UNIVERSITY OF CALIFORNIA BERKELEY

Numerical Algorithms Applied to Computational Quantum Chemistry
Homework 2: Analytical and Numerical Integration for Overlap Integrals

Dr. Mayank Agrawal, with Yao Shen (CHEM279)

Due Monday Oct 7th, 2024

This homework constructs the key component (overlap integrals over gaussian functions) that you will need to write your own extended Huckel program in problem set 3.

Some small suggestions to save time:

- Keep your code as modular as possible to follow good coding practices.
- You don't need to comment your code everywhere. You just need to comment major blocks of code and any other parts you find tricky.
- Make sure you print out lots of intermediate stuff or use gdb, to track down your errors.

1 NUMERICAL INTEGRATION IN ONE DIMENSION.

In this problem you will write some software to numerically evaluate the indefinite 1-d overlap integral between two gaussians, $G_A(x)$ and $G_B(x)$. The first one, $G_A(x)$ is centered at x_A, and is given by:

$$G_A(x) = (x - X_A)^{l_A} \exp\left[-\alpha (x - X_A)^2\right] \tag{1.1}$$

If $l_A = 0$, this is a gaussian approximation to an s-type atomic orbital (in 1-d.... 3-d is coming in q. 2!). If $l_A = 1$, the gaussian is p_x-like, etc. $G_B(x)$ is analogous, and centered at x_B. The 1-d overlap integral that you will evaluate is:

$$\begin{aligned} S_x^{AB} &= \int_x G_A(x) G_B(x) && (1.2) \\ &= \int_x (x - X_A)^{l_A} (x - X_B)^{l_B} \exp\left[-\alpha (x - X_A)^2 - \beta (x - X_B)^2\right] && (1.3) \end{aligned}$$

The centers, the exponents (α, β), and the polynomial powers (l_A and l_B) are inputs that you will specify.

While we are asking you to evaluate a *specific* integral, you are going to employ a *general* numerical integration procedure to do it. Therefore your code should separate the evaluation of the *specific* integrand from the *general* numerical integration method.

For numerical integration, this is your chance to enjoy the delights of Numerical Recipes, chapter 4, and implement one of the algorithms described there. The easiest one could be the extended trapezoidal rule, using equally spaced points. To make this *definite* integration approach work for our *indefinite* integral, you will have to set the magnitudes of the upper and lower limits of integration not too safely (i.e. not too big) and not too aggressively (i.e. not too small), using properties of the function we plan to integrate.

Whatever you implement, demonstrate convergence towards a limiting result (you will get that exactly below, but you should also see it as you make the grid finer, etc). Please explore at least the following test cases:

1. Two s-type functions ($l_A = l_B = 0$) both centered at the origin (you can look up the standard integral to get the reference result)
2. An s-type function and a p-type function both centered at the origin
3. Repeat the above with an offset of 1 (in dimensionless units).

You may then use this functionality again to debug the results of problem 2 below....

2 Analytical 3-D overlap integral of primitive gaussians.

Implement a C++ code to analytically evaluate the overlap integrals between two shells of normalized cartesian gaussian functions. What is that you ask? Well, a cartesian gaussian function (which we will call a primitive gaussian) may be defined as:

$$\omega(\mathbf{r}) = N(x-X)^l \left(y-Y\right)^m (z-Z)^n \exp\left[-\alpha\,(\mathbf{r}\text{-}\mathbf{R})^2\right] \tag{2.1}$$

Here N is an optional normalization constant (we'll ignore it in *this* problem set), (l, m, n) are the powers of x, y, z, $\mathbf{R}$ is the center of the gaussian, and α is its exponent. Primitive gaussians group naturally into "shells", which are defined by a constant value of L such that $L = l+m+n$. An s shell has $L = 0$ and contains a single function. A p shell has $L = 1$ and contains 3 functions, $p_x = (1,0,0)$, $p_y = (0,1,0)$, $p_z = (0,0,1)$. Similarly $L = 2$ contains 6 cartesian d functions: $d_{xx}, d_{xy}, d_{xz}, d_{yy}, d_{yz}, d_{zz}$, as opposed to the 5 "pure" $L = 2$ d functions.

As mentioned already, please remember principles of good software design even as you do a small project like this. Aim for modular and extensible code so that your components can be re-used for later projects! You will need this capability in your next problem set!

Your code should:

- Read in the information about 2 primitive cartesian gaussian shells, A and B. Specifically this should be their centers $(\mathbf{R}_A, \mathbf{R}_B)$, angular momentum (L_A, L_B), and their exponents (α, β).
- Build a function or a class that evaluates the overlap integrals associated with this pair of shells, using analytical integration for S_x^{AB} as discussed below.
- Print the overlap matrix whose dimensions are the number of functions in A and B.

Here is one approach to evaluating the overlap integrals that you can implement, but feel free to search google for other alternatives mentioned at the end of the assignment. You could also simplify your code at the expense of some efficiency by having an inner function that does the overlap integral between two primitive gaussian functions, without using the shell structure or the product structure discussed below. This type of choice is up to you.

The overlap integral between 2 primitive gaussian functions, ω_A and ω_B, is:

$$S^{AB} = \int_x \int_y \int_z \omega_A(\mathbf{r})\omega_B(\mathbf{r}) \tag{2.2}$$

$$= S_x^{AB} S_y^{AB} S_z^{AB} \tag{2.3}$$

S_x^{AB} is exactly the function you numerically integrated in the previous problem (see Eq. 1.2). Cartesian gaussian functions are very convenient because S^{AB} factors into independent contributions from integration over x, y, z.

Equally nice, the product of two gaussians is also a gaussian: you may wish to prove a basic version of the gaussian product theorem (the version for a pair of s-type gaussians):

$$\exp\left[-\alpha (x - X_A)^2 - \beta (x - X_B)^2\right] = \exp\left[-\frac{\alpha\beta (X_A - X_B)^2}{\alpha + \beta}\right] \exp\left[-\left(\alpha + \beta\right)(x - X_P)^2\right] \tag{2.4}$$

The center of the product is $\mathbf{R}_P$, where:

$$\mathbf{R}_P = \frac{\alpha \mathbf{R}_A + \beta \mathbf{R}_B}{\alpha + \beta} \tag{2.5}$$

The polynomial terms entering S_x^{AB}, which are $(x - X_A)^{l_A}$ and $(x - X_B)^{l_B}$, can be re-expressed using the binomial theorem to yield:

$$[x - X_A]^{l_A} = [(x - X_P) + (X_P - X_A)]^{l_A} \tag{2.6}$$

$$= \sum_{i=0}^{l_A} \binom{l_A}{i} (x - X_P)^i (X_P - X_A)^{l_A - i} \tag{2.7}$$

Note that the binomial expression above involves integer prefactors which are the number of combinations defined for $m > n$ as:

$$\binom{m}{n} = \frac{m!}{n!(m-n)!} \tag{2.8}$$

Together with standard integrals for the indefinite integration, we (i.e. not just me, but also you!) can derive:

$$S_x^{AB} = \exp\left[-\frac{\alpha\beta(X_A - X_B)^2}{\alpha+\beta}\right]\sqrt{\frac{\pi}{\alpha+\beta}}\sum_{i=0}^{l_A}\sum_{j=0}^{l_B}\binom{l_A}{i}\binom{l_B}{j}\frac{(i+j-1)!!(X_P - X_A)^{l_A-i}(X_P - X_B)^{l_B-j}}{\left[2(\alpha+\beta)\right]^{(i+j)/2}} \tag{2.9}$$

Note (for the sake of your code) that only those terms with $i + j$ equal to an even number contribute a non-zero result (i.e. the odd terms are to be skipped or discarded). Note the appearance of the "double factorial", $n!!$, which is defined as the product of all the integers from 1 to n that have the same parity (odd or even) as n. For example $5!! = 5 \times 3 \times 1$. It is a contrast with the usual factorial function, $n!$, that is simply the product of all integers from 1 to n.

The expression for S_x^{AB} above, while complicated-looking, can be coded. You should do this in easy stages, and make a plan before you code. Here are a few things to think about.

- You need to implement the factorial and double factorial functions. This can be done as a loop, where you might most simply have separate cases for odd and even $n!!$.
- Evaluate the center of the product gaussian, $\mathbf{R}_P$
- Evaluate the exponential prefactor of Eq. 2.9, and the associated square root.
- Decide how general your code should be. At a minimum, we need it to work for s shells and p shells. If you can get it to work for higher angular momentum shells as well, that would be cool, but is not essential.
- Write some code to execute the double summation for a pair of l_A and l_B values associated with the shell pair. For instance, for a $p-s$ shell pair, $l_A = 0, 1$ and $l_B = 0$ meaning there are 2 possible (l_A, l_B) pairs, while there is only one for an $s-s$ shell pair.
- Check your analytical integration code for S_x^{AB} against numerical integration from the first problem! Check that cases which should give zero do give zero. We will discuss some of these issues in the compute lab.
- Assemble the set of integrals for a shell pair by calling your S_x^{AB} routine to evaluate S_y^{AB} and S_z^{AB} and forming the final product shown in Eq. 2.3. There will be 1, 3, 9 numbers for $s-s, p-s, p-p$ shell pairs. Compare with what your class-mates are getting and what we will put on the github!

3 Going further, but only if you want to!

- **More Advanced Numerical Integration Algorithms** You can look at a more accurate method using equally spaced points, like the Romberg rule. You can also explore Gauss-Hermite quadrature, a powerful approach that can obtain significantly higher accurate for integrals of the form $\int f(x)\exp(-x^2)$, which looks very similar to ours! You can also look at variable transformations to mutate the infinite domain, $(-\infty,\infty)$ to a finite domain, and then apply the extended trapezoidal rule, or Romberg integration. See the discussion of the double exponential (DE) transformation in Numerical Recipes, for example.

- **Alternative Analytical Algorithms for Overlap Integrals** There are more efficient algorithms for calculating overlap and other kinds of integrals, like recurrence relations or orthogonal polynomials that give exact quadratures. You can try whatever you want!
- **Improve your algorithm's efficiency** I expect that your function would just do one integral at a time (perhaps easier to code). You can try to assemble the full shell pair of integrals at once (more efficient).