Fast computation of close-coupling exchange integrals using polynomials in a tree representation

Markus Wallerberger[a,*], Katharina Igenbergs[a], Josef Schweinzer[b], Friedrich Aumayr[a]

[a]*Institute of Applied Physics, TU Wien – Vienna Univ. of Technology, Association* Euratom-ÖAW, *1040 Vienna, Austria*
[b]*Max-Planck-Institut für Plasmaphysik, Association* Euratom, *85748 Garching, Germany*

Abstract

The semi-classical atomic-orbital close-coupling method is a well-known approach for the calculation of cross sections in ion-atom collisions. It strongly relies on the fast and stable computation of exchange integrals. We present an upgrade to earlier implementations of the Fourier transform method.

For this purpose, we implement an extensive library for symbolic storage of polynomials, relying on sophisticated tree structures to allow fast manipulation and numerically stable evaluation. Using this library, we considerably speed up creation and computation of exchange integrals. This enables us to compute cross sections for more complex collision systems.

Keywords: exchange integral, polynomial, symbolic manipulations, close-coupling

---

NEW VERSION PROGRAM SUMMARY

Manuscript Title: Fast computation of close-coupling exchange integrals using polynomials in a tree representation
Authors: M. Wallerberger, K. Igenbergs
Program Title: TXINT
Journal Reference:
Catalogue identifier:

*Corresponding author.
E-mail address: wallerberger@iap.tuwien.ac.at

Licensing provisions: none
Programming language: Fortran 95
RAM: depends heavily on input, usually less than 100 MiB
Keywords: exchange integrals, symbolic manipulations, polynomials
Classification: 16.10 Wave Functions and Integrals
Nature of problem:
Analytical calculation of one- and two-center exchange matrix elements for the close coupling method in the impact parameter model.
Solution method:
Similar to the code of Hansen and Dubois [1], we use the Fourier transform method suggested by Shakeshaft [2] to compute the integrals. However, we heavily speed up calculation using a library for symbolic manipulation of polynomaials.
Restrictions:
We restrict ourselves to a defined collision system in the impact parameter model.
Unusual features:
A library for symbolic manipulation of polynomials, where polynomials are stored in a space-saving left-child right-sibling binary tree. This provides stable numerical evaluation and fast mutation while maintaining full compatibility with the original code.
Running time: heavily depending on input, usually less than one CPU second.

LONG WRITE-UP

1. Introduction

The semi-classical close-coupling method provides a framework for the calculation of cross section data for atomic collisions at low to intermediate energies [3]. We have already used it to calculate state-selective cross sections for charge exchange in $\mathrm{Be}^{4+} - \mathrm{H}(n = 1, 2)$ collisions [4]. For heavier highly charged ions A^{q+}, the number of basis states required for a convergent representation increases roughly with $q^{9/4}$ [5]. This makes more efficient numerical methods desirable.

Basically, the close-coupling method is driven by overlap and coupling of basis states centered at different collision partners [6]. We assume that wave functions and potentials can be expanded into cartesian coordinates in the following way:

$$\sum_i A_i \exp(-\alpha_i r + \imath \vec{\beta}_i \vec{r}) r^{\rho_i} x^{\xi_i} y^{\eta_i} z^{\zeta_i}$$

which is true for all Laguerre-type states, Slater-type orbitals and the most common potentials used in the method. Furthermore, to shorten our notation, we introduce vector powers $\vec{n} \in \mathbb{Z}^3$ by defining [7]:

$$(\vec{x})^{\vec{n}} := \prod_{i=1}^{3} (x_i)^{n_i} \quad \text{and} \quad \nabla^{\vec{n}} := \prod_{i=1}^{3} \left(\frac{\partial}{\partial x_i}\right)^{n_i}.$$

We can now give the most general form of exchange matrix elements in terms of *exchange integrals*:

$$I(n_1, \vec{l}_1, n_2, \vec{l}_2) := \int d^3 r_A r_A^{n_1-2} r_B^{n_2-2} (\vec{r}_A)^{\vec{l}_1} (\vec{r}_B)^{\vec{l}_2} \exp(\imath \vec{a} \vec{r}_A + \imath \vec{b} \vec{r}_B - c r_A - d r_B)$$

Using the Fourier-transform method suggested by Shakeshaft [2], we can relate the infinite three-dimensional integral to a sum over one-dimensional integrals over a finite range:

$$I(n_1, \vec{l}_1, n_2, \vec{l}_2) = 2\pi (-\imath)^{\vec{l}_1 + \vec{l}_2} (\nabla_{\vec{a}})^{\vec{l}_1} (\nabla_{\vec{b}})^{\vec{l}_2} \left(-\frac{\partial}{\partial c}\right)^{n_1-1} \left(-\frac{\partial}{\partial d}\right)^{n_2-1} \int_0^1 dy \frac{\exp(\imath \vec{B} \vec{R} - AR)}{A} \tag{1}$$

where we introduced the quantities $A^2 := y(1-y)|\vec{a}+\vec{b}|^2 + yc^2 + (1-y)d^2$ and $\vec{B} := y\vec{a} - (1-y)\vec{b}$.

If we assume a spherically symmetric potential on both centers and use the so-called impact parameter model, we can set $\vec{b} = 0$. Furthermore, by assuming that the incoming nucleus travels on a straight line, we get for two-center exchange integrals $\vec{a} = v\vec{e}_z$ and for one-center elements $\vec{a} = 0$.

2. Computational challenges

The Fourier transform method requires symbolic manipulation: as a result, we need to store and mutate symbolic forms of polynomials in up to 9 variables, multiplied by the base integral $I(1, \vec{0}, 1, \vec{0})$. Symbolic differentiation in particular is problematic because with each differentiation step, the number of terms at least triples, but at the same time lots of linearly dependent terms arise. As a result, we need to (i) insert terms in a fast fashion and (ii) detect and add up duplicates early.

Evaluation poses another problem: if we evaluate a polynomial with high powers on a term-by-term basis, we massively impair performance. Moreover, cancellation errors occur and, as the number of terms increases, may even become dominant. Therefore, we want to collapse the polynomial using the Horner scheme:

$$\sum_{n=a}^{b} c_n x^n = (\cdots((c_b x + c_{b-1})x + c_{b-2})x + \cdots)x + c_a x^a \tag{2}$$

This requires (iii) the *monomials* (terms) to be in *lexicographic order* (sorted by powers of the first variable, then by powers of next variable, and so forth). Finally, to find a fast way to use the Horner scheme, we need to find (iv) a fast way to know up to which variable the powers of two adjacent monomials agree.

The well-known Fortran 77 code by Hansen and Dubois [1] stores the monomials in an unordered fashion within arrays for coefficients and powers of the variables. This is however inefficient with respect to aspects (i) – (iv): Provided that we insert n terms into our polynomial,

1. detecting duplicate terms requires $O(n^2)$, since ordered insert into an array requires costly array manipulation and is therefore not feasible.
2. ordering the terms is of complexity $O(n \log n)$. Moreover, comparing two terms means comparing all variables, which is time-expensive.
3. evaluating an unordered set may lead to cancellation and is time-expensive.

For small basis sets or small quantum numbers, this is usually no problem. With increasing size of our problem, on the other hand, considerable computing time is spent on the creation of symbolic structures (see Figure 2).

3. Polynomial library

We improve this approach using the new features of Fortran 90: the polynomial is now stored in a derived type `polynomial`, and its terms are modelled in a *left-child right-sibling binary tree* (Figure 1), a popular method to store n-ary trees. This considerably speeds up sorted insert, duplicate detection and evaluation at the expense of increasing the time needed to find a specific term (which we do not need to do).

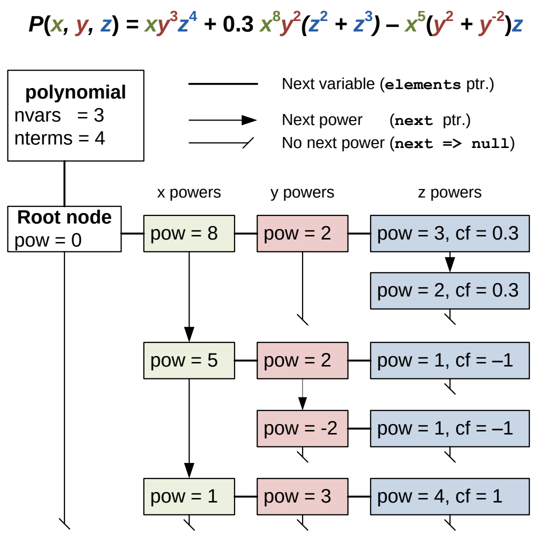


Figure 1: Internal representation of an instance of type `polynomial`. The nodes, implemented in type `polyterm`, form a *LC-RS tree* structure where the right branch (`elements` pointer, lines) distinguishes different variables and the down branch (`next` pointer, arrows) points to the next power in the same variable, enforcing a descending order of powers.

Unlike a binary search tree, the two children of a node (for example x^8) have different meanings: the down branch points to the subsequent power (in our example x^5), represented by an arrow. The right branch, on the other hand, points to the next variable, distiguishing ranks in the tree.

You can restore and evaluate a polynomial from the tree representation in Figure 1 as follows: Start at the root node (top left) and replace each node by the respective variables and powers $cf \cdot var^{pow}$. Always follow the horizonal lines first, replacing each of them with an opening bracket. Then follow the vertical lines, replacing each arrow with a '+' and each dead end $\nmid$ with a closing bracket.

In this representation, many common operations and calculus reduce to simple tree mutation, being considerably faster than operating on (even sorted) arrays.

4. Computation, performance and testing

The computation is performed in a similar way to the code of Hansen and Dubois [1, 8], which we have optimized for our collision system, but with the new Fortran 90 polynomial library. For one-center exchange matrix elements, we also improved the detection of vanishing terms.

The performance of both methods is illustrated by Figure 2: we observe that for all cases, the performance of the new tree method is a lower boundary for the running time of a modified CRERS routine [1]. Studying the asymptotic behaviour of the radial part, we find that the complexity is reduced from $O(n^{7.17})$ to $O(n^{4.80})$ for the exchange integral $I(\frac{n}{2}, \vec{0}, \frac{n}{2}, \vec{0})$ with $c \neq d$. This improves running time by $O(n^{2.37})$, speeding up large integrals.

For the two-center angular part, we investigated the integral $I(1, \frac{n}{2}, 1, \frac{n}{2})$. The complexity of the algorithm was improved from $O(n^{10.3})$ to $O(n^{6.48})$, a massive performance gain by $O(n^{3.82})$. All calculations were performed on an Intel i5 750 processor with 2 GB of RAM running Ubuntu 10.04 x64 edition. Both codes were compiled using gfortran 4.4.1 with maximum optimization levels.

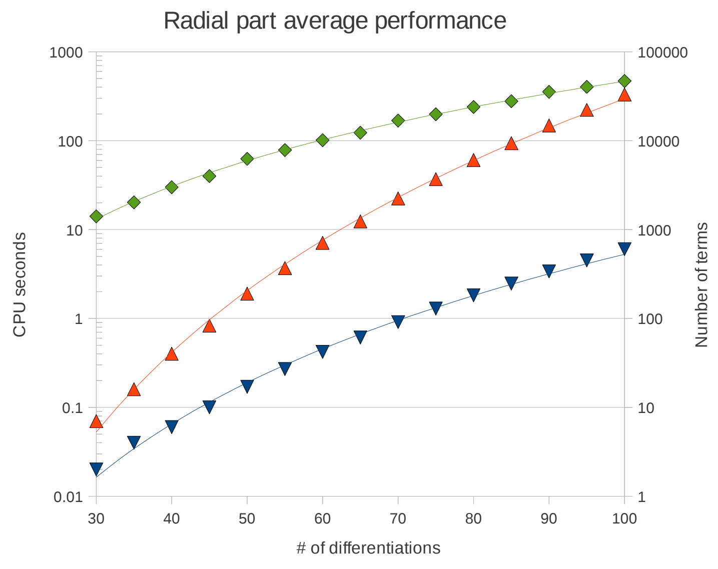
We checked the code both analytically and numerically against known results and the original method for a wide array of integrals, showing exact agreement with the previous results. Perfect agreement with the CRERS method is achieved provided that the complex accuracy of both methods match.

5. Program structure

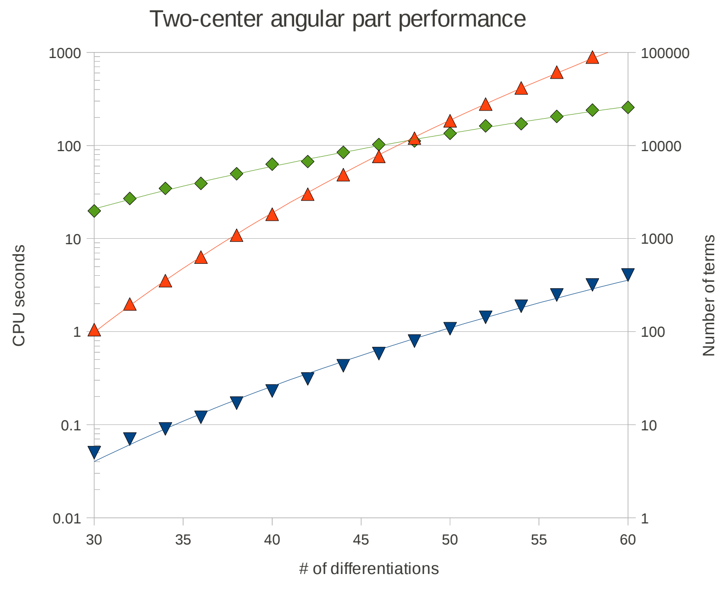
An overview of the program's static structure is given in Figure 3, depicted in an UML 2.0 class diagram.

Polynomial. The derived type polynomial is the basic building block of the program, allowing us to model complex polynomials of arbitrary size in n variables using the structure presented in Section 3. All members of this type are private, so all access is performed through "methods" `poly*`. The accuracy of the polynomial can be tweaked using the `polykind` parameter.

Exchangeint. The derived type stores the integral as well as its defining parameters. The method `xintcreate` is the core method of the type and the equivalent of the `CRERS` method. Like the previous code, it first performs the radial symbolic differentiations on the base integral and then evaluates the integral for certain c and d. After this, it performs the angular differentiations. For vanishing components of $\vec{a}$, it tracks which terms may reach the power 0 and thus survive the evaluation as constant terms.



(a)



(b)

Figure 2: Performance of our tree method (∇) vs. the CRERS method ($\triangle$) for the creation of exchange integrals on an Intel i5 750 processor over the number n of differentiations involved: (a) radial exchange integral $I(\frac{n}{2}, 0, \frac{n}{2}, 0)$ for $c \neq d$, (b) two-center angular exchange integral $I(1, \frac{l}{2}, 1, \frac{l}{2})$. A power regression was laid through the measurement points. The number of terms ($\diamond$, secondary y-axis) is identical for both methods.

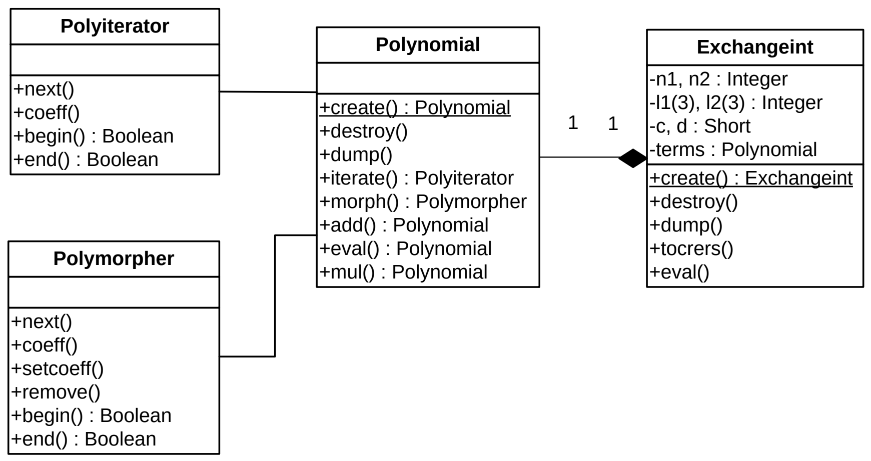


Figure 3: Structure of the program in an UML class diagram [9]. Each block represents a static unit ("class"), where fields are given in the second and methods are given in the third section.

Fast evaluation using the Horner scheme and Gauss-Legendre quadrature is performed by `xinteval` (which requires `xintsetup`).

The method `xinttocrers` provides an interface to the array storage used by JANAL and ALAIN methods [1]. Other methods provide auxiliaries and life-cycle management of the exchange integrals.

Test program. Provides simple user interaction, allowing the user to specify the exchange integral desired. The user interface is printed to standard error, whereas the results are printed to standard output, allowing you to redirect or discard it.

Acknowledgements

This work, supported by the European Communities under the Contract of Association between EURATOM and the Austrian Academy of Sciences, was carried out within the framework of the European Fusion Development Agreement. The views and opinions herein do not necessarily reflect those of the European Commission.

Katharina Igenbergs is a fellow of the Friedrich Schiedel Foundation for Energy Technology.

References

[1] Hansen, J.-P. and Dubois, A., Comput. Phys. Commun. **67** (1992) 456 .

[2] Shakeshaft, R., J. Phys. B: At. Mol. Opt. Phys. **8** (1975) L134.

[3] Bates, D. R., Proc. R. Soc. A **247** (1958) 294.

[4] Igenbergs, K., Schweinzer, J., and Aumayr, F., J. Phys. B: At. Mol. Opt. Phys. **42** (2009) 235206.

[5] Burgdörfer, J., Morgenstern, R., and Niehaus, A., J. Phys. B: At. Mol. Opt. Phys. **19** (1986) L507.

[6] Fritsch, W. and Lin, C. D., Physics Reports **202** (1991) 1.

[7] Kocbach, L. and Liska, R., J. Phys. B: At. Mol. Opt. Phys. **27** (1994) L619.

[8] Hansen, J. P., Comput. Phys. Commun. **58** (1990) 217 .

[9] Object Management Group, *Unified Modelling Language 2.0: Superstructure*, online, 2005.