Intception: Automatic generation of code for the evaluation of molecular integrals

James C. Womack

University of Bristol

11th February 2015
Molecular integrals

\[ E = \frac{\langle \Psi | \hat{H} | \Psi \rangle}{\langle \Psi | \Psi \rangle} \]

\[ E^{(2)} = \frac{1}{4} \sum_{ijab} \frac{|\langle ij | ab \rangle - \langle ij | ba \rangle|^2}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} \]

\[ F_{\alpha \beta} = \langle \alpha | \hat{h} | \beta \rangle + \sum_{\delta \gamma} P_{\delta \gamma} \left[ (\alpha \beta | \gamma \delta) - \frac{1}{2} (\alpha \delta | \gamma \beta) \right] \]

\[ \langle ij | ab \rangle \equiv (ia|jb) = \int dr_1 dr_2 \psi_i^*(1) \psi_j^*(2) r_{12}^{-1} \psi_a(1) \psi_b(2) \]

\[ |i\rangle \equiv \psi_i(r) = \sum_{\alpha} C_{i\alpha} \phi_{\alpha}(r) \]

\[ |\alpha\rangle \equiv \phi_{\alpha}(r; A, a) = \sum_{m} d_{m\alpha} g(r; \zeta_{m\alpha}, A, a) \]
Molecular integrals: GTOs

\[ |a) \equiv g(r; \zeta_a, A, a) = (x - A_x)^a_x(y - A_y)^a_y(z - A_z)^a_z \exp(-\zeta_a |r - A|^2) = \prod_{i=x,y,z} (r_i - A_i)^{a_i} \exp(-\zeta_a (r_i - A_i)^2) \]

\[ A = (A_x, A_y, A_z) \]
\[ a = (a_x, a_y, a_z) \]
\[ l_a = a_x + a_y + a_z \]

\[
\begin{align*}
\text{s: } l &= 0 & (0, 0, 0) \\
\text{p: } l &= 1 & (1, 0, 0), (0, 1, 0), (0, 0, 1) \\
\text{d: } l &= 2 & (2, 0, 0), (1, 1, 0), \ldots
\end{align*}
\]

Plot adapted from Szabo, A. & Ostlund, N. S. Modern Quantum Chemistry (Dover, 1996), pp.153–159.
Evaluating molecular integrals

\[(a|b) = \int dr \, g(r; \zeta_a, A, a) g(r; \zeta_b, B, b)\]

Recurrence relations: Obara-Saika scheme

\[(0_A|0_B) = (\pi/\zeta)^{3/2} \exp(-\xi|A - B|^2)\]

\[\frac{\partial}{\partial A_i}|a) = 2\zeta_a|a + 1_i) - a_i|a - 1_i)\]

\[(a + 1_i|b) = PA_i(a|b) + \frac{a_i}{2\zeta}(a - 1_i|b) + \frac{b_i}{2\zeta}(a|b - 1_i)\]
Evaluating molecular integrals: “ERIs”

\[ (a b | r_{12}^{-1} | c) = \int dr_1 dr_2 g_a(r_1) g_b(r_1) r_{12}^{-1} g_c(r_2) \]

Auxiliary indexes

\[ ((a + 1i)b|c)^{(m)} = \text{VRR} \left\{ (ab|c)^{(m)}, (ab|c)^{(m+1)}, \ldots \right\} \]

\[ (ab|c)^{(0)} \equiv (ab|r_{12}^{-1}|c) \]

The Boys function

\[ (0_A 0_B|0_C)^{(m)} = f(\zeta_a, \zeta_b, \zeta_c, A, B, C) F_m(T) \]

\[ F_m(T) = \int_0^1 dt \ t^{2m} \exp(-Tt^2) \]

Evaluating molecular integrals

Contracted functions (AOs)

\[ \phi_\alpha(r; A, a) = \sum_{m} d_{m\alpha} g(r; \zeta_{m\alpha}, A, a) \]

Horizontal recurrence relations (HRRs)

\[ (a(b + 1_i)|c) = ((a + 1_i)b|c) + AB_i(ab|c) \]

... can be applied to contracted integrals.

Density fitting for three-electron integrals in explicitly correlated electronic structure theory

James C. Womack and Frederick R. Manby

Centre for Computational Chemistry, School of Chemistry, University of Bristol, Bristol BS8 1TS, United Kingdom

(Received 27 November 2013; accepted 10 January 2014; published online 31 January 2014)

The principal challenge in using explicitly correlated wavefunctions for molecules is the evaluation of nonfactorizable integrals over the coordinates of three or more electrons. Immense progress was made in tackling this problem through the introduction of a single-particle resolution of the identity. Decompositions of sufficient accuracy can be achieved, but only with large auxiliary basis sets. Density fitting is an alternative integral approximation scheme, which has proven to be very reliable for two-electron integrals. Here, we extend density fitting to the treatment of all three-electron integrals that appear at the MP2-F12/3*A level of theory. We demonstrate that the convergence of energies with respect to auxiliary basis size is much more rapid with density fitting than with the traditional resolution-of-the-identity approach. © 2014 AIP Publishing LLC. [http://dx.doi.org/10.1063/1.4863136]
DF3-MP2-F12/3*A(FIX) theory

\[ v_{ij,kl} = \sum_m \langle ijm| r_{12}^{-1} f_{23} | mlk \rangle \]

\[ a_{kl, mn}^{(1)} = \sum_i \langle kli|[f_{12}, \hat{t}_1] f_{23}| inm \rangle \]

\[ a_{kl, mn}^{(2)} = \sum_i \langle kli|[f_{12}, \hat{t}_2] f_{23}| inm \rangle \]

<table>
<thead>
<tr>
<th>label</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-F</td>
<td>(a</td>
</tr>
<tr>
<td></td>
<td>(ab</td>
</tr>
<tr>
<td></td>
<td>(a</td>
</tr>
<tr>
<td></td>
<td>(a</td>
</tr>
<tr>
<td>F-F</td>
<td>(a</td>
</tr>
<tr>
<td></td>
<td>(ab</td>
</tr>
<tr>
<td></td>
<td>(a</td>
</tr>
<tr>
<td></td>
<td>(a</td>
</tr>
<tr>
<td>F-FX</td>
<td>(a</td>
</tr>
<tr>
<td></td>
<td>(ab</td>
</tr>
<tr>
<td></td>
<td>(a</td>
</tr>
<tr>
<td></td>
<td>(a</td>
</tr>
<tr>
<td>F-FD</td>
<td>(a</td>
</tr>
<tr>
<td></td>
<td>(ab</td>
</tr>
<tr>
<td></td>
<td>(a</td>
</tr>
<tr>
<td></td>
<td>(a</td>
</tr>
<tr>
<td>FT1-F</td>
<td>(ab</td>
</tr>
<tr>
<td>FT2-F</td>
<td>(a</td>
</tr>
</tbody>
</table>

Intception

Purpose

Automatic generation of source code for the evaluation of molecular integrals in quantum chemistry

Motivation

- Integral evaluation code is time consuming to write.
- The software and hardware landscape is constantly shifting.

Cray 2 supercomputer

x86 workstation

Nvidia Tesla GPGPU

x86 workstation image by Vernon Chan [CC-BY-2.0], via Wikimedia Commons
Multi-core CPU
- Memory per core is large
- Several sophisticated cores
- Multiple different threads
- Large on-die caches
- OpenMP, MPI

GPGPU
- Memory per core is small
- Many stream processors
- Many similar threads
- Data transfer bottlenecks
- CUDA, OpenCL

New hardware and software revisions may also introduce changes to behaviour that require code modification!
Automatic code generation in quantum chemistry

Not a new idea!
There are many projects using some form of code generation for:
- Derivation and simplification of equations.
- Implementation of electronic structure methods.
- Optimized method implementation.

A few examples
- **Tensor Contraction Engine (NWChem)**
- **(Local) Integrated Tensor Framework (Molpro)**
- **Libint** Valeev group, [http://www.valeevgroup.chem.vt.edu/software.html](http://www.valeevgroup.chem.vt.edu/software.html)
Inception of Intception

Writing a general input parser is difficult!

- Avoid this by building on top of the Python language.
- In Python, (nearly) “everything is an object”.
- A domain-specific language (DSL) can be created using Python objects with overloaded operators methods.
Intception is written entirely in Python 3.
The input is a Python script, written using DSL objects.
C is a lingua franca, allowing wide compatibility with other software packages (we use the C99 standard).
Implementation of Intception: DSL

```python
>>> from intception.dsl import *
>>> ga = dsl_cartesian_gaussian(name='a')
>>> gb = dsl_cartesian_gaussian(name='b')
>>> ga * gb
<intception.dsl.dsl_binop object at 0x7f05ff58f358>
```

dsl_cartesian_gaussian ga  *  dsl_cartesian_gaussian gb

dsl_binop  dsl_op
op_mul

def __mul__(self, other):
    return dsl_binop(op_dict['*'], self, other)
Implementation of Intception: DSL

\[(\pi \times o_o_{xp})^{1.5} \times \exp(-x_{xb_o_{xp}} \times RAB2)\]
Implementation of Intception: Modularity

**DSL**

- `dsl_integral“overlap_2idx”`
- `dsl_cartesian_gaussian ga`
  - `dsl_scalar xa`
  - `dsl_position A`
- `dsl_cartesian_gaussian gb`
  - `dsl_scalar xb`
  - `dsl_position B`
- `dsl_rr vrra`
  - In DSL expression:
    - `dsl_scalar o_o_2xp`
    - `dsl_position PA`
    - etc...
- `dsl_rr vrrb`
  - In DSL expression:
    - `dsl_scalar o_o_2xp`
    - `dsl_position PB`
    - etc...
- `base`
  - In DSL expression:
    - `dsl_scalar xaxb_o_xp`
    - `RAB2`
    - etc...

*User input*

**generator**

- `src_dsl_integral`
  - `dsl_integral“overlap_2idx”`
  - Source code specific data:
    - work array counters
    - array indexing
    - etc...

- Further processing

- **output**
  - integral evaluation
  - source code

Southampton, 11th February 2015
Implementation of Intception: Input

# Define 2-idx overlap integral class
ab = dsl_integral(ga,gb,name='overlap_2idx')
ab.set_base( ( pi * o_o_xp )**(3/2) * exp( -xaxb_o_xp*RAB2 ) )
ab.add_vrr('vrr1', gb, \n    PB * ab.int(ga,gb-1) + o_o_2xp * (gb - 1) * ab.int(ga,gb-2) \n    + o_o_2xp * ga * ab.int(ga-1,gb-1) \n)
ab.add_vrr('vrr2', ga, \n    PA * ab.int(ga-1,gb) + o_o_2xp * (ga - 1) * ab.int(ga-2,gb) \n    + o_o_2xp * gb * ab.int(ga-1,gb-1) \n)

# Generate code for evaluation of primitive 2-idx overlap integrals
opt = generator_options( output_directory = "./output_test_primitive/src", \n    contracted = False)

gen = generator( ab, opt = opt )
gen.out()
Current status

Currently implemented
- Serial code.
- Integrals over primitive Cartesian Gaussians.
- VRR-only and VRR+HRR algorithms.
- Boys function / auxiliary indexes.

Work in progress
- Contraction and spherical transformation.
- Optimized serial code [1].
- Parallel implementation [1].

1. Collaboration with Tom Rumsey (MEng project).
Testing and preliminary results

<table>
<thead>
<tr>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing of generated code for a range of 2- and 3-index integral classes against Molpro’s built-in primitive integral routines:</td>
</tr>
<tr>
<td>- Numerical accuracy of generated code verified against Molpro.</td>
</tr>
<tr>
<td>- Serial execution times typically within $2 \times$ Molpro.</td>
</tr>
</tbody>
</table>

| Preliminary data example: $(a|r_{12}^{-1}|b)$ |
|---------------------------------------------|
| Comparison of generated and Molpro primitive Coulomb integral evaluation subroutines: |
| - All combinations of exponents and angular momentum values from uncontracted cc-pVTZ (C) basis set. |
| - Representative range of function centres ($\pm 2.65$ Å). |
| Max difference: $1.02 \times 10^{-12}$  
Max RMSD: $4.74 \times 10^{-13}$ |

Werner, H.-J. et al. MOLPRO, development version (http://www.molpro.net)
Preliminary data example: \((a | r_{12}^{-1} | b)\)
Is Intception a compiler?

- **High level**
  - Source code
    - Abstract problem implicit
    - Algorithm explicit
    - Target specified by language choice and compiler options
  - Compiler
    - Intermediate representation
    - Analysis
    - Optimization
    - Translation
  - Object code
    - Optimized for target
    - Algorithm defined in source code

- **Python**
  - Input script
    - Abstract problem explicit
    - Algorithm parts specified
    - Target specified in script
  - Intception
    - Analysis
    - Optimization
    - Assemble algorithm parts using template
    - Translation

- **C**
  - Source code
    - Optimized for target
    - Constructed using algorithm template

---

Southampton, 11th February 2015
Is Intception a compiler?

- Abstract problem
- Intception via input script
- Source code
- Compiler
- Object code

High level

Lower level
Challenges of automatic code generation

Formal definition of implementation details

**Hand-coding:** one-off, specific implementation.  
**Code generation:** formal, general implementation.

Example: dynamic memory allocation

For a given integral class \((abcd \cdots)\) how much memory is required? Consider:

- Number of indexes.
- Type of indexes (primitive, contracted).
- Evaluation algorithm (VRR, HRR).

... generate routine for array sizing based on runtime information.
A block of \((p|p)\) integrals \((l_a = l_b = 1)\):

\[
2 \times \text{VRR} \quad \text{VRR} + \text{HRR} \quad 2 \times \text{VRR} \text{ with auxiliary index}
\]

\[
(a + 1_i|b) = \text{VRR} \{ (a|b), (a - 1_i|b), (a|b - 1_i) \}
\]

\[
(a|b + 1_i) = \text{HRR} \{ (a|b), (a + 1_i|b) \}
\]
The “competition”

Similar projects

- **Libint**[1]: DSL, C++ output, Obara-Saika type RRIs, emphasises pre-generated optimized libraries.
- **Libcint**[2]: C output, Dupuis-Rys-King (Gaussian quadrature) method.

Why Intception?

- Simple DSL as user input.
- Focus on generality, new integral classes.
- Designed for modularity and extensibility.

... software ecosystem diversity is also important!

---


Southampton, 11th February 2015
The future

Near future
- Contraction and spherical transformation.
- Optimized parallel code output (multi-core CPU, GPGPU) [1].
- Robust testing and benchmarking of generated code in quantum chemical calculations.
- Public release in the near future (licensing TBC).

Long term possibilities
- Multi-component integral support (e.g. multipole moment).
- Other basis function types (e.g. plane waves).
- Automatic generation of wrapper code.
- Simple API for generated library.

1. Collaboration with Tom Rumsey (MEng project).
I would like to thank the following people and organizations:

- Prof Fred Manby
- Tom Rumsey
- Centre for Computational Chemistry
- EPSRC
- University of Bristol
Max difference: $7.11 \times 10^{-15}$

Max RMSD: $4.29 \times 10^{-15}$