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

James C. Womack

University of Bristol

22nd April 2015
Molecular integrals

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

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

\[ \langle ij|ab\rangle \equiv (ia|jb) = \int \text{d}r_1 \text{d}r_2 \psi_i^*(1)\psi_j^*(2) r_{12}^{-1} \psi_a(1) \psi_b(2) \]

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

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

\[ = \prod_{i=x,y,z} (r_i - A_i)^{a_i} \exp(-\zeta_a(r_i - A_i)^2) \]

\[ A = (A_x, A_y, A_z) \]

\[ a = (a_x, a_y, a_z) \]

\[ l_a = a_x + a_y + a_z \]

s: \( l = 0 \) \((0,0,0)\)

p: \( l = 1 \) \((1,0,0), (0,1,0), (0,0,1)\)

d: \( l = 2 \) \((2,0,0), (1,1,0), \ldots\)

Evaluating molecular integrals

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

(a|b) → (a|b) → (a|b) → (i|a)

**Primitive (l=0)**

**Primtive**

**Contracted (AO)**

**MO**

**Recurrence relations: Obara-Saika scheme**

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

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

\[(a + 1_i|b) = PA_i(a|b) + \frac{a_i}{2\zeta} (a - 1_i|b) + \frac{b_i}{2\zeta} (a|b - 1_i)\]

Evaluating molecular integrals

\[ (0\,|\,0) \xrightarrow{\text{VRR}} (a\,|\,0) \xrightarrow{\text{Contract}} (a\,|\,0) \xrightarrow{\text{HRR}} (a\,|\,b) \]

Primitive \((l_a = l_b = 0)\)

Primative \((l_b = 0)\)

Contracted \((l_b = 0)\)

Contracted

**Contracted functions (AOs)**

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

**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]
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
Intception is written entirely in Python 3.

The input is written in a domain specific language (DSL).

The DSL is constructed using Python objects.

C is a *lingua franca*, allowing wide compatibility with other software packages (we use the C99 standard).
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.
- Optimised method implementation.

Similar projects

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

---

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>
```

```
def __mul__(self, other):
    return dsl_binop(op_dict['*'], self, other)
```

```
dsl_cartesian_gaussian ga * dsl_cartesian_gaussian gb

dsl_cartesian_gaussian ga

dsl_cartesian_gaussian gb

dsl_binop dsl_op
op_mul

dsl_cartesian_gaussian ga

dsl_cartesian_gaussian gb
```
Implementation of Intception: Input

```python
# 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,
    PB * ab.int(ga,gb-1) + o_o_2xp * (gb - 1) * ab.int(ga,gb-2)
    + o_o_2xp * ga * ab.int(ga-1,gb-1)
)
ab.add_vrr('vrr2', ga,
    PA * ab.int(ga-1,gb) + o_o_2xp * (ga - 1) * ab.int(ga-2,gb)
    + o_o_2xp * gb * ab.int(ga-1,gb-1)
)

# Generate code for evaluation of primitive 2-idx overlap integrals
opt = generator_options( output_directory = "./output_test_primitive/src",
    contracted = False)
gen = generator( ab, opt = opt )
gen.out()
```
Implementation of Intception: Output

- **VRR sequence**
- **Contraction sequence**
- **HRR sequence**
- **Copy to output array**

**Base function**

1. $(00|0)^{(m)}$ (Contracted)
2. $(a0|c)$
3. $(ab|c)$
4. $(ab|c)$

**Primitives**

1. $(00|0)^{(m)}$ (Primitive)
2. $(a0|c)$
3. $(ab|c)$
4. $(ab|c)$

Manchester, 22nd April 2015
Structured input in the DSL

**Intception**

- Process DSL input
- Construct optimized algorithm
- Output integral evaluation code

Source code for integral evaluation

Manchester, 22nd April 2015
### Results 1: *It works!*

<table>
<thead>
<tr>
<th>Class</th>
<th>Molpro /s</th>
<th>Intception / s</th>
<th>Max. diff.</th>
<th>RMSD</th>
<th>Mean value</th>
</tr>
</thead>
<tbody>
<tr>
<td>((a</td>
<td>b))</td>
<td>(10^{-3})</td>
<td>0.001</td>
<td>(10^{-15})</td>
<td>(10^{-16})</td>
</tr>
<tr>
<td>((a</td>
<td>r^{-1}_C</td>
<td>b))</td>
<td>(10^{-3})</td>
<td>0.001</td>
<td>(10^{-15})</td>
</tr>
<tr>
<td>((a</td>
<td>\hat{T}</td>
<td>b))</td>
<td>(10^{-3})</td>
<td>0.002</td>
<td>(10^{-15})</td>
</tr>
<tr>
<td>((a</td>
<td>r^{-1}_{12}</td>
<td>b))</td>
<td>(10^{-3})</td>
<td>0.001</td>
<td>(10^{-13})</td>
</tr>
<tr>
<td>((ab</td>
<td>r^{-1}_{12}</td>
<td>c))</td>
<td>(10^{-1})</td>
<td>0.128</td>
<td>(10^{-14})</td>
</tr>
<tr>
<td>((ab</td>
<td>r^{-1}_{12}</td>
<td>cd))</td>
<td>–</td>
<td>30.971</td>
<td>(10^{-14})</td>
</tr>
<tr>
<td>((a</td>
<td>r^{-1}_{12}</td>
<td>b</td>
<td>g^{\nu}_{23}</td>
<td>c))</td>
<td>1.237</td>
</tr>
<tr>
<td>((a</td>
<td>g^{\mu}_{12}</td>
<td>b</td>
<td>g^{\nu}_{23}</td>
<td>c))</td>
<td>1.151</td>
</tr>
</tbody>
</table>

Glycine monomer \((\text{NH}_2\text{CH}_2\text{COOH})\), cc-pVDZ basis set (95 functions), serial execution

---

**DFT calculation on naphthalene \((C_{10}H_8)\)**

No XC, cc-pVDZ AO basis and CFIT-Ahlrichs DF basis, comparing an in-house code against Molpro.

\[ E_{\text{elec}} = -329.497952607 \text{ (Intception),} -329.497952586 \text{ (Molpro) Hartree (\sim 10s)} \]

---

A single nuclear centre \(C\) was used for all nuclear integrals. \(\mu, \nu\) were set to a constant values for all 3-electron integrals. Serial execution on 1 core of an Intel Pentium (Haswell) CPU G3420 @ 3.20GHz.

Results 2: *It’s flexible!*

<table>
<thead>
<tr>
<th>Class</th>
<th>Algorithm</th>
<th>Molecule</th>
<th>Max. diff.</th>
<th>RMSD</th>
<th>Mean value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>VRR</td>
<td>Naphthalene</td>
<td>$10^{-14}$</td>
<td>$10^{-15}$</td>
<td>$10^0$</td>
</tr>
<tr>
<td>(a</td>
<td>b)</td>
<td>VRR</td>
<td>Naphthalene</td>
<td>$10^{-15}$</td>
<td>$10^{-17}$</td>
</tr>
<tr>
<td>(a</td>
<td>b)</td>
<td>VRR/HRR</td>
<td>Naphthalene</td>
<td>$10^{-15}$</td>
<td>$10^{-16}$</td>
</tr>
<tr>
<td>(a</td>
<td>b</td>
<td>c)</td>
<td>VRR</td>
<td>Naphthalene</td>
<td>$10^{-15}$</td>
</tr>
<tr>
<td>(a</td>
<td>b</td>
<td>c)</td>
<td>VRR/HRR</td>
<td>Naphthalene</td>
<td>$10^{-15}$</td>
</tr>
<tr>
<td>(a</td>
<td>b</td>
<td>c</td>
<td>d)</td>
<td>VRR</td>
<td>Benzene</td>
</tr>
<tr>
<td>(a</td>
<td>b</td>
<td>c</td>
<td>d)</td>
<td>VRR/HRR</td>
<td>Benzene</td>
</tr>
</tbody>
</table>

Naphthalene ($C_{10}H_8$), cc-pVDZ basis set (180 functions), serial execution.

Benzene ($C_6H_6$), cc-pVDZ basis set (114 functions), serial execution.

---

Serial execution on 1 core of an Intel Pentium (Haswell) CPU G3420 @ 3.20GHz
Results 3: *It’s…not slow!*

| Generated code algorithm          | Naphthalene (C_{10}H_{8}), (ab|r_{12}^{-1}|C) |
|----------------------------------|-----------------------------------------------|
| A 3 VRRs                         | AO: cc-pVDZ (180 functions)                   |
| B 3 VRRs (1 simplified [1])      | DF: cc-pVDZ/MP2FIT (672 functions)            |
| C 2 VRRs + 1 HRR                 |                                               |
| D 2 VRRs (1 simplified [1]) + 1 HRR |                                               |

AO, DF:

Naphthalene (C_{10}H_{8}), (ab|r_{12}^{-1}|C)

Serial execution, Intel Pentium CPU G3420 @ 3.20GHz


Summary

Conclusions

- **It works...**
  Numerical accuracy for a large range of integral classes.

- **It’s flexible...**
  Different algorithms can be rapidly generated and tested.

- **It’s not (so) slow...**
  Serial execution speed is close to Molpro.

Why Intception?

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

... software ecosystem diversity is also important!
Acknowledgements

- Prof Fred Manby
- Tom Rumsey
- Centre for Computational Chemistry
- EPSRC
- University of Bristol