Quantum Chemistry Done Wrong: One-Electron Systems

Nikita Lisitsa
lisyarus@gmail.com

April 18, 2020

Revision 79

Contents

1 What it is all about

It has already been a month of me trying to come up with a perfect introduction for this book when I decided I better do something more practical. Let's get straight to the point: what is this book about? Or, rather, what is it *not* about? This is *not* a

- Chemistry textbook
- Quantum mechanics textbook
- Linear algebra textbook
- Python tutorial

But what it *is*, then? Well, a sturdy mix of all the above, aimed at getting physically significant results as soon as possible, yet without sacrificing understanding. We are going to write some Python code using `numpy` and `scipy` libraries (and ocasionally `matplotlib`, if you want some fancy plots & images) and obtain energies and orbital densities for some pretty tiny chemical systems (that is, systems composed of nuclei & electrons). You will also most probably need a file called `hgto.py` (that just contains routines for evaluating certain integrals) which you can download directly from my github.

The reader is expected to have *at least* a bit of knowledge of physics/chemistry, basic mathematics and Python programming language. For getting into the physics and maths involved, I recommend either reading the first few chapters of [1] or at least not skipping the Variational method section. For learning Python, I recommend googling some tutorials/courses/etc., for there are way too many of them to recommend anything in particular.

The following few sections briefly discuss some essential theoretical background, starting with basic mechanics. I'd especially recommend reading the Variational method section so that you understand what all those matrices even mean and what those solvers are actually solving. The actual coding starts at the Harmonic oscillator section.

Beware that I am by no means a scientist, a quantum chemist or even a physicist. I am barely an amateur possessing some unconventional hobbies. The methods that we are going to make use of are **not** the methods used by real quantum chemical software[1] and, as far as I understand, are only applicable to very tiny systems (with, say, 4 or 5 electrons, while systems of practical interest can contain *thousands* of them). This book only covers one-electron systems for the reason of having a sufficiently narrowed scope.

Finally, feel free to email me if you have some questions, I would love to be of help. Oh, and follow me on twitter, I post weird simulations/computations/etc. sometimes.

This work was notably inspired by Peter Shirley's Ray Tracing in One Weekend series, which is an amazing source for learning about modern photorealistic rendering.

[1] This largely explains the title of the book. If you want to learn about more sophisticated methods, the keywords are Hartree-Fock, post Hartree-Fock, and density functional theory.

2 A tale of energies

So, what we are going to do is compute the lowest-energy states of atoms & ions. Why is energy so important?

In any decent physical framework (e.g. classical or quantum mechanics) the total energy of a closed system is conserved. It consists of several parts (e.g. kinetic + potential) which don't necessarily remain constant by themselves, but conservation of total energy implies that a decrease in one part leads to an increase in another. Consider a perfectly reasonable situation of a rock falling from the sky: the higher it is, the higher is the potential energy; as the rock falls down, the potential energy decreases, leading to an increase in kinetic energy, i.e. acceleration (some people call this *gravity*).

We know that even if the rock was thrown up in the air, it would eventually fall down anyway. This is how all mechanical systems behave: they accelerate towards areas of less potential energy (this is Newton's second law). This doesn't generally mean that the system eventually falls into a local minimum of potential energy. Remember that as potential energy decreases, the kinetic energy increases, so upon reaching a local minimum the system would run with insane speeds and miss the minimum completely. This doesn't happen to rocks, though, since energy is converted into stresses in the ground when the rock finally hits it, but this does happen to pendulums: the potential energy minimum is when the pendulum is at its lowest point, yet the pendulum flies through it at high speed over and over again, never stopping there (it does eventually stop due to friction, but that's another story).

A system can, however, reach a local minimum of potential energy if it has some energy sink that can absorb excess kinetic energy. Typically, this sink is just the environment: a particle constantly hitting another particles quickly loses spare kinetic energy and reaches so-called thermal equilibrium. Recall that temperature is simply average kinetic energy; thus, the lower the surrounding temperature, the higher are the chances that the system will be found in a local minimum of potential energy. And *this* effect is the major reason why we seek for lowest-energy states: most molecules in real life are found in those states supplemented with small kinetic fluctuations (real molecules aren't static, they wiggle and bend randomly).

Interactions can likewise be described in terms of energy: attractive force between particles means that the closer they are, the lower the energy, and repulsive force means the opposite. A *bond* happens when the interaction between two systems (say, particles) has a local minimum of potential energy at some specific distance between the systems. The energy of the bond is the difference between the energy of two systems combined when the bonding takes place and the energy when they are so far that no interaction is present at all. Thus, creating a bond releases energy, while breaking it requires energy. This, coupled with some termodynamics, leads to equations that allow us to predict temperature changes in chemical reactions, provided we know the energies of the bonds. By the way, a chemical reaction is a process of forming and breaking chemical bonds. Bonds are called chemical when they happen between atomic nuclei and electrons, and usually involve at least three participants (two nuclei and an electron holding them together). As an extreme example, a piece of metal can be considered to be a single bond between all particles involved (this is so-called *metallic bonding*).

I think I have convinced the reader that energies are of great utility and importance. They allow us to predict which reactions can happen and which cannot, and at what temperature. They can let us visualize what molecules look like, i.e. what is the geometry of the molecules. What is especially nice is that chemists can actually observe the energies directly — or, rather, the differences between them, — simply by looking at the molecules at some special conditions: this is called *spectroscopy*[2]. The word "simply" might have been slightly unfair, but the point is that there are methods that can verify energy calculations with great precision.

[2]Performing this to a hydrogen atom was one of the things that led to quantum mechanics.

3 A classical failure

One of the first things I tried to implement when I started feeling sufficiently self-confident about my skills in 3D graphics and simulations was an H_2 molecule. I failed miserably: most configurations were disastrous, with a few exceptions of unstable equilibria. There was no bonding, no stability, no *molecule* being formed.

It turns out I've rediscovered one of the many problems of XIX-century physics: it fails to explain how atoms & molecules work. There are many reasons for that, a few well-known chemistry-related examples being

- Earnshaw's theorem: a system of charged particles cannot reach stable equilibrium by means of electrostatic forces alone, — this is exactly what I've faced when trying to simulate those
- Electromagnetic radiation: an electron orbiting a nucleus would lose energy by emitting electromagnetic waves, thus eventually falling onto the nucleus
- Discrete emission spectrum: it was observed via spectroscopy that the possible energy levels of hydrogen are discrete, although classical mechanics always predicts a continuous range of possible energy levels

Quantum mechanics came to the rescue. The advantage of this new theory is that it correctly describes an enormous number of previously unexplained phenomena. The disadvantage, however, is that the mathematics involved is quite hard, sometimes ridiculously hard. In classical mechanics, to predict the motion of a particle, you would solve an *ordinary differential equation* — pick up your favourite method (4th-order Runge-Kutta seemingly being the most popular one) and you are done. In quantum mechanics, you have to solve a *hyperbolic partial differential equation*, — which are notoriously hard to solve numerically, — not to mention that we don't have the notions of exact particle's posisition or momentum anymore. In classical mechanics, if you have N particles, you'd just solve N coupled ordinary differential equations using the same methods — they work here as well. In quantum mechanics, if you have N particles, you still have a single hyperbolic partial differential equation, but in $3N$*-dimensional space*! All conventional space discretization schemes are useless here: if you wanted to discretize each spatial direction into, say, 10 nodes, – which is still too few to be useful[3], — you'd need 10^{3N} nodes overall. If N is the number of electrons in a water molecule, that is, $N = 10$, we get a nice value of 10^{30} nodes. In comparison, your computer probably has something about 10^{10} bytes of RAM.

Gladly, being difficult is very far from being impossible, and we are going to employ this nice observation.

N.B. The theory of relativity plays an important role as well, though in chemistry it is mostly responsible for much finer effects. It also limits the applicability of non-relativistic quantum mechanics to atoms with atomic number being 137 and higher (as of today, only 118 have been identified as existing), and I've written a twitter thread on the details of this.

[3] See also my miserable attempts.

4 Quantum mechanics: quick and dirty

From now on, we'll start being more technical.

The reader will learn from any inroductory material on quantum mechanics (e.g. [1]) that in Schrödinger representation everything boils down to the *time-independent Schrödinger equation*:

$$\hat{H}\psi = E\psi \tag{1}$$

Physicists and chemists use the $\widehat{hat}$ to distinguish operators $\hat{H}$ from numbers E. This number E is the energy of the system. It isn't always well-defined (there may be some probabilities involved), but we are interested in cases when it is. ψ is the *wavefunction*, which encodes everything about the state of our physical system (though, one state corresponds to many wavefunctions, so this encoding is not unique). Technically, it is a vector from some Hilbert space, but in many real situations this Hilbert space consists of functions, so ψ is just a function, like $\psi(x) = \sin(3x)$. The unknowns of this equation are usually both ψ and E.

In quantum mechanics, every measurable physical property is encoded as a linear operator. Energy is no exception: $\hat{H}$ is the energy operator, also called *the Hamiltonian* (in honour of William Rowan Hamilton, who has nothing to do with quantum mechanics[4], in accordance with the Arnold Principle, and who had also discovered quaternions). The energy operator is arguably the most important one: after all, it governs the evolution of the system (through *time-dependent Schrödinger equation*, which we don't need here). All we need to know to start reasoning about a system is the energy operator (and the Hilbert space it operates on, though purists claim this piece of data is built into the operator itself, and I tend to agree). In our cases, this operator applies to functions, usually by multiplying them by other functions, or differentiating them, or both, e.g. $\hat{H}\psi = -\frac{1}{2}\frac{d^2}{dx^2}\psi$. Out or pure curiosity, let us apply this operator to the ψ from previous paragraph:

$$\hat{H}\psi = -\frac{1}{2}\frac{d^2}{dx^2}\sin(3x) = \frac{9}{2}\sin(3x) \tag{2}$$

because $\frac{d^2}{dx^2}\sin(kx) = -k^2\sin(kx)$. We can see that for such $\hat{H}$, ψ is a solution of (1). Such sine-like wavefunctions are called *plane waves*, while the physical system described by the operator $\hat{H}$ is called a *free one-dimensional particle*. In this cases, we call ψ the *eigenfunction* or *eigenvector* of the operator $\hat{H}$, and E is the corresponding *eigenvalue*.

Let us repeat:

- A system (say, a water molecule) is described by an operator $\hat{H}$
- A specific state of a system (a water molecule at that point, flying in that direction with that speed) is described by a wavefunction ψ (we'll ocasionally use other letters for it, but always Greek ones)
- We seek solutions to the equation $\hat{H}\psi = E\psi$, where $\hat{H}$ is known, while both E and ψ are unknowns.

There are, in general, many solutions to this equation. It so happens in physical systems that the possible values of E are bounded below (see here for a proof in case of chemical systems). If the minimal E is attained by some wavefunction ψ, we call this state the *ground state*. It is the state of most importance, since in general it is the most likely state a random system will be found in (the lower the temperature, the more the probability of ground state, see Gibbs distribution). It is this state that we are interested in the most.

N.B. There are severe technical difficulties in the actual mathematical formulation of quantum mechanics. They can be overcome, but the mathematics involved is far beyond the scope of this book. You can find a reasonably graceful introduction to the matter in [3].

[4] He did however make significant contribution to *classical* mechanics, formulating the so-called *Hamiltonian formalism*, where the Hamiltonian is the energy function that corresponds directly to our operator $\hat{H}$.

5 Variational method

Let's recap what we've got so far: we have some $\hat{H}$, and we seek for ψ that has the minimal possible value of E. How can we achieve this? It is the time to lose our very last hope and accept the inevitable: there is no good way. If the Hilbert space we are working with was finite-dimensional, this would be an ordinary (yet by no means trivial!) problem of numerical linear algebra. Our Hilbert space is the space of some functions or combinations thereof, and it is *infinite-dimensional*. Oops.

The trick is to reformulate our problem. First of all, however, we should talk about Hilbert spaces a bit more. They are not just vector spaces, they come with a special operation called the *inner product*. If you are familiar with the dot product of vectors in geometry, this is essentially the same. Take any two functions — say, ψ and ϕ, — and you can get their inner product, denoted by $\langle \psi, \phi \rangle$ or, using Dirac notation[5], $\langle \psi | \phi \rangle$. This is just a number that can be used to measure lengths, angles, etc., — even in infinite-dimensional spaces. The definition that we are going to use (which is a pretty standard one) is the L^2 *inner product*, which is the integral of one function times the complex conjugate[6] of the other, i.e.

$$\langle \psi, \phi \rangle = \int \psi(x)\overline{\phi(x)}dx \tag{3}$$

It so happens[1] that solutions to $\hat{H}\psi = E\psi$ are exactly the stationary points of the functional[7][8]

$$R(\psi) = \frac{\langle \psi, \hat{H}\psi \rangle}{\langle \psi, \psi \rangle} \tag{4}$$

called the Rayleigh quotient. In particular, the value of this functional on *any* wavefunction ψ is *not less* than the minimal possible value of E. Thus, we have our first simplest method to solve the equation: pick any random ψ that you like, plug it into R, and the value you get is an upper bound to the true ground state energy. Hurray!

Of course, this is still too simple to be useful, yet it provides a good foundation. Instead of picking one ψ, we could parametrize it with some parameter c so that the value of R depends on c as well and we can find the minimum of R with respect to c using techniques from undergraduate analysis![1] We could even take multiple c's and minimize a function of many arguments. This is what is called the *variational method* — it works by *varying* the wavefunction with respect to parameters and finding the minimum of R, and can be used to get some valuable analytical estimates for various systems[1].

It is not that simple to minimize a function of many parameters, though, unless it depends on the parameters in some primitive way. Say, pick some functions $\phi_1, \phi_2, \ldots, \phi_k$ and let $\psi(x) = c_1\phi_1(x) + \cdots + c_k\phi_k(x)$. This is the idea of *linear variational method*[9], where the dependence of ψ on the parameters is, well, linear. After taking a few derivatives and doing a bit of algebra one arrives[1] at this matrix equation:

$$Hc = ESc \tag{5}$$

where

- H is an $k \times k$-matrix with entries $H_{ij} = \langle \phi_i, \hat{H}\phi_j \rangle = \langle \hat{H}\phi_i, \phi_j \rangle$ (the second equality is true because $\hat{H}$ is *self-adjoint*).
- S is an $k \times k$-matrix (called the *overlap matrix*[10]) with entries $S_{ij} = \langle \phi_i, \phi_j \rangle$
- c is the column vector of parameters $c_1, \ldots, c_k$

This is known as a *generalized eigenvalue problem* (as opposed to the regular *eigenvalue problem* that lacks S), and it is **this** equation that we will try to solve.

[5] Dirac notation is not used in this book.
[6] If you are afraid of complex numbers, do not let fear stop you: we won't actually need complex numbers in this book.
[7] A functional is a function that applies to functions and outputs a number.
[8] See also: Courant-Fischer-Weyl min-max principle
[9] Also known as the *Rayleigh-Ritz method*
[10] In mathematics, this one is better known as Gramian matrix.

6 Harmonic oscillator

Before we get into chemical systems, atomic nuclei and electrons, let's first try our hand at a simpler toy quantum system — the one-dimensional *harmonic oscillator*. It works roughly like a pendulum, being forced back the more it moves off the center[11]. The Hilbert space for this system is denoted $L^2(\mathbb{R})$ — the space of functions of one real argument that have a finite integral of their modulus squared (in other words, $\langle\psi,\psi\rangle$ is finite). We should be working with complex-valued functions, but as I'll explain in the next section, we can actually restrict ourselves to real numbers everywere.

The Hamiltonian for this system looks like this:

$$\hat{H}\psi = -\frac{1}{2}\frac{d^2}{dx^2}\psi + \frac{x^2}{2}\psi \tag{6}$$

And the energy eigenvalues are[1]

$$E_k = k + \frac{1}{2}, k = 0, 1, 2, \dots \tag{7}$$

Thus, the ground state energy is $E_0 = \frac{1}{2}$. Let's try to get this number without solving the equation (1) directly, employing the equation (5) instead!

All we need to do is pick a good selection of *basis functions* ϕ_k, also called the *basis set*. What makes a basis set better or worse? We have a dilemma here: on the one hand, the basis should be somewhat resembling the actual ground state, so that we can get closer to real ground state energy; on the other hand, we'll have to compute a bunch of integrals $\langle\phi_i,\phi_j\rangle$ and $\langle\phi_i,\hat{H}\phi_j\rangle$, so the basis functions should be simple enough so that the integrals are tractable[12].

There are many options here, and I encourage you to try your own basis set if you are not afraid of computing a few integrals. The one I'll use will be

$$\phi_a(x) = e^{-(x-a)^2} \tag{8}$$

This *gaussian* functions are sufficiently localized and easy to integrate at the same time, which makes them a good choice. The parameter a can be any real number here.

First of all, we need to compute those integrals. You may skip the derivations and go straight to the code, since we will only need the final formulas. The overlap integral is the simplest:

$$\begin{aligned}\langle\phi_a,\phi_b\rangle &= \int_{\mathbb{R}} \phi_a(x)\phi_b(x)dx = \int_{\mathbb{R}} e^{-(x-a)^2}e^{-(x-b)^2}dx = \int_{\mathbb{R}} e^{-2x^2+2(a+b)x-\left(a^2+b^2\right)}dx = \\ &= \int_{\mathbb{R}} e^{-2\left(x^2-2\frac{a+b}{2}x+\left(\frac{a+b}{2}\right)^2\right)}e^{\frac{(a+b)^2}{2}-\left(a^2+b^2\right)}dx = \int_{\mathbb{R}} e^{-2\left(x-\frac{a+b}{2}\right)^2}e^{-\frac{(a-b)^2}{2}}dx = \sqrt{\frac{\pi}{2}}e^{-\frac{(a-b)^2}{2}}\end{aligned} \tag{9}$$

We have used the following well-known Gaussian integral:

$$\int_{\mathbb{R}} e^{-p(x-a)^2}dx = \sqrt{\frac{\pi}{p}} \tag{10}$$

The equality

$$e^{-(x-a)^2}e^{-(x-b)^2} = e^{-2\left(x-\frac{a+b}{2}\right)^2}e^{-\frac{(a-b)^2}{2}} \tag{11}$$

is occasionally known as the Gaussian product theorem and is super useful for integrals with several gaussians.

[11] Any system that is sufficiently close to a local potential energy minimum behaves like a harmonic oscillator. The reason is that in a minimum the derivative of potential energy is zero, so it is well-approximated by the quadratic term: $U(x+\delta) = U(x) + \frac{1}{2}U''(x)\delta^2$.

[12] We could approach them numerically, but this is orders of magnitude slower and less accurate.

Next, we need a few handy observations:

$$\frac{d}{dx}e^{-p(x-a)^2} = -2p(x-a)e^{-p(x-a)^2} \tag{12}$$

$$\int\limits_{\mathbb{R}} (x-a)e^{-p(x-a)^2} \overset{\text{by (12)}}{=\!=} -\frac{1}{2p}\int\limits_{\mathbb{R}} \left[\frac{d}{dx}e^{-p(x-a)^2}\right] dx = -\frac{1}{2p}e^{-p(x-a)^2}\Big|_{-\infty}^{+\infty} = 0 \tag{13}$$

(by the fundamental theorem of calculus).

$$\begin{gathered}\int\limits_{\mathbb{R}} xe^{-p(x-a)^2}dx = \int\limits_{\mathbb{R}} \left[(x-a)+a\right]e^{-p(x-a)^2}dx = \int\limits_{\mathbb{R}} (x-a)e^{-p(x-a)^2}dx + a\int\limits_{\mathbb{R}} e^{-p(x-a)^2}dx = \\ \overset{\text{by (13)}}{=\!=} 0 + a\int\limits_{\mathbb{R}} e^{-p(x-a)^2}dx \overset{\text{by (10)}}{=\!=} a\sqrt{\frac{\pi}{p}}\end{gathered} \tag{14}$$

$$\frac{d^2}{dx^2}e^{-p(x-a)^2} \overset{\text{by (12)}}{=\!=} \frac{d}{dx}\left[-2p(x-a)e^{-p(x-a)^2}\right] = \left[-2p+4p^2(x-a)^2\right]e^{-p(x-a)^2} \tag{15}$$

$$\begin{gathered}\int\limits_{\mathbb{R}} (x-a)^2e^{-p(x-a)^2}dx = \frac{1}{4p^2}\int\limits_{\mathbb{R}} 4p^2(x-a)^2e^{-p(x-a)^2}dx = \\ = \frac{1}{4p^2}\int\limits_{\mathbb{R}} \left[4p^2(x-a)^2-2p\right]e^{-p(x-a)^2}dx + \frac{1}{2p}\int\limits_{\mathbb{R}} e^{-p(x-a)^2}dx \overset{\text{by (15)}}{=\!=} \\ = \frac{1}{4p^2}\int\limits_{\mathbb{R}} \left[\frac{d^2}{dx^2}e^{-p(x-a)^2}\right]dx + \frac{1}{2p}\sqrt{\frac{\pi}{p}} = \frac{1}{4p^2}\left[\frac{d}{dx}e^{-p(x-a)^2}\right]\Big|_{-\infty}^{+\infty} + \frac{1}{2p}\sqrt{\frac{\pi}{p}} = \\ = 0 + \frac{1}{2p}\sqrt{\frac{\pi}{p}} = \sqrt{\frac{\pi}{4p^3}}\end{gathered} \tag{16}$$

(again, using the fundamental theorem of calculus).

Now we are ready for the energy integral $\langle\phi_a, \hat{H}\phi_b\rangle$. We will split it into kinetic $-\frac{1}{2}\frac{d^2}{dx^2}$ and potential $\frac{1}{2}x^2$ parts (see the definition(6) of $\hat{H}$ for harmonic oscillator).

$$\begin{gathered}\left\langle \phi_a, \frac{1}{2}x^2\phi_b\right\rangle = \int\limits_{\mathbb{R}} e^{-(x-a)^2}\frac{1}{2}x^2e^{-(x-b)^2}dx \overset{\text{by (11)}}{=\!=} \frac{1}{2}e^{-\frac{(a-b)^2}{2}}\int\limits_{\mathbb{R}} x^2e^{-2\left(x-\frac{a+b}{2}\right)^2}dx = \\ = \frac{1}{2}e^{-\frac{(a-b)^2}{2}}\int\limits_{\mathbb{R}} \left[\left(x-\frac{a+b}{2}\right)^2 + x(a+b) - \frac{(a+b)^2}{4}\right]e^{-2\left(x-\frac{a+b}{2}\right)^2}dx \overset{\text{by (16)}}{=\!=} \\ = \frac{1}{2}e^{-\frac{(a-b)^2}{2}}\left[\sqrt{\frac{\pi}{32}} + \int\limits_{\mathbb{R}} \left[x(a+b) - \frac{(a+b)^2}{4}\right]e^{-2\left(x-\frac{a+b}{2}\right)^2}dx\right] \overset{\text{by (14)}}{=\!=} \\ = \frac{1}{2}e^{-\frac{(a-b)^2}{2}}\left[\sqrt{\frac{\pi}{32}} + \frac{(a+b)^2}{2}\sqrt{\frac{\pi}{2}} - \frac{(a+b)^2}{4}\int\limits_{\mathbb{R}} e^{-2\left(x-\frac{a+b}{2}\right)^2}dx\right] \overset{\text{by (10)}}{=\!=} \\ = \frac{1}{2}e^{-\frac{(a-b)^2}{2}}\left[\sqrt{\frac{\pi}{32}} + \frac{(a+b)^2}{2}\sqrt{\frac{\pi}{2}} - \frac{(a+b)^2}{4}\sqrt{\frac{\pi}{2}}\right] = \\ = \frac{1}{2}e^{-\frac{(a-b)^2}{2}}\sqrt{\frac{\pi}{2}}\left[\frac{1}{4} + \frac{(a+b)^2}{4}\right] \overset{\text{by (9)}}{=\!=} \frac{1+(a+b)^2}{8}\langle\phi_a, \phi_b\rangle\end{gathered} \tag{17}$$

Finally, the kinetic integral:

$$\left\langle \phi_a, -\frac{1}{2}\frac{d^2}{dx^2}\phi_b \right\rangle = -\int\limits_{\mathbb{R}} e^{-(x-a)^2}\frac{1}{2}\left[\frac{d^2}{dx^2}e^{-(x-b)^2}\right]dx \overset{\text{by (15)}}{=\!=}$$

$$= -\int\limits_{\mathbb{R}} \left[2(x-b)^2 - 1\right] e^{-(x-a)^2}e^{-(x-b)^2}dx =$$

$$= -\int\limits_{\mathbb{R}} \left[2x^2 - 4bx + 2b^2 - 1\right] e^{-(x-a)^2}e^{-(x-b)^2}dx =$$

$$= -4\int\limits_{\mathbb{R}} e^{-(x-a)^2}\frac{1}{2}x^2e^{-(x-b)^2}dx + 4b\int\limits_{\mathbb{R}} xe^{-(x-a)^2}e^{-(x-b)^2}dx - (2b^2-1)\int\limits_{\mathbb{R}} e^{-(x-a)^2}e^{-(x-b)^2}dx =$$

$$= -4\left\langle \phi_a, \frac{1}{2}x^2\phi_b \right\rangle + 4b\int\limits_{\mathbb{R}} xe^{-(x-a)^2}e^{-(x-b)^2}dx - (2b^2-1)\langle\phi_a,\phi_b\rangle \overset{\text{by (17)}}{=\!=}$$

$$= -\frac{1+(a+b)^2}{2}\langle\phi_a,\phi_b\rangle + 4b\int\limits_{\mathbb{R}} xe^{-(x-a)^2}e^{-(x-b)^2}dx - (2b^2-1)\langle\phi_a,\phi_b\rangle = \tag{18}$$

$$= -\left[\frac{1+(a+b)^2}{2} + 2b^2 - 1\right]\langle\phi_a,\phi_b\rangle + 4b\int\limits_{\mathbb{R}} xe^{-(x-a)^2}e^{-(x-b)^2}dx \overset{\text{by (11)}}{=\!=}$$

$$= -\left[\frac{1+(a+b)^2}{2} + 2b^2 - 1\right]\langle\phi_a,\phi_b\rangle + 4be^{-\frac{(a-b)^2}{2}}\int\limits_{\mathbb{R}} xe^{-2\left(x-\frac{a+b}{2}\right)^2}dx \overset{\text{by (14)}}{=\!=}$$

$$= -\left[\frac{1+(a+b)^2}{2} + 2b^2 - 1\right]\langle\phi_a,\phi_b\rangle + 4be^{-\frac{(a-b)^2}{2}}\frac{a+b}{2}\sqrt{\frac{\pi}{2}} \overset{\text{by (9)}}{=\!=}$$

$$= -\left[\frac{1+(a+b)^2}{2} + 2b^2 - 1 - 4b\frac{a+b}{2}\right]\langle\phi_a,\phi_b\rangle =$$

$$= -\frac{1}{2}\left[1+(a+b)^2 + 4b^2 - 2 - 4b(a+b)\right]\langle\phi_a,\phi_b\rangle =$$

$$= \frac{1-(a-b)^2}{2}\langle\phi_a,\phi_b\rangle$$

We could as well put this into WolframAlpha and get the results immediately. Anyways, we are ready to code!

The final source code for the following section is available online.

First of all, import the required libraries:

```
import math
import numpy
import scipy.linalg
import matplotlib.pyplot as pyplot
```

Write down the formulas for overlap(9), kinetic(18), and potential(17) integrals in python code:

```
def overlap(a, b):
    return math.sqrt(math.pi / 2) * numpy.exp(- (a - b)**2 / 2)


def potential(a, b):
    return ((a + b)**2 + 1) / 8 * overlap(a,b)


def kinetic(a, b):
    return (1 - (a-b)**2) / 2 * overlap(a,b)
```

I am using `numpy.exp` instead of `math.exp` so that we can pass numpy arrays in place of `a` and `b`.

Now, time to build the matrices of our main equation(5): $H\psi = ES\psi$. We need a function that, given a description of a basis set, returns a pair of matrices `(H,S)`. Our basis functions ψ_a are parametrized by a single parameter `a`, so a basis set can be described as a list of numbers that correspond to different `a`'s.

```
def equation(basis):
    D = len(basis)
    H = numpy.zeros((D,D))
    S = numpy.zeros((D,D))

    for i in range(D):
        for j in range(D):
            ai = basis[i]
            aj = basis[j]

            H[i,j] = kinetic(ai,aj) + potential(ai,aj)
            S[i,j] = overlap(ai,aj)

    return (H, S)
```

Next, we need a function that will solve the generalized eigenvalue problem(5). Luckily, `scipy` package already knows how to deal with these! We also need to normalize the resulting wavefunctions, but `scipy.linalg.eigh` does this as well.

```
def solve(H, S):
    return scipy.linalg.eigh(H, S)
```

We have all the building blocks to run the code. What basis shall we pick, though? Let's try something silly, like a basis of $e^{-(x+2)^2}$, $e^{-(x+1)^2}$, e^{-x^2}, $e^{-(x-1)^2}$, and $e^{-(x-2)^2}$:

```
basis = [-2, -1, 0, 1, 2]
energy, coeffs = solve(*equation(basis))
print("Energy:", energy)
```

Here, `energy` is an array of energy eigenvalues (values of E in $H\psi = ES\psi$) while `coeffs` is a matrix with its k-th *column* containing the coefficients c_i in the sum $\psi(x) = \sum c_i\phi_i(x)$ for the quantum state ψ that corresponds to the k-th value of E.

If you run the script, you should get something like

```
Energy: [0.50028959 1.50654872 2.50906592 3.65198693 4.52712922]
```

We got the ground-state energy of `0.50028959`, correct within 0.06%! Even the next two energies are very close to the real values of `1.5` and `2.5`. Congratulations! You've just completed your (probably) first numerical quantum-mechanical computation. Before we move on to chemical systems, let's see what else we can do with our current code.

We can optimize the `equation` function by using `numpy.meshgrid`, since operations on `numpy` arrays are generally faster than python loops:

```
def equation(basis):
    Ai, Aj = numpy.meshgrid(basis, basis)

    H = kinetic(Ai, Aj) + potential(Ai, Aj)
    S = overlap(Ai, Aj)

    return (H, S)
```

We can plot the wavefunction that we've found and compare it against the true ground-state wavefunction $\frac{1}{\sqrt[4]{\pi}} e^{-\frac{x^2}{2}}$. For that matter, define a function that computes ϕ_a:

```
def phi(a, x):
        return numpy.exp(-(x-a)**2)
```

Then, a function that evaluates the wavefunction:

```
def wavefunction(basis, coeffs, x):
    assert len(basis) == len(coeffs)

    y = numpy.zeros(x.shape)
    for i in range(len(basis)):
        y += coeffs[i] * phi(basis[i], x)
    return y
```

For convenience, let's define the real wavefunction as well:

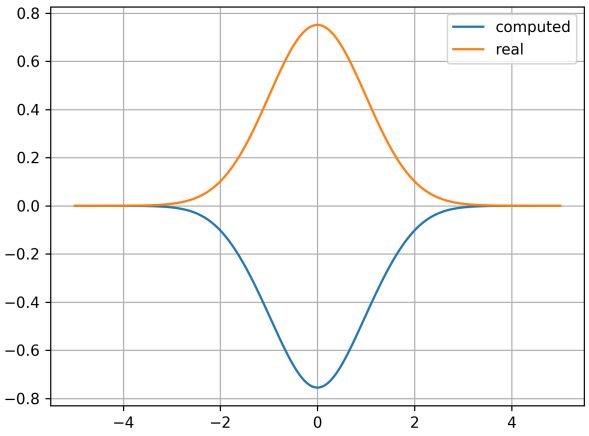
```
def ground_state(x):
        return 1.0 / math.pow(math.pi, 0.25) * numpy.exp(- x**2 / 2.0)
```

To plot the results, add this to the end of the script:

```
x = numpy.linspace(-5, 5, 1001)
computed_gound_state = wavefunction(basis, coeffs[:,0], x)
real_ground_state = ground_state(x)

pyplot.plot(x, computed_gound_state, label="computed")
pyplot.plot(x, real_ground_state, label="real")
pyplot.grid()
pyplot.legend()
pyplot.show()
```

The `:` in `coeffs[:,0]` is a special placeholder that mean *all possible indices*. Thus, `coeffs[:,0]` takes the 0-th column of `coeffs`.

If you run the code, you might (or might not!) get something like this:



Why is your wavefunction upside down?! There's nothing wrong with that, actually: if ψ is a solution, then $-\psi$ is a solution as well. In fact, these two represent *the same quantum state*. We could negate our wavefunction if it is negative at $x = 0$:

```
x = numpy.linspace(-5, 5, 1001)
computed_gound_state = wavefunction(basis, coeffs[:,0], x)
real_ground_state = ground_state(x)

if wavefunction(basis, coeffs[:,0], 0.0) < 0.0:
    computed_gound_state *= -1.0

pyplot.plot(x, computed_gound_state, label="computed")
pyplot.plot(x, real_ground_state, label="real")
pyplot.grid()
pyplot.legend()
pyplot.show()
```

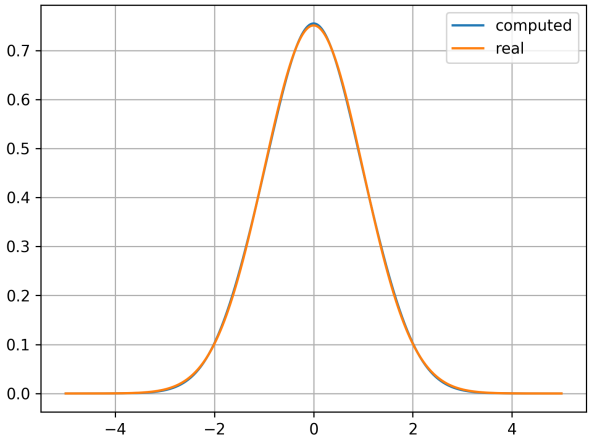
This code will result in an error like this:

```
AttributeError: 'float' object has no attribute 'shape'
```

Oops, we need to fix `wavefunction` so that it works in case `x` is a number and not a `numpy` array:

```
def zero_like(x):
    if type(x) == numpy.ndarray:
        return numpy.zeros(x.shape)
    elif type(x) == float:
        return 0.0
    else:
        raise RuntimeError("Unknown type {}".format(type(x)))


def wavefunction(basis, coeffs, x):
    assert len(basis) == len(coeffs)

    y = zero_like(x)
    for i in range(len(basis)):
        y += coeffs[i] * phi(basis[i], x)
    return y
```

This time, we should have the two wavefunctions on the same side of the x-axis:



As you can see, our computed solution is very close the the true one! Can we get any closer? Sure, but we need a better basis than just `[-2, -1, 0, 1, 2]`. We need to tweak a bunch of numbers so that

another number (the energy) gets smaller — this is precisely what is called mathematical optimization. Luckily, `scipy` already comes with routines that do just that! Add this to imports:

```
import scipy.optimize
```

Now, define our target function that we want to minimize — the ground-state energy:

```
def target(basis):
        energy, coeffs = solve(*equation(basis))
        return energy[0]
```

The main script now goes as follows: we pick initial basis parameters, optimize them, and see the energy values:

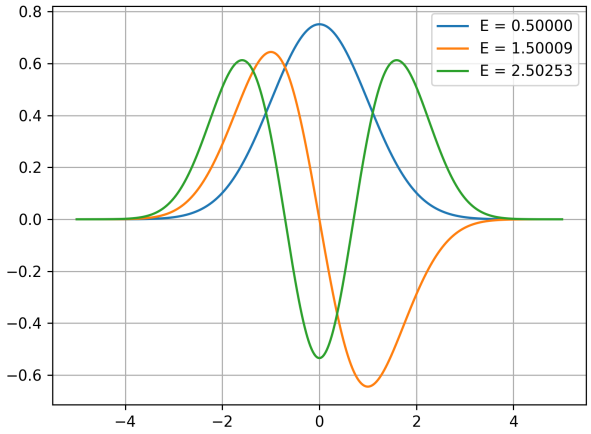
```
def optimize(basis):
        result = scipy.optimize.minimize(target, basis)
        if not result.success:
            raise RuntimeError("Energy minimization failed")
        return result.x


basis = optimize([-2, -1, 0, 1, 2])
print("Basis:", basis)

energy, coeffs = solve(*equation(basis))
print("Energy:", energy)
```

The results are pretty convincing:

```
Basis: [-1.62551507e+00 -7.69713217e-01 -1.08490137e-04  7.69683879e-01
  1.62560700e+00]
Energy: [0.50000139 1.50009365 2.50253397 3.53463459 4.78613676]
```

We can plot some more wavefunctions, not only the ground state:

```
x = numpy.linspace(-5, 5, 1001)

for k in range(3):
    state = wavefunction(basis, coeffs[:,k], x)
    pyplot.plot(x, state, label="E = {}".format(energy[k]))

pyplot.grid()
pyplot.legend()
pyplot.show()
```



We could try the same thing starting with a larger basis, say, from -5 to 5:

```
basis = optimize(range(-5, 6))
energy, coeffs = solve(*equation(basis))
print("Energy:", energy)
```

Notice how the ground-state energy gets even closer to the true value of 0.5.

```
Energy: [ 0.50000007  1.50000621  2.5002545   3.50575575  4.56901491
  5.85678673  6.52567398  9.508999    9.70153557 14.49609236
 14.55364916]
```

Beware, though, that this method is **not guaranteed to converge**. Indeed, in the following sections we will see that it fails often enough. The upside is the method's simplicity compared to what is used by professional software.

This section is already 8 pages long. I hope you got the basis idea, for it is time to move on to real atoms.

7 Practical considerations

We have seen that in order to use the linear variational method we need to pick a nice basis set — a bunch of parametrized functions that are well-suited for integrals. A particularly nice choice for chemical systems are *gaussian-type orbitals*, often abbreviated as GTOs, multiplied by Hermite polynomials. Usually, *cartesian* GTOs (CGTOs) are used, where a gaussian is multiplied by monomials of the form $x^a y^b z^c$. However, Hermite polynomials make the integrals much easier, — in fact, the derivations of integrals for CGTOs often include steps to convert them to Hermite polynomials, so why not use them directly instead? Furthemore, this cartesian gaussian-type orbitals are usually used indirectly: the actual basis set used is Slater-type orbitals (STOs), with functions taken directly from eigenstates of the hydrogen atom, but those are internally represented as a linear combination of GTOs (known as *contracted* GTOs or cGTOs) to simplify the integrals. It is however a lot easier to work with gaussian-type functions directly instead. The general form of such a function looks like this:

$$\phi_{R,\sigma,n}(x,y,z) = H_{n_x}\left(\frac{x-R_x}{\sigma}\right) H_{n_y}\left(\frac{y-R_y}{\sigma}\right) H_{n_z}\left(\frac{z-R_z}{\sigma}\right) e^{-\frac{(x-R_x)^2+(y-R_y)^2+(z-R_z)^2}{\sigma^2}} \tag{19}$$

It has three integer parameters n_x, n_y, n_z and four real parameters R_x, R_y, R_z, σ. This function is localized around the point (R_x, R_y, R_z), with the parameter σ controlling the spread (less σ means more localized function). $H_n(x)$ is the n-th Hermite polynomial, and the integers n_x, n_y, n_z are responsible for that. They make these basis functions look more like hydrogen p-orbitals, d-orbitals, f-orbitals, and so on. Note that in case $n_x = n_y = n_z = 0$, the basis function is just a gaussian.

I've put the derivations of the integrals with this basis functions to the appendix. These integrals are arguably the most difficult and the least interesting or instructive part of the whole process. You may look into the appendix and type the formulas into python (or whatever language you use) yourself, or you may download a package from github that implements them. The code in the following sections assumes the latter.

As I've already said, although complex numbers are invaluable in quantum mechanics, we don't need them. The reason is that in chemical systems the energy eigenstates can always be made real[13]. Note however that a general state is a mixture of those eigenstates with *complex coefficients*.

Any real-world computation is unimaginable without a choice of units. There is a specialized unit system called the *atomic units*, where most constants relevant to quantum chemistry are set to 1, specifically

- The electron mass m_e
- The electron charge e
- The reduced Planck constant $\hbar$
- The Coulomb constant k_e

The energy unit in this system is called *hartree* and equals approximately 27.2 electron-volts or 4.35×10^{-18} joules. There's a nice energy converter online. The speed of light in this unit system is just 137. Keep all that in mind when reading the equations in the following sections: there are no physical constants in them, not because they shouldn't be there, but because they are all equal to 1.

One final note: all that follows assumes the Borh-Oppenheimer approximation, which states that since atomic nuclei are much[14] heavier than the electrons, the latter should move orders of magnitude faster, thus it is possible to separate the movement of nuclei from that of electrons, effectively treating the nuclei as being static. This is exactly what we are going to do: solve the equations for the electrons assuming static nuclei, and solve for the nuclei separately.

[13] Technically, this boils down to the fact that the Hamiltonian commutes with complex conjugation, thus also with projectors on spaces of real-valued and imaginary-valued functions. If an eigenstate is purely imaginary-valued, multiply it by i to get a real-valued one, otherwise apply the projector to real-valued functions.

[14] A proton is 1836 times more massive than an electron, see a corresponding wikipedia article.

8 Hydrogen atom H

At last we arrived to an actual chemical problem: the hydrogen atom. Assuming that the nucleus is at the origin, the Hamiltonian looks like this:

$$\hat{H}\psi = -\frac{1}{2}\Delta\psi - \frac{1}{|r|}\psi \tag{20}$$

Here,

- ψ is a function (the wavefunction) of a three-component vector r[15] (technically, ψ is an element of the space $L^2(\mathbb{R}^3)$)
- Δ is the Laplace operator $\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ and accounts for the kinetic energy of an electron
- $|r| = \sqrt{x^2 + y^2 + z^2}$ is the distance to nucleus
- $-\frac{1}{|r|}$ is the Coulomb potential energy of the attraction between electron (at distance $|r|$) and hydrogen nucleus.

It is one of the few real-world quantum systems with known analytical solutions (see here). The energies are (in atomic units!) $E_n = -\frac{1}{2n^2}$, where $n = 1, 2, 3, \ldots$ is that number written in front of atomic orbitals like 1s, 2s, 2p, 3s, 3p, 3d, etc. The corresponding wavefunctions look like those funny blobs[16] from any introductory chemistry textbook.

The final source code for the following section is available online.

As I've said in the Practical considerations section, the code assumes you have the `hgto` module that implements all the integrals for Hermite-Gaussian orbitals. Start with the imports:

```
import numpy
import scipy.linalg
import scipy.optimize
import hgto
```

Now, the already familiar equation-building function:

```
def equation(basis):

    B = len(basis)

    H = numpy.zeros((B,B))
    S = numpy.zeros((B,B))

    n = hgto.Nucleus(1.0)

    for i in range(B):
        for j in range(B):
            bi = basis[i]
            bj = basis[j]
            S[i,j] = hgto.overlap(bi, bj)
            H[i,j] = hgto.kinetic(bi, bj) + hgto.nuclear_attraction(bi, bj,

    return (H, S)
```

[15] r may be thought of as representing the position of the electron, but care should be taken, for there is no *the* position of an electron due to Heisenberg uncertainty, and each possible r has a probability of finding the electron in that particular place.

[16] See also my interactive visualizer.

You've probably noticed that we have a new thing here: the nucleus, represented as an object of type `hgto.Nucleus`. The constructor has a second parameter `origin` that corresponds to the location of the nucleus. In the case of a single atom, it is reasonable to leave it at the default value of `[0.0,0.0,0.0]`. The `nuclear_attraction` function computes the integral corresponding to attraction to that single nucleus. `basis` is a list of basis functions represented as objects of type `hgto.HGTO`.

Next, the function that solves the generalized eigenvalue equation doesn't change at all:

```
def solve(H, S):
    return scipy.linalg.eigh(H, S)
```

Now, time to pick a set of basis functions. The constructor of the `hgto.HGTO` class looks like this:

```
    def __init__(self, sigma, degree=[0,0,0], origin=[0.0,0.0,0.0]):
        ...
```

The `sigma`, `degree`, and `origin` parameters correspond to σ, $[n_x, n_y, n_z]$, and $[R_x, R_y, R_z]$ parameters of an HGTO-orbital $H_{n_x}(x)H_{n_y}(y)H_{n_z}(z)e^{-\frac{(x-R_x)^2+(y-R_y)^2+(z-R_z)^2}{\sigma^2}}$, respectively. Since we are working with a single atom, it is reasonable to set the origin of all basis functions to the location of the nucleus.

The `degree` parameter is a bit more subtle. The value of `[0,0,0]` corresponds to a basis function that is radially symmetric, much like the s-orbitals of hydrogen, so let's start with them. Thus, our only adjustable parameters will be `sigma`'s. Write a function that takes a list of `sigma` and outputs a corresponding basis:

```
def make_basis(params):
    basis = []

    for sigma in params:
        basis.append(hgto.HGTO(sigma))

    return basis
```

As before, define the optimization target function and the optimization routine. This time, we will select the optimization method other than the default one: I figured that the *Nelder-Mead* simplex-based method works pretty well (you may try other methods, of course!). We'll say it to be adaptive, so that it tweaks the internal parameters as needed. It is also helpful to provide a callback function that will record optimization progress.

```
def target(params):
    basis = make_basis(params)
    energy, coeffs = solve(*equation(basis))
    return energy[0]


def callback(params):
    print(params)


def optimize(params):
    result = scipy.optimize.minimize(target, params, method='Nelder-Mead',
        options={'adaptive': True}, callback=callback)
    if not result.success:
        raise RuntimeError("Energy minimization failed")
    return result.x
```

Finally, pick a random set of, say, five initial parameters, invoke the optimization, and look at the results. How do I know to choose the parameters in the $[0.5, 5.0]$ range? This is where our choice of

units helps: all related constants are set to 1, and we can expect all physically significant properties of the system to be of the same order of magnitude, that is, close to 1. Thus, we choose some arbitrary range that is close to 1.

```
params = optimize(numpy.random.uniform(0.5, 5.0, 5))

basis = make_basis(params)
energy, coeffs = solve(*equation(basis))
print("Energy:", list(energy))
```

You might see some errors here. Firstly, the minimization routine might not succeed after some default maximal number of iterations, — this can probably be fixed by adjusting the solver options. Secondly, you may encounter an error like this:

```
numpy.linalg.LinAlgError: the leading minor of order 2 of 'b' is not
positive definite. The factorization of 'b' could not be completed and
no eigenvalues or eigenvectors were computed.
```

This comes from the Cholesky decomposition that is invoked as part of `scipy.linalg.eigh` and means that the basis that we supplied is too close to being linearly dependent, — in particular, if two of the `sigma` are too close to each other. You might also notice that the error pops up a lot when the `sigma` become negative: due to numerical errors having $\sigma_1 \approx -\sigma_2$ leads to erroneously low values of energy, while the corresponding basis functions are almost the same (since only σ^2 is involved in the formulas). It would be better to use constrained optimization (to require $\sigma > 0$), or use singular value decomposition and restrict to a better-behaving subbasis.

It is however much easier to just call the script again in hope that this time the random initial parameters would be better, or even wrap the `optimize` routine with a loop that starts over and over again until the optimization succeeds without an exception. In any case, you should get something like this:

```
Energy: [-0.4998098322278446, 0.023987247113695483, 1.5010945028423488,
 9.012692390104283, 60.979054059578246]
```

The ground-state energy is very close to the actual value of -0.5, while the others are complete garbage (they are even positive, meaning that they do not correspond to bound states). This is a general phenomenon: is you optimize for something, don't expect something else to be optimized as well. There are several ways to make this work for excited ($n = 2, 3, \ldots$) states. One could find the ground state first, then employ the same procedure but with an extra step of reprojecting the basis on the subspace orthogonal to the ground state to find the first excited state, then doing the same but orthogonal to the two already found states to find the next excited state, and so on. This makes the basis grow rapidly (since you need both the old basis and the new one to make the new one orthogonal to the old). We may, however, try to trick our solver a bit, and make it optimize for the *sum* of a few lowest energies (say, three of them), at the price of having a lower overall precision:

```
def target(params):
    basis = make_basis(params)
    energy, coeffs = solve(*equation(basis))
    return sum(energy[0:3])
```

The results are not that bad:

```
Energy: [-0.4976228254475445, -0.12462108469029497, -0.05542836872909056,
 0.5864866278502727, 7.547798891619299]
```

The first three energy values are pretty close to the actual values of -0.5, -0.125, and $-\frac{1}{18} = -0.0(5)$, corresponding to 1s, 2s, and 3s hydrogen orbitals. Time to visualize them!

How does one visualize a three-dimensional function? Usually this requires volume rendering, isosurfaces, or slicing the space. However, while we are dealing with highly symmetric functions, we can manage without such sophisticated machinery. Recall that s-orbitals are spherically symmetric, meaning that we can just plot a cross-section along the X axis.

As before, add `matplotlib` to the imports:

```
import matplotlib.pyplot as pyplot
```

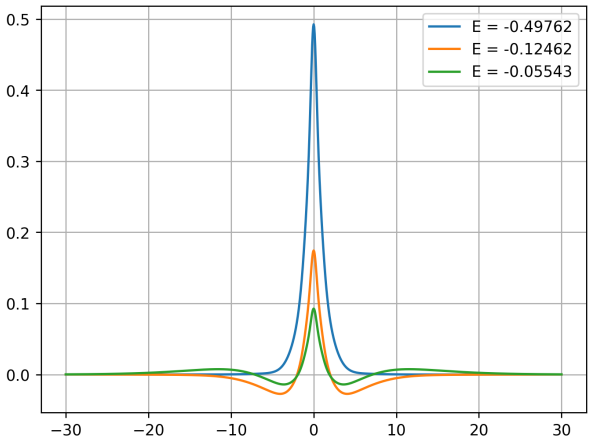
Write the function that evaluates our wavefunction at a specific point (`x`, `y`, and `z` may as well be `numpy` arrays):

```
def zero_like(x):
    if type(x) == numpy.ndarray:
        return numpy.zeros(x.shape)
    elif type(x) == float:
        return 0.0
    else:
        raise RuntimeError("Unknown type {}".format(type(x)))


def wavefunction(basis, coeffs, x, y, z):
    assert len(basis) == len(coeffs)

    f = zero_like(x)
    for i in range(len(basis)):
        f += coeffs[i] * basis[i](x, y, z)
    return f
```

Now, evaluate our three lowest states along the X axis and plot them:

```
L = 30
N = 1001
X = numpy.linspace(-L, L, N)

for k in range(3):
    state = wavefunction(basis, coeffs[:,k], X, 0, 0)
    pyplot.plot(X, state, label="E = {0:.5f}".format(energy[k]))

pyplot.grid()
pyplot.legend()
pyplot.show()
```

This results in something like (remember that the wavefunctions may be upside down)



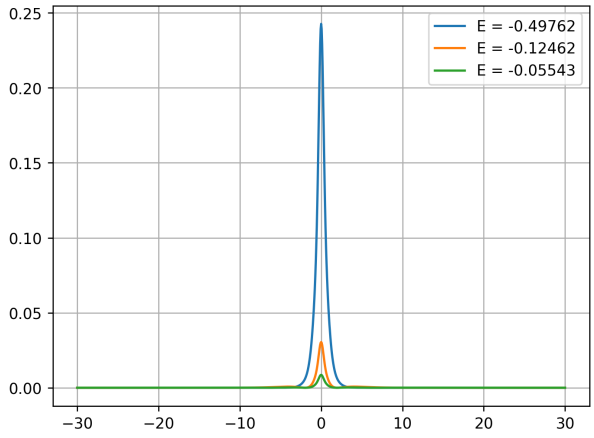
A few notable things can be read from this plot:

- All states have singularities[17] at the origin, where the nucleus is located
- Higher-energy orbitals are less concentrated near the nucleus and are more spread around
- 1s orbital has the same sign everywhere, 2s orbital changes sign once, 3s changes sign twice, etc.

Let's plot the electron density along the X axis as well:

```
def density(basis, coeffs, x, y, z):
    f = wavefunction(basis, coeffs, x, y, z)
    return f * f
```

```
for k in range(3):
    state = density(basis, coeffs[:,k], X, 0, 0)
    pyplot.plot(x, state, label="E = {0:.5f}".format(energy[k]))

pyplot.grid()
pyplot.legend()
pyplot.show()
```

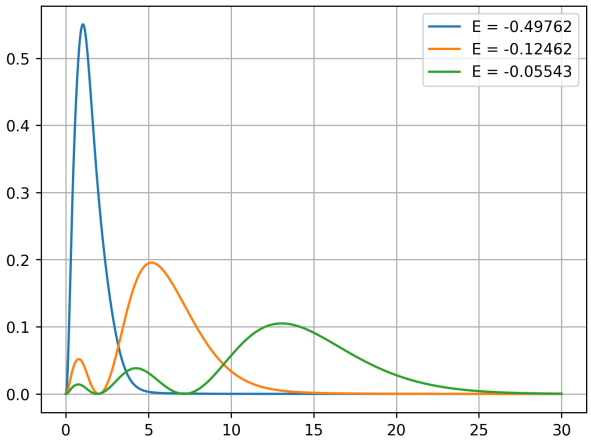


This is hardly useful, since the density gets extremely small as we get futher from the nucleus. This, however, is the electron density *at a specific point*, while we might be interested in density *of a specific distance from the nucleus*, i.e. density integrated over a sphere of a specific radius. Since the states are spherically symmetric, the integral over a sphere is a value at a point times the area of the sphere $4\pi r^2$:

```
def radial_density(basis, coeffs, r):
    f = density(basis, coeffs, r, 0, 0)
    return f * numpy.pi * 4 * (r**2)
```

```
R = numpy.linspace(0, L, N)

for k in range(3):
    state = radial_density(basis, coeffs[:,k], R)
    pyplot.plot(R, state, label="E = {0:.5f}".format(energy[k]))
```

[17] Even for many-electron systems one can reconstruct the original system knowing this cusps only, see Kato's cusp condition. This is especially important for the foundations of density functional theory (DFT).



This plot is my favorite concerning s-orbitals. You can easily see where are those orbitals localized in space, where are the sign changes, etc. The distance of maximal probability for the 1s-orbital is known as Bohr radius (in atomic units it is exactly equal to 1).

Let's try some p-orbitals!

The final source code for the following section is available online.

Hydrogen (and hydrogen-like) orbitals are very special. While s-orbitals are spherically symmetric, p-orbitals are symmetric with respect to rotations around a particular axis and antisymmetric (change sign) upon reflection through that axis. As an example, the p_x-orbital doesn't change when rotated around the X-axis and flips sign when the X-coordinate changes sign.[18]

All this, together with basic group representation theory, lets us pick specific basis functions for specific orbitals based on symmetry. We have already used spherically symmetric basis functions for s-orbitals, now we should pick appropriate ones for p-orbitals. Taking HGTO's with `degree` equal to `[1,0,0]` seems reasonable: they have the same symmetries as p_x-orbitals.

Change the `make_basis` function to

```
def make_basis(params):
    basis = []

    for sigma in params:
        basis.append(hgto.HGTO(sigma, [1,0,0]))

    return basis
```

Optimize again for the lowest energy:

```
def target(params):
    basis = make_basis(params)
    energy, coeffs = solve(*equation(basis))
    return energy[0]
```

As before, pick some random starting sigmas and optimize:

```
params = optimize(numpy.random.uniform(0.5, 5.0, 5))

basis = make_basis(params)
energy, coeffs = solve(*equation(basis))
print("Energy:", list(energy))
```

[18] There's much more to it: hydrogen orbitals correspond to irreducible representations of the group $SO(3)$ of rotations in three dimensions.

There are no 1p-orbitals, only 2p, 3p, etc. This means that the lowest energy value we should get is -0.125:

```
Energy: [-0.124990568560368, -0.04460409913403034, 0.10989269961085461,
 0.7056560089020736, 3.5767938893908897]
```

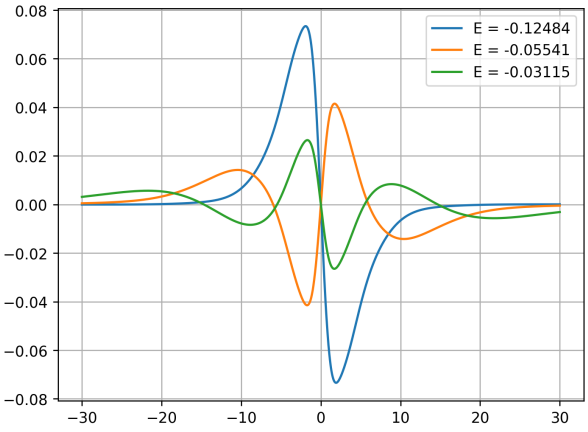
The first value is surprisingly close to the real answer. Let's use the same trick of optimizing the sum of a few first energies to get more orbitals:

```
def target(params):
    basis = make_basis(params)
    energy, coeffs = solve(*equation(basis))
    return sum(energy[0:3])
```

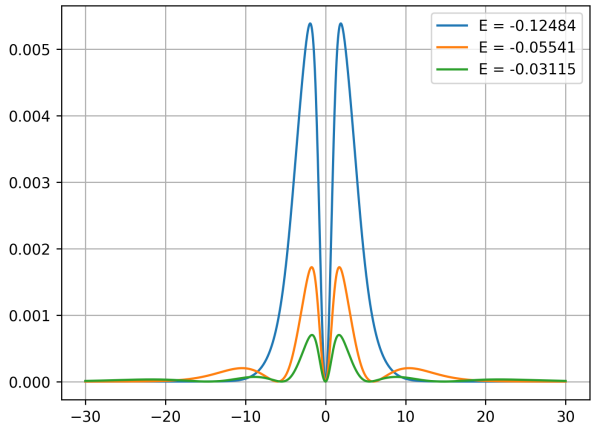
```
Energy: [-0.1248431191321489, -0.05541009666729667, -0.031146344230485228,
 0.0501113155120981, 0.6710920124828863]
```

Again, the precision of each orbital is questionable compared to actual values of -0.125, $-\frac{1}{18} = -0.0(5)$, and $-\frac{1}{32} = -0.03125$, but we got three p-orbitals ($2p_x$, $3p_x$, and $4p_x$) using only five basis functions! We may plot them along the X axis, as before:

```
L = 30
N = 1001
X = numpy.linspace(-L, L, N)

for k in range(3):
    state = wavefunction(basis, coeffs[:,k], X, 0, 0)
    pyplot.plot(X, state, label="E = {0:.5f}".format(energy[k]))
```



We can plot the density, as well:



Radial density doesn't make much sense in this case, since p-orbitals are not spherically symmetric.

You may try getting more sophisticated orbitals (3d, 4d, 4f, ...) with higher `degree` of HGTO's, but the basis starts being close to degenerate, and `numpy.linalg.LinAlgError` appears more and more often.[19] At this point, we move on to other systems.

[19] As I've said before, it is possible to overcome this using SVD, but this is out of the scope of this book.

9 Helium cation He^+

The next one-electron system we are going to explore is the helium +1 cation — a helium atom that lost one of its two electrons. This one is very similar to the hydrogen atom, the only difference being the nucleus charge. The Hamiltonian changes to

$$\hat{H}\psi = -\frac{1}{2}\Delta\psi - \frac{Z}{|r|}\psi \tag{21}$$

Look at wikipedia again to see what the energies are: $-\frac{Z^2}{2n^2}$, where Z is the charge of the nucleus: $Z = 1$ for hydrogen, $Z = 2$ for helium. Thus, we expect the energies to be 4 times the energies for hydrogen.

The final source code for the following section is available online.

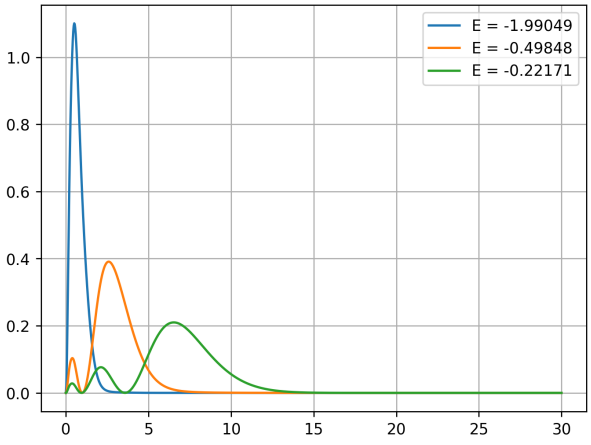
Change the nucleus charge to 2:

```
def equation(basis):
    ...
    n = hgto.Nucleus(2.0)
    ...
```

to get

```
Energy: [-1.990491312820971, -0.49848433212591076, -0.22171347050750198,
 2.3459781841680547, 30.19206227078643]
```

The three first energies are close to being right. Plotting the radial density



we can see that the orbitals are closer to the nucleus compared to the hydrogen orbitals. This makes sense: the higher the nuclear charge, the stronger is the nucleus-electron attraction. This leads to an interesting phenomenon: all atoms are *roughly* of the same size! Larger atomic number means higher nucleus charge, so the electrons are attracted more and fly closer to the nucleus; however, more electrons means they repel each other and don't concentrate near the nucleus.[20]

We can find and plot the p-orbitals as well, which will again look the same as hydrogen p-orbitals, but closer to the nucleus and having higher energy.

Everything interesting related to helium starts when we add a second electron. Since this book considers one-electron systems only, we move on.[21]

[20] This absence of electron concentration is also largely due to the Pauli principle, in turn related to the fact that electrons are fermions. Bosons, however, *can* concentrate in one place, which leads to the existence of Bose-Einstein condensate.

[21] A *rough* description of what happens in a real helium atom is that the two-electron density looks a lot like the 1s-orbital, but slightly more spread out due to electron-electron repulsion. Pauli exclusion principle doesn't change anything in two-electron systems: we have two electrons and two possible spins as well. See also here and there.

10 Lithium 2+ cation Li^{2+}

This is, again, almost the same as hydrogen. Lithium nucleus has charge +3, and the lithium atom has 3 electrons at normal conditions. If we steal two of them, we get the lithium +2 cation, which is described the same way as helium, but with $Z = 3$.

The final source code for the following section is available online.

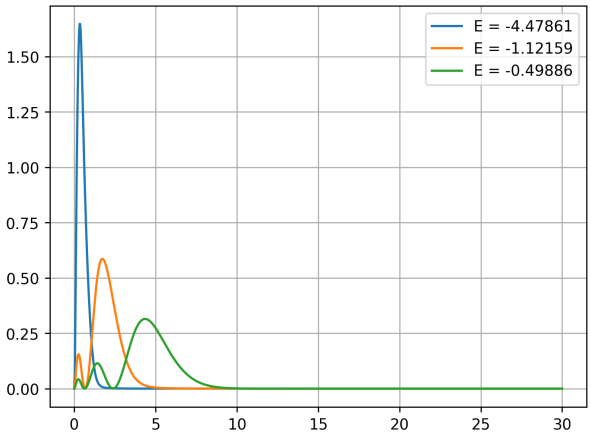
Change the nucleus charge to 3:

```
def equation(basis):
    ...
    n = hgto.Nucleus(3.0)
    ...
```

and get something like

```
Energy: [-4.478605460280523, -1.1215897686285095, -0.49885528082305414,
 5.2782304665358195, 67.92747569252208]
```

Close enough to the true -4.5, -1.125, and -0.5. The orbitals are even closer to the nucleus this time:



Excercise: guess the dependence of orbitals' spread on the value of Z, then look at wikipedia one more time and see if you guessed right.[22]

One can proceed with all atoms the same way, but this is starting to get dull. More importantly, these one-electron atoms are rare in nature: it takes more and more energy[23] to remove all electrons but one from a heavy atom. We haven't seen *chemical bonding* yet — the time has come.

[22] Hint: look at the exponential term of the radial part of the wavefunction.
[23] See also here.

11 Dihydrogen cation H_2^+

The nature of chemical bonding is quite a delicate one, even in a system of two protons (that is, two hydrogen nuclei) bonded through a single electron. The conventional argument[24] states that accumulating electron density between two nuclei lowers the potential energy, thus providing a bond. A more careful analysis uncovers that bonding is achieved via redistribution of energy between kinetic and potential parts[25]. Luckily, we don't need explanations to throw numbers at linear algebra solvers.

The Hamiltonian for dihydrogen cation (also called the *hydrogen molecular ion*) is

$$\hat{H}\psi = -\frac{1}{2}\Delta\psi - \frac{1}{|r-R_1|}\psi - \frac{1}{|r-R_2|}\psi + \frac{1}{|R_1-R_2|}\psi \tag{22}$$

We have a few new terms here:

- R_1 is the position of the first nucleus
- $|r-R_1|$ is the distance from electron to the first nucleus
- $-\frac{1}{|r-R_1|}$ is the energy of attraction to the first nucleus
- R_2 is the position of the second nucleus
- $|r-R_2|$ is the distance from electron to the second nucleus
- $-\frac{1}{|r-R_2|}$ is the energy of attraction to the second nucleus
- $\frac{1}{|R_1-R_2|}$ is the energy of repulsion between the nuclei

We can reuse the code from previous section just by introducing a second nucleus...where? We can, without loss of generality, assume that both nuclei lay on the X axis, at some distance R between them. Say, $R_1 = [-\frac{R}{2}, 0.0, 0.0]$ and $R_2 = [\frac{R}{2}, 0.0, 0.0]$ (no particular reason for that choice except symmetry). Yet how do we know R? We could google the bond length[26] to get the value of bit less than 2 (in atomic units!), but that wouldn't be fair. Instead, let's optimize the bond length R the same way we optimize `sigma`.

The final source code for the following section is available online.

First of all, we cannot simply hardcode the nuclei in `equation` function now, so make it a function agument, and update the function to handle multiple nuclei:

```
def equation(basis, nuclei):

    B = len(basis)

    H = numpy.zeros((B,B))
    S = numpy.zeros((B,B))

    total_nuclear_repulsion = 0.0
    for i in range(len(nuclei)):
        for j in range(i+1, len(nuclei)):
            total_nuclear_repulsion += hgto.nuclear_repulsion(nuclei[i],
                nuclei[j])

    for i in range(B):
        for j in range(B):
            bi = basis[i]
            bj = basis[j]
```

[24] See also wikipedia.
[25] See [1], section 8.2 *An application: the hydrogen molecule-ion.*
[26] See [8]

```
            S[i,j] = hgto.overlap(bi, bj)
            H[i,j] = hgto.kinetic(bi, bj)
            for n in nuclei:
                H[i,j] += hgto.nuclear_attraction(bi, bj, n)
            H[i,j] += S[i,j] * total_nuclear_repulsion

    return (H, S)
```

`hgto.nuclear_repulsion` is just evaluating the elecrostatic potential energy, which we sum over all pairs of distinct nuclei. Recall that we consider nuclei to be static *within a single eigenvalue problem*, so the total energy doesn't depend on the chosen basis for one-electron wavefunction. However, the H matrix is not just the energy, but contains the value of the integral $\langle H\phi_i, \phi_j\rangle$. Specifically, for the nuclei repulstion term we get

$$\left\langle \frac{1}{|R_1 - R_2|}\phi_i, \phi_j \right\rangle = \int_{\mathbb{R}^3} \frac{1}{|R_1 - R_2|}\phi_i(r)\phi_j(r)dr = \frac{1}{|R_1 - R_2|}\int_{\mathbb{R}^3} \phi_i(r)\phi_j(r)dr = \frac{1}{|R_1 - R_2|}\langle \phi_i, \phi_j\rangle \quad (23)$$

(this happens every time when a specific energy term doesn't depend on r). Thus, we reuse the overlap integral $\langle \phi_i, \phi_j\rangle$ already computed into `S[i,j]`.

Next, we need to update the `make_basis` function: since the nuclei also depend on the optimized parameters, it would be nice for the function to return both the nuclei and the chosen basis. The first number in the parameters list will be the internuclear distance R, while all the others will be the `sigma`'s, and for each `sigma` we'll make two basis functions, each centered on one of the two nuclei:

```
def make_basis(params):
    R = params[0]

    nuclei = [
        hgto.Nucleus(1.0, [-R/2, 0.0, 0.0]),
        hgto.Nucleus(1.0, [ R/2, 0.0, 0.0]),
    ]

    basis = []

    for sigma in params[1:]:
        for n in nuclei:
            basis.append(hgto.HGTO(sigma, origin=n.position))

    return basis, nuclei
```

Now we need to update the target function:

```
def target(params):
    energy, coeffs = solve(*equation(*make_basis(params)))
    return energy[0]
```

And invoke the optimization, starting with a random configuration again:

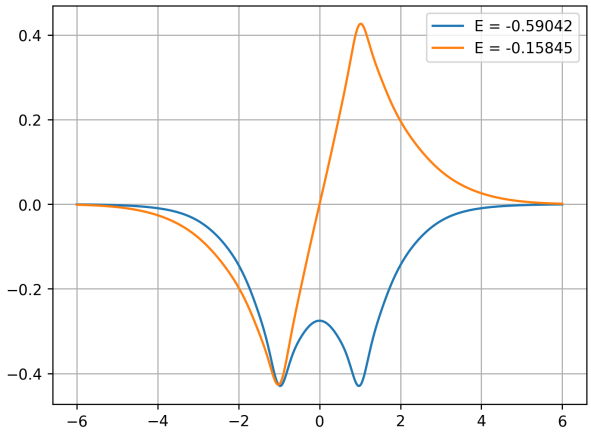
```
params = optimize(numpy.random.uniform(0.5, 5.0, 5))

basis, nuclei = make_basis(params)
energy, coeffs = solve(*equation(basis, nuclei))
print("R:", params[0])
print("Energy:", list(energy))
```

resulting in

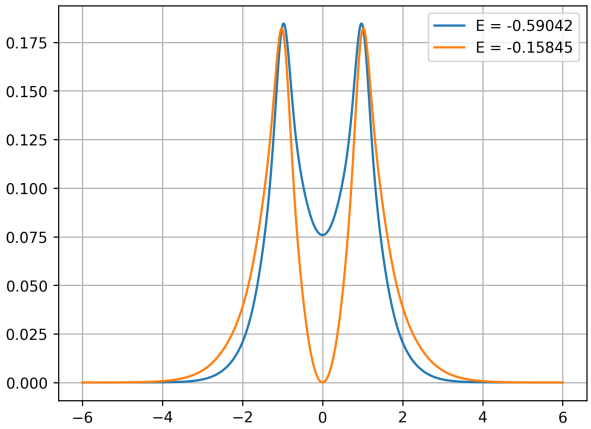
```
R: 1.9812752254550632
Energy: [-0.5904203018386721, -0.158456775787351, 0.41327243101286076,
 0.5952619487434113, 3.918972615674728, 3.9258967834545113,
  29.407028344934997, 29.55225299315845]
```

First of all, why did we get 8 values of energy (and 8 eigenstates)? We have 5 parameters, one of which is R, and the other 4 are `sigma`. Each `sigma` corresponds to 2 basis functions (one for each nucleus), giving a total of 8 basis functions.

The true values are something close to $R = 1.99$ and $E = -0.6$, so we are reasonably close. What is the second orbital with energy -0.158? Before we talk on that, let's plot the two first wavefunctions along the X axis, as usual:



and the corresponding densities:



We can note a few differencies between these states. The state with $E \approx -0.59$

- Is symmetric with respect to swapping the nuclei, thus called a *gerade*[27] state
- Has a notable concentration of density between the nuclei
- Has energy lower than -0.5

while the state with $E \approx -0.16$

- Is antisymmetric with respect to swapping the nuclei, thus called an *ungerade*[28] state

[27] This is German for *even*.
[28] This is German for *odd*.

- Has low density between the nuclei and spreads a bit further to the sides
- Has energy much higher than -0.5.

It seems to be a general phenomenon that the ground state will be as symmetric as it can be[29], emphasizing the difference between the two states.

How do we know whether these states correspond to something real, occurring in nature? Consider the limit case of $R \to \infty$. The repulsion between the nuclei would be negligibly small, and the usual hydrogenic orbitals of the two atoms would be almost non-overlapping (thus have a zero overlap integral, thus be orthogonal). The two lowest energy eigenstates will be the 1s orbital on the first atom and the 1s orbital on the second atom, both having energy -0.5. Due to the variational principle, any state of this system will have energy of at least -0.5. The gerade state of H_2^+, however, has a *lower* energy of ≈ -0.59, meaning that this state is more preferable than having two completely separate nuclei.

The ungerade state has energy much higher than -0.5, meaning that an ion could not exist long in this state. This state is just an eigenstate of the system with fixed nuclei, and would break if you let the nuclei move. This is exactly what happens if you try our trick of optimizing for the sum of the first two energy values: two highly separated hydrogen nuclei give a sum of $-0.5 - 0.5 = -1.0$, while two eigenstates of H_2^+ give a higher value of about -0.75, meaning that optimization would prefer the former, and no ions will be found.

A hydrogen molecule H_2 looks pretty much the same, but the internuclear distance is smaller, about $R \approx 1.4$, because there are two electrons now[30], but the density is a bit more spread out due to electron-electron repulsion.

This is a good place for some excercises: try adding more sophisticated basis functions (like p_x), or just more s-like orbitals, and see if the energy gets better (lower).

The final source code for the following section is available online.

It is also useful to plot the dependence of energy on R. We'll have to tweak the functions a bit, since we don't need to optimize for R now:

```
def make_basis(R, params):
    nuclei = [
        hgto.Nucleus(1.0, [-R/2, 0.0, 0.0]),
        hgto.Nucleus(1.0, [ R/2, 0.0, 0.0]),
    ]

    basis = []

    for sigma in params:
        for n in nuclei:
            basis.append(hgto.HGTO(sigma, origin=n.position))

    return basis, nuclei


def target(R, params):
    energy, coeffs = solve(*equation(*make_basis(R, params)))
    return energy[0]


def optimize(R, params):
    result = scipy.optimize.minimize(lambda params: target(R, params), para
    if not result.success:
        raise RuntimeError("Energy minimization failed")
    return result.x
```

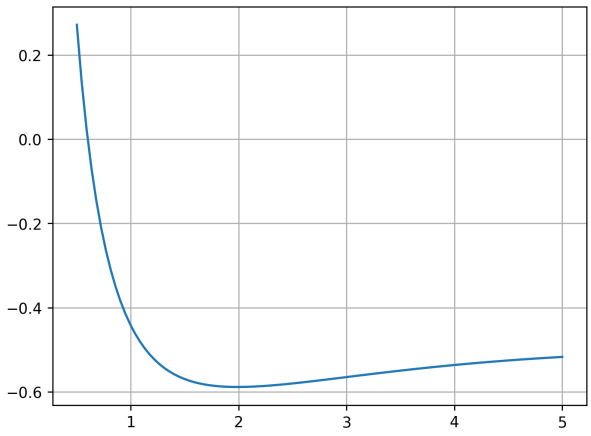
[29] A proof of a similar statement can be found in [2]. See also this math.stackexchange question.
[30] As in helium atom, the Pauli principle hardly changes anything here: two electrons, two spins.

(I've removed the callback since it would have cluttered the output). Now, run the optimization for every R sampled from some range:

```
R = numpy.linspace(0.5, 5.0, 101)
E = numpy.zeros(R.shape)

for i in range(len(R)):
    print("{}/{}".format(i, len(R)))
    while True:
        try:
            params = optimize(R[i], numpy.random.uniform(0.5, 5.0, 3))
            break
        except Exception:
            pass

    basis, nuclei = make_basis(R[i], params)
    energy, coeffs = solve(*equation(basis, nuclei))

    E[i] = energy[0]
```

Finally, plot it:

```
pyplot.plot(R, E)
pyplot.grid()
pyplot.show()
```



We can clearly see that it has a minimum at about $R \approx 2$, as expected. The wikipedia article features a similar plot.

What happens if we add another hydrogen atom?

12 Trihydrogen 2+ cation H_3^{2+}

Let's try the same with three hydrogen nuclei and one electron. This time all our code is already suited to work with an arbitrary number of nuclei, and we only need to change the `make_basis` function. We can assume that the first nucleus is located at the origin, the second is somewhere along the X axis at $[X_2, 0, 0]$, and the third nucleus is somewhere on the XY-plane at $[X_3, Y_3, 0]$. To optimize for this, we need to include X_2, X_3, and Y_3 to the optimized parameters:

```
def make_basis(params):
    X2 = params[0]
    X3 = params[1]
    Y3 = params[2]

    nuclei = [
        hgto.Nucleus(1.0, [0.0, 0.0, 0.0]),
        hgto.Nucleus(1.0, [X2, 0.0, 0.0]),
        hgto.Nucleus(1.0, [X3, Y3, 0.0]),
    ]

    basis = []

    for sigma in params[3:]:
        for n in nuclei:
            basis.append(hgto.HGTO(sigma, origin=n.position))

    return basis, nuclei
```

Now, do the usual optimization. You won't get anything useful, though, for the optimization will always fail, either trying to separate all three nuclei (both X_2 and $\max(X_3, Y_3)$ growing rapidly), or try to separate one of the nuclei from two others and build up a dihydrogen cation H_2^+ on the latter (you will notice this by either $|X_2|$ or $\sqrt{X_3^2 + Y_3^2}$ equal to ≈ 1.99).

It turns out that there is no such thing as trihydrogen 2+ cation H_3^{2+}. No matter what, a single electron cannot bind three hydrogen nuclei together. The real trihydrogen cation H_3^+ has two electrons, and is beyond the scope for now.

We can, however, try to do something at least slightly useful. Basic chemistry builds up on the idea[31] that electrons independently occupy single-electron orbitals in pairs[32]. So, if we arrange the nuclei the way they exist in H_3^+ and don't let them move, we will get a one-electron approximation to what really happens in this cation.

Looking at the wikipedia article, we see that the trihydrogen 1+ cation is an equilateral triangle with side equal to 0.9Å[33], which is about 1.7 in atomic units. Let us arrange the nuclei so that the center of this triangle is at the origin (again, for purely aesthetic reasons), the first nucleus is along the X axis, and two others are obtained by rotating 120° and 240°:

```
def make_basis(params):
    A = 1.7
    R = A / numpy.sqrt(3.0)

    nuclei = [
        hgto.Nucleus(1.0, [   R,  0.0, 0.0]),
        hgto.Nucleus(1.0, [-R/2,  A/2, 0.0]),
        hgto.Nucleus(1.0, [-R/2, -A/2, 0.0]),
```

[31] I never liked it, since it completely ignores electron-electron repulsion and, probably even more importantly, ignores that a multi-electron wavefunction is **not** simply composed of independent single-electron wavefunctions, but is an element of a tensor product space of several single-electron Hilbert spaces. Nevertheless, this simplistic point of view provides a useful, if not always correct, mental model for chemists, which is a feature of extreme value.

[32] Because of Pauli principle and having two possible spins.

[33] This is the angstrom unit of distance equal to $1Å = 10^{-10}m$.

```
    ]

    basis = []

    for sigma in params:
        for n in nuclei:
            basis.append(hgto.HGTO(sigma, origin=n.position))

    return basis, nuclei
```

and call the optimization:

```
params = optimize(numpy.random.uniform(0.5, 5.0, 3))

basis, nuclei = make_basis(params)
energy, coeffs = solve(*equation(basis, nuclei))
print("Energy:", list(energy))
```

(decreasing the number of parameters to 3 so that it goes a little faster). I got this:

```
Energy: [-0.10152107345559061, 0.6528801984407289, 0.6528801984407303,
 1.3377309465627292, 1.6843031987455306, 1.6843031987455341, ...]
```

We got the ground-state energy of about -0.1 (in atomic units!). The energy of nuclear repulsion here is about 1.76 (excercise: obtain this number yourself), so the pure one-electron energy (ignoring nuclei repulsion) is about -1.86. If there were two electrons, like in a real H_3^+, together they would have the energy of $-1.86-1.86=-3.72$. Adding the nuclei repulsion back, we get the total energy of the H_3^+ cation of $-3.72+1.76=-1.96$. The actual energy will, of course, be a lot higher (about -0.34, though I might be incorrect here) due to electron-electron repulsion.

What if the H_3^+ cation gets another electron? It will become the triatomic hydrogen molecule, although a very unstable one. It quickly decomposes, leading to a reaction

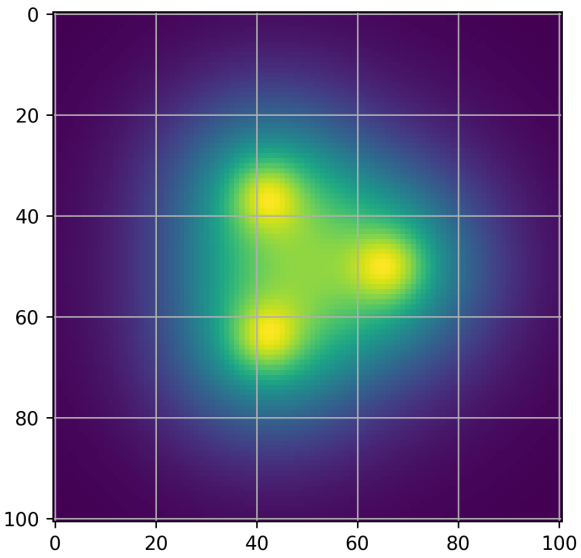
$$H_3^+ + e^- \to H_3 \to H_2 + H \tag{24}$$

which happens a lot in the interstellar medium. In fact, the H_3^+ ion is one of the most abundant ions in the universe!

We have already seen that the ground state of a chemical system tends to be highly symmetric. The most symmetric function in our case would be the one that doesn't change upon any permutation of the three nuclei. Let us plot[34] the ground state we've found to confirm this. Plotting along the X axis if pretty useless now, since the molecule is not linear[35], so we'll have to use 2D image plots and plot the XY-plane cross-section of the wavefunction:

```
L = 3
N = 101
X = numpy.linspace(-L, L, N)
Y = numpy.linspace(-L, L, N)

X, Y = numpy.meshgrid(X, Y)

state = wavefunction(basis, coeffs[:,k], X, Y, 0)
pyplot.imshow(state)

pyplot.grid()
pyplot.show()
```

[34] A simpler yet boring way to see this is to inspect the coefficients of the wavefunction.
[35] Does not lie on a straight line.

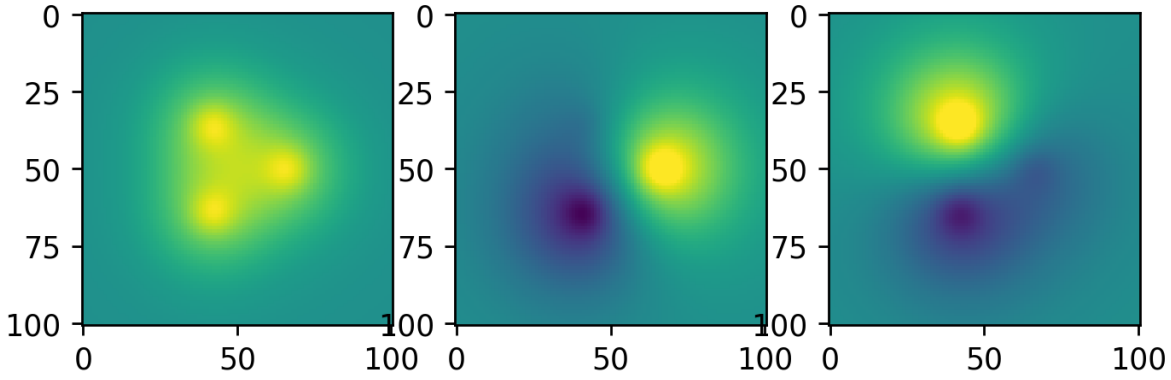


(Remember that you might get the opposite colors — a negated wavefunction represents the same state). We can clearly see that the wavefunction is concentrated the same way around each nucleus.

Before we move on, let's have a look at the next two energy levels (with energy of ≈ 0.65). We've ignored positive-energy states up to now, since having positive energy means that it would be more energy-efficient to tear all particles apart and move them away from each other, reducing energy to zero. However, these two states are interesting from a mathematical perspective. Let's plot their cross-sections alongside the ground state:

```
figure = pyplot.figure()
for k in range(3):
    state = wavefunction(basis, coeffs[:,k], X, Y, 0)
    figure.add_subplot(1, 3, k+1)
    pyplot.imshow(state, vmin=-0.4, vmax=0.4)
        pyplot.grid()

pyplot.show()
```

I am setting the possible range of values `vmin..vmax` to something fixed so that all three images have the same value to color matching.



The two wavefunctions on the right look pretty much the same, one being a rotated and/or reflected version of the other[36]. What is interesting about them is that if we take the coefficients of one of these

[36] See also my tweet with an alternative visualization.

states corresponding to three basis functions with the same `sigma` but different origin and sum these three numbers, we get a zero. Let's check this:

```
for k in [1,2]:
    c = coeffs[:,k]
    for i in range(len(params)):
        print(sum(c[3*i:3*i+3]))
```

Recall that `params` are the values of `sigma`, and each `sigma` corresponds to three consecutive basis functions, thus the summation from `3*i` to `3*i+3` (exclusive on the end). You should get something like

```
-5.828670879282072e-16
1.887379141862766e-15
8.465450562766819e-16
2.7755575615628914e-16
-5.551115123125783e-16
3.3306690738754696e-16
```

These are within a numerical error from zero[37]. Clearly something's happening here.

Welcome to the miraculous and frightening world of representation theory! This is an enormous beautiful part of mathematics that is concerned with viewing any object (e.g. elements of some group) as linear operators on some vector space, and trying to squeeze as much information as possible, with applications from coding theory[38] to particle physics[39]. In our case, the groups is S_3 — the group of permutations of three objects (the three nuclei). The system is clearly invariant under the action of this group (since there are three exactly identical nuclei sitting on the vertices of an equilateral triangle), so representation theory tells us that the eigenstates should be *irreducible representations* of this group. Now, mathematics tells us that there are two irreducible representations of this group:

- The one-dimensional *trivial* representation, which is completely invariant under group action — the ground state,
- The two-dimensional *standard* representation, with coefficients that sum to zero — the $k = 1$ and $k = 2$ states.

So, representation theory can predict certain properties of chemical species simply by looking at the symmetry groups. In fact, most chemists study representation theory as part of their curriculum. For a chemical overview on the subject, see [1]. For a deep dive in, see [6].

[37] Curiously enough, the machine epsilon for double precision floating point is about 10^{-16}.

[38] Fourier series and Fourier transform, both continuous and discrete, are instances of representation theory of groups $\mathbb{T} \cong \mathbb{R}/\mathbb{Z}$, $\mathbb{R}$, and $C_n \cong \mathbb{Z}/n\mathbb{Z}$.

[39] Physicists love the mantra that elementary particles are the irreducible representations of the Poincaré group, see here.

13 Helium hydride 2+ cation HeH^{2+}

At this point you might already suspect that the helium hydride 2+ cation HeH^{2+} is not a thing either, while the 1+ cation HeH^{+} (having two electrons) is. Let us check all that as well.

Put the helium and hydrogen atoms symmetrically on the X axis. This time there are no reasons to expect any symmetry between the two atoms, so we'll have two separate sets of `sigma`'s, one for each atom:

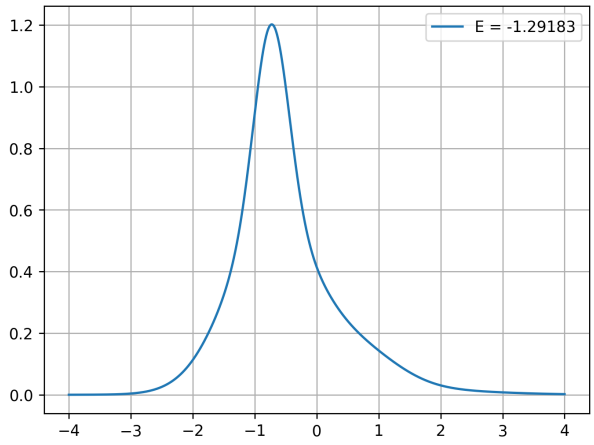
```
def make_basis(params):
    R = params[0]

    nuclei = [
        hgto.Nucleus(2.0, [-R/2, 0.0, 0.0]),
        hgto.Nucleus(1.0, [ R/2, 0.0, 0.0]),
    ]

    N = len(params) // 2

    sigma_He = params[1:N]
    sigma_H = params[N:]

    basis = []

    for sigma in sigma_He:
        basis.append(hgto.HGTO(sigma, origin=nuclei[0].position))

    for sigma in sigma_H:
        basis.append(hgto.HGTO(sigma, origin=nuclei[1].position))

    return basis, nuclei
```

This won't get us anywhere (`R` will tend to infinity), for there is indeed no such thing as HeH^{2+}. We could instead take $R = 0.772\text{Å} = 1.459$ from the HeH^{+} cation and look at the orbitals:

```
def make_basis(params):
    R = 1.459

    nuclei = [
        hgto.Nucleus(2.0, [-R/2, 0.0, 0.0]),
        hgto.Nucleus(1.0, [ R/2, 0.0, 0.0]),
    ]

    N = len(params) // 2

    sigma_He = params[0:N]
    sigma_H = params[N:]

    basis = []

    for sigma in sigma_He:
        basis.append(hgto.HGTO(sigma, origin=nuclei[0].position))

    for sigma in sigma_H:
        basis.append(hgto.HGTO(sigma, origin=nuclei[1].position))

    return basis, nuclei
```

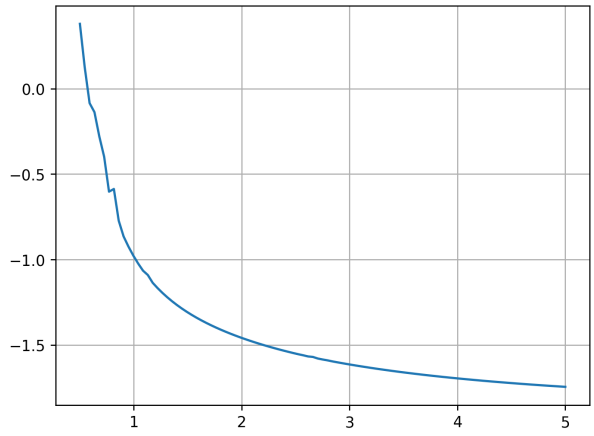
```
Energy: [-1.291834165736075, 0.08084412988221856, 1.191697875250631, ...]
```

We can plot the ground state now:



The wavefunction is mostly concentrated around the helium atom as expected (since it has twice the charge). As in the Dihydrogen cation H_2^+ section, we can plot the dependence of energy on the internuclear distance.

The final source code for the following section is available online.



There is no clear minimum (except for that tiny bump at $R \approx 0.7$ that stems from our method imprefections), which confirms that this HeH^{2+} ion is non-existing.

There isn't much else we can do at this point. Most chemical systems have a lot more than just a single electron. While one-electron orbitals are nice to look at, they are an extremely crude approximation to what is really going on in multi-electron systems. If this gets enough feedback, I will write a text on two-electron, and then multi-electron systems. To engage more into quantum chemistry, look into this amazing book[1], this video series[9], and this free high-quality python package[10]. As for now, I hope you've learnt something from this book.

References

[1] Peter Atkins, Ronald Friedman, *Molecular Quantum Mechanics*

[2] Elliot Lieb, *The Stability of Matter: From Atoms to Stars*

[3] Brian Hall, *Quantum Theory for Mathematicians*

[4] Tosio Kato, *On the eigenfunctions of many-particle systems in quantum mechanics*

[5] Tosio Kato, *Fundamental properties of Hamiltonian operators of Schrödinger type*

[6] William Fulton, Joe Harris, *Representation Theory: A First Course*

[7] A. R. Edmonds, *Angular Momentum in Quantum Mechanics*

[8] S. B. Doma, M. Abu-Shady, F. N. El-Gammal, A. A. Amer, *Ground States of the Hydrogen Molecule and Its Molecular Ion in the Presence of Magnetic Field Using the Variational Monte Carlo Method* https://arxiv.org/pdf/1509.00477.pdf

[9] TMP Chem, www.youtube.com/channel/UC3dZQdfv67X49cZkoXWYSwQ

[10] Psi4NumPy, www.github.com/psi4/psi4numpy

[11] T. Petersson, B. Hellsing, *Detailed derivation of Gaussian orbital based matrix elements in electron structure calculations*

A Hermite polynomials

Among the vast number of different polynomial families, Hermite polynomials are particularly suited to dealing with Gaussians. We define[40] them as follows:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2} \tag{25}$$

The first few of them are

$$H_0(x) = 1 \tag{26}$$
$$H_1(x) = 2x \tag{27}$$
$$H_2(x) = 4x^2 - 2 \tag{28}$$
$$H_3(x) = 8x^3 - 12x \tag{29}$$

The definition (25) can also be written in terms of repeated application of a certain operator:

$$(25) = \left[-e^{x^2} \frac{d}{dx} e^{-x^2}\right] \cdot \left[-e^{x^2} \frac{d}{dx} e^{-x^2}\right] \cdot \cdots\cdots \left[-e^{x^2} \frac{d}{dx} e^{-x^2}\right] \cdot 1 = \left[-e^{x^2} \frac{d}{dx} e^{-x^2}\right]^n 1 \tag{30}$$

Notice how the exponents in the middle cancel out. This equation gives a recursive formula for Hermite polynomials:

$$H_{n+1}(x) = -e^{x^2} \frac{d}{dx} e^{-x^2} H_n(x) \tag{31}$$

Expanding this, we get

$$H_{n+1}(x) = -e^{x^2} \left(-2xe^{-x^2} H_n(x) + e^{-x^2} H_n'(x)\right) = 2xH_n(x) - H_n'(x) = \left[2x - \frac{d}{dx}\right] H_n(x) \tag{32}$$

leading to another operator-power type equation:

$$H_n(x) = \left[2x - \frac{d}{dx}\right]^n 1 \tag{33}$$

We now use (32) and induction to confirm that $H_n'(x) = 2nH_{n-1}(x)$:

$$\begin{gathered} H_1'(x) = (2x)' = 2 = 2 \cdot 1 \cdot H_0(x) \\ H_{n+1}'(x) = (2xH_n(x) - H_n'(x))' = 2H_n(x) + 2xH_n'(x) - H''(n) = 2H_n(x) + 2x \cdot 2nH_{n-1}(x) - 2nH_{n-1}'(x) = \\ = 2H_n(x) + 2n(2xH_{n-1}(x) - H_{n-1}'(x)) \overset{\text{by (32)}}{=\!=} 2H_n(x) + 2nH_n(x) = 2(n+1)H_n(x) \end{gathered} \tag{34}$$

Substituting (34) into (32), we get

$$H_{n+1}(x) = 2xH_n(x) - H_n'(x) = 2xH_n(x) - 2nH_{n-1}(x) \tag{35}$$

which can be used to recursively evaluate Hermite polynomials of any order.

[40] There are two conventions; we use the physicist's convention here. See wikipedia.

B Derivation of HGTO integrals

This derivation is partially based on [11]. Beware that it contains loads of hard-to-detect typos, as the following text might do as well. Please, email me if you find some.

Recall that we defined Hermite-Gaussian-type orbitals (HGTO's) as

$$\phi_{R,\sigma,n}(x,y,z) = H_{n_x}\left(\frac{x-R_x}{\sigma}\right) H_{n_y}\left(\frac{y-R_y}{\sigma}\right) H_{n_z}\left(\frac{z-R_z}{\sigma}\right) e^{-\frac{(x-R_x)^2+(y-R_y)^2+(z-R_z)^2}{\sigma^2}} \tag{36}$$

where $R \in \mathbb{R}^3$ is the coordintate vector of the function's origin, $n \in \mathbb{N}^3$ is its degree, while $\sigma \in \mathbb{R}$ controls its spread, so an HGTO is defined by these 4 real numbers and 3 non-negative integers.

The key to integrating them is the decomposition into a product of 1D-functions

$$\phi_{R,\sigma,n}(x,y,z) = \phi_{R_x,\sigma,n_x}(x)\phi_{R_y,\sigma,n_y}(y)\phi_{R_z,\sigma,n_z}(z) \tag{37}$$

where we defined

$$\phi_{R,\sigma,n}(x) = H_n\left(\frac{x-R}{\sigma}\right) e^{-\frac{(x-R)^2}{\sigma^2}} \tag{38}$$

Here, $R, \sigma \in \mathbb{R}$ and $n \in \mathbb{N}$. Since the 3D and 1D functions have different number of parameters, reusing the letters should cause no confusion.

Recall (25) that

$$\frac{d^n}{dx^n} e^{-x^2} = (-1)^n H_n(x) e^{-x^2} \tag{39}$$

Thus

$$\frac{d^n}{dx^n} e^{-\left(\frac{x-R}{\sigma}\right)^2} = (-1)^n \sigma^{-n} H_n\left(\frac{x-R}{\sigma}\right) e^{-\left(\frac{x-R}{\sigma}\right)^2} \tag{40}$$

and

$$\frac{d^n}{dR^n} e^{-\left(\frac{x-R}{\sigma}\right)^2} = \sigma^{-n} H_n\left(\frac{x-R}{\sigma}\right) e^{-\left(\frac{x-R}{\sigma}\right)^2} \tag{41}$$

and also

$$\frac{d^n}{dR^n}\frac{d^m}{dx^m} e^{-\left(\frac{x-R}{\sigma}\right)^2} = \frac{d^n}{dR^n}(-1)^m \frac{d^m}{dR^m} e^{-\left(\frac{x-R}{\sigma}\right)^2} = (-1)^m \sigma^{-(m+n)} H_{m+n}\left(\frac{x-R}{\sigma}\right) e^{-\left(\frac{x-R}{\sigma}\right)^2} \tag{42}$$

(41) leads to

$$\phi_{R,\sigma,n}(x) = \sigma^n \frac{d^n}{dR^n} e^{-\left(\frac{x-R}{\sigma}\right)^2} \tag{43}$$

This observation makes it possible to only compute the integrals for the case $n = 0$ and then use differentiation on the resulting formula.

The final ingredient is the result occasionally known as the Gaussian product theorem: a product of Gaussians is again a Gaussian.

$$\left(\frac{x-R_1}{\sigma_1}\right)^2 + \left(\frac{x-R_2}{\sigma_2}\right)^2 = \left(\frac{1}{\sigma_1^2}+\frac{1}{\sigma_2^2}\right)x^2 - 2\left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)x + \left(\frac{R_1^2}{\sigma_1^2}+\frac{R_2^2}{\sigma_2^2}\right) \tag{44}$$

Denote

$$\Sigma = \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right)^{-\frac{1}{2}} = \left(\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}\right)^{\frac{1}{2}} \tag{45}$$

so that

$$\begin{aligned}
(44) &= \frac{1}{\Sigma^2}\left[x^2 - 2\left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)\Sigma^2 x\right] + \left(\frac{R_1^2}{\sigma_1^2}+\frac{R_2^2}{\sigma_2^2}\right) = \\
&= \frac{1}{\Sigma^2}\left[x^2 - 2\left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)\Sigma^2 x + \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)^2\Sigma^4 - \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)^2\Sigma^4\right] + \left(\frac{R_1^2}{\sigma_1^2}+\frac{R_2^2}{\sigma_2^2}\right) = \\
&= \frac{1}{\Sigma^2}\left[x - \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)\Sigma^2\right]^2 - \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)^2\Sigma^2 + \left(\frac{R_1^2}{\sigma_1^2}+\frac{R_2^2}{\sigma_2^2}\right) = \\
&= \frac{1}{\Sigma^2}\left[x - \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)\Sigma^2\right]^2 + \Sigma^2\left[-\left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)^2 + \left(\frac{R_1^2}{\sigma_1^2}+\frac{R_2^2}{\sigma_2^2}\right)\Sigma^{-2}\right] = \\
&= \frac{1}{\Sigma^2}\left[x - \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)\Sigma^2\right]^2 + \Sigma^2\left[-\left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)^2 + \left(\frac{R_1^2}{\sigma_1^2}+\frac{R_2^2}{\sigma_2^2}\right)\left(\frac{1}{\sigma_1^2}+\frac{1}{\sigma_2^2}\right)\right] = \\
&= \frac{1}{\Sigma^2}\left[x - \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)\Sigma^2\right]^2 + \Sigma^2\left[-\frac{R_1^2}{\sigma_1^4} - 2\frac{R_1R_2}{\sigma_1^2\sigma_2^2} - \frac{R_2^2}{\sigma_2^4} + \frac{R_1^2}{\sigma_1^4} + \frac{R_1^2}{\sigma_1^2\sigma_2^2} + \frac{R_2^2}{\sigma_1^2\sigma_2^2} + \frac{R_2^2}{\sigma_2^4}\right] = \\
&= \frac{1}{\Sigma^2}\left[x - \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)\Sigma^2\right]^2 + \Sigma^2\left[\frac{(R_1-R_2)^2}{\sigma_1^2\sigma_2^2}\right] = \frac{1}{\Sigma^2}\left[x - \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)\Sigma^2\right]^2 + \frac{(R_1-R_2)^2}{\sigma_1^2+\sigma_2^2}
\end{aligned} \tag{46}$$

Using (46), we obtain

$$\exp\left[-\left(\frac{x-R_1}{\sigma_1}\right)^2\right]\exp\left[-\left(\frac{x-R_2}{\sigma_2}\right)^2\right] = \exp\left[-\left(\frac{x-A}{\Sigma}\right)^2\right]\exp\left[-\frac{(R_1-R_2)^2}{\sigma_1^2+\sigma_2^2}\right] \tag{47}$$

where

$$A = \left(\frac{R_1}{\sigma_1^2}+\frac{R_2}{\sigma_2^2}\right)\Sigma^2 \tag{48}$$

We will also need the well-known Gaussian integral:

$$\int_{-\infty}^{\infty}\exp\left[-\left(\frac{x-a}{b}\right)^2\right]dx = b\sqrt{\pi} \tag{49}$$

and a slightly different form of it:

$$\int_{-\infty}^{\infty}\exp\left[-ax^2-bx\right]dx = \int_{-\infty}^{\infty}\exp\left[-a\left(x-\frac{b}{2a}\right)^2+\frac{b^2}{4a}\right]dx = = \sqrt{\frac{\pi}{a}}\exp\frac{b^2}{4a} \tag{50}$$

B.1 Overlap integral

We are ready to compute the overlap integral of two HGTO's:

$$\langle\phi_{R_1,\sigma_1,n_1},\phi_{R_2,\sigma_2,n_2}\rangle = \int_{\mathbb{R}^3}\phi_{R_1,\sigma_1,n_1}(x,y,z)\phi_{R_2,\sigma_2,n_2}(x,y,z)dxdydz \tag{51}$$

First, employ the decomposition into 1D functions (37) to separate the integrals:

$$(51) = \left[\int_{-\infty}^{\infty}\phi_{R_{1x},\sigma_1,n_{1x}}(x)\phi_{R_{2x},\sigma_2,n_{2x}}(x)dx\right]\cdot[\ldots dy]\cdot[\ldots dz] \tag{52}$$

We will temporarily focus on the x-integral only, and drop the x subscript.

Now, use (43) to remove the Hermite polynomials:

$$\int_{-\infty}^{\infty} \phi_{R_1,\sigma_1,n_1}(x)\phi_{R_2,\sigma_2,n_2}(x)dx = \sigma_1^{n_1}\sigma_2^{n_2}\frac{d^{n_1}}{dR_1^{n_1}}\frac{d^{n_2}}{dR_2^{n_2}}\int_{-\infty}^{\infty}\exp\left[-\left(\frac{x-R_1}{\sigma_1}\right)^2\right]\exp\left[-\left(\frac{x-R_2}{\sigma_2}\right)^2\right]dx \tag{53}$$

Next, use (47)

$$\begin{aligned}
(53) &= \sigma_1^{n_1}\sigma_2^{n_2}\frac{d^{n_1}}{dR_1^{n_1}}\frac{d^{n_2}}{dR_2^{n_2}}\int_{-\infty}^{\infty}\exp\left[-\left(\frac{x-A}{\Sigma}\right)^2\right]\exp\left[-\frac{(R_1-R_2)^2}{\sigma_1^2+\sigma_2^2}\right]dx \overset{\text{by (47)}}{=\!=} \\
&= \sigma_1^{n_1}\sigma_2^{n_2}\cdot\left[\frac{d^{n_1}}{dR_1^{n_1}}\frac{d^{n_2}}{dR_2^{n_2}}\exp\left[-\frac{(R_1-R_2)^2}{\sigma_1^2+\sigma_2^2}\right]\right]\cdot\int_{-\infty}^{\infty}\exp\left[-\left(\frac{x-A}{\Sigma}\right)^2\right]dx \overset{\text{by (49)}}{=\!=} \\
&= \sigma_1^{n_1}\sigma_2^{n_2}\Sigma\sqrt{\pi}\cdot\left[\frac{d^{n_1}}{dR_1^{n_1}}\frac{d^{n_2}}{dR_2^{n_2}}\exp\left[-\left(\frac{R_1-R_2}{\sqrt{\sigma_1^2+\sigma_2^2}}\right)^2\right]\right] \overset{\text{by (41)}}{=\!=} \\
&= \sigma_1^{n_1}\sigma_2^{n_2}\Sigma\sqrt{\pi}(-1)^{n_1}(\sigma_1^2+\sigma_2^2)^{-\frac{n_1+n_2}{2}}H_{n_1+n_2}\left(\frac{R_1-R_2}{\sqrt{\sigma_1^2+\sigma_2^2}}\right)\exp\left[-\frac{(R_1-R_2)^2}{\sigma_1^2+\sigma_2^2}\right]
\end{aligned} \tag{54}$$

This is the final formula for the overlap integral of two 1D HGTO's. The integral for 3D functions is the product of three 1D integrals (52). Everything here is just arithmetic, except square roots and exponentials, which can be computed using standard routines in most programming languages, and Hermite polynomials, which can be computed using (35).

B.2 Kinetic integral

The kinetic integral, representing the kinetic energy, is

$$\begin{aligned}
\langle\phi_{R_1,\sigma_1,n_1}, -\frac{1}{2}\Delta\phi_{R_2,\sigma_2,n_2}\rangle &= -\frac{1}{2}\int_{\mathbb{R}^3}\phi_{R_1,\sigma_1,n_1}(x,y,z)\Delta\phi_{R_2,\sigma_2,n_2}(x,y,z)dxdydz = \\
&= -\frac{1}{2}\int_{\mathbb{R}^3}\phi_{R_1,\sigma_1,n_1}(x,y,z)\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)\phi_{R_2,\sigma_2,n_2}(x,y,z)dxdydz = \\
&= -\frac{1}{2}\int_{\mathbb{R}^3}\phi_{R_1,\sigma_1,n_1}(x,y,z)\frac{\partial^2}{\partial x^2}\phi_{R_2,\sigma_2,n_2}(x,y,z)dxdydz - \frac{1}{2}\int\ldots dy - \frac{1}{2}\int\ldots dz
\end{aligned} \tag{55}$$

We now focus on the first integral (the one with $\frac{\partial^2}{\partial x^2}$), the other two can be computed in a similar way. We also drop the $-\frac{1}{2}$ factor in what follows.

$$\begin{aligned}
&\int_{\mathbb{R}^3}\phi_{R_1,\sigma_1,n_1}(x,y,z)\frac{\partial^2}{\partial x^2}\phi_{R_2,\sigma_2,n_2}(x,y,z)dxdydz \overset{\text{by (37)}}{=\!=} \\
&\left[\int_{-\infty}^{\infty}\phi_{R_{1x},\sigma_1,n_{1x}}(x)\frac{\partial^2}{\partial x^2}\phi_{R_{2x},\sigma_2,n_{2x}}(x)dx\right]\cdot\left[\int_{-\infty}^{\infty}\phi_{R_{1y},\sigma_1,n_{1y}}(y)\phi_{R_{2y},\sigma_2,n_{2y}}(y)dy\right]\cdot\left[\int_{-\infty}^{\infty}\phi_{R_{1z},\sigma_1,n_{1z}}(z)\phi_{R_{2z},\sigma_2,n_{2z}}(z)dz\right]
\end{aligned} \tag{56}$$

Note that the integrals over y and z are just 1D overlap integrals computed earlier (54), so we are left to deal with the first term.

Exploiting again the handy properties of Hermite polynomials, we get

$$\frac{d^2}{dx^2}\phi_{R,\sigma,n}(x) \overset{\text{by (43)}}{=\!=\!=} \sigma^n \frac{d^2}{dx^2}\frac{d^n}{dR^n} e^{-\left(\frac{x-R}{\sigma}\right)^2} \overset{\text{by (42)}}{=\!=\!=} = \sigma^{-2} H_{n+2}\left(\frac{x-R}{\sigma}\right) e^{-\left(\frac{x-R}{\sigma}\right)^2} \overset{\text{by (43)}}{=\!=\!=} \sigma^{-2}\phi_{R,\sigma,n+2}(x) \tag{57}$$

so that the integral in question reduces to

$$\int_{-\infty}^{\infty} \phi_{R_{1x},\sigma_1,n_{1x}}(x)\frac{\partial^2}{\partial x^2}\phi_{R_{2x},\sigma_2,n_{2x}}(x)dx = \sigma_{2x}^{-2}\int_{-\infty}^{\infty} \phi_{R_{1x},\sigma_1,n_{1x}}(x)\phi_{R_{2x},\sigma_2,n_{2x}+2}(x)dx \tag{58}$$

which is again an overlap integral, multiplied by a constant.

So, the kinetic integral is $-\frac{1}{2}$ times a sum of three terms (55), each being a product of three overlap integrals (56): two integrals being the same as in (54) and one (58) having a 2+ higher degree and a σ^{-2} factor.

B.3 Nuclear attraction integral

The nuclear attraction integral corresponding to a nucleus of charge Z at position R_0 is

$$\langle \phi_{R_1,\sigma_1,n_1}, -\frac{Z}{|r-R_0|}\phi_{R_2,\sigma_2,n_2}\rangle = -Z\int_{\mathbb{R}^3} \phi_{R_1,\sigma_1,n_1}(x,y,z)\frac{1}{|r-R_0|}\phi_{R_2,\sigma_2,n_2}(x,y,z)dxdydz \tag{59}$$

Here, r is the vector (x,y,z).

This integral doesn't split in such an easy way. In fact, we won't even get a closed formula for it, but only an algorithm receipt. We start with the same trick (43) of using differentiation to get rid of Hermite polynomials:

$$\begin{aligned}(59) = -Z\sigma_1^{n_{1x}+n_{1y}+n_{1z}}\sigma_2^{n_{2x}+n_{2y}+n_{2z}} \frac{d^{n_{1x}}}{dR_{1x}^{n_{1x}}}\frac{d^{n_{1y}}}{dR_{1y}^{n_{1y}}}\frac{d^{n_{1z}}}{dR_{1z}^{n_{1z}}}\frac{d^{n_{2x}}}{dR_{2x}^{n_{2x}}}\frac{d^{n_{2y}}}{dR_{2y}^{n_{2y}}}\frac{d^{n_{2z}}}{dR_{2z}^{n_{2z}}} \\ \int_{\mathbb{R}^3} \exp\left[-\left(\frac{r-R_1}{\sigma_1}\right)^2\right]\exp\left[-\left(\frac{r-R_2}{\sigma_2}\right)^2\right]\frac{1}{|r-R_0|}dxdydz\end{aligned} \tag{60}$$

(here, squaring a vector like $(r-R_1)^2$ should be understood in terms of dot product).

Retaining sanity at this point poses considerable difficulties. We concentrate on the integral first, analyze the resulting expression as a function of R_1, R_2, and discuss how one can compute this 6 nested higher-order derivatives.

To make the $\frac{1}{|r-R_0|}$ term more manageable, we use the Gaussian integral (49) in reverse, introducing a dummy variable u:

$$\frac{1}{|r-R_0|} = \frac{1}{\sqrt{\pi}}\int_{-\infty}^{\infty} \exp\left[-u^2\cdot(r-R_0)^2\right]du \tag{61}$$

This way we get

$$\begin{aligned}\int_{\mathbb{R}^3} \exp\left[-\left(\frac{r-R_1}{\sigma_1}\right)^2\right]\exp\left[-\left(\frac{r-R_2}{\sigma_2}\right)^2\right]\frac{1}{|r-R_0|}dxdydz = \\ \frac{1}{\sqrt{\pi}}\int_{\mathbb{R}^4} \exp\left[-\left(\frac{r-R_1}{\sigma_1}\right)^2 - \left(\frac{r-R_2}{\sigma_2}\right)^2 - u^2\cdot(r-R_0)^2\right]dxdydzdu\end{aligned} \tag{62}$$

Note that each term inside the exponential is actually a dot product, thus consists of three terms, e.g. $(r-R_1)^2$ really is $(x-R_{1x})^2+(y-R_{1y})^2+(z-R_{1z})^2$, etc.

Use the Gaussian product theorem (47) to reduce the number of terms:

$$(62) = \frac{1}{\sqrt{\pi}} \exp\left[-\frac{(R_1-R_2)^2}{\sigma_1^2+\sigma_2^2}\right] \int\limits_{\mathbb{R}^4} \exp\left[-\left(\frac{r-A}{\Sigma}\right)^2 - u^2\cdot(r-R_0)^2\right] dxdydzdu \tag{63}$$

Here, $\Sigma = \left(\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}\right)^{\frac{1}{2}}$ as before, and A is a vector with coordinates

$$A_i = \left(\frac{R_{1i}}{\sigma_1^2} + \frac{R_{2i}}{\sigma_2^2}\right)\Sigma^2 \tag{64}$$

with $i \in \{x,y,z\}$. Note that A depends linearly on R_1 and R_2, which will help us with differentiation later.

Denote

$$N = \frac{1}{\sqrt{\pi}} \exp\left[-\frac{(R_1-R_2)^2}{\sigma_1^2+\sigma_2^2}\right] \tag{65}$$

and gather everything that depends on r together:

$$\begin{aligned}
(63) &= N \int\limits_{\mathbb{R}^4} \exp\left[-\left(\frac{r-A}{\Sigma}\right)^2 - u^2\cdot(r-R_0)^2\right] dxdydzdu = \\
&= N \int\limits_{\mathbb{R}^4} \exp\left[-\frac{r^2}{\Sigma^2} + 2\frac{r\cdot A}{\Sigma^2} - \frac{A^2}{\Sigma^2} - u^2r^2 + 2u^2(r\cdot R_0) - u^2R_0^2\right] dxdydzdu = \\
&= N \int\limits_{\mathbb{R}} \exp\left[-\frac{A^2}{\Sigma^2} - u^2R_0^2\right] \left[\int\limits_{\mathbb{R}^3} \exp\left[-\left(\Sigma^{-2}+u^2\right)r^2 - 2\left(A\Sigma^{-2}+R_0u^2\right)\cdot r\right] dxdydz\right] du \overset{\text{by (50)}}{=\!=} \\
&= N \int\limits_{\mathbb{R}} \exp\left[-\frac{A^2}{\Sigma^2} - u^2R_0^2\right] \left[\sqrt{\frac{\pi}{\Sigma^{-2}+u^2}}^3 \exp\left[\frac{(A\Sigma^{-2}+R_0u^2)^2}{\Sigma^{-2}+u^2}\right]\right] du = \\
&= N\sqrt{\pi}^3 \int\limits_{\mathbb{R}} (\Sigma^{-2}+u^2)^{-\frac{3}{2}} \exp\left[-\frac{A^2}{\Sigma^2} - u^2R_0^2 + \frac{(A\Sigma^{-2}+R_0u^2)^2}{\Sigma^{-2}+u^2}\right] du
\end{aligned} \tag{66}$$

Lets rearrange the expression inside the exponential:

$$\begin{aligned}
-\frac{A^2}{\Sigma^2} - u^2R_0^2 + \frac{(A\Sigma^{-2}+R_0u^2)^2}{\Sigma^{-2}+u^2} &= \frac{(-A^2\Sigma^{-2} - R_0^2u^2)(\Sigma^{-2}+u^2) + (A\Sigma^{-2}+R_0u^2)^2}{\Sigma^{-2}+u^2} = \\
&= \frac{-A^2\Sigma^{-4} - A^2\Sigma^{-2}u^2 - R_0^2\Sigma^{-2}u^2 - R_0^2u^4 + A^2\Sigma^{-4} + 2(A\cdot R_0)\Sigma^{-2}u^2 + R_0^2u^4}{\Sigma^{-2}+u^2} = \\
&= \frac{-A^2\Sigma^{-2}u^2 - R_0^2\Sigma^{-2}u^2 + 2(A\cdot R_0)\Sigma^{-2}u^2}{\Sigma^{-2}+u^2} = \frac{\Sigma^{-2}u^2}{\Sigma^{-2}+u^2}(-A^2 - R_0^2 + 2(A\cdot R_0)) = \\
&= -(\Sigma^2+u^{-2})^{-1}(A-R_0)^2
\end{aligned} \tag{67}$$

Take a break here and admire how nicely did it simplify.

We now have

$$(66) = N\sqrt{\pi}^3 \int\limits_{\mathbb{R}} (\Sigma^{-2}+u^2)^{-\frac{3}{2}} \exp\left[-(\Sigma^2+u^{-2})^{-1}(A-R_0)^2\right] du \tag{68}$$

Perform a change of variables

$$
\begin{gathered}
t = u(\Sigma^{-2} + u^2)^{-\frac{1}{2}} \\
dt = (\Sigma^{-2} + u^2)^{-\frac{1}{2}} du - u\frac{1}{2}(\Sigma^{-2} + u^2)^{-\frac{3}{2}} 2udu = (\Sigma^{-2} + u^2)(\Sigma^{-2} + u^2)^{-\frac{3}{2}} du - u^2(\Sigma^{-2} + u^2)^{-\frac{3}{2}} du = \\
= \Sigma^{-2}(\Sigma^{-2} + u^2)^{-\frac{3}{2}} du
\end{gathered}
\tag{69}
$$

so that

$$
\begin{gathered}
\Sigma^{-2}t^2 = \Sigma^{-2}u^2(\Sigma^{-2} + u^2)^{-1} = \left(\frac{\Sigma^{-2} + u^2}{\Sigma^{-2}u^2}\right)^{-1} = (\Sigma^2 + u^{-2})^{-1} \\
\Sigma^2 dt = (\Sigma^{-2} + u^2)^{-\frac{3}{2}} du
\end{gathered}
\tag{70}
$$

Substituting, we obtain

$$
(68) = N\sqrt{\pi}^3\Sigma^2 \int_{-1}^{1} \exp\left[-t^2\left(\frac{A - R_0}{\Sigma}\right)^2\right] dt \tag{71}
$$

Recall that both N and A depend on R_1 and R_2. Moreover, expanding in terms of separate coordinates, we get

$$
\begin{gathered}
(71) = \sqrt{\pi}^3\Sigma^2 \frac{1}{\sqrt{\pi}} \exp\left[-\frac{(R_1 - R_2)^2}{\sigma_1^2 + \sigma_2^2}\right] \int_{-1}^{1} \exp\left[-t^2\left(\frac{A - R_0}{\Sigma}\right)^2\right] dt = \\
= \pi\Sigma^2 \exp\left[-\frac{(R_{1x} - R_{2x})^2}{\sigma_1^2 + \sigma_2^2} - \frac{(R_{1y} - R_{2y})^2}{\sigma_1^2 + \sigma_2^2} - \frac{(R_{1z} - R_{2z})^2}{\sigma_1^2 + \sigma_2^2}\right] \cdot \\
\cdot \int_{-1}^{1} \exp\left[-t^2\left(\frac{A_x - R_{0x}}{\Sigma}\right)^2 - t^2\left(\frac{A_y - R_{0y}}{\Sigma}\right)^2 - t^2\left(\frac{A_z - R_{0z}}{\Sigma}\right)^2\right] dt = \\
= \pi\Sigma^2 \int_{-1}^{1} \left(\exp\left[-\frac{(R_{1x} - R_{2x})^2}{\sigma_1^2 + \sigma_2^2}\right] \exp\left[-t^2\left(\frac{A_x - R_{0x}}{\Sigma}\right)^2\right]\right) \cdot \\
\cdot \left(\exp\left[-\frac{(R_{1y} - R_{2y})^2}{\sigma_1^2 + \sigma_2^2}\right] \exp\left[-t^2\left(\frac{A_y - R_{0y}}{\Sigma}\right)^2\right]\right) \cdot \\
\cdot \left(\exp\left[-\frac{(R_{1z} - R_{2z})^2}{\sigma_1^2 + \sigma_2^2}\right] \exp\left[-t^2\left(\frac{A_z - R_{0z}}{\Sigma}\right)^2\right]\right) dt
\end{gathered}
\tag{72}
$$

Here, among the six exponentials, only two depend on R_{1x} and R_{2x}, two on R_{1y} and R_{2y}, and two on R_{1z} and R_{2z}, which simplifies differentiating the whole expression. In light of (60) and (72), we need to compute

$$
\begin{gathered}
(60) = -Z\sigma_1^{n_{1x}+n_{1y}+n_{1z}}\sigma_2^{n_{2x}+n_{2y}+n_{2z}}\pi\Sigma^2 \cdot \\
\cdot \int_{-1}^{1} \left(\frac{d^{n_{1x}}}{dR_{1x}^{n_{1x}}}\frac{d^{n_{2x}}}{dR_{2x}^{n_{2x}}} \exp\left[-\frac{(R_{1x} - R_{2x})^2}{\sigma_1^2 + \sigma_2^2}\right] \exp\left[-t^2\left(\frac{A_x - R_{0x}}{\Sigma}\right)^2\right]\right) \cdot \\
\cdot \left(\frac{d^{n_{1y}}}{dR_{1y}^{n_{1y}}}\frac{d^{n_{2y}}}{dR_{2y}^{n_{2y}}} \exp\left[-\frac{(R_{1y} - R_{2y})^2}{\sigma_1^2 + \sigma_2^2}\right] \exp\left[-t^2\left(\frac{A_y - R_{0y}}{\Sigma}\right)^2\right]\right) \cdot \\
\cdot \left(\frac{d^{n_{1z}}}{dR_{1z}^{n_{1z}}}\frac{d^{n_{2z}}}{dR_{2z}^{n_{2z}}} \exp\left[-\frac{(R_{1z} - R_{2z})^2}{\sigma_1^2 + \sigma_2^2}\right] \exp\left[-t^2\left(\frac{A_z - R_{0z}}{\Sigma}\right)^2\right]\right) dt
\end{gathered}
\tag{73}
$$

Lets focus on the x coordinate (as before, the others are completely analogous). We'll have to employ the generalized Leibniz rule for two variables and two functions:

$$\frac{\partial^n}{\partial a^n}\frac{\partial^m}{\partial b^m}(f\cdot g)=\sum_{\substack{i+i'=n\\ j+j'=m}}\binom{n}{i}\cdot\binom{m}{j}\cdot\left[\frac{\partial^i}{\partial a^i}\frac{\partial^j}{\partial b^j}f\right]\cdot\left[\frac{\partial^{i'}}{\partial a^{i'}}\frac{\partial^{j'}}{\partial b^{j'}}g\right] \tag{74}$$

Using this, we can treat the derivatives of $\exp\left[-\frac{(R_{1x}-R_{2x})^2}{\sigma_1^2+\sigma_2^2}\right]$ and $\exp\left[-t^2\left(\frac{A_x-R_{0x}}{\Sigma}\right)^2\right]$ separately.

$$\begin{gathered}\frac{d^{n_{1x}}}{dR_{1x}^{n_{1x}}}\frac{d^{n_{2x}}}{dR_{2x}^{n_{2x}}}\exp\left[-\frac{(R_{1x}-R_{2x})^2}{\sigma_1^2+\sigma_2^2}\right]\overset{\text{by (42)}}{=\!=\!=}\\ (-1)^{n_{1x}}(\sigma_1^2+\sigma_2^2)^{-\frac{n_{1x}+n_{2x}}{2}}H_{n_{1x}+n_{2x}}\left(\frac{R_{1x}-R_{2x}}{\sqrt{\sigma_1^2+\sigma_2^2}}\right)\exp\left[-\frac{(R_{1x}-R_{2x})^2}{\sigma_1^2+\sigma_2^2}\right]\end{gathered} \tag{75}$$

Next, since A_x depends on R_{1x} and R_{2x} linearly, we have

$$\begin{gathered}\frac{d^{n_{1x}}}{dR_{1x}^{n_{1x}}}\frac{d^{n_{2x}}}{dR_{2x}^{n_{2x}}}\exp\left[-t^2\left(\frac{A_x-R_{0x}}{\Sigma}\right)^2\right]=\\ =\left(\frac{\partial A_x}{\partial R_{1x}}\right)^{n_{1x}}\left(\frac{\partial A_x}{\partial R_{2x}}\right)^{n_{2x}}\frac{d^{n_{1x}+n_{2x}}}{dA^{n_{1x}+n_{2x}}}\exp\left[-t^2\left(\frac{A_x-R_{0x}}{\Sigma}\right)^2\right]\overset{\text{by (64)}}{=\!=\!=}\\ =\left(\frac{\Sigma^{n_{1x}+n_{2x}}}{\sigma_1^{n_{1x}}\sigma_2^{n_{2x}}}\right)^2\frac{d^{n_{1x}+n_{2x}}}{dA_x^{n_{1x}+n_{2x}}}\exp\left[-t^2\left(\frac{A_x-R_{0x}}{\Sigma}\right)^2\right]=\\ =\left(\frac{\Sigma^{n_{1x}+n_{2x}}}{\sigma_1^{n_{1x}}\sigma_2^{n_{2x}}}\right)^2\frac{d^{n_{1x}+n_{2x}}}{dA_x^{n_{1x}+n_{2x}}}\exp\left[-\left(\frac{A_x-R_{0x}}{\Sigma/t}\right)^2\right]\overset{\text{by (40)}}{=\!=\!=}\\ =\left(\frac{\Sigma^{n_{1x}+n_{2x}}}{\sigma_1^{n_{1x}}\sigma_2^{n_{2x}}}\right)^2\left(-\frac{t}{\Sigma}\right)^{n_{1x}+n_{2x}}H_{n_{1x}+n_{2x}}\left(t\frac{A_x-R_{0x}}{\Sigma}\right)\exp\left[-t^2\left(\frac{A_x-R_{0x}}{\Sigma}\right)^2\right]\end{gathered} \tag{76}$$

Finally, we need to deal with integration with respect to t. Close inspection of (73), (75), and (76) reveals that only the derivatives of $\exp\left[-t^2\left(\frac{A_x-R_{0x}}{\Sigma}\right)^2\right]$ depend on t, which are of the form $p(t)e^{-at^2}$, where p is some polynomial and a is some constant (in our case, $a=\left[-t^2\left(\frac{A-R_0}{\Sigma}\right)^2\right]$). Since integration is linear, we can expand $p(t)=\sum p_k t^k$ and concentrate on integrating $t^k e^{-at^2}$.

This can be deal with through induction. Call $I_n(a)=\int_{-1}^{1}t^n e^{-at^2}dt$. Note that $I_n(a)=0$ when n is odd, since we are integrating an odd function on a symmetric interval. For even n we get

$$\begin{gathered}I_0(a)=\int_{-1}^{1}e^{-at^2}dt=\sqrt{\frac{\pi}{a}}\operatorname{erf}\sqrt{a}\\ I_{n+2}(a)=\int_{-1}^{1}t^{n+2}e^{-at^2}dt=\int_{-1}^{1}-\frac{t^{n+1}}{2a}(-2ate^{-at^2})dt=\\ =\int_{-1}^{1}-\frac{t^{n+1}}{2a}de^{-at^2}=-\frac{t^{n+1}}{2a}e^{-at^2}\Big|_{-1}^{1}-\int_{-1}^{1}-e^{-at^2}d\frac{t^{n+1}}{2a}=\\ =\frac{e^{-a}}{a}+\frac{n+1}{2a}I_n(a)\end{gathered} \tag{77}$$

Thus, we obtain a recursive formula for $I_n(a)$. Here, erf is the error function. Routines that compute it are usually present in the standard mathematical library of any programming language.

Note that this formula is highly unstable when $a \approx 0$, and produces infinities and/or NaN's when $a = 0$ (which happens e.g. when using HGTO's centered on the nucleus, i.e. happens a lot). In this situation, one can use Taylor expansion to compute the integral (assuming even n):

$$\begin{aligned} I_n(a) &= \int_{-1}^{1} t^n e^{-at^2} dt = \int_{-1}^{1} t^n \left(\sum_{k=0}^{\infty} (-a)^k \frac{t^{2k}}{k!} \right) dt = \\ &= \sum_{k=0}^{\infty} \frac{(-a)^k}{k!} \int_{-1}^{1} t^{n+2k} dt = \sum_{k=0}^{\infty} \frac{(-a)^k}{k!} \frac{2}{n+2k} \end{aligned} \tag{78}$$

A reasonable solution is to use recursion for $|a| \geq \varepsilon$ and use series when $|a| < \varepsilon$, for some threshold ε. I used an arbitrarily-chosen value of $\varepsilon = \frac{1}{16}$.

Lets summarize the receipt for computing the nuclear attraction integral:

- A constant factor of $-Z\sigma_1^{n_{1x}+n_{1y}+n_{1z}}\sigma_2^{n_{2x}+n_{2y}+n_{2z}}\pi\Sigma^2$
- An integral (73) of a product of three terms, corresponding to x, y, z
- Each term is a higher-order derivative of a product two functions
- After expanding all the derivatives, we are left with a polynomial in t times e^{-at^2} for some constant a
- After expanding the polynomial into coefficients, the integral can be computed using either a recursive formula (77) or a series (78).