

THE RAPSODIA TOOL FOR FAST HIGHER-ORDER DERIVATIVE TENSOR
COMPUTATIONS

I. Charpentier

Laboratoire de Physique et Mécanique des Matériaux, Metz, France

J. Utke

Argonne National Laboratory, Mathematics and Computer Science Division, Argonne ,USA

OUTLINE

IONIZATION APPLICATION

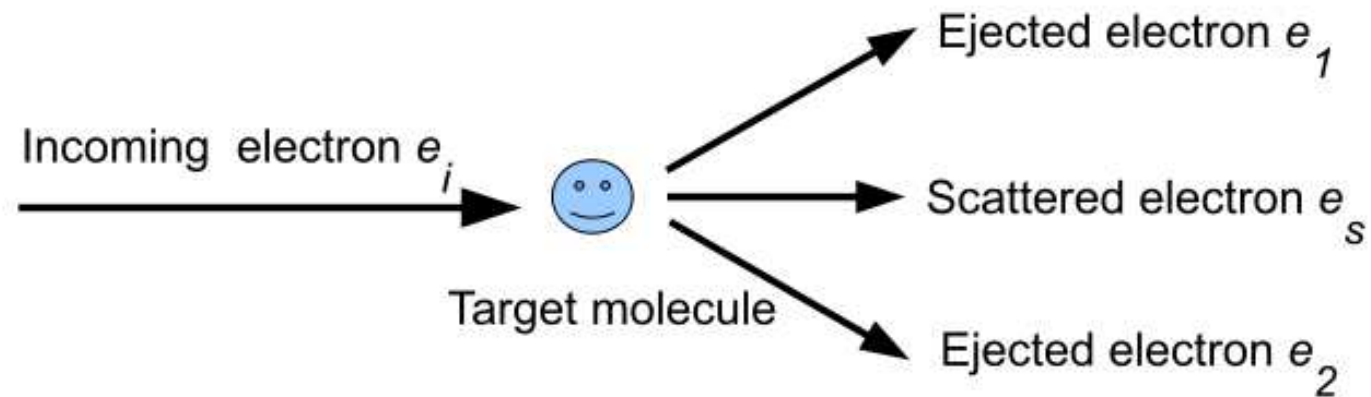
- hypergeometric function
- higher order tensors

RAPSODIA

- operators and intrinsic functions

PERFORMANCES

DOUBLE IONIZATION OF ATOMS AND MOLECULES



Two cases:

1. e_i ejects e_1 , then e_i ($= e_s$) ejects e_2 (simple modelling),
2. e_i ejects e_1 , then e_1 ejects e_2 .

THEORY

Computation of the fully differential cross section (first Born approximation)

$$\frac{\partial^5 \sigma}{\partial \Omega_s \partial \Omega_1 \partial \Omega_2 \partial E_1 \partial E_2} = \frac{k_1 k_2 k_s}{k_i} |M|^2 \quad (1)$$

where Ω_i are solid angles, E_1 and E_2 are energies of the ejected electrons, \vec{k}_i are vectors (momenta of incident, scattered, first ejected, and second ejected electrons).

Matrix element M is a 9-dimensional integral

$$M = \frac{1}{2\pi} \int \psi_f^*(\vec{r}_1, \vec{r}_2) e^{i\vec{k}_s \cdot \vec{r}_0} V \psi_i(\vec{r}_1, \vec{r}_2) e^{i\vec{k}_i \cdot \vec{r}_0} d\vec{r}_0 d\vec{r}_1 d\vec{r}_2 \quad (2)$$

that may be reduced to a six dimensional integral using $e^{i\vec{k} \cdot \vec{r}_0} k^{-2} = 4\pi^{-1} \int e^{i\vec{k} \cdot \vec{r}} |\vec{r} - \vec{r}_0|^{-1} d\vec{r}$.

Wavefunctions ψ_i and ψ_f are the solutions of the Schrödinger equation for Helium. No exact formulas exist for ψ_i and ψ_f .

NUMERICAL APPROACHES

Bound state wavefunction ψ_i is approximated by means of the Hylleraas-type wavefunction

$$\psi_N(\vec{r}_1, \vec{r}_2) = \sum_{i,j,l \geq 0} c_{ijl} (r_1^i r_2^j r_{12}^l + r_1^j r_2^i r_{12}^l) r_{12}^l \quad (3)$$

Two numerical approaches:

- 3 occurrences of “ ${}_1F_1$ ” and a 6-dimensional \int (Jones & madison, 2003)
→ expensive in computer time)
- 1 occurrence of “ ${}_2F_1$ ” and a 2-dimensional \int
→ higher order tensors (Brauner, Briggs & Klar, 1989)

BRAUNER METHOD (BRAUNER, BRIGGS & KLAR, 1989)

Based on $D = e^{-ar_1} e^{-br_2} e^{-cr_{12}} / (r_1 r_2 r_{12})$ whose third-order derivative yields the simple wavefunction

$$\psi_i(\vec{r}_1, \vec{r}_2) = e^{-ar_1} e^{-br_2} e^{-cr_{12}}.$$

$\rightarrow r_1^i r_2^j r_{12}^l e^{-ar_1} e^{-br_2} e^{-cr_{12}}$ may be written as a mixed derivative of D of order $i + j + l + 3$.

CHALLENGES	AD answers
Complex arithmetics	no AD tool
Hypergeometric function	no AD tool
Higher order tensors	HOT ¹
Efficiency	

¹ HOT: Griewank, Utke & Walther, 2000.

RAPSODIA TOOL (CHARPENTIER & UTKE)

CHALLENGE	AD answer	Rapsodia
complex arithmetics	no AD tool	included
hypergeometric function	no AD tool	ODE ²
higher order tensors	HOT	included
efficiency		unrolled loops

² Charpentier, Dal Cappello & Utke, AD2008.

BASIC IDEAS FOR ANOTHER OVERLOADING LIBRARY

- Implement explicit unrolling via code generation for given fixed degree & number of directions
 - iteration count (degree & directions) limited by memory → slicing,
 - inner iteration count variable (convolutions),
 - Active type with flat structure
- Advantages
 - compiler can optimize simple code
 - core logic shared across languages
 - code generator for Fortran and C++ (real and complex arithmetics)

ALONG COMES **Rapsodia**

Rapide surcharge d'opérateur pour la **différentiation automatique**.

- get it for free :-) from `mcs.anl.gov/rapsodia`
- code generation done with Python
- user gives a degree and number of directions
- generator makes AST for active types, intrinsic declarations and overloading logic
- unparses the AST into C++ or Fortran files making up a library
- optional inlining (C++)
- provides C++ and Fortran implementations for seeding & interpolation to compute higher order tensors (HOT)
- we even have a manual !

DISCLAIMER!

- this is only one of the many possible source code incarnations of Taylor coefficient computation
- tool was needed for Fortran (incl. computations with complex numbers):
- AD02: not free and no univariate coefficient propagation
- Adol-F: no more supported
- Adol-C: no tapeless higher order mode.

COMPARISON TEST IN FORTRAN (REAL ARITHMETICS)

Application in underwater acoustics (using real arithmetics) described in (Charpentier & Roux, 2004)

		AD02				Rapsodia					
		g95	ifort	NAG				g95	ifort	NAG	
o	n	-O3	-O2	-O2	-O4	d*	d	-O3	-O2	-O2	-O4
2	5	0.599	0.460	0.543	0.658	15	15	0.072	0.106	0.087	0.086
4	3	40.97	11.97	13.67	14.41	15	15	0.161	0.255	0.181	0.176
6	3	185.4	58.88	73.63	71.21	14	28	0.514	0.794	0.538	0.515
8	2	105.8	36.39	45.41	41.56	9	9	0.250	0.366	0.262	0.257
8	3	651.1	*	289.8	285.2	15	45	1.157	1.762	1.172	1.101
10	3	1958.	*	+	+	11	66	2.453	3.523	2.474	2.420
13	3	+	*	+	+	10	105	5.677	8.656	5.673	5.638

(Somewhat unfair: AD02 does not have univariate Taylor coefficient propagation.)

Note: We do not want to compete with compiler optimization - may be Rapsodias loop unrolling will become obsolete eventually.

IONIZATION APPLICATION: CHALLENGES

- Complex arithmetics: Ok with Rapsodia
- Hypergeometric function
- Higher order tensors
- Efficiency

THE HYPERGEOMETRIC FUNCTION ${}_2F_1(a, b, c; z)$ (IN BRIEF)

- solution of the hypergeometric differential equation

$$z(1-z)\frac{d^2\varphi(z)}{dz^2} + [c - (a+b+1)z]\frac{d\varphi(z)}{dz} - ab\varphi(z) = 0 \quad (4)$$

- can be expressed as a series

$${}_2F_1(a, b, c; z) = \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{(c)_n} \frac{z^n}{n!} \quad \text{with} \quad (x)_n = \frac{(x+n-1)!}{(x-1)!}$$

- derivatives given by

$${}_2F_1^{(n)}(a, b, c; z) = \frac{\partial^n {}_2F_1(a, b, c; z)}{\partial z^n} = \frac{(a)_n (b)_n}{(c)_n} {}_2F_1(a+n, b+n, c+n; z) \quad (5)$$

may be avoided by deriving a recurrence formula for the Taylor coefficients from (4).

Note: Every second-order ODE with three regular singular points can be transformed into (4).

LOW-ORDER DERIVATIVES OF THE $v = {}_2F_1$

Notations and assumptions

- write the ${}_2F_1$ ODE as: $\alpha(z)\phi^{(2)}(z) + \beta(z)\phi^{(1)}(z)\gamma\phi(z) = 0$,
- write $v(t) = v^{(0)}(t) = \phi(z(t))$

SECOND ORDER DERIVATIVE

1. Differentiating $v(t)$: $v^{(1)} = \phi^{(1)}z^{(1)}$ and $v^{(2)} = \phi^{(2)}(z^{(1)})^2 + \phi^{(1)}z^{(2)}$,

2. assuming $z^{(1)} = 0$: $\phi^{(1)} = \frac{v^{(1)}}{z^{(1)}}$ and $\phi^{(2)} = \frac{v^{(2)}(v^{(1)})/z^{(1)}z^{(2)}}{(z^{(1)})^2}$

3. substituting into (6): $\alpha \frac{v^{(2)}(v^{(1)}/z^{(1)})z^{(2)}}{(z^{(1)})^2} + \beta \frac{v^{(1)}}{z^{(1)}} - \gamma v^{(0)}$

yield:

$$\begin{cases} v^{(0)} = \phi \\ v^{(1)} = \phi^{(1)}z^{(1)} \\ v^{(2)} = \frac{\gamma v^{(0)}(z^{(1)})^3 \beta v^{(1)}(z^{(1)})^2 + \alpha v^{(1)}z^{(2)}}{\alpha z^{(1)}} \end{cases} \quad (6)$$

HIGHER-ORDER DERIVATIVES OF THE $v = {}_2F_1$ (Charpentier & Utke, AD08)

May be computed by:

1. Faa di Bruno formula

$$v^{(n)} = (\phi \circ z)^{(n)} = \sum_{k=1}^n \phi^{(k)} B_{n,k}(z^{(1)}, \dots, z^{(n-k+1)}), \quad (7)$$

(theoretical base to discuss the treatment of poles)

2. Taylor Coefficient Propagation,
3. Overloading (7).

ADVANTAGES / DRAWBACKS

- Poles appear in the last 2 formulas (Charpentier, Dal Cappello & Utke, AD08)
- Option 3 easily and efficiently implemented using Diamant. (see complexity results discussed as in Chang & Corliss, 1982).

ACTUAL IONIZATION FRAMEWORK

$$\psi_N(\vec{r}_1, \vec{r}_2) = \sum_{i,j,l \geq 0} c_{ijl} (r_1^i r_2^j + r_1^j r_2^i) r_{12} \quad (8)$$

3 “AUTOMATIC DIFFERENTIATION” TOOLS

- my colleague in Metz (Dal Cappello):
 - derivatives at order 7 of the original code (23 lines, \simeq 150 ops.) in 9 directions
 - fully handcoded and with some optimization,
 - a few Fortran functions (*, /, 2 F1) replicate operator overloading AD.
- Rapsodia for elementary operations and intrinsics functions,
- Diamant for the hypergeometric function (we also have some hand coded implementation of Faa di Bruno formula).

RUNTIMES FOR ${}_2F_1$ COMPUTATIONS

To compute tensor elements:

- HOT (Griewank, Utke & Walther, 2000), or
- some tricky differentiation (here):
 - use “my colleague” to perform a 4th-order differentiation with respect to r_{12} ,
 - overload its code using Rapsodia or Diamant to obtain 6 derivatives more (in r_1 and r_2) by a linear combination of six directions (for coef. c_{223}).

Table 1: CPU time consumptions for the ψ_{14} function

N	$P_{original}$ without ${}_2F_1$	$P_{original}$	$P_{Fa\acute{a} di Bruno}$	$P_{Diamant_2}$	$P_{Diamant_0}$
14	2.812	x	5.9	4.7	8.4

→ 2-level overloading strategy (with $Diamant_2$) is about 25% less time consuming than the Faá di Bruno formula implementation realized in $P_{Fa\acute{a} di Bruno}$.

NONZERO c_{ijl} COEFFICIENTS FOR THE ψ_N FUNCTIONS WITH $N = 3, 5, 9$ AND 14
 (Ancarani & al, 2008)

c_{ijl}	ψ_3	ψ_5	ψ_9	ψ_{14}	$K = i + j + l + 3$
c_{000}	×	×	×	×	3
c_{200}	×	×	×	×	5
c_{220}		×	×	×	7
c_{300}		×	×	×	3
c_{320}			×	×	8
c_{400}	×			×	7
c_{002}		×	×	×	5
c_{202}			×	×	7
c_{222}				×	9
c_{302}			×	×	8
c_{402}			×	×	9
c_{003}				×	6
c_{223}				×	10
c_{004}				×	7
$\langle -E \rangle_N$	2.90196	2.90286	2.90327	2.90342	

CONCLUSIONS ON RAPSODIA

- free software
- really help us in our physics application (results)

And now ?

My colleague (quantum chemistry) asks for more and more derivatives.

- full evaluation of tensor needs \rightarrow limits of our approach (even with HOT) ?
- complex biological molecules (DNA)
- differentiation of a generic code in 9 directions