac{d^2}{dx^2}\left(\frac{\delta v}{\delta b_1}\right) + v + b_1\frac{\delta v}{\delta b_1} + 2b_2 v\frac{\delta v}{\delta b_1} = 0 \qquad 0 \le x \le 1$$

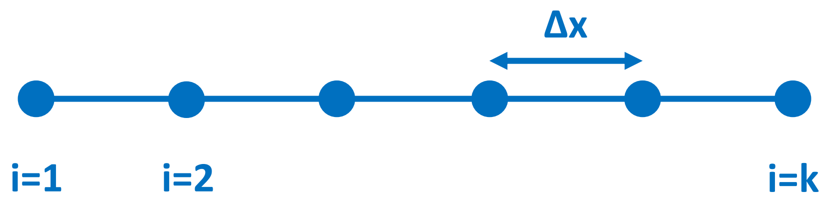
$$\frac{\delta v}{\delta b_1}(0) = 0 \quad ◆ \quad \frac{\delta v}{\delta b_1}(1) = 0$$

N DD-equations to be solved.
CPU cost scales with **N**.
Programming is needed.

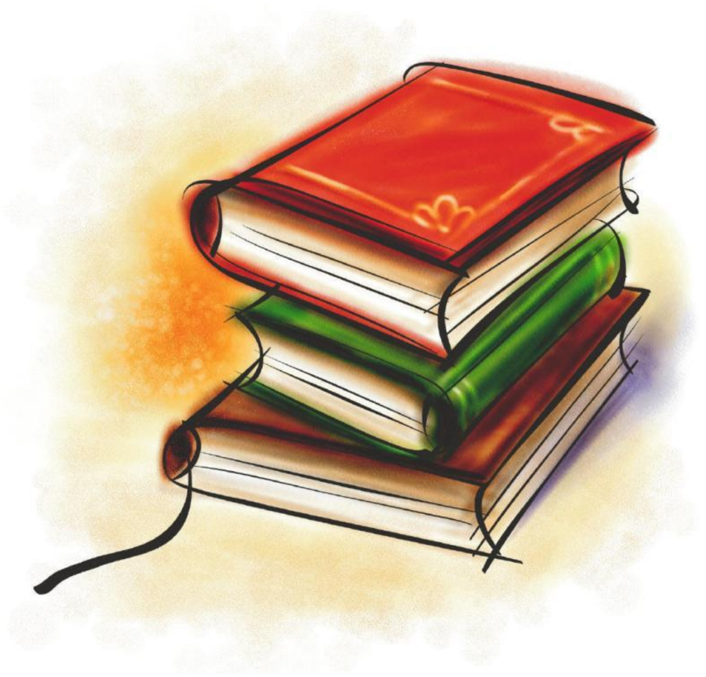
Second DD-ODE:

$$\frac{d^2}{dx^2}\left(\frac{\delta v}{\delta b_2}\right) + b_1\frac{\delta v}{\delta b_2} + v^2 + 2b_2 v\frac{\delta v}{\delta b_2} = 0 \qquad 0 \le x \le 1$$

$$\frac{\delta v}{\delta b_2}(0) = 0 \quad ◆ \quad \frac{\delta v}{\delta b_2}(1) = 0$$



Discrete DD



Grad(F) computation using the Adjoint (Variable, AV) Method

The Adjoint Method (AV)

Continuous Adjoint Method:

1. Differentiate the primal equations (PDEs). Derive adjoint equations in the form of PDEs.
2. Discretize & solve the primal and adjoint equations (PDEs).

First Differentiate – then Discretize

Discrete Adjoint Method:

1. Discretize the primal equations (PDEs) and differentiate them in discrete form.
2. Derive the adjoint equations in discrete form (linear system) and solve them.

First Discretize – then Differentiate

Adjoint is the art of computing the derivatives of J (or F) w.r.t. b_n (n=1,…,N) without first computing the fields of the derivatives of the primal variables w.r.t. b_n (skip DD). **The adjoint method makes the cost of computing the gradient of F independent from N!**

Towards Discrete Adjoint (1/7)

Objective Function: Min.

$$F = F(\overrightarrow{U}(\overrightarrow{b}), \overrightarrow{b}) \qquad \text{or} \qquad F = F(\overrightarrow{U}, \overrightarrow{b})$$

Indirect dependence

Direct dependence

Subject to

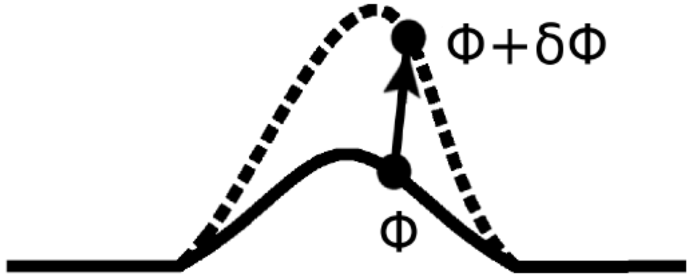
$$\overrightarrow{R} = \overrightarrow{R}(\overrightarrow{U}(\overrightarrow{b}), \overrightarrow{b}) = 0 \qquad \text{or} \qquad \overrightarrow{R} = \overrightarrow{R}(\overrightarrow{U}, \overrightarrow{b}) = 0$$

Towards Discrete Adjoint (2/7)

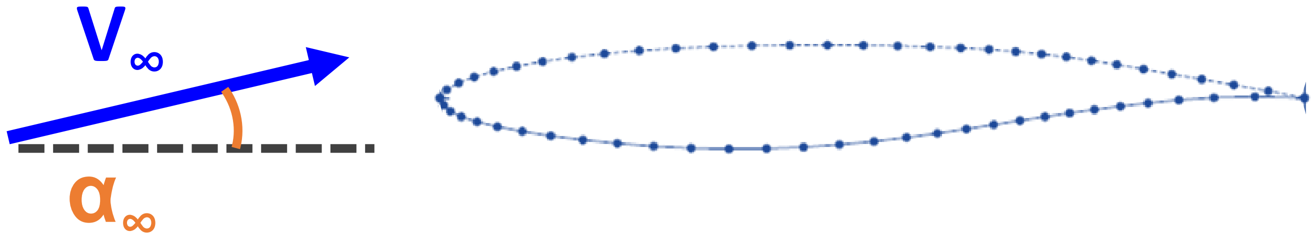
Or, compute $$\frac{\delta F}{\delta \vec{b}} = \frac{\partial F}{\partial \vec{b}} + \frac{\partial F}{\partial \vec{U}}\frac{\delta \vec{U}}{\delta \vec{b}}$$ (since $F = F(\vec{U}(\vec{b}), \vec{b})$)

Subject to $$\frac{\delta \vec{R}}{\delta \vec{b}} = \frac{\partial \vec{R}}{\partial \vec{b}} + \frac{\partial \vec{R}}{\partial \vec{U}}\frac{\delta \vec{U}}{\delta \vec{b}} = 0$$ (since $\vec{R} = \vec{R}(\vec{U}(\vec{b}), \vec{b}) = 0$)

Total vs. Partial Derivatives



$$F = \int_S p\vec{n}\cdot\vec{a}_\infty dS = \sum p_i \vec{n}_i \cdot \vec{a}_\infty \Delta S_i$$



Towards Discrete Adjoint (3/7)

Understanding $\frac{\delta F}{\delta \vec{b}}$ **:**

$$\frac{\delta F}{\delta \vec{b}} = \frac{\partial F}{\partial \vec{b}} + \frac{\partial F}{\partial \vec{U}}\frac{\delta \vec{U}}{\delta \vec{b}}$$

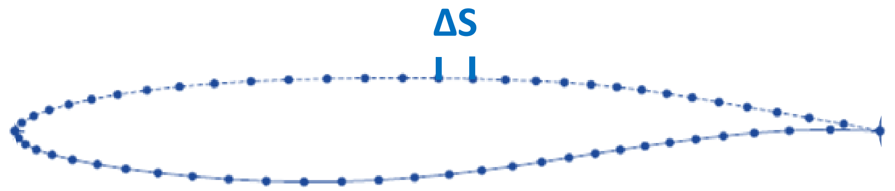
$$\frac{\partial F}{\partial \vec{b}} = \underset{\leftarrow N \rightarrow}{\begin{array}{|c|c|c|}\hline \quad & \cdots & \quad \\ \hline \end{array}} \updownarrow 1 \qquad \frac{\partial F}{\partial \vec{U}} = \underset{\leftarrow \quad \kappa \quad \rightarrow}{\begin{array}{|c|c|c|c|c|c|c|}\hline \quad & \quad & \quad & \cdots & \quad & \quad & \quad \\ \hline \end{array}} \updownarrow 1 \qquad \frac{\delta \vec{U}}{\delta \vec{b}} = \underset{\leftarrow N \rightarrow}{\begin{array}{|c|c|c|}\hline \quad & \cdots & \quad \\ \hline \quad & \cdots & \quad \\ \hline \quad & \cdots & \quad \\ \hline \quad & \vdots & \quad \\ \hline \quad & \cdots & \quad \\ \hline \quad & \cdots & \quad \\ \hline \quad & \cdots & \quad \\ \hline \end{array}} \begin{array}{c} \uparrow \\ \kappa \\ \downarrow \end{array}$$

Computing the derivatives of the primal variables w.r.t. the design variables (κ x N) corresponds to Discrete DD. The computational cost of DD scales with N. This is avoided using the Adjoint Method.

Towards Discrete Adjoint (4/7)

Understanding $\dfrac{\partial F}{\partial \vec{b}}$ **and** $\dfrac{\partial F}{\partial \vec{U}}$

$$\frac{\partial F}{\partial \vec{b}} = \underbrace{\boxed{}\boxed{\;\cdots\;}\boxed{}}_{\leftarrow N \rightarrow} \updownarrow 1 \qquad \frac{\partial F}{\partial \vec{U}} = \underbrace{\boxed{}\boxed{}\boxed{}\boxed{\;\cdots\;}\boxed{}\boxed{}\boxed{}}_{\leftarrow \quad \kappa \quad \rightarrow} \updownarrow 1$$



Example:

$$F = \oint_{airfoil} p\, dS = \sum p_i \Delta S_i$$

This is a naïve example without any physical meaning.

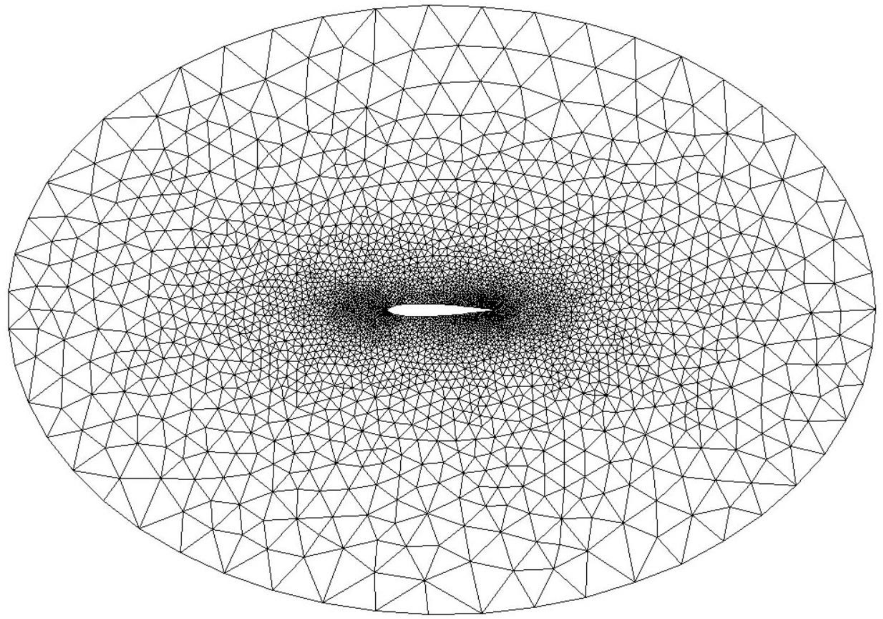
$$\frac{\partial F}{\partial U_\lambda} \approx \frac{\partial F}{\partial p_\lambda} = \Delta S_\lambda$$

$$\frac{\partial F}{\partial b_n} = \sum p_i \frac{\partial \Delta S_i}{\partial b_n}$$

Towards Discrete Adjoint (5/7)

Understanding $\frac{\delta \vec{U}}{\delta \vec{b}}$

$$\frac{\delta \vec{U}}{\delta \vec{b}} = \begin{array}{cc} \begin{array}{|c|c|c|}\hline & \cdots & \\ \hline & \cdots & \\ \hline & \cdots & \\ \hline & \vdots & \\ \hline & \cdots & \\ \hline & \cdots & \\ \hline & \cdots & \\ \hline \end{array} & \begin{array}{c} \uparrow \\ \\ \\ \kappa \\ \\ \\ \downarrow \end{array} \\ \leftarrow N \rightarrow & \end{array}$$



2D Flow Problem,
3469 nodes, 6787 triangles

$$\vec{U} = \begin{array}{cc} \begin{array}{|c|}\hline \\ \hline \\ \hline \\ \hline \vdots \\ \hline \\ \hline \\ \hline \\ \hline \end{array} & \begin{array}{c} \uparrow \\ \\ \\ \kappa \\ \\ \\ \downarrow \end{array} \\ 1 & \end{array}$$

κ= $\begin{cases} \textbf{3469x3 if 2D incompressible laminar flow} \\ \textbf{3469x4 if 2D compressible laminar flow} \end{cases}$

Towards Discrete Adjoint (6/7)

Understanding $\dfrac{\delta \overrightarrow{R}}{\delta \overrightarrow{b}}$ **and** $\dfrac{\partial \overrightarrow{R}}{\partial \overrightarrow{U}}$

$$\frac{\delta \overrightarrow{R}}{\delta \overrightarrow{b}} = \frac{\partial \overrightarrow{R}}{\partial \overrightarrow{b}} + \frac{\partial \overrightarrow{R}}{\partial \overrightarrow{U}} \frac{\delta \overrightarrow{U}}{\delta \overrightarrow{b}} = 0$$

$$\frac{\partial \overrightarrow{R}}{\partial \overrightarrow{b}} = \begin{array}{|c|c|c|c}
\hline
 & \cdots & & \uparrow \\ \hline
 & \cdots & & \\ \hline
 & \cdots & & \\ \hline
 & \vdots & & \kappa \\ \hline
 & \cdots & & \\ \hline
 & \cdots & & \\ \hline
 & \cdots & & \downarrow \\ \hline
\multicolumn{3}{c}{\leftarrow N \rightarrow} &
\end{array}$$

$$\frac{\partial \overrightarrow{R}}{\partial \overrightarrow{U}} = A = \begin{array}{|c|c|c|c|c|c|c|c}
\hline
 & & & \cdots & & & & \uparrow \\ \hline
 & & & \cdots & & & & \\ \hline
 & & & \cdots & & & & \\ \hline
 & & & \vdots & & & & \kappa \\ \hline
 & & & \cdots & & & & \\ \hline
 & & & \cdots & & & & \\ \hline
 & & & \cdots & & & & \downarrow \\ \hline
\multicolumn{7}{c}{\leftarrow \quad \kappa \quad \rightarrow} &
\end{array}$$

Towards Discrete Adjoint (7/7)

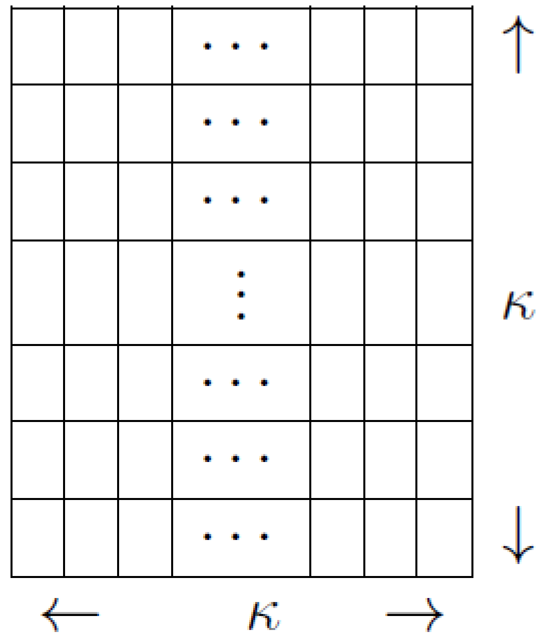
Have we seen $\dfrac{\partial \vec{R}}{\partial \vec{U}}$ **again?**

$$\vec{R}(\vec{U}) = 0$$

$$\left.\frac{d\vec{R}(\vec{U})}{d\vec{U}}\right|^{old} \Delta\vec{U} = -\vec{R}\,(\vec{U}^{old})$$

$$\vec{U}^{new} = \vec{U}^{old} + \Delta\vec{U}$$

$$\frac{\partial \vec{R}}{\partial \vec{U}} = A =$$



Jacobian! (However, here, you cannot use an easily invertible approximation to the real Jacobian!

Discrete DD

(1) Solve the N DD equations resulting from

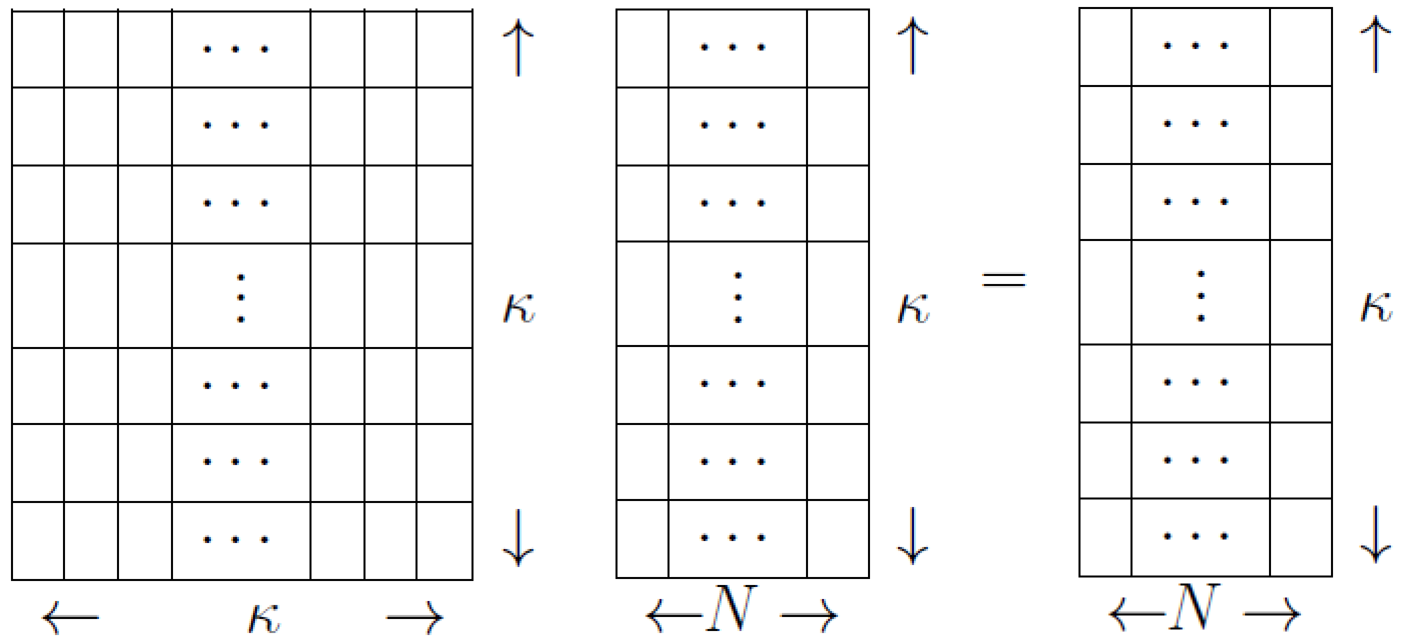
$$\frac{\delta \vec{R}}{\delta \vec{b}} = \frac{\partial \vec{R}}{\partial \vec{b}} + \frac{\partial \vec{R}}{\partial \vec{U}} \frac{\delta \vec{U}}{\delta \vec{b}} = 0$$

i.e.

$$\frac{\partial \vec{R}}{\partial \vec{U}} \frac{\delta \vec{U}}{\delta \vec{b}} = - \frac{\partial \vec{R}}{\partial \vec{b}}$$

(2) Compute grad(F):

$$\frac{\delta F}{\delta \vec{b}} = \frac{\partial F}{\partial \vec{b}} + \frac{\partial F}{\partial \vec{U}} \frac{\delta \vec{U}}{\delta \vec{b}}$$

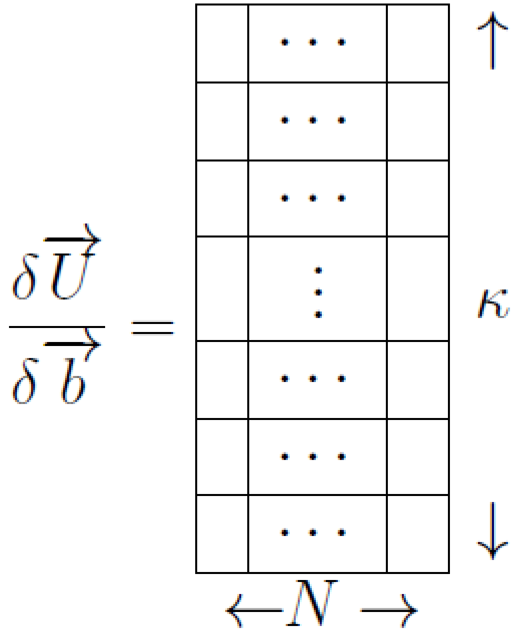


Compact representation of N independent linear systems κ x κ

Why Discrete Adjoint?

To avoid computing the derivatives of the flow variable w.r.t. the design variables (κxN matrix), i.e. avoid Direct Differentiation (DD).

Instead, use the Adjoint Method, either in its discrete or continuous form.



Discrete Adjoint (1/4)

$$\frac{\delta \vec{R}}{\delta \vec{b}} = \frac{\partial \vec{R}}{\partial \vec{b}} + \frac{\partial \vec{R}}{\partial \vec{U}}\frac{\delta \vec{U}}{\delta \vec{b}} = 0 \Rightarrow \frac{\delta \vec{U}}{\delta \vec{b}} = -\left(\frac{\partial \vec{R}}{\partial \vec{U}}\right)^{-1}\frac{\partial \vec{R}}{\partial \vec{b}}$$

$$\frac{\delta F}{\delta \vec{b}} = \frac{\partial F}{\partial \vec{b}} + \frac{\partial F}{\partial \vec{U}}\frac{\delta \vec{U}}{\delta \vec{b}} = \frac{\partial F}{\partial \vec{b}} - \frac{\partial F}{\partial \vec{U}}\left(\frac{\partial \vec{R}}{\partial \vec{U}}\right)^{-1}\frac{\partial \vec{R}}{\partial \vec{b}}$$

Set: $\boxed{\frac{\partial F}{\partial \vec{U}}\left(\frac{\partial \vec{R}}{\partial \vec{U}}\right)^{-1} = \vec{\Psi}^T} \Rightarrow \boxed{\left(\frac{\partial \vec{R}}{\partial \vec{U}}\right)^T \vec{\Psi} = \left(\frac{\partial F}{\partial \vec{U}}\right)^T}$

Field Adjoint Equations (FAE) in discrete form

Sensitivity Derivatives (SD) $\boxed{\frac{\delta F}{\delta \vec{b}} = \frac{\partial F}{\partial \vec{b}} - \vec{\Psi}^T\frac{\partial \vec{R}}{\partial \vec{b}}}$

Discrete Adjoint - Alternative Development (2/4)

Augmented Objective Function or Lagrangian:

$$F_{aug} = F - \vec{\Psi}^T \vec{R} \qquad \frac{\delta F_{aug}}{\delta \vec{b}} = \frac{\delta F}{\delta \vec{b}} - \vec{\Psi}^T \frac{\delta \vec{R}}{\delta \vec{b}}$$

$$\frac{\delta F_{aug}}{\delta \vec{b}} = \frac{\partial F}{\partial \vec{b}} + \frac{\partial F}{\partial \vec{U}}\frac{\delta \vec{U}}{\delta \vec{b}} - \vec{\Psi}^T \left(\frac{\partial \vec{R}}{\partial \vec{b}} + \frac{\partial \vec{R}}{\partial \vec{U}}\frac{\delta \vec{U}}{\delta \vec{b}} \right)$$

$$\frac{\delta F_{aug}}{\delta \vec{b}} = \underbrace{\frac{\partial F}{\partial \vec{b}} - \vec{\Psi}^T \frac{\partial \vec{R}}{\partial \vec{b}}} + \underbrace{\left(\frac{\partial F}{\partial \vec{U}} - \vec{\Psi}^T \frac{\partial \vec{R}}{\partial \vec{U}} \right)} \frac{\delta \vec{U}}{\delta \vec{b}}$$

Sensitivity Derivatives (SD)

$$\boxed{\frac{\delta F}{\delta \vec{b}} = \frac{\partial F}{\partial \vec{b}} - \vec{\Psi}^T \frac{\partial \vec{R}}{\partial \vec{b}}}$$

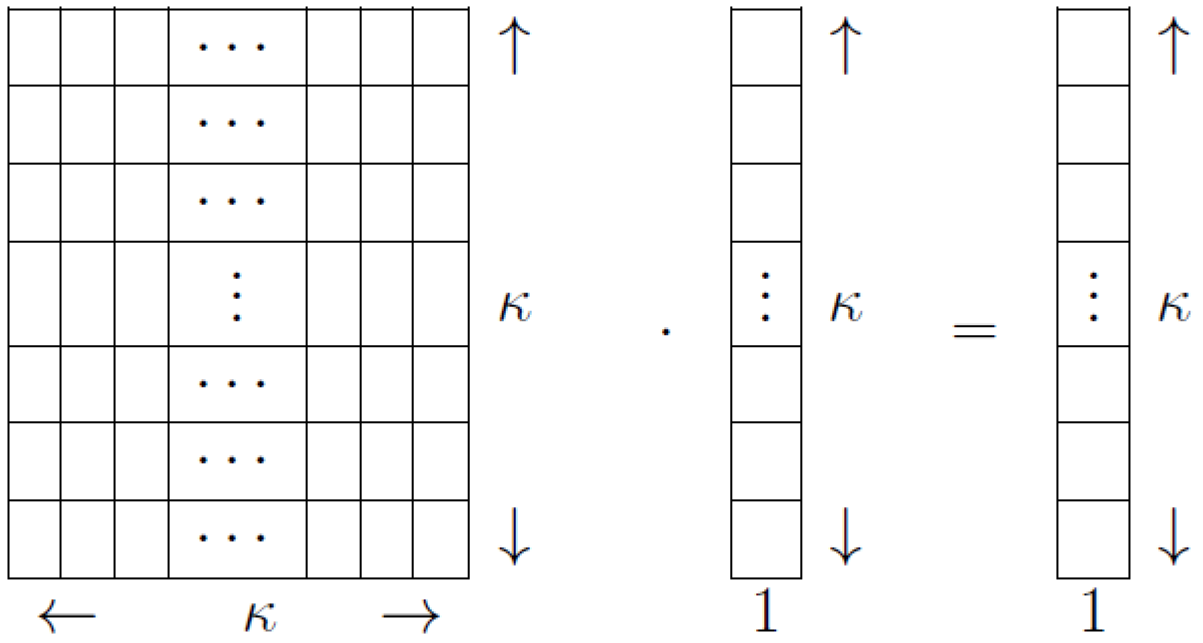
$$\boxed{\left(\frac{\partial \vec{R}}{\partial \vec{U}} \right)^T \vec{\Psi} = \left(\frac{\partial F}{\partial \vec{U}} \right)^T}$$

Field Adjoint Equations (FAE) in discrete form

Discrete Adjoint – Adjoint Equations (3/4)

Understanding the Discrete Adjoint Equation

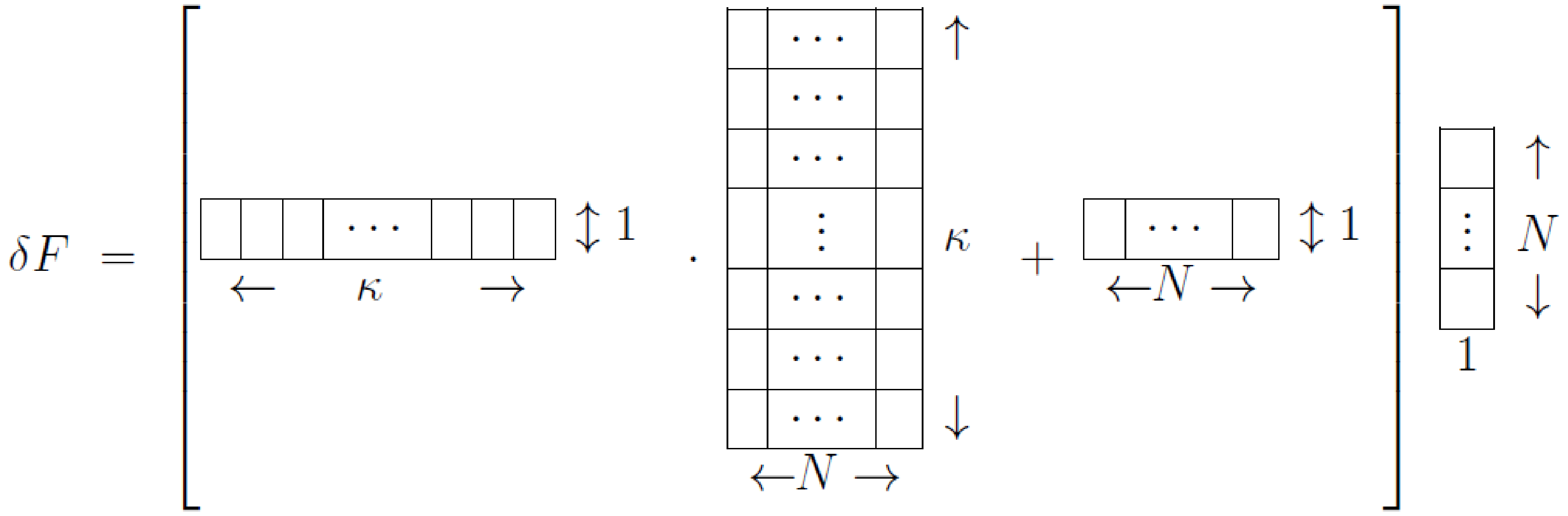
$$\left(\frac{\partial \overrightarrow{R}}{\partial \overrightarrow{U}}\right)^T \overrightarrow{\Psi} = \left(\frac{\partial F}{\partial \overrightarrow{U}}\right)^T$$



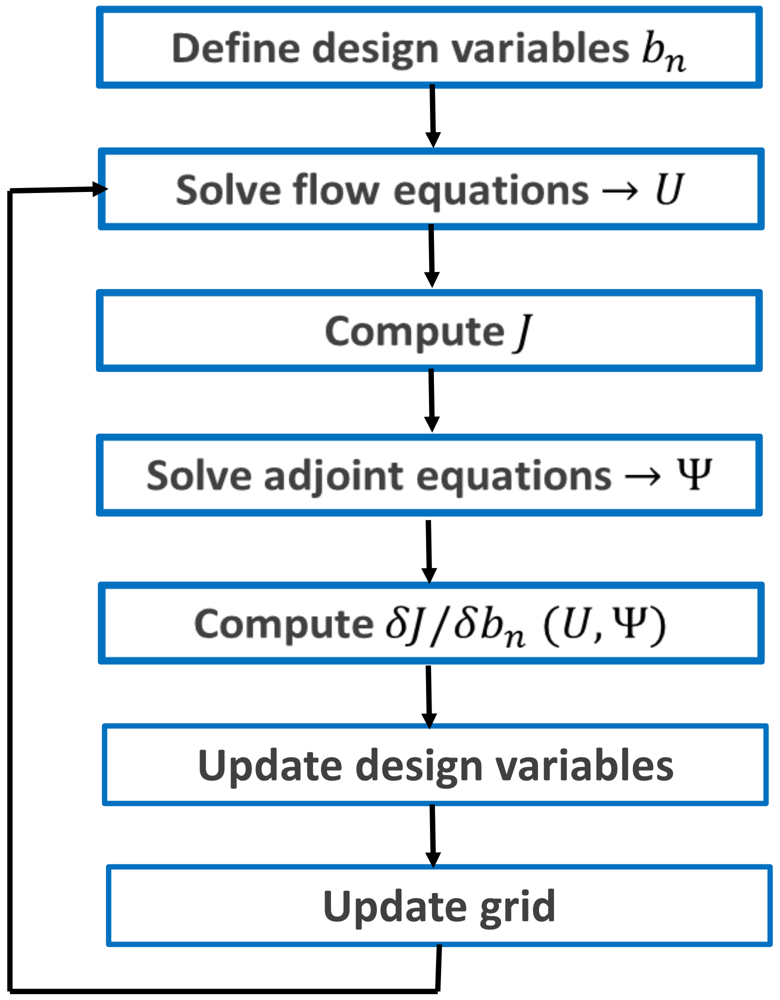
Discrete Adjoint – Sensitivity Derivatives (4/4)

$$\boxed{\frac{\delta F}{\delta \vec{b}} = \frac{\partial F}{\partial \vec{b}} - \vec{\Psi}^T \frac{\partial \vec{R}}{\partial \vec{b}}}$$

$$\delta F = \left(-\vec{\Psi}^T \frac{\partial \vec{R}}{\partial \vec{b}} + \frac{\partial F}{\partial \vec{b}} \right) \delta \vec{b}$$



Adjoint-based Optimisation Loop (for min. J)


Define design variables b_n
Solve flow equations $\rightarrow U$
Compute J
Solve adjoint equations $\rightarrow \Psi$
Compute $\delta J/\delta b_n \; (U,\Psi)$
Update design variables
Update grid

Computational Cost: 1 TU

Computational Cost: ~ 1TU

Using the Adjoint Method, the computational cost per optimisation cycle is equal to 2 TU,
IRRESPECTIVE OF THE NUMBER (N) OF THE DESIGN VARIABLES.

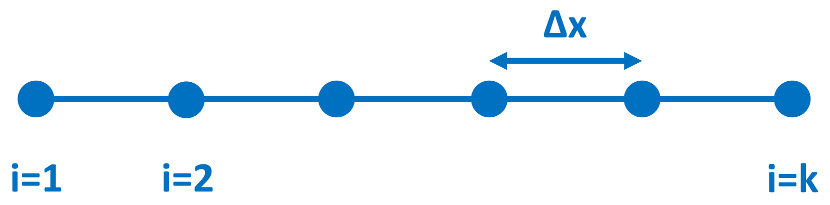
Example Revisited – The Primal Problem in Discrete Form (1/8)

Example: (Primal) ODE:

$$\frac{d^2 v}{dx^2} + b_1 v + b_2 v^2 - 5 = 0$$

$$0 \leq x \leq 1$$

Boundary conditions: $\begin{cases} v(0) = 1 \\ v(1) = 3 \end{cases}$



Discretisation using equidistant nodes.

Linearisation / Delta formulation: $(\ v_i^{new} = v_i^{old} + \Delta v_i\ )$

$$\frac{d^2\left(v^{old} + \Delta v\right)}{dx^2} + b_1\left(v^{old} + \Delta v\right) + b_2\left((v^{old})^2 + 2v^{old}\Delta v\right) - 5 = 0 \qquad 2 \leq i \leq k-1$$

or

$$\frac{d^2 \Delta v}{dx^2} + b_1 \Delta v + 2b_2 v^{old}\Delta v = -\frac{d^2 v^{old}}{dx^2} - b_1 v^{old} - b_2 (v^{old})^2 + 5$$

$$\boxed{\begin{array}{ll} i = 1 & \Delta v_1 = 0 \\ i = k & \Delta v_k = 0 \end{array}}$$

Example Revisited – The Primal Problem in Discrete Form (2/8)

$$\frac{d^2\Delta v}{dx^2} + b_1\Delta v + 2b_2 v^{old}\Delta v = -\frac{d^2 v^{old}}{dx^2} - b_1 v^{old} - b_2(v^{old})^2 + 5$$

$$\left[\frac{1}{\Delta x^2}\right]\Delta v_{i+1} + \left[b_1 + b_2 v_i^{old} - \frac{2}{\Delta x^2}\right]\Delta v_i + \left[\frac{1}{\Delta x^2}\right]\Delta v_{i+1} = -\frac{v_{i+1}^{old} - 2v_i^{old} + v_{i-1}^{old}}{\Delta x^2} - b_1 v_i^{old} - b_2(v^{old})^2 + 5$$

If:

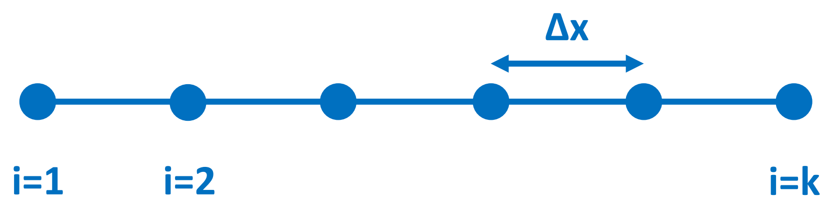
$$\alpha = \frac{1}{\Delta x^2} \qquad \gamma_i = b_1 + 2b_2 v_i^{old} - \frac{2}{\Delta x^2} = b_1 + 2b_2 v_i^{old} - 2\alpha$$

$$R_1 = v_1 - 1 = 0$$

$$R_6 = v_6 - 3 = 0$$

$$\boxed{\begin{bmatrix} 1 & & & & & \\ \alpha & \gamma_2 & \alpha & & & \\ & \alpha & \gamma_3 & \alpha & & \\ & & \alpha & \gamma_4 & \alpha & \\ & & & \alpha & \gamma_5 & \alpha \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} \Delta v_1 \\ \Delta v_2 \\ \Delta v_3 \\ \Delta v_4 \\ \Delta v_5 \\ \Delta v_6 \end{bmatrix} = -\begin{bmatrix} 0 \\ R_2^{old} \\ R_3^{old} \\ R_4^{old} \\ R_5^{old} \\ 0 \end{bmatrix}}$$

A matrix



(for k=6 nodes)

Example Revisited – Discrete Adjoint (3/8)

Objective Function (in discrete form):

$$F = \int_0^1 \upsilon(x)dx = \frac{\Delta x}{2}\upsilon_1 + \sum_2^5 \upsilon_i \Delta x + \frac{\Delta x}{2}\upsilon_6$$

$$\frac{\vartheta F}{\vartheta \upsilon_i} = \Delta x \quad \text{for} \quad 2 \le i \le 5$$

$$\frac{\vartheta F}{\vartheta \upsilon_1} = \frac{\vartheta F}{\vartheta \upsilon_6} = \frac{\Delta x}{2}$$

$$\frac{\vartheta F}{\vartheta \vec{b}} = [0 \quad 0] \quad \text{since} \quad \frac{\vartheta F}{\vartheta b_1} = \frac{\vartheta F}{\vartheta b_2} = 0$$

Example Revisited – Discrete Adjoint (4/8)

$$R_i = \alpha v_{i-1} + b_1 v_i + b_2 v_i^2 - 2\alpha v_i + \alpha v_{i+1} - 5 \neq 0 \qquad \alpha = \frac{1}{\Delta x^2}$$

$$\frac{\vartheta R_i}{\vartheta v_{i-1}} = \alpha \qquad \frac{\vartheta R_i}{\vartheta v_{i+1}} = \alpha \qquad \frac{\vartheta R_i}{\vartheta v_i} = b_1 + 2b_2 v_i - 2\alpha = \gamma_i$$

Discrete Adjoint Equation:

$$\boxed{\left(\frac{\vartheta \vec{R}}{\vartheta \vec{v}}\right)^T \vec{\Psi} = \left(\frac{\vartheta F}{\vartheta \vec{v}}\right)^T}$$

or

$$A^T \vec{\Psi} = \left(\frac{\vartheta F}{\vartheta \vec{v}}\right)^T$$

or

$$A^T \begin{bmatrix} \Psi_1 \\ \Psi_2 \\ \Psi_3 \\ \Psi_4 \\ \Psi_5 \\ \Psi_6 \end{bmatrix} = \begin{bmatrix} \Delta x/2 \\ \Delta x \\ \Delta x \\ \Delta x \\ \Delta x \\ \Delta x/2 \end{bmatrix}$$

Example Revisited – Discrete Adjoint (5/8)

$$R_i = \alpha v_{i-1} + b_1 v_i + b_2 v_i^2 - 2\alpha v_i + \alpha v_{i+1} - 5 \neq 0 \qquad \alpha = \frac{1}{\Delta x^2}$$

$$\frac{\vartheta R_i}{\vartheta b_1} = v_i \qquad \frac{\vartheta R_i}{\vartheta b_2} = v_i^2 \quad \text{for} \quad 2 \le i \le 5 \qquad \frac{\vartheta R_1}{\vartheta b_1} = \frac{\vartheta R_1}{\vartheta b_2} = 0 \qquad \frac{\vartheta R_6}{\vartheta b_1} = \frac{\vartheta R_6}{\vartheta b_2} = 0$$

Sensitivity Derivatives (SD):

$$\boxed{\frac{\delta F}{\delta \vec{b}} = \underbrace{\cancel{\frac{\vartheta F}{\vartheta \vec{b}}}}_{0} - \vec{\Psi}^T \frac{\vartheta \vec{R}}{\vartheta \vec{b}}}$$

or

$$\begin{bmatrix} \frac{\delta F}{\delta b_1} \\ \frac{\delta F}{\vartheta \delta} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}^{\mathrm{T}} - \begin{bmatrix} \Psi_1 & \Psi_2 & \Psi_3 & \Psi_4 & \Psi_5 & \Psi_6 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ v_2 & v_2^2 \\ v_3 & v_3^2 \\ v_4 & v_4^2 \\ v_5 & v_5^2 \\ 0 & 0 \end{bmatrix}$$

Example Revisited – Continuous Adjoint (6/8)

$$F_{aug} = F + \int_0^1 \Psi\left(\frac{d^2 \upsilon}{dx^2} + b_1 \upsilon + b_2 \upsilon^2 - 5\right) dx$$

$$\Rightarrow \delta F_{aug} = \delta \int_0^1 \upsilon dx + \int_0^1 \Psi\delta\left(\frac{d^2 \upsilon}{dx^2} + b_1 \upsilon + b_2 \upsilon^2 - 5\right) dx$$

$$\Rightarrow \delta F_{aug} = \int_0^1 \delta\upsilon dx + \underbrace{\int_0^1 \Psi \frac{d^2 \delta\upsilon}{dx^2} dx}_{\textbf{T1}} + \underbrace{\int_0^1 \Psi\delta(b_1 \upsilon) dx}_{\textbf{T2}} + \underbrace{\int_0^1 \Psi\delta(b_2 \upsilon^2) dx}_{\textbf{T3}}$$

Example Revisited – Continuous Adjoint (7/8)

⇨ $$T1\colon \int_0^1 \Psi \frac{d^2\delta\upsilon}{dx^2}dx = \int_0^1 \frac{d}{dx}\left(\Psi\frac{d\delta\upsilon}{dx}\right)dx - \int_0^1 \frac{d\Psi}{dx}\frac{d\delta\upsilon}{dx}dx =$$

$$= \left[\Psi\frac{d\delta\upsilon}{dx}\right]_0^1 - \int_0^1 \frac{d}{dx}\left(\delta\upsilon\frac{d\Psi}{dx}\right)dx + \int_0^1 \frac{d^2\Psi}{dx^2}\delta\upsilon dx =$$

$$= \left[\Psi\frac{d\delta\upsilon}{dx}\right]_0^1 - \left[\cancel{\delta\upsilon}\frac{d\Psi}{dx}\right]_0^1 + \int_0^1 \frac{d^2\Psi}{dx^2}\delta\upsilon dx$$

⇨ $$T2\colon \int_0^1 \Psi\delta(b_1\upsilon)dx = b_1\int_0^1 \Psi\delta\upsilon dx + \int_0^1 \Psi\upsilon\delta b_1 dx$$

⇨ $$T3\colon \int_0^1 \Psi\delta(b_2\upsilon^2)dx = 2b_2\int_0^1 \Psi\upsilon\delta\upsilon dx + \int_0^1 \Psi\upsilon^2\delta b_2 dx$$

Example Revisited – Continuous Adjoint (8/8)

$$\delta F_{aug} = \int_0^1 \delta v \left[\frac{d^2\Psi}{dx^2} + b_1\Psi + 2b_2 v\Psi + 1\right] dx + \left[\Psi \frac{d\delta v}{dx}\right]_0^1 + \int_0^1 \Psi v \delta b_1 dx + \int_0^1 \Psi v^2 \delta b_2 dx$$

Field Adjoint Equation (FAE):

$$\frac{d^2\Psi}{dx^2} + b_1\Psi + 2b_2 v\Psi + 1 = 0$$

Adjoint Boundary Conditions (ABC):

$$\Psi(x=0) = 0 \qquad \Psi(x=1) = 0$$

Sensitivity Derivatives (SD):

$$\frac{\delta F}{\delta b_1} = \Psi v \qquad \frac{\delta F}{\delta b_2} = \Psi v^2$$

Discrete Adjoint for other than Optimization Purposes

Compute $F = \vec{g}^T \vec{U}, \qquad \vec{g}, \vec{U} \in R^N$

where $A\vec{U} = \vec{q}, \qquad \vec{q} \in R^N, \quad A \in R^{N \times N}$

Direct Computation

Step 1: $A\vec{U} = \vec{q} \longrightarrow \vec{U}$

Step 2: $F = \vec{g}^T \vec{U} \longrightarrow F$

Adjoint Computation

Step 1: $A^T \vec{\Psi} = \vec{g} \longrightarrow \vec{\Psi}$

Step 2: $F = \vec{\Psi}^T \vec{q} \longrightarrow F$

Their Equivalence:

$$\vec{\Psi}^T \vec{q} = \vec{\Psi}^T \left(A\vec{U}\right) = \vec{\Psi}^T A \vec{U} = \left(A^T \vec{\Psi}\right)^T \vec{U} = \vec{g}^T \vec{U}$$

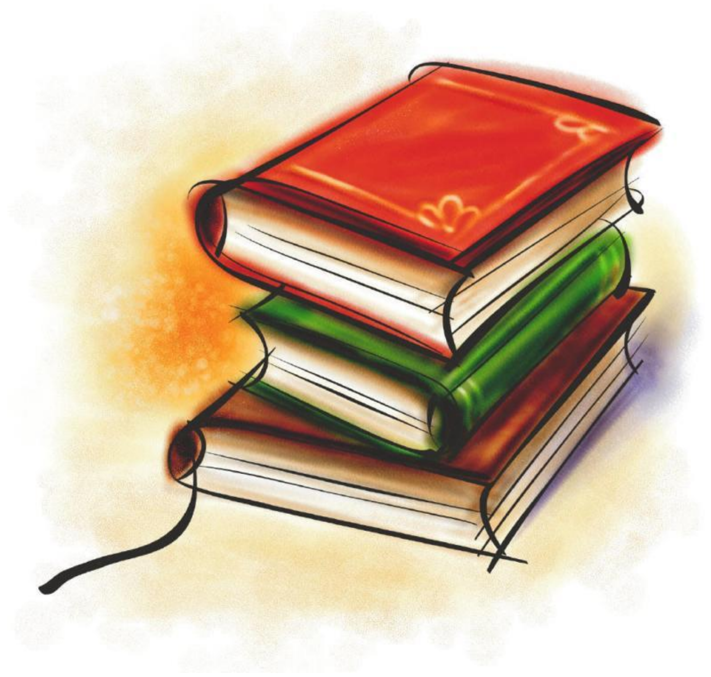
Benefits:

Assume we have m vectors $\vec{q}$

and n vectors $\vec{g}$

Use the **Direct Computation** if m<n.

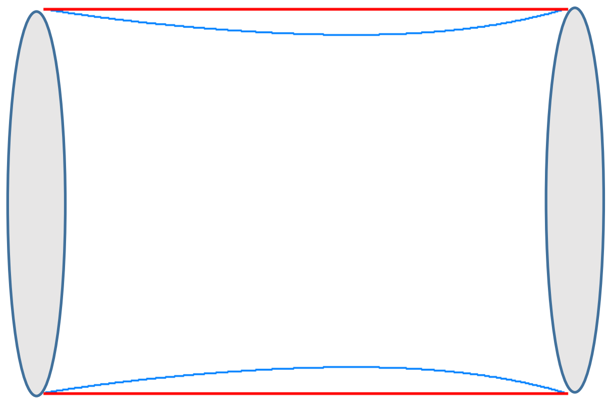
Use the **Adjoint Computation** if m>n.



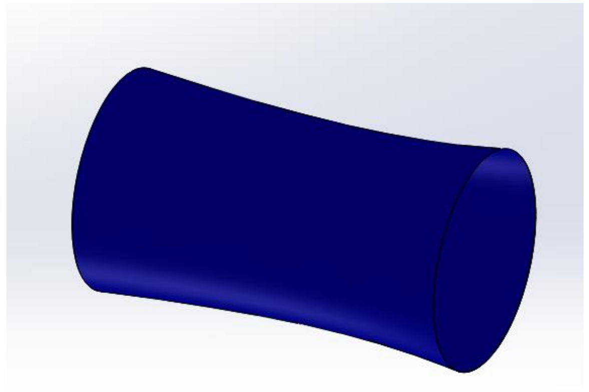
Inverse Design of a Quasi-1D Duct

Inverse Design of a Quasi-1D Duct – Incompressible Flow

Cross Section

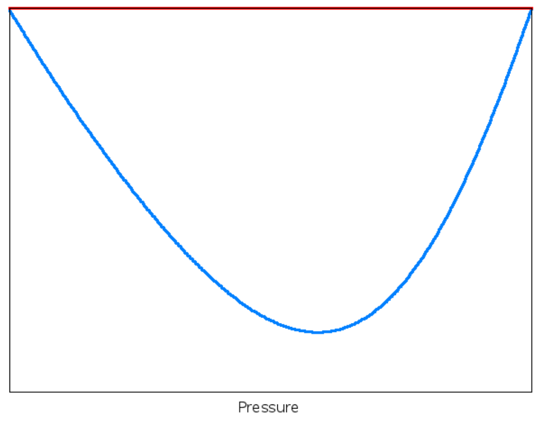


Target Duct

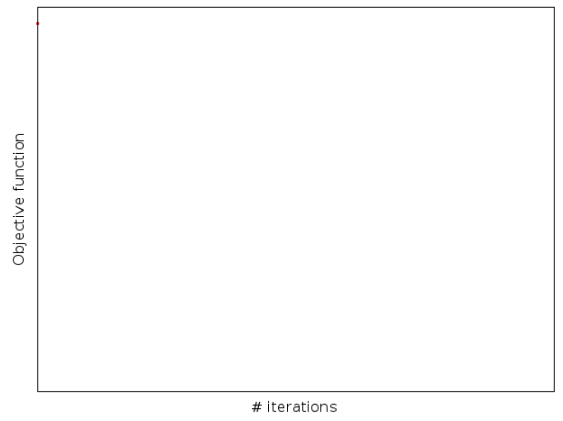


$$J = \frac{1}{2}\int_0^1 \left(p(x) - \underbrace{p_{tar}(x)}_{\text{given}}\right)^2 dx$$

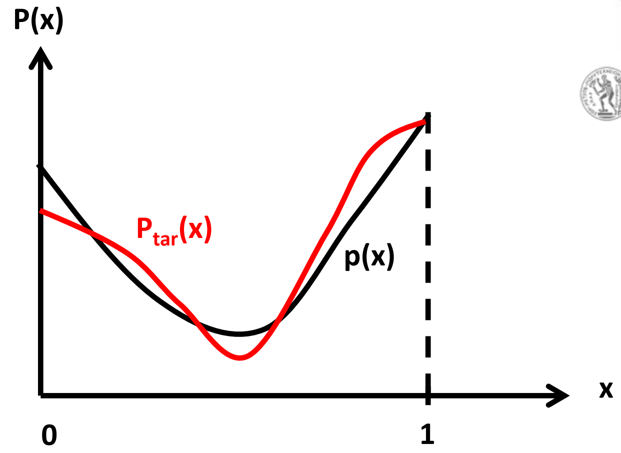
Pressure



Convergence History



Inverse Design of a Quasi-1D Duct – Incompressible Flow



Objective Function (min.):

$$J = \frac{1}{2}\int_0^1 \left(p(x) - p_{tar}(x)\right)^2 dx$$

Shape (Cross-Sectional Area) Parametrisation:

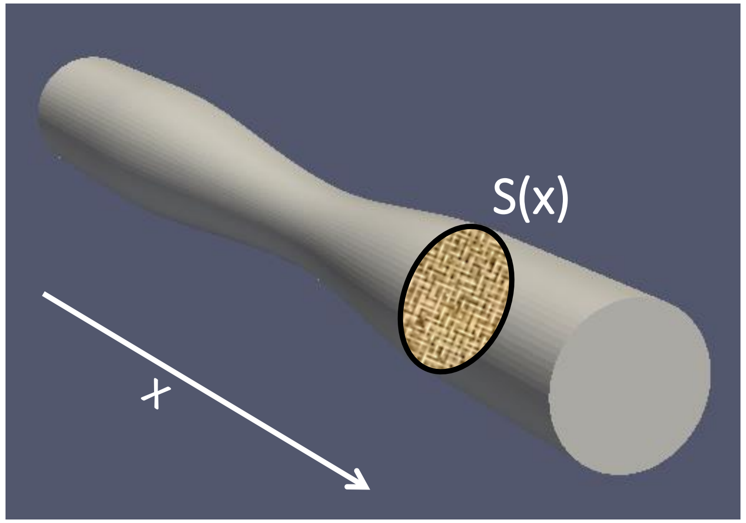
$$S(x, b_1, b_2, b_3, b_4) = b_1\left(-x^3+3x^2-3x+1\right) + b_2\left(3x^3-6x^2+3x\right) + b_3\left(-3x^3+3x^2\right) + b_4 x^3$$

Design/Optimisation Variables:

$$b_n,\ \ n = 1, 2, 3, 4$$

State/Primal/Flow Equations (ODEs):

$$\frac{d(vS)}{dx} = 0 \qquad v\frac{dv}{dx} + \frac{dp}{dx} = 0$$



State/Primal/Flow Boundary Conditions:

$$v|_{x=0} = v_0 \qquad p|_{x=1} = p_0$$

Inverse Design of a Quasi-1D Duct – Incompressible Flow

Differentiation of the objective function:

$$\frac{\delta J}{\delta b_n} = \int_0^1 \left(p(x) - p_{tar}(x)\right) \frac{\delta p}{\delta b_n} dx, \;\; n = 1, \ldots, N$$

Differentiation of the flow (primal or state) equations (<u>non-conservative</u>):

$$\left.\begin{aligned} &S\frac{d}{dx}\left(\frac{\delta v}{\delta b_n}\right) + \frac{dS}{dx}\frac{\delta v}{\delta b_n} + \frac{dv}{dx}\frac{\delta S}{\delta b_n} + v\frac{d}{dx}\left(\frac{\delta S}{\delta b_n}\right) = 0 \\ &\frac{\delta v}{\delta b_n}\frac{dv}{dx} + v\frac{d}{dx}\left(\frac{\delta v}{\delta b_n}\right) + \frac{d}{dx}\left(\frac{\delta p}{\delta b_n}\right) = 0 \end{aligned}\right\} \;\; n = 1, \ldots, N$$

Differentiation of the flow (primal or state) boundary conditions:

$$\left.\frac{\delta v}{\delta b_n}\right|_{x=0} = 0, \quad \left.\frac{\delta p}{\delta b_n}\right|_{x=1} = 0$$

Direct Differentiation (DD)

Adjoint is the art of computing $\frac{\delta J}{\delta b_n}$ without first computing $\frac{\delta v}{\delta b_n}$ and $\frac{\delta p}{\delta b_n}$.

Inverse Design of a Quasi-1D Duct – Incompressible Flow

Define & differentiate the augmented objective function or Lagrangian of J:

$$L = J + \int_0^1 q \frac{d(vS)}{dx} dx + \int_0^1 u \left[v\frac{dv}{dx} + \frac{dp}{dx} \right] dx$$

$$\frac{\delta L}{\delta b_n} = \frac{\delta J}{\delta b_n} + \int_0^1 q \frac{d}{dx}\left[\frac{\delta(vS)}{\delta b_n}\right] dx + \int_0^1 u \left[\frac{\delta v}{\delta b_n}\frac{dv}{dx} + v\frac{d}{dx}\left(\frac{\delta v}{\delta b_n}\right) + \frac{d}{dx}\left(\frac{\delta p}{\delta b_n}\right) \right] dx$$

where q and u are the adjoint pressure and velocity (1D) fields.
Integrate by parts:

$$\frac{\delta L}{\delta b_n} = -\int_0^1 \left[-u\frac{dv}{dx} + \frac{d(vu)}{dx} + S\frac{dq}{dx} \right] \frac{\delta v}{\delta b_n} dx + \int_0^1 \left(-\frac{du}{dx} + p - p_{tar} \right) \frac{\delta p}{\delta b_n} dx$$

$$- \int_0^1 v \frac{dq}{dx}\frac{\delta S}{\delta b_n} dx + \left[(vu + qS)\frac{\delta v}{\delta b_n} \right]_{x=0}^{x=1} + \left[u\frac{\delta p}{\delta b_n} \right]_{x=0}^{x=1} + \left[vq\frac{\delta S}{\delta b_n} \right]_{x=0}^{x=1}$$

Inverse Design of a Quasi-1D Duct – Incompressible Flow

Adjoint field equations:

$$\frac{du}{dx} = p - p_{tar} \qquad u\frac{dv}{dx} - \frac{d(vu)}{dx} - S\frac{dq}{dx} = 0$$

Adjoint boundary conditions:

$$u|_{x=0} = 0, \quad q|_{x=1} = -\left.\frac{vu}{S}\right|_{x=1}$$

Compare with the primal problem equations & boundary conditions:

$$\frac{d(vS)}{dx} = 0 \qquad v\frac{dv}{dx} + \frac{dp}{dx} = 0$$

$$v|_{x=0} = v_0 \qquad p|_{x=1} = p_0$$

Inverse Design of a Quasi-1D Duct – Incompressible Flow

Sensitivity derivatives:

$$\frac{\delta L}{\delta b_n}=\frac{\delta J}{\delta b_n}=-\int_0^1 v\frac{dq}{dx}\frac{\delta S}{\delta b_n}dx+vq\frac{\delta S}{\delta b_n}\bigg|_{x=1}-vq\frac{\delta S}{\delta b_n}\bigg|_{x=0}, \quad n=1,\ldots,4$$

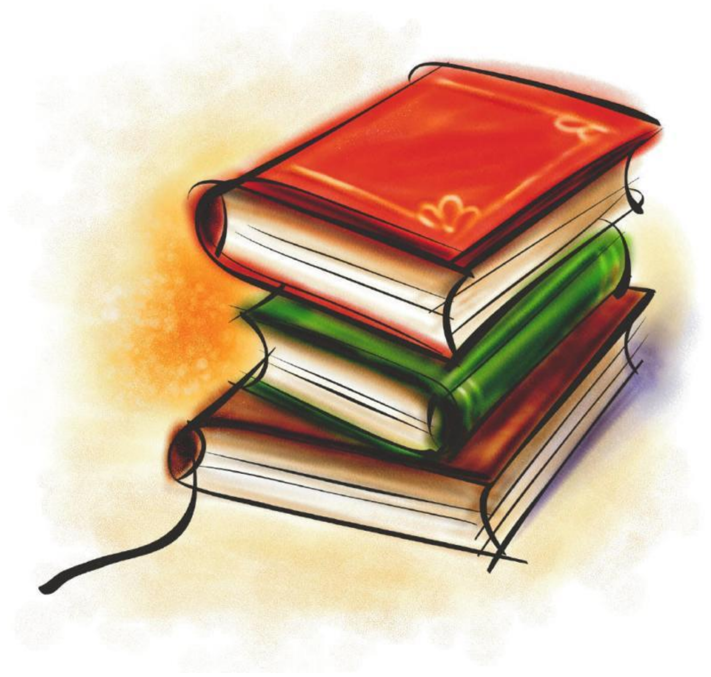
or:

$$\frac{\delta J}{\delta b_1}=-\int_0^1 v\frac{dq}{dx}\left(-x^3+3x^2-3x+1\right)dx-vq|_{x=0}$$

$$\frac{\delta J}{\delta b_2}=-\int_0^1 v\frac{dq}{dx}\left(3x^3-6x^2+3x\right)dx$$

$$\frac{\delta J}{\delta b_3}=-\int_0^1 v\frac{dq}{dx}\left(-3x^3+3x^2\right)dx$$

$$\frac{\delta J}{\delta b_4}=-\int_0^1 v\frac{dq}{dx}x^3dx+vq|_{x=1}$$



Computing the Hessian Matrix

Computation of Hessian (1/4)

WHY? Hessian matrix is needed either in Newton's method or in Robust-Design Optimisation (RDO), based on the Method of Moments. Differentiating w.r.t. the design or uncertain variables is irrelevant.

(Exact) Newton's Method:

$$\overrightarrow{b}^{new} = \overrightarrow{b}^{old} - \nabla^2 F(\overrightarrow{b}^{old})^{-1} \nabla F(\overrightarrow{b}^{old})^T$$

The most efficient approach is: DD-AV.
(DD to compute the gradient, AV=adjoint to compute the Hessian).

Computation of Hessian (2/4)

Hessian Matrix Computation using the DD-DD method:

$$\frac{dF}{db_i} = \frac{\partial F}{\partial b_i} + \frac{\partial F}{\partial U_k}\frac{dU_k}{db_i}$$

$$\frac{d^2 F}{db_i db_j} = \frac{\partial^2 F}{\partial b_i \partial b_j} + \frac{\partial^2 F}{\partial b_i \partial U_k}\frac{dU_k}{db_j} + \frac{\partial^2 F}{\partial U_k \partial b_j}\frac{dU_k}{db_i} + \frac{\partial^2 F}{\partial U_k \partial U_m}\frac{dU_k}{db_i}\frac{dU_m}{db_j} + \frac{\partial F}{\partial U_k}\frac{d^2 U_k}{db_i db_j}$$

k=1,…,N design variables

$$\frac{dR_m}{db_i} = \frac{\partial R_m}{\partial b_i} + \frac{\partial R_m}{\partial U_k}\frac{dU_k}{db_i} = 0$$

$$\frac{d^2 R_n}{db_i db_j} = \frac{\partial^2 R_n}{\partial b_i \partial b_j} + \frac{\partial^2 R_n}{\partial b_i \partial U_k}\frac{dU_k}{db_j} + \frac{\partial^2 R_n}{\partial U_k \partial b_j}\frac{dU_k}{db_i} + \frac{\partial^2 R_n}{\partial U_k \partial U_m}\frac{dU_k}{db_i}\frac{dU_m}{db_j} + \frac{\partial R_n}{\partial U_k}\frac{d^2 U_k}{db_i db_j} = 0$$

The cost for computing the Hessian via the DD-DD approach scales with N^2.

Computation of Hessian (3/4)

$$\frac{dR_m}{db_i}=\frac{\partial R_m}{\partial b_i}+\frac{\partial R_m}{\partial U_k}\frac{dU_k}{db_i}=0 \quad \Rightarrow \quad \boxed{\frac{dU_k}{db_i}} \qquad \boxed{N}\ \text{TU}$$

$$\frac{\partial F}{\partial U_k}+\hat{\Psi}_n\frac{\partial R_n}{\partial U_k}=0 \quad \Rightarrow \quad \boxed{\hat{\Psi}_n} \qquad \boxed{1}\ \text{TU}$$

The Adjoint equation is <u>the same</u> to that used to compute Grad(F) !!!

$$\begin{aligned}\frac{d^2\hat{F}}{db_idb_j}=&\frac{\partial^2 F}{\partial b_i\partial b_j}+\hat{\Psi}_n\frac{\partial^2 R_n}{\partial b_i\partial b_j}+\frac{\partial^2 F}{\partial U_k\partial U_m}\frac{dU_k}{db_i}\frac{dU_m}{db_j}+\hat{\Psi}_n\frac{\partial^2 R_n}{\partial U_k\partial U_m}\frac{dU_k}{db_i}\frac{dU_m}{db_j}\\&+\frac{\partial^2 F}{\partial b_i\partial U_k}\frac{dU_k}{db_j}+\hat{\Psi}_n\frac{\partial^2 R_n}{\partial b_i\partial U_k}\frac{dU_k}{db_j}+\frac{\partial^2 F}{\partial U_k\partial b_j}\frac{dU_k}{db_i}+\hat{\Psi}_n\frac{\partial^2 R_n}{\partial U_k\partial b_j}\frac{dU_k}{db_i}\\&+\cancel{\left(\frac{\partial F}{\partial U_k}+\hat{\Psi}_n\frac{\partial R_n}{\partial U_k}\right)}\frac{d^2U_k}{db_idb_j}\end{aligned}$$

$$\frac{d^2F^\lambda}{db_idb_j}db_j^\lambda=-\frac{dF^\lambda}{db_i}$$

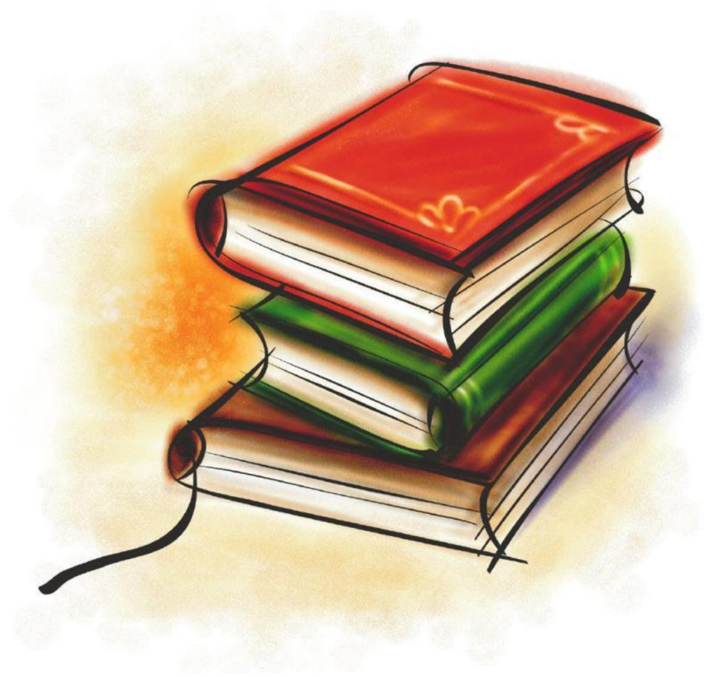
- ▶ **The cost per Newton cycle is N+1+1=N+2 EFS! Scales with N, not N2.**
- ▶ **DD-AV is the most efficient approach (among DD-DD, AV-DD, AV-AV)!**

Computation of Hessian (4/4)

Though the development was presented for the Discrete Adjoint Method, a similar development can be carried out for the Continuous Adjoint Method.

Read more in:

- D.I. Papadimitriou and K.C. Giannakoglou, “Direct, adjoint and mixed approaches for the computation of hessian in airfoil Design Problems”, International Journal for Numerical Methods in Fluids 2007; 56(10):1929-1943.
- D.I. Papadimitriou and K.C. Giannakoglou, “Computation of the Hessian matrix in aerodynamic inverse design using continuous adjoint formulations”, Computers & Fluids 2008; 37(8):1029-1039.
- D.I. Papadimitriou and K.C. Giannakoglou, “The continuous direct-adjoint approach for second order sensitivities in viscous aerodynamic inverse design problems”, Computers & Fluids 2009; 38(8):1539-1548.



The Think-Discrete Do-Continuous (TDDC) Adjoint

The Think-Discrete Do-Continuous (TDDC) Adjoint – Why?

- **Continuous Adjoint (First differentiate-then discretize)**
- **Discrete Adjoint (First discretize-then differentiate)**

Adjoint	Physical Insight	Low Memory Footprint	Consistency	Possible Modifications
Continuous	✓	✓	✗	✓
Discrete	✗	✗	✓	✗

The Think-Discrete Do-Continuous (TDDC) Adjoint – Why?

- **Continuous Adjoint (First differentiate-then discretize)**
- **Discrete Adjoint (First discretize-then differentiate)**

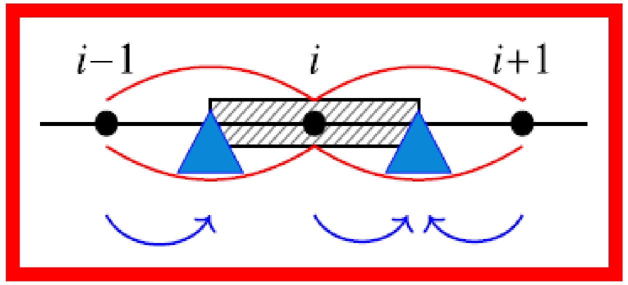
Adjoint	Physical Insight	Low Memory Footprint	Consistency	Possible Modifications
Continuous	✓	✓	✗	✓
Discrete	✗	✗	✓	✗
Think-Discrete Do-Continuous	✓	✓	✓	✓

The *TDDC* Adjoint bridges the gap between the two adjoint variants and combines the best of both worlds. The concept is to develop consistent discretisation schemes for the continuous adjoint equations, inspired by (hand-differentiated) discrete adjoint.

The Think-Discrete Do-Continuous (TDDC) Adjoint – The Concept (1/4)

Example: a VERY SIMPLE term in a nonlinear 1D ODE:

❑ **Flow Equation & FV Discretisation:** $$\int_{i-1/2}^{i+1/2} \boxed{\frac{du^2}{dx}} dx \simeq u_{i+1/2} u_i^{\text{upwind}} - u_{i-1/2} u_{i-1}^{\text{upwind}}$$



❑ **First-order Upwind Discretisation →**

$$\frac{1}{2}(u_i + u_{i+1})\, u_i - \frac{1}{2}(u_i + u_{i-1})\, u_{i-1}$$

❑ **Second-order Upwind Discretisation →**

$$\frac{1}{2}(u_i + u_{i+1})\left(u_i + \frac{1}{4}u_{i+1} - \frac{1}{4}u_{i-1}\right) - \frac{1}{2}(u_i + u_{i-1})\left(u_{i-1} + \frac{1}{4}u_i - \frac{1}{4}u_{i-2}\right)$$

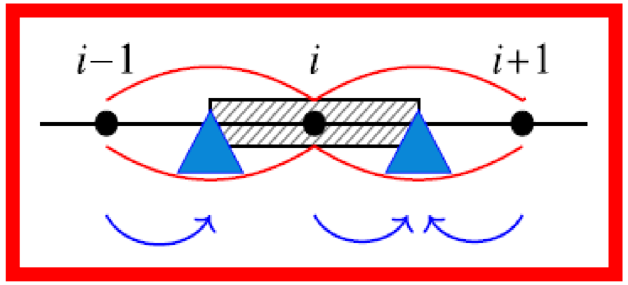
The Think-Discrete Do-Continuous (TDDC) Adjoint – The Concept (2/4)

❑ **Continuous adjoint; corresponding term:**

$$\boxed{\frac{du^2}{dx} \longrightarrow -2u\frac{d\Psi}{dx}}$$

❑ **"Standard" First-order Downwind Discretisation →**

$$\mathcal{I}_i = -2\int_{i-1/2}^{i+1/2} u\frac{d\Psi}{dx}dx \simeq -2u_i\left(\Psi_{i+1}-\Psi_i\right)$$



❑ **Hand-Differentiated Discrete Adjoint →**

First-order:

$$\mathcal{I}_i \simeq \frac{1}{2}\Psi_{i-1}u_{i-1} - \frac{1}{2}\Psi_i u_{i-1} + \Psi_i u_i + \frac{1}{2}\Psi_i u_{i+1} - \Psi_{i+1}u_i - \frac{1}{2}\Psi_{i+1}u_{i+1}$$

Second-order:

$$\mathcal{I}_i \simeq -\frac{1}{8}\Psi_{i-1}u_{i-2} + \frac{5}{8}\Psi_{i-1}u_{i-1} + \frac{1}{4}\Psi_{i-1}u_i + \frac{1}{8}\Psi_i u_{i-2} - \frac{6}{8}\Psi_i u_{i-1} + \frac{3}{4}\Psi_i u_i + \frac{5}{8}\Psi_i u_{i+1}$$
$$+\frac{1}{8}\Psi_{i+1}u_{i-1} - \Psi_{i+1}u_i - \frac{6}{8}\Psi_{i+1}u_{i+1} - \frac{1}{8}\Psi_{i+1}u_{i+1} + \frac{1}{8}\Psi_{i+2}u_{i+1} + \frac{1}{8}\Psi_{i+2}u_{i+2}$$

The Think-Discrete Do-Continuous (TDDC) Adjoint – The Concept (3/4)

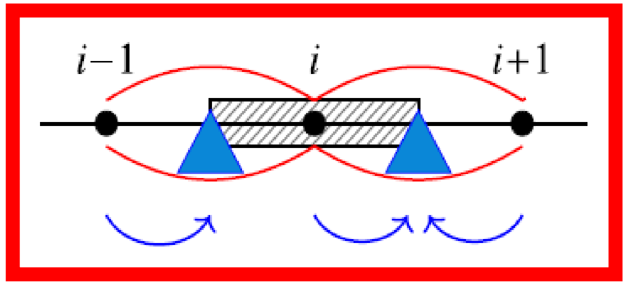
❑ **TDDC adjoint** (Downwind) Discretisation:

$$\mathcal{I}_i = -2\int_{i-1/2}^{i+1/2} u\frac{d\Psi}{dx}dx \longrightarrow \boxed{\mathcal{I}_i \simeq -2\left(\mathrm{a}\,Q_{i-1/2} + \mathrm{b}\,Q_{i+1/2} + \mathrm{c}\,Q_{i+3/2}\right)}$$

With

$$Q_{\lambda+1/2} = \bar{u}_{\lambda+1/2}\left.\frac{d\Psi}{dx}\right|_{\lambda+1/2}\Delta x = \bar{u}_{\lambda+1/2}\frac{\Psi_{\lambda+1}-\Psi_{\lambda}}{\Delta x}\Delta x$$

Downwind effect!



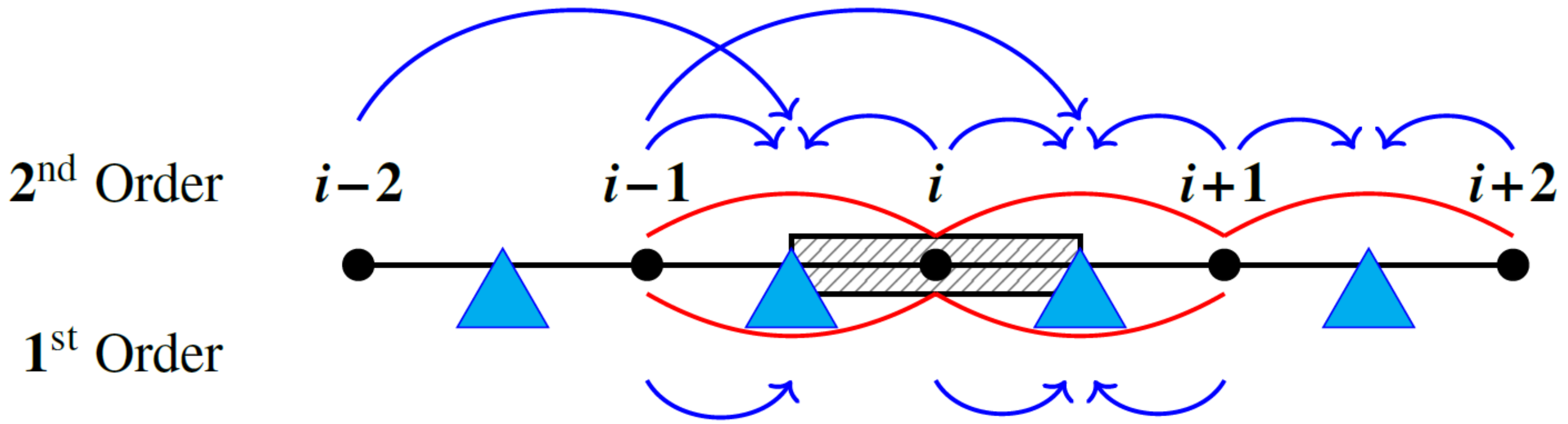
❑ **Consistent TTDC Adjoint** Discretisation:

	a	b	c	$\bar{u}_{i-1/2}$	$\bar{u}_{i+1/2}$	$\bar{u}_{i+3/2}$
1st Order	1/4	3/4	0	u_{i-1}	$\frac{1}{3}(2u_i + u_{i+1})$	0
2nd Order	3/8	6/8	−1/8	$\frac{1}{6}(-u_{i-2} + 5u_{i-1} + 2u_i)$	$\frac{1}{12}(-u_{i-1} + 8u_i + 5u_{i+1})$	$\frac{1}{2}(u_{i+1} + u_{i+2})$

!

The Think-Discrete Do-Continuous (TDDC) Adjoint – The Concept (4/4)

❑ **Graphical Interpretation of the TDDC Discretisation:**



First-order: Red curves indicate the central scheme for the gradients of Ψ and blue arrows contributions of nodal u values to the midnodes for first- (low) and second-order (up) schemes.

TDDC Adjoint:

$$\mathcal{I}_i = -2\int_{i-1/2}^{i+1/2} u\frac{d\Psi}{dx}dx \simeq -2\left[\frac{1}{4}u_{i-1}\left(\Psi_i-\Psi_{i-1}\right)-\frac{3}{4}\frac{2u_i+u_{i+1}}{3}\left(\Psi_{i+1}-\Psi_i\right)\right]$$

First-Order:

Standard Continuous:

$$\mathcal{I}_i = -2\int_{i-1/2}^{i+1/2} u\frac{d\Psi}{dx}dx \simeq -2u_i\left(\Psi_{i+1}-\Psi_i\right)$$

Read more in:

- M.G. Kontou, “The Continuous Adjoint Method with Consistent Discretization Schemes for Transitional Flows and the Use of Deep Neural Networks in Shape Optimization in Fluid Mechanics”, PhD Thesis, National Technical University of Athens, 2023.
- M.G. Kontou, V.G. Asouti, X.S. Trompoukis, K.C. Giannakoglou, “Consistently discretized continuous adjoint equations: The Think-Discrete Do-Continuous adjoint”, Computer Methods in Applied Mechanics and Engineering 2026; 449:118529.