Using Multicomplex Variables for Automatic Computation of High-Order Derivatives

Gregory Lantoine
Mission Design Engineer JPL
(previously School of Aerospace Engineering, Georgia Tech)

Ryan P. Russell
Assistant Professor, The University of Texas at Austin
(previously School of Aerospace Engineering, Georgia Tech)

Thierry Dargent *
Research & development directorate, Thales Alenia Space

13rd European Workshop on Automatic Differentiation
Monday 10th and Tuesday 11th June 2013
INRIA Sophia-Antipolis, France
Motivations

- Working on 2\textsuperscript{nd} order optimal control methods... (HDDP- variant of differential dynamic programming)

- Frustrated with expense of derivative calculations...
  - Analytic... tedious to code or not always possible
  - Finite differencing... easy but inaccurate, slow
  - Automatic Differentiation... not straightforward, slow
  - New complex step method (Squire & Trapp 1998, Martens 2003) ...accurate for first order derivs only

- **OBJECTIVE**: *Extend complex method to higher order derivatives*
Motivations

- Sensitivity Analysis
  - Partial Derivatives of outputs w.r.t. inputs

\[ \nabla_X f = \begin{bmatrix}
  \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\
  \vdots & \ddots & \vdots \\
  \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n}
\end{bmatrix} \]

- Computations of Sensitivities highly desirable in many fields:
  - Design Optimization: gradient-based on satellite trajectory optimization, and optimal control
  - Inverse Problem (Data assimilation)
  - Curve fitting
  - Parameter identification
  - Nonlinear PDEs
Requirements of Sensitivity Computations

in order of importance (to us):

1. Accurate
   - Improvement of algorithm convergence
   - Compute adjoint state dynamic with the same précision as the state dynamic

2. Fast (enough)

3. Easy-to-implement
   - Low setup time for one problem
   - Generalized for different problems
Existing methods

• **Analytical (Manual) differentiation**
  - *Can be* accurate and efficient (depends on the programmer)
  - Knowledge of computer language needed for model implementation
  - Development time is long
  - Error prone
  - Maintaining derivatives an additional burden
  - Difficult on large numerical model

• **Symbolic differentiation**
  - Implies using software for symbolic manipulation such as *Maple or Mathematica*
  - Reduces model development time
  - Reduces errors associated with mathematical manipulations
  - Still requires human efforts for further model implementation
  - Might lead to non-efficient expressions
Existing methods

- **Finite Differencing**

  Central second-order accurate difference:

  \[
  \frac{\partial F}{\partial x_i} \bigg|_{x=x_0} \approx \frac{F(x_0 + he_i) - F(x_0 - he_i)}{2h}
  \]

  Error introduced

  - Easiest method to implement: only original computer program is required → Development time is minimal
  - Accuracy is step dependent
  - Computationally intensive: N derivatives will require N+1 function evaluations
  → Often inaccurate or inefficient

  ![Graph showing step size and function behavior](image)

  Step size?
  Large: Averaging
  Small: round off error in subtraction

  ~dfdx
  dfdx

  f

  x
**Existing methods**

- **Automatic Differentiation**
  - Analytic differentiation of elementary functions
  - Repetitive application of the *chain rule*
  - Can be implemented in two ways:
    - Source transformation: Produce new code to calculate derivative based on original code of a function (ex: ADIFOR, TAPENADE, ...)
    - Operator overloading: Each elementary operation is replaced by a new one, working on pairs of value and its derivative (doublet) (ex: AD02 from the Harwell Subroutine Library)
  - Derivatives of any order
  - Accurate to machine precision: No round-off errors
  - Complete Partial Derivatives with single execution
  - Computationally more efficient than FD
  - Not straightforward method to implement
**Existing methods**

**Complex-step method** (Squire & Trapp 1998, Martens 2003)
- Use complex variables instead of real variables
- Found from Taylor Series expansion:
  \[ f(x + ih) = f(x) + \frac{df}{dx} ih + O(h^2) \]
  \[ \frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{\text{Im}[f(x + ih)]}{h} \]
  - Avoid the subtraction involved in finite difference
  - Accurate to working precision for VERY small \( h < 10^{-8} \)
  - Very easy implementation
  - Separate simulations for each gradient required (like finite differencing)
  - Increased computational time due to complex arithmetic
  - Limited to first-order derivatives
    - (Lai, Crassidis: give tuning methods to find optimal step size for 2nd order approximations, still not machine accurate...)

No subtraction!
MultiComplex Numbers

- MultiComplex numbers ($C^n$) are extensions of complex numbers in higher dimensions.

\[ z = x_0 + x_1i_1 + \ldots + x_ni_n + x_{12}i_1i_2 + \ldots + x_{n-1n}i_{n-1}i_n + \ldots + x_{1\ldots n}i_1\ldots i_n \]

- Same formal representation as complex numbers can be used: $z = z_1 + z_2i_n$ where $z \in C^n$ and $z_1, z_2 \in C^{n-1}$
**MultiComplex Numbers**

- Example: bicomplex numbers

\[
z = x_0 + x_1 i_1 + x_2 i_2 + x_{12} i_1 i_2 \quad \text{where } x_1, x_2, x_3, x_4 \in \mathbb{R}, i_1^2 = -1, i_2^2 = -1, i_1 i_2 = i_2 i_1
\]

\[
z = z_1 + z_2 i_2 \quad \text{where } z_1, z_2 \in \mathbb{C}, i_2^2 = -1
\]

- Example: tricomplex numbers

\[
z = x_0 + x_1 i_1 + x_2 i_2 + x_3 i_3 + x_{12} i_1 i_2 + x_{13} i_1 i_3 + x_{23} i_2 i_3 + x_{123} i_1 i_2 i_3
\]

\[
z = z_1 + z_2 i_3 \quad \text{where } z_1, z_2 \in \mathbb{C}^2, i_3^2 = -1
\]

imaginary terms can be represented as matrix operator example bicomplex \( i_1 \):

\[
i_1 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}
\]
Example 2\textsuperscript{nd} Derivative From Taylor Series

\[ f\left[ x + h(i_1 + i_2) \right] = f(x) + h(i_1 + i_2) f'(x) + \frac{h^2}{2} (i_1 + i_2)^2 f''(x) + H.O.T. \text{ (ignore...)} \]

\[ = f(x) + h(i_1 + i_2) f'(x) + \frac{h^2}{2} (i_1i_1 + 2i_1i_2 + i_2i_2) f''(x) \]

\[ = f(x) + h(i_1 + i_2) f'(x) + \frac{h^2}{2} (-2 + 2i_1i_2) f''(x) \]

\[ = f(x) + h(i_1 + i_2) f'(x) + h^2 (i_1i_2) f''(x) - h^2 f''(x) \]

take now the coefficient of \( i_1i_2 \) from both sides, divide by \( h^2 \)

\[ f''(x) = \frac{\text{Im}_{12} \left( f\left[ x + h(i_1 + i_2) \right] \right)}{h^2} + O\left( h^2 \right) \]

we choose \( h \) extremely small, say \( 1e-100 \) and there is no subtraction error

\( (h^2 \text{ must be representable with double precision}) \)
MultiComplex-Step Differentiation

- Fundamental formula:

\[ f^{(n)}(x) = \frac{\text{Im}_{1 \ldots n} \left[ f(x + hi_1 + \ldots + hi_n) \right]}{h^n} + O(h^2) \]

- Use multicomplex variables instead of real variables
- Found from Taylor Series expansion
- Derivatives up to any order \( n \)
- Same other advantages as complex method
- See paper for mathematical formalities:
  - Using Multicomplex Variables for Automatic Computation of High-Order Derivatives, Gregory Lantoine, Ryan P. Russell & Thierry Dargent, ACM Transactions on Mathematical Software (TOMS) Volume 38 Issue 3, April 2012, Article No. 16
- Details:
  - Step with \( h \) in each of the imaginary directions
  - Evaluate multi-complex function
  - Resulting derivatives retrieved from the coefficients of the multi-complex function result
Example Third Derivative Calculation

Multiple variables: \((x,y,z)\)

\[
\frac{\partial^3 f}{\partial x \partial y \partial z} = \text{Im}_{123}\left( f\left[ x + hi_1, y + hi_2, z + hi_3 \right] \right)
\]

\[
\frac{\partial^3 f}{\partial x \partial y \partial y} = \text{Im}_{123}\left( f\left[ x + hi_1, y + hi_2 + hi_3, z \right] \right)
\]

\[
\frac{\partial^3 f}{\partial y \partial y \partial y} = \text{Im}_{123}\left( f\left[ x, y + hi_1 + hi_2 + hi_3, z \right] \right)
\]
MultiComplex-Step Differentiation

- simple example:

\[ f(x) = \frac{e^x}{\sqrt{\sin(x)^3 + \cos(x)^3}} \]

1st Derivative

<table>
<thead>
<tr>
<th>Normalized error</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^5</td>
</tr>
<tr>
<td>10^0</td>
</tr>
<tr>
<td>10^-5</td>
</tr>
<tr>
<td>10^-10</td>
</tr>
<tr>
<td>10^-15</td>
</tr>
<tr>
<td>10^-20</td>
</tr>
</tbody>
</table>

ACCURACY:
- MCX and CX: ~16 digits large range
- FD: ~10 digits, requires tuning

h~1e-8
MultiComplex-Step Differentiation

- simple example:

\[ f(x) = \frac{e^x}{\sqrt{\sin(x)^3 + \cos(x)^3}} \]

2nd Derivative

ACCURACY:
MCX: ~16 digits
large range

CX and FD:
~7 digits, requires tuning
**MultiComplex-Step Differentiation**

- simple example: \[ f(x) = \frac{e^x}{\sqrt{\sin(x)^3 + \cos(x)^3}} \]

![Graph showing 3rd Derivative](image)

**ACCURACY:**
- MCX: ~16 digits
- Large range
- CX and FD: ~5-7 digits, requires tuning
Implementation

Declare all the variables as complex type

Or only the independent variables and the associated intermediate variables
**Implementation**

- Declare all the variables as complex type
- Redefine all the functions with complex definitions

Or only the independent variables and the associated intermediate variables

Easy implementation using the same form of operator overloading functions as complex numbers
Implementation

- Declare all the variables as complex type
- Redefine all the functions with complex definitions
- Add complex step (h) to the desired variable

Or only the independent variables and the associated intermediate variables

Easy implementation using the same form of operator overloading functions as complex numbers

\[ x + hi_1 + ... + hi_n \]
Implementation

- Declare all the variables as complex type
- Redefine all the functions with complex definitions
- Add complex step (h) to the desired variable

Partial Derivative Calculations

\[
f^{(n)}(x) \approx \text{Im}_{1...n}\left[\frac{f(x + hi_1 + ... + hi_n)}{h^n}\right]
\]

Or only the independent variables and the associated intermediate variables

Easy implementation using the same form of operator overloading functions as complex numbers

\[x + hi_1 + ... + hi_n\]
Implementation

Declare all the variables as complex type

Redefine all the functions with complex definitions

Add complex step \( h \) to the desired variable

Partial Derivative Calculations

\[
\begin{align*}
\text{Or only the independent variables and the associated intermediate variables} \quad &\quad \text{Easy implementation using the same form of operator overloading functions as complex numbers} \\

\left\{ \begin{array}{l}
\quad x + hi_1 + \ldots + hi_n \\
\quad f^{(n)}(x) \approx \frac{\text{Im}_{1\ldots n}[f(x + hi_1 + \ldots + hi_n)]}{h^n}
\end{array} \right.
\end{align*}
\]

Repeat as necessary for more independent variables
Implementation

Declare all the variables as complex type

Redefine all the functions with complex definitions

Add complex step (h) to the desired variable

Partial Derivative Calculations

Or only the independent variables and the associated intermediate variables

Easy implementation using the same form of operator overloading functions as complex numbers

\[ f^{(n)}(x) \approx \frac{\text{Im}_{1\ldots n}\left[f(x + h_1 + \ldots + h_n)\right]}{h^n} \]

Repeat as necessary for more independent variables

Currently have working modules in Matlab and Fortran
Overloading example in FORTRAN for + operator

```fortran
! Bicomplex overloading

type bicomplex
    COMPLEX*16 :: a
    COMPLEX*16 :: b
end type

interface operator(+)
    module procedure bicplx_plus_bicplx, bicplx_plus_bicplx_array, bicplx_plus_dble, bicplx_plus_dble_array
end interface

function bicplx_plus_bicplx(q1,q2) result(q3)
    implicit none

    type(bicomplex), intent(in) :: q1, q2
    type(bicomplex) :: q3

    q3%a = q1%a + q2%a
    q3%b = q1%b + q2%b

    return
end function bicplx_plus_bicplx
```
Step-size Limits for High order Derivatives

1) Since error is $O(h^2)$, then $h^2 > \delta = 1e-16$
   therefore: $h > \sim 1e-8$

2) Because $h^n$ appears in derivative approximation:
   $h^n > \epsilon = 1e-308$,
   therefore:
   - $h > 10^{-308/n}$
   - From above: $1e-8 > 10^{-308/n}$
   - $n < \sim 38$ for double precision
     (this is upperbound, in practice should be smaller due to dynamic range of variables, margin on error estimates...)
   - Note that a complex number with $n = 35$ is represented with $2^{35} > 10^{10}$ real numbers!!!

Elementary computation cost comparison to compute a product and its derivative

- **MCX**: $Z_1 \cdot Z_2 = (x_1 + \text{i}h_1) \cdot (x_2 + \text{i}h_2) = x_1x_2 - h_1h_2 + \text{i}h_1x_2 + \text{i}h_2x_1$
- **AD**: \{x_1x_2; d(x_1 \cdot x_2) = dx_1x_2 + x_1 \, dx_2 \}

Over cost of MCX versus AD: the Product $h_1h_2$
Example: \( \cos(x), \ldots, \frac{d^n}{dx^n} \cos(x) \)

Relative errors on 1\textsuperscript{st}-6\textsuperscript{th} derivative of \( \cos(x) \)

- \( h^5 \) & \( h^6 \) under-flow limitation
- for double precision, \((1+\delta)=1\) if \( \delta < 2e-16 \)
- Large choice for step \( h \)

\( (1+\delta=1) \) if \( \delta < 2e-16 \)
Test Case: Trajectory

- Trajectory State Transition Matrix
  - Satellite subject to gravitational force and constant inertial thrust
  - Segment propagated for 6 days
  - 1\textsuperscript{st} and 2\textsuperscript{nd}-order State Transition Matrices useful in trajectory optimization (399 terms)

<table>
<thead>
<tr>
<th>Method</th>
<th>Sample 2\textsuperscript{nd}-order STM</th>
<th>Accuracy</th>
<th>Total Relative Compute Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytical</td>
<td>-2.092290564266828 10\textsuperscript{-2}</td>
<td>NA</td>
<td>1.0*</td>
</tr>
<tr>
<td>MultiComplex</td>
<td>-2.092290564266829 10\textsuperscript{-2}</td>
<td>5.3 10\textsuperscript{-15}</td>
<td>1.7</td>
</tr>
<tr>
<td>TAPENADE</td>
<td>-2.092290564266826 10\textsuperscript{-2}</td>
<td>3.7 10\textsuperscript{-14}</td>
<td>2.1</td>
</tr>
<tr>
<td>AD02</td>
<td>-2.0922905642668233 10\textsuperscript{-2}</td>
<td>4.0 10\textsuperscript{-14}</td>
<td>4.4</td>
</tr>
<tr>
<td>Finite Differences</td>
<td>-2.092290785071782 10\textsuperscript{-2}</td>
<td>2.8 10\textsuperscript{-6}</td>
<td>4.5</td>
</tr>
</tbody>
</table>

*analytic computation of STMs do not take advantage of symmetry, time could likely be reduced in half, but effort is nontrivial
Test Case: Gravity field

- Gravity field derivatives
  - 20x20 Lunar Gravity Field
  - Up to third-order
  - Useful for satellite geodesy and trajectory optimization

<table>
<thead>
<tr>
<th>Method</th>
<th>Sample 3\textsuperscript{rd}-order Sensitivity</th>
<th>Maximum Relative Difference with Analytic (across all 3\textsuperscript{rd} order terms)</th>
<th>Total Relative Computational Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytical</td>
<td>-4.239541972305253 \times 10^{-12}</td>
<td>NA</td>
<td>1.0</td>
</tr>
<tr>
<td>MultiComplex-Step</td>
<td>-4.239541972305250 \times 10^{-12}</td>
<td>6.6 \times 10^{-15}</td>
<td>20.9</td>
</tr>
<tr>
<td>TAPENADE</td>
<td>-4.239541972305257 \times 10^{-12}</td>
<td>2.6 \times 10^{-15}</td>
<td>30.1</td>
</tr>
<tr>
<td>AD02</td>
<td>-4.239541972305255 \times 10^{-12}</td>
<td>2.9 \times 10^{-15}</td>
<td>154.9</td>
</tr>
</tbody>
</table>
### New Sensitivity Landscape

<table>
<thead>
<tr>
<th></th>
<th>FD</th>
<th>Analytical</th>
<th>MCX</th>
<th>AD*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute Speed</td>
<td>Slow</td>
<td>Fast</td>
<td>Medium</td>
<td>Medium (1) / slow (2)</td>
</tr>
<tr>
<td>Ease of implementation</td>
<td>Easiest</td>
<td>Hardest</td>
<td>Medium</td>
<td>Medium (1)/Hard(2)</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Poor</td>
<td>Near Exact**</td>
<td>Near Exact **</td>
<td>Near Exact **</td>
</tr>
<tr>
<td>Special requirements</td>
<td>None: function call CAN BE a library or “black box”</td>
<td>Function call and derivatives <em>must</em> be consistent. CAN BE a library or “black box”</td>
<td>MCX module +function source</td>
<td>AD module +function source</td>
</tr>
</tbody>
</table>

*only considered TAPENADE (1) & AD02 (2) of the many AD toolboxes available

** subject to round off, order of operations, dynamic variable range errors, but not subtraction error
Conclusion

• MultiComplex-Step method was developed for computation of partial derivatives up to any order: extension of complex step method to any order

• MultiComplex-Step differentiation combines the best of finite difference, complex, and automatic differentiation

• Further increasing in tool flexibility is possible through:
  – Prototype Modules for Fortran and Matlab
  – Developing a script to automatically process source codes
  – Matrix and array operations in Matlab
Thank you!

- We want your feedback!!