Han-sur-Lesse 2017

How to write your own *ab initio* electronic structure program.

Ivo Filot - i.a.w.filot@tue.nl

Inorganic Materials Chemistry

TU/e Technische Universiteit Eindhoven University of Technology

Prerequisites

- Linear algebra
  - matrix diagonalization, eigenvalues and eigenvectors
  - matrix multiplication arithmetic
- Basic quantum chemistry
  - Basis sets, linear combination of molecular orbitals
- Basic programming skills
  - *For* and *while* loops, vectors (or other type of container), functions, variables

Learning goals

- **Understand** the SCF algorithm in Hartree-Fock theory
- **Identify** the building blocks underlying the HF-SCF code
- **Build** critical pieces of the routine from scratch
- Effectively **use** pre-built libraries to avoid re-inventing the wheel

What is not included…

- A detailed derivation of the Roothaan equations
- Complex mathematics (I try to make things as simple as possible, though I will probably abysmally fail to do that…)
- How to write a proper, user-friendly and efficient electronic structure program (that in itself is like 3 PhD degrees)

Ingredients

What makes the self-consistent field algorithm?

- The Schrödinger equation
- Born-Oppenheimer approximation
- Slater determinant
- Minimization principle
- (Gaussian) Basis set

Han-sur-Lesse 2017

I. Hartree-Fock Equation

Ivo Filot - i.a.w.filot@tue.nl

Inorganic Materials Chemistry

TU/e Technische Universiteit Eindhoven University of Technology

The Schrödinger equation

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

OK, this one we know, we just inventorize all relevant interactions for the Hamiltonian and plug them in.

$$\hat{H} = -\sum_{i=1}^{N}\frac{1}{2}\nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>1}^{N}\frac{1}{r_{ij}}$$

kinetic energy

nuclear attraction

electron-electron repulsion

Here, I applied the Born-Oppenheimer approximation. I don't consider the motion of the nuclei, they are much heavier than the electrons...

Wave function

We have a molecule with multiple electrons, so we need a wave function that can handle that...

Let's use this:

$$\psi = \chi_i(x_1)\chi_j(x_2)\cdots\chi_N(x_N)$$

This thing is termed the Hartree Product

Cool! If we plug this in the Schrödinger equation

$$\hat{H} = -\sum_{i=1}^{N}\frac{1}{2}\nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>1}^{N}\frac{1}{r_{ij}}$$

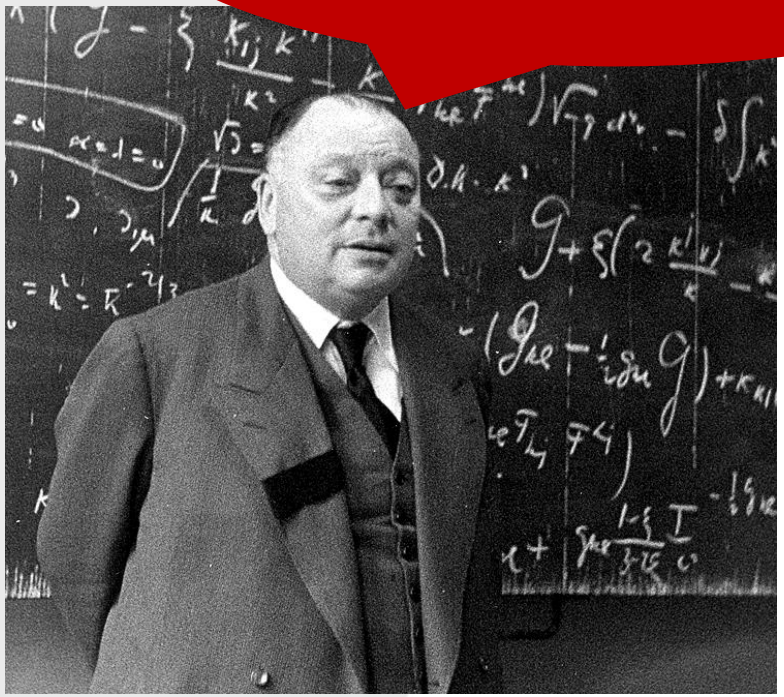
we get: $E = \varepsilon_i + \varepsilon_j + \ldots + \varepsilon_N$

I have assumed that the χ's form an orthonormal set, because I am lazy

And we're done, right?

Wave function

WRONG!!!



$$|\psi|^2 = |\chi_i(x_1)|^2 |\chi_j(x_2)|^2 \cdots |\chi_N(x_N)|^2$$

The above is simply the chance to find a particular electron. That chance does not depend on the position of the other electrons. That cannot be right!

The Slater determinant

$$\hat{H}\psi = E\psi \qquad \hat{H} = -\sum_{i=1}^{N}\frac{1}{2}\nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M}\frac{Z_A}{r_{iA}} + \sum_{i=1}^{N}\sum_{j>1}^{N}\frac{1}{r_{ij}}$$

Remember: the x's are electrons and the χ's are the spin-orbitals.

$$\psi = (N!)^{-1/2}\begin{vmatrix} \chi_i(x_1) & \chi_j(x_1) & \cdots & \chi_N(x_1) \\ \chi_i(x_2) & \chi_j(x_2) & \cdots & \chi_N(x_2) \\ \vdots & \vdots & & \vdots \\ \chi_i(x_N) & \chi_j(x_N) & \cdots & \chi_N(x_N) \end{vmatrix}$$

The Slater determinant

$$\psi = (N!)^{-1/2} \begin{vmatrix} \chi_i(x_1) & \chi_j(x_1) & \cdots & \chi_N(x_1) \\ \chi_i(x_2) & \chi_j(x_2) & \cdots & \chi_N(x_2) \\ \vdots & \vdots & & \vdots \\ \chi_i(x_N) & \chi_j(x_N) & \cdots & \chi_N(x_N) \end{vmatrix}$$

- Satisfies anti-symmetry principle (try swapping two x's, the thing gets a minus sign!)
- Electrons become (exchange)-correlated!
- Nevertheless, we call this an uncorrelated wave function because electrons with opposite spin remains uncorrelated…

The Hartree-Fock approximation

$$E = \langle \psi | \hat{H} | \psi \rangle = \overbrace{\sum_i^N \langle \chi_i | -\frac{1}{2}\nabla^2 | \chi_i \rangle}^{\text{kinetic energy}} + \overbrace{\sum_i^N \langle \chi_i | \sum_{A=1}^M \frac{Z_A}{r_{iA}} | \chi_i \rangle}^{\text{nuclear attraction}} \ldots$$

$$\ldots + \underbrace{\sum_i^N \langle \chi_i | \frac{1}{r_{ij}} \sum_{j \neq i}^N \langle \chi_j | \chi_j \rangle | \chi_i \rangle}_{\text{coulomb repulsion}} - \underbrace{\sum_i^N \langle \chi_i | \frac{1}{r_{ij}} \sum_{j \neq i}^N \langle \chi_j | \chi_i \rangle | \chi_j \rangle}_{\text{exchange}}$$

The Hartree-Fock approximation

$$E = \langle \psi | \hat{H} | \psi \rangle = \sum_i^N \langle \chi_i | -\frac{1}{2}\nabla^2 | \chi_i \rangle + \sum_i^N \langle \chi_i | \sum_{A=1}^M \frac{Z_A}{r_{iA}} | \chi_i \rangle \ldots$$

$$\ldots + \sum_i^N \sum_{j>i}^N \langle \chi_i \chi_j | \frac{1}{r_{ij}} | \chi_i \chi_j \rangle - \sum_i^N \sum_{j>i}^N \langle \chi_i \chi_j | \frac{1}{r_{ij}} | \chi_j \chi_i \rangle$$

We can transform the above to the eigenvalue form:

$$\left[-\frac{1}{2}\nabla_1^2 - \sum_{A=1}^M \frac{Z_A}{r_{1A}} + \frac{1}{r_{ij}} \sum_{j \neq i}^N \langle \chi_j | \chi_j \rangle - \frac{1}{r_{ij}} \sum_{j \neq i}^N \langle \chi_j | \chi_i \rangle \right] | \chi_i(1) \rangle = \varepsilon_i | \chi_i(1) \rangle$$

And by introducing a couple additional operators:

$$\left[h(1) + \sum_{j \neq i}^N J_j(1) - \sum_{j \neq i}^N K_j(1) \right] | \chi_i(1) \rangle = \varepsilon_i | \chi_i(1) \rangle$$

Interim summary

Hamiltonian + Slater Determinant =
Hartree-Fock equation

$$\left[h(1) + \sum_{j \neq i}^{N} J_j(1) - \sum_{j \neq i}^{N} K_j(1) \right] \left| \chi_i(1) \right\rangle = \varepsilon_i \left| \chi_i(1) \right\rangle$$

This is an integro-differential equation, as J and K depend on all the other spin-orbitals. Thus, the equation cannot be solved exactly, and we resort to numerical methods.

Han-sur-Lesse 2017

II. Energy minimization

Ivo Filot - i.a.w.filot@tue.nl

Inorganic Materials Chemistry

TU/e Technische Universiteit Eindhoven University of Technology

Basis sets & energy minimization

Each spin-orbital is given by a linear expansion of a set of known basis functions

$$\left|\chi_\alpha(1)\right\rangle = \sum_i^N c_i \left|\varphi_i\right\rangle$$

We want to find the best set of coefficients that minimizes the energy

"WE have THE BEST coefficients that minimize the energy!"

The linear variational principle

Let's start simple...

$$|\psi\rangle = \sum_{i}^{N} c_i |\varphi_i\rangle$$

Minimize a single-electron wave function which is a linear expansion of a set of known basis functions.

$E = \langle\psi|H|\psi\rangle$ subject to the constraint that $\langle\psi|\psi\rangle - 1 = 0$

i.e., that the wave function remains normalized

Notice: the $\{\varphi_i\}$ does not need to be an orthonormalized set

The linear variational principle

Great, this we recognize!

$$\sum_i H_{ij} c_j = E \sum_i S_{ij} c_j$$

$$\mathbf{Hc} = E\mathbf{Sc}$$

Coefficient vector

Overlap matrix

Hamiltonian matrix

But instead of a single-electron wave function, we have a single determinant. How do we minimize this?

The linear variational principle

$$f\,|\chi_i\rangle = \sum_{b=1}^{N} \varepsilon_{ji}\,|\chi_j\rangle$$

But in Quantum Chemistry 101 I learned this:

$$f\,|\chi_i\rangle = \varepsilon_i\,|\chi_i\rangle$$

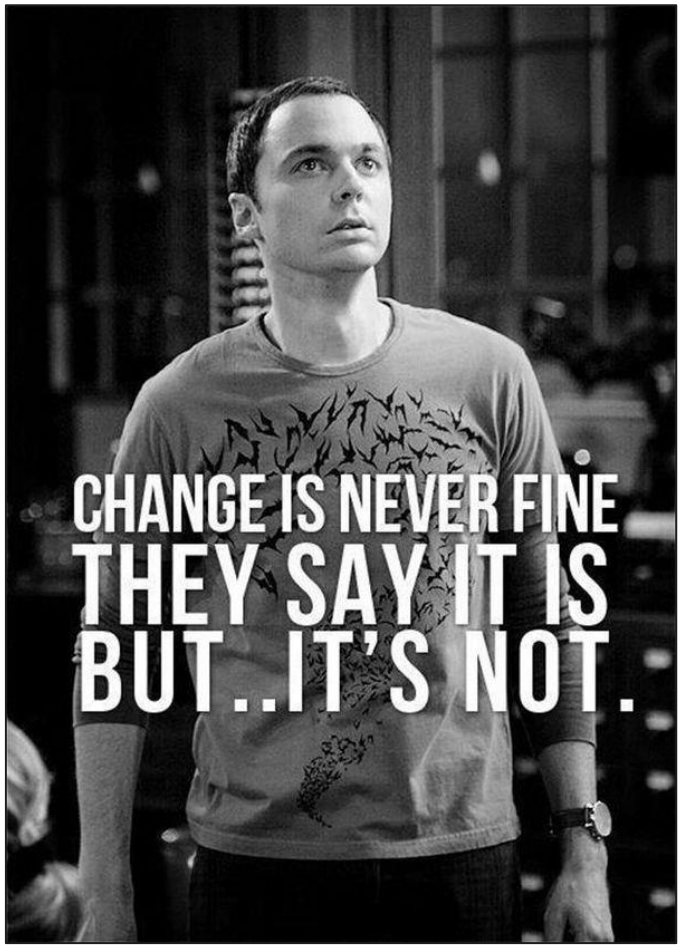
Unitary transformations

This solution says that there are an infinite number of solutions. This is not OK, because we do not like change and would like to have a unique solution.

$$f\,|\chi_i\rangle = \sum_{b=1}^{N} \varepsilon_{ji}\,|\chi_j\rangle$$

Solution: Introduce a unitary matrix

$$f\,|\chi_i\rangle = \varepsilon_i\,|\chi_i\rangle$$



Unitary transformations

$$\chi_i' = \sum_j \chi_j U_{ji}$$

Introduce a unitary matrix...

... to obtain a new set of spin-orbitals from a given set.

$$U^\dagger = U^{-1}$$

$$UU^{-1} = UU^\dagger = \mathbf{1}$$

Such a unitary transformation does <u>not</u> affect J or K, as they depend purely on the sums of the spin-orbitals. Hence, the Fock operator *f* is invariant to an arbitrary unitary transformation.

$$f|\chi_i\rangle = \sum_{b=1}^{N} \varepsilon_{ji} |\chi_j\rangle$$

Note that ε_{ji} forms a Hermitian matrix.

Unitary transformations

$$f\,|\chi_i\rangle = \sum_{b=1}^{N} \varepsilon_{ji}\,|\chi_j\rangle$$

$$\langle\chi_a|\,f\,|\chi_b\rangle = \sum_{i=1}^{N} \varepsilon_{ib}\,\langle\chi_a|\chi_i\rangle = \varepsilon_{ab}$$

The energies are matrix elements of the Fock operator

$$\langle\chi_c'|\,f\,|\chi_d'\rangle = \varepsilon'_{cd}$$

$$\langle\chi_c'|\,f\,|\chi_d'\rangle = \sum_{cd} U^*_{ca} U_{db}\,\langle\chi_a|\,f\,|\chi_b\rangle = \varepsilon'_{cd}$$

$$\varepsilon'_{cd} = \sum_{cd} U^*_{ca}\,\varepsilon_{ab}\,U_{db}$$

$$\boldsymbol{\varepsilon}' = \mathbf{U}^\dagger \boldsymbol{\varepsilon}\mathbf{U}$$

In words, I can introduce a unitary transformation U such that ε becomes a diagonal matrix.

Interim summary

$$f\,|\chi_i\rangle = \sum_{b=1}^{N} \varepsilon_{ji}\,|\chi_j\rangle$$

I choose a unitary transformation such that ε diagonalizes

$$f\,|\chi_i'\rangle = \varepsilon_i'\,|\chi_i'\rangle$$

And I drop the primes

$$f\,|\chi_i\rangle = \varepsilon_i\,|\chi_i\rangle$$

Roothaan equations

Last step: how to minimize the energy of a single determinant that is constructed from a known basis set?

$$f\left|\chi_i\right\rangle = \varepsilon_i\left|\chi_i\right\rangle \text{ and } \left|\chi_i\right\rangle = \sum_{j=1}^{N} C_{ji}\varphi_j$$

Plug linear expansion into eigenvalue equation

$$f(1)\sum_{j=1}^{N} C_{ji}\varphi_j(1) = \varepsilon_i \sum_{j=1}^{N} C_{ji}\varphi_j(1)$$

Multiply by φ_j on the left and change into matrix equation

$$\mathbf{FC} = \mathbf{SC\varepsilon}$$

These are the Roothaan equations

Interim summary

By applying the linear variational principle to a single-determinant wave function, we obtain the Roothaan equations:

Fock-matrix → $\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon}$ ← Energy matrix (diagonal)

Coefficient matrix

Overlap matrix

Han-sur-Lesse 2017

III. The SCF procedure

Ivo Filot - i.a.w.filot@tue.nl

Inorganic Materials Chemistry

TU/e Technische Universiteit Eindhoven University of Technology

Matrices

$$\mathbf{FC} = \mathbf{SC\varepsilon}$$

$$S_{ij} = \langle \varphi_i | \varphi_j \rangle$$

$$F_{ij} = \langle \varphi_i | f | \varphi_j \rangle = \langle \varphi_i | h + \sum_{b=1}^{N} J_j - \sum_{b=1}^{N} K_j | \varphi_j \rangle$$

$$\chi_i = \sum_{j=1}^{K} C_j \varphi_j$$

Auxiliary matrices

$$P_{ij} = 2\sum_{a}^{N/2} C_{ia} C_{ja}^{*}$$

$$T_{ij} = \langle \varphi_i | -\frac{1}{2}\nabla^2 | \varphi_j \rangle$$

$$V_{ij} = \langle \varphi_i | -\sum_{A} \frac{Z_A}{|r - R_A|} | \varphi_j \rangle$$

$$H_{ij}^{\text{core}} = T_{ij} + V_{ij}$$

$$F_{ij} = H_{ij}^{\text{core}} + \sum_{kl} P_{kl} \left[\langle ik | lj \rangle - \frac{1}{2} \langle il | kj \rangle \right]$$

$$F_{ij} = H_{ij}^{\text{core}} + G_{ij}$$

$$S_{ij} = \langle \varphi_i | \varphi_j \rangle$$

$$F_{ij} = \langle \varphi_i | f | \varphi_j \rangle = \langle \varphi_i | h + \sum_{b=1}^{N} J_j - \sum_{b=1}^{N} K_j | \varphi_j \rangle$$

$$\chi_i = \sum_{j=1}^{K} C_j \varphi_j$$

We introduce a density matrix **P** and separate **F** into an one-electron $\mathbf{H}^{\text{core}}$ and two-electron **G** part.

The SCF procedure: Step 1

Step 1: The system

- M nuclei at positions $\{R_M\}$
- N electrons
- K basis function $\{\varphi_i\}$

Step 2: Calculate matrices which do not change over the course of the SCF algorithm

$$S_{ij} = \langle \varphi_i | \varphi_j \rangle$$

$$T_{ij} = \langle \varphi_i | -\frac{1}{2}\nabla^2 | \varphi_j \rangle$$

$$V_{ij} = \langle \varphi_i | -\sum_A \frac{Z_A}{|r - R_A|} | \varphi_j \rangle$$

$$H_{ij}^{\text{core}} = T_{ij} + V_{ij}$$

$$\langle \varphi_i \varphi_j | \varphi_k \varphi_l \rangle = \langle \varphi_i \varphi_j | r_{kl}^{-1} | \varphi_k \varphi_l \rangle$$

- Overlap
- Kinetic energy
- Nuclear attraction
- Core hamiltonian
- Two-electron integrals

Step 3: Orthonormalize the basis set

Construct a transformation X that the basis functions form an orthonormal set

$$\mathbf{C} = \mathbf{X}\mathbf{C}' \qquad \langle \varphi_i' | \varphi_j' \rangle = \delta_{ij}$$

We can apply this transformation to F to get F', which we can solve and always back-transform to our original basis set.

$$\mathbf{F}'\mathbf{C}' = \mathbf{C}'\boldsymbol{\varepsilon} \qquad \mathbf{F}' = \mathbf{X}^{\dagger}\mathbf{F}\mathbf{X}$$

Step 3: Orthonormalize the basis set

Typically, a canonical transformation is used for this purpose, which is defined as

$$\mathbf{S} = \mathbf{U}\mathbf{s}\mathbf{U}^{\dagger}$$

1. We diagonalize the overlap matrix **S**, which gives us the diagonal matrix **s** (eigenvalues) and a unitary matrix **U** (eigenvectors).

$$\mathbf{X} = \mathbf{U}\mathbf{s}^{-1/2}$$

2. We construct the transformation **X** from the eigenvalues and eigenvectors.

Step 4: Obtain a guess for the density matrix P

$$\mathbf{P} = \mathbf{0}$$

We simply use the null (zero) matrix

Step 5: Calculate G and F and F'

$$G_{ij} = \sum_{kl} P_{kl} \left[\langle ik | lj \rangle - \frac{1}{2} \langle il | kj \rangle \right]$$

$$F_{ij} = H_{ij}^{\text{core}} + G_{ij}$$

$$\mathbf{F}' = \mathbf{X}^{\dagger} \mathbf{F} \mathbf{X}$$

Recall: we already obtained the values for all two-electron integrals in step 2 and **X** in step 3.

Step 6: Diagonalize F' to obtain C' and ε

$$\mathbf{F}' = \mathbf{C}'\boldsymbol{\varepsilon}\mathbf{C}'^{\dagger}$$

We obtain the eigenvectors, i.e. the linear coefficients, and the energies of the molecular orbitals in the orthonormalized basis.

! Notice: This is the same diagonalization algorithm as employed in step 3.

Step 7: Calculate C from C'

$$\mathbf{C} = \mathbf{X}\mathbf{C}'$$

To get the linear coefficients in our regular base, we simply use the transformation matrix **X**. Because **X** is a unitary matrix, the know that the energies of the molecular orbitals in the regular basis are the same as in the orthonormalized basis!

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}'$$

Step 8: Form a new density matrix P from C

$$P_{ij} = 2\sum_{a}^{N/2} C_{ia} C_{ja}^{*}$$

Determine whether this new matrix P is the same as the old matrix P (within some margin). If it is, self-convergence is achieved, if not, go back to step 5.

Instead of checking whether the density matrices are the converged, you can also probe the total energy.

The SCF procedure : Overview

$$S_{ij} = \langle \varphi_i | \varphi_j \rangle$$

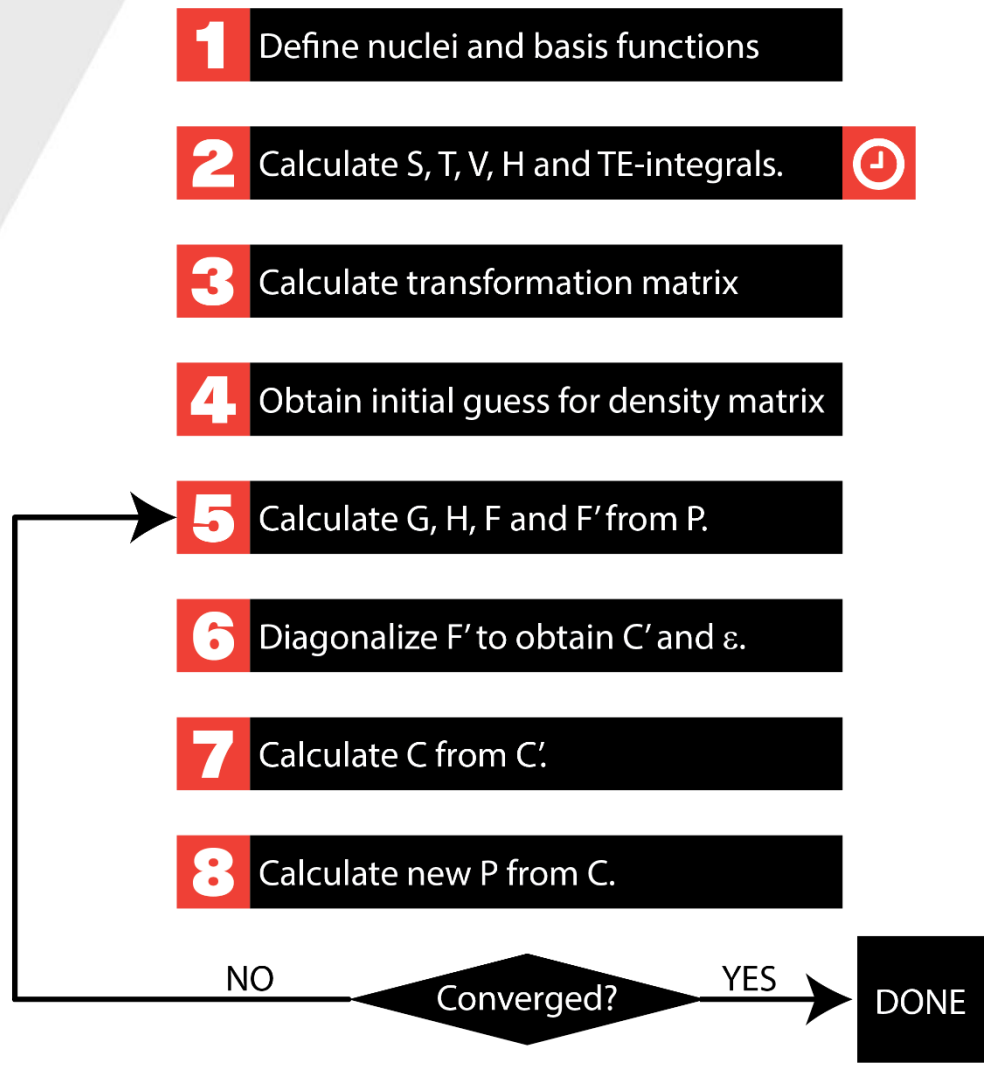
$$T_{ij} = \langle \varphi_i | -\frac{1}{2}\nabla^2 | \varphi_j \rangle$$

$$V_{ij} = \langle \varphi_i | -\sum_A \frac{Z_A}{|r - R_A|} | \varphi_j \rangle$$

$$H_{ij}^{\text{core}} = T_{ij} + V_{ij}$$

$$\langle \varphi_i \varphi_j | \varphi_k \varphi_l \rangle = \langle \varphi_i \varphi_j | r_{kl}^{-1} | \varphi_k \varphi_l \rangle$$

Calculating the two-electron integrals is the most time-consuming step and scales with N^4. (step 2)



Example: H_2O @ STO-3G (1/5)

- 3 atoms
- 7 CGF (2 for H, 5 for O)

$$\mathbf{S} = \begin{array}{c} \\ \varphi_{1s}^{O} \\ \varphi_{2s}^{O} \\ \varphi_{2p_x}^{O} \\ \varphi_{2p_y}^{O} \\ \varphi_{2p_z}^{O} \\ \varphi_{1s}^{H1} \\ \varphi_{1s}^{H2} \end{array} \begin{array}{c} \begin{array}{ccccccc} \varphi_{1s}^{O} & \varphi_{2s}^{O} & \varphi_{2p_x}^{O} & \varphi_{2p_y}^{O} & \varphi_{2p_z}^{O} & \varphi_{1s}^{H1} & \varphi_{1s}^{H2} \end{array} \\ \begin{bmatrix} 1 & 0.23114 & 0 & 0 & 0 & 0.15239 & 0.15239 \\ & 1 & 0 & 0 & 0 & 0.79121 & 0.79121 \\ & & 1 & 0 & 0 & 0.35345 & -0.35345 \\ & & & 1 & 0 & 0.27361 & 0.27361 \\ & & & & 1 & 0 & 0 \\ & & & & & 1 & 0.61996 \\ & & & & & & 1 \end{bmatrix} \end{array}$$

Example: H_2O @ STO-3G (2/5)

- Canonical orthogonalization of **S** gives **X**

(can you perhaps identify symmetry aspects here…)

$$\mathbf{X} = \begin{array}{c} \\ \varphi_{1s}^{O} \\ \varphi_{2s}^{O} \\ \varphi_{2p_x}^{O} \\ \varphi_{2p_y}^{O} \\ \varphi_{2p_z}^{O} \\ \varphi_{1s}^{H1} \\ \varphi_{1s}^{H2} \end{array} \begin{array}{c} \begin{array}{ccccccc} \varphi_{1s}^{O} & \varphi_{2s}^{O} & \varphi_{2p_x}^{O} & \varphi_{2p_y}^{O} & \varphi_{2p_z}^{O} & \varphi_{1s}^{H1} & \varphi_{1s}^{H2} \end{array} \\ \begin{bmatrix} 1.02259 & -0.14215 & 0 & -0.00769 & 0 & 0.00666 & 0.00666 \\ & 2.26169 & 0 & 0.42701 & 0 & -0.90516 & -0.90516 \\ & & 1.41635 & 0 & 0 & -0.67573 & 0.67573 \\ & & & 1.14695 & 0 & -0.31189 & -0.31189 \\ & & & & 1 & 0 & 0 \\ & & & & & 2.13618 & -0.46540 \\ & & & & & & 2.13618 \end{bmatrix} \end{array}$$

Example: H_2O @ STO-3G (3/5)

Integral evaluation yields (for example) the following H matrix:

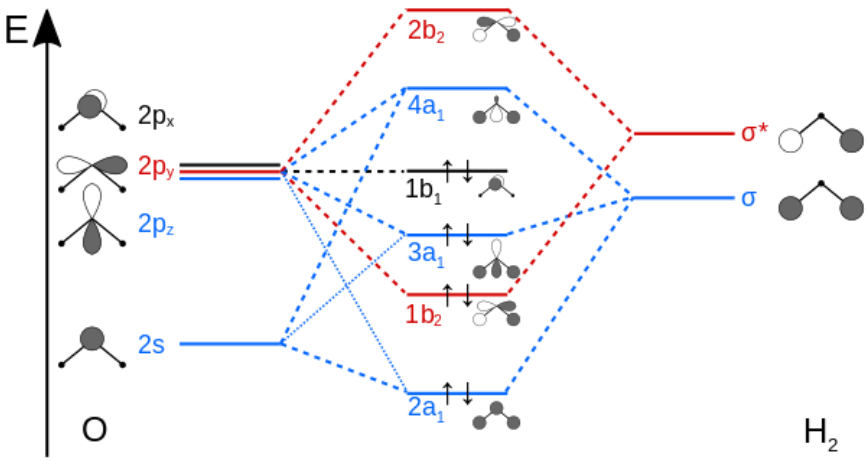
$$\mathbf{H} = \begin{array}{c} \\ \varphi_{1s}^{O} \\ \varphi_{2s}^{O} \\ \varphi_{2p_x}^{O} \\ \varphi_{2p_y}^{O} \\ \varphi_{2p_z}^{O} \\ \varphi_{1s}^{H1} \\ \varphi_{1s}^{H2} \end{array} \begin{array}{c} \begin{array}{ccccccc} \varphi_{1s}^{O} & \varphi_{2s}^{O} & \varphi_{2p_x}^{O} & \varphi_{2p_y}^{O} & \varphi_{2p_z}^{O} & \varphi_{1s}^{H1} & \varphi_{1s}^{H2} \end{array} \\ \begin{bmatrix} -34.02030 & -7.52954 & 0 & -0.07106 & 0 & -5.00637 & -5.00637 \\ & -9.90374 & 0 & -0.45101 & 0 & -7.39309 & -7.39309 \\ & & -8.38763 & 0 & 0 & -2.72350 & 2.72350 \\ & & & -8.23916 & 0 & -2.36381 & -2.36381 \\ & & & & -8.01716 & 0 & 0 \\ & & & & & -7.74241 & -5.31508 \\ & & & & & & -7.74241 \end{bmatrix} \end{array}$$

Example: H_2O @ STO-3G (4/5)

Solutions (eigenvectors and eigenvalues)

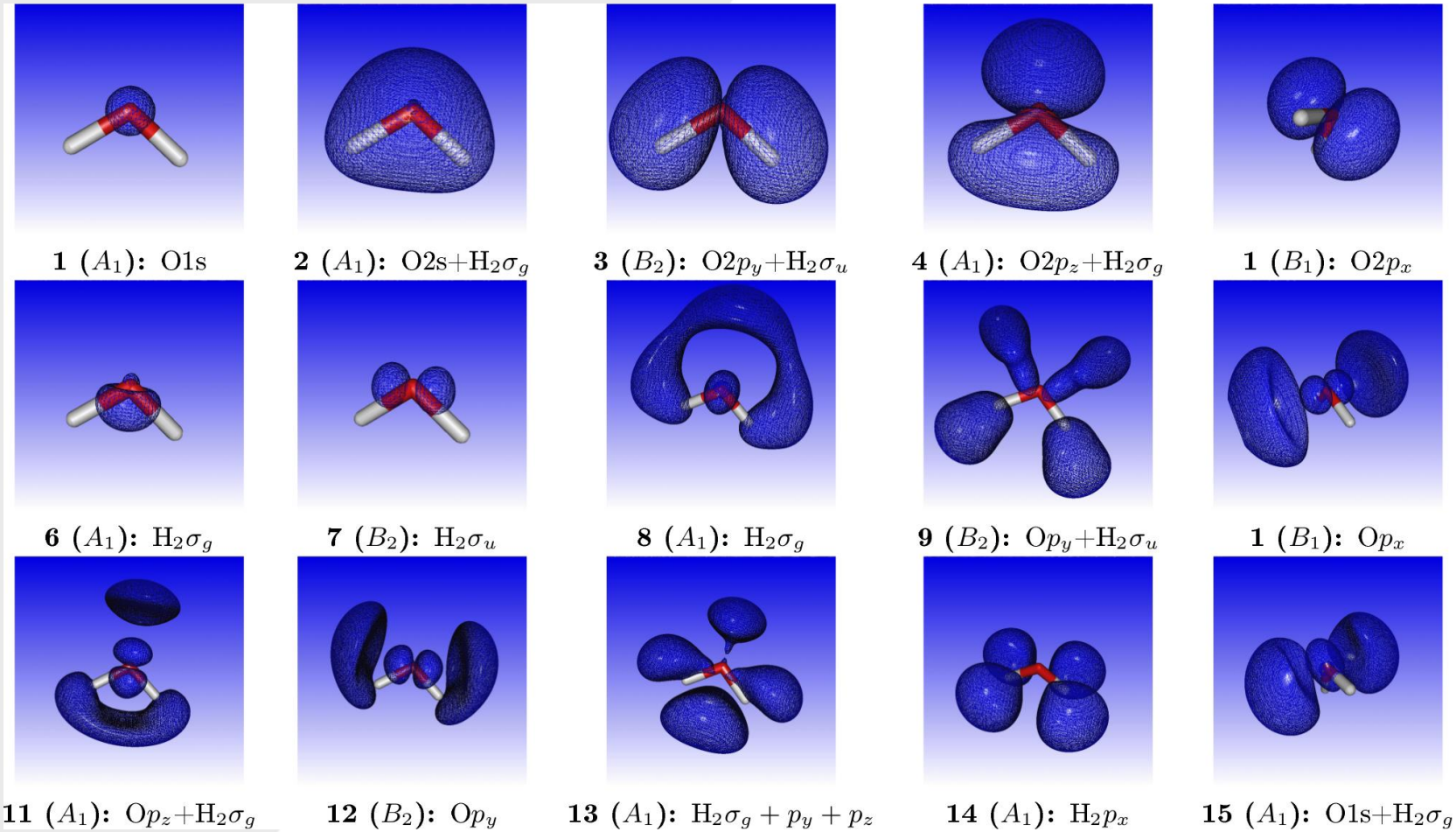
$$\mathbf{C} = \begin{bmatrix} & -20.7292555611 & -1.6971769512 & -0.9543621271 & -0.5943021376 & -0.5560416689 & 1.0154742617 & 1.5060329764 \\ \varphi_{1s}^{O} & -0.9959 & 0.2284 & 0 & -0.0984 & 0 & 0.1112 & -0.0000 \\ \varphi_{2s}^{O} & -0.0324 & -0.8682 & 0 & 0.4427 & 0 & -2.4506 & -0.0000 \\ \varphi_{2p_x}^{O} & -0.0000 & -0.0000 & 0.7151 & 0 & 0 & -0.0000 & -1.5517 \\ \varphi_{2p_y}^{O} & -0.0099 & -0.3737 & 0 & -0.9056 & 0 & -0.8559 & 0.0000 \\ \varphi_{2p_z}^{O} & -0.0000 & -0.0000 & 0 & 0 & 1 & 0 & 0 \\ \varphi_{1s}^{H1} & 0.0115 & -0.0461 & 0.3766 & -0.0201 & 0 & 1.5198 & 1.9232 \\ \varphi_{1s}^{H2} & 0.0115 & -0.0461 & -0.3766 & -0.0201 & 0 & 1.5198 & -1.9232 \end{bmatrix}$$

From the solution you can (for instance) generate an **MO diagram**



Example: H_2O @ STO-3G (5/5)

Visualize orbitals



Han-sur-Lesse 2017

IV. Building the code

Ivo Filot - i.a.w.filot@tue.nl

Inorganic Materials Chemistry
TU/e Technische Universiteit Eindhoven University of Technology

Basis functions

For the basis functions, we are going to use Gaussian Type Orbitals (GTOs)

$$\phi_{\mathrm{GTO}} = d \cdot \left(x - X_A\right)^l \cdot \left(y - Y_A\right)^m \cdot \left(z - Z_A\right)^n \cdot \exp\left(-\alpha \left|r - R_A\right|^2\right)$$

Which in turn can form Contracted Gaussian Orbitals (CGFs) as:

$$\varphi_{\mathrm{CGF}} = \sum_{i=1}^{L} c_i \phi_i \left(\alpha, l, m, n, \vec{\mathrm{R}}\right)$$

Example: STO-3G for H at (0,0,0)

$$\varphi_{\mathrm{CGF}} = \sum_{i=1}^{L} c_i \phi_i \left(\alpha, l, m, n, \vec{\mathrm{R}}\right)$$

$$L = 3$$

$$l = m = n = 0$$

$$R_A = (0,0,0)$$

1S (H)	α	c_i
1	3.425251	0.154329
2	0.623914	0.535328
3	0.168855	0.444635

C++ code

```
// construct cgf
const vec3 pos1(0.0, 0.0, 0.0);
CGF cgf1(pos1);
cgf1.add_gto(CGF::GTO_S, 3.4252509099999999,  0.15432897000000001, pos1);
cgf1.add_gto(CGF::GTO_S, 0.62391373000000006, 0.53532813999999995, pos1);
cgf1.add_gto(CGF::GTO_S, 0.16885539999999999, 0.44463454000000002, pos1);
```

Integral library (integrals.cpp)

Types of integrals

- Overlap integrals
- Kinetic
- Nuclear
- Two-electron integrals

One class that solves all types of integrals given a set of CGFs (which are also a class).

C++ code

```
// overlap integrals
integrator.overlap(cgf1, cgf2);

// kinetic energy integrals
integrator.kinetic(cgf1, cgf2);

// nuclear attraction integrals
integrator.nuclear(cgf1, cgf2, pos, charge);

// two-electron integrals
integrator.repulsion(cgf1, cgf2, cgf3, cgf4);
```

Integral library (integrals.cpp)

Gaussian Type Orbital Integral evaluation is **non-trivial**. It relies on a relatively complex set of routines. Nevertheless, it is fast (relatively speaking) and can be (rather) trivially parallelized.

$$\varphi_{\mathrm{i,CGF}} \cdot \varphi_{\mathrm{j,CGF}} = \varphi_{\mathrm{p,CGF}}$$

The product of two GTOs, is a **new** GTOs. And the product of four GTOs is by extension also a new GTOs. Hence, all type of integral evaluations can be back-transformed to **single-GTO** integrals.

Integral library: Overlap integrals

$$\langle \varphi_i | \varphi_j \rangle = \exp\left(\frac{\alpha_i a_j \left| \vec{R}_i - \vec{R}_j \right|^2}{\alpha_i + a_j} \right) S_x S_y S_z$$

$$S_x = \sqrt{\frac{\pi}{\alpha_i + a_j}} \sum_{n=0}^{\frac{l_i + l_j}{2}} f_{2n}\left(l_i, l_j, \left| X_P - X_i \right|, \left| X_P - X_j \right| \right) \frac{(2n-1)!!}{\left(2\left(\alpha_i + a_j \right) \right)^n}$$

$$f_n(l, m, a, b) = \sum_{k=\max(0, n-m)}^{\min(j,l))} \binom{l}{k} \binom{m}{n-k} a^{l-k} b^{m+k-n}$$

Don't focus too much on this. I just want to show how it's done.

Integral library: Kinetic integrals

$$-\frac{1}{2}\nabla^2|\varphi\rangle = \alpha\left(2(l+m+n)+3\right)\varphi\left(\alpha,l,m,n,\vec{\mathrm{R}}\right)$$

$$-2\alpha^2\underbrace{\left[\varphi\left(\alpha,l+2,m,n,\vec{\mathrm{R}}\right)+\varphi\left(\alpha,l,m+2,n,\vec{\mathrm{R}}\right)+\varphi\left(\alpha,l,m,n+2,\vec{\mathrm{R}}\right)\right]}_{\text{overlap integrals of two orders higher}}$$

$$-\frac{1}{2}\underbrace{\left[l(l-1)\varphi\left(\alpha,l-2,m,n,\vec{\mathrm{R}}\right)+m(m-1)\varphi\left(\alpha,l,m-2,n,\vec{\mathrm{R}}\right)+n(n-1)\varphi\left(\alpha,l,m,n-2,\vec{\mathrm{R}}\right)\right]}_{\text{overlap integrals of two orders lower}}$$

In other words: the kinetic integral can be expanded as a **set of overlap integrals**.

! Don't focus too much on this. I just want to show how it's done.

Integral library: Nuclear integrals

$$\langle \varphi_i | \frac{1}{\vec{R}_C} | \varphi_j \rangle = \frac{2\pi}{\alpha_i + \alpha_j} \exp\left(\frac{\alpha_i a_j \left| \vec{R}_i - \vec{R}_j \right|^2}{\alpha_i + a_j} \right) \cdot$$

$$\underbrace{\sum_{l=0}^{\frac{l_i+l_j}{2}} \sum_{r=0}^{\frac{l}{2}} \sum_{i=0}^{\frac{l-2r}{2}} A_{l,r,i}\left(l_i, l_j, X_i, X_j, X_C, \alpha_i + \alpha_j\right) \cdot}_{\text{x-component}}$$

$$\underbrace{\sum_{m=0}^{\frac{m_i+m_j}{2}} \sum_{s=0}^{\frac{m}{2}} \sum_{j=0}^{\frac{m-2s}{2}} A_{m,s,j}\left(l_i, l_j, Y_i, Y_j, \mathrm{Y}_C, \alpha_i + \alpha_j\right) \cdot}_{\text{y-component}}$$

$$\underbrace{\sum_{n=0}^{\frac{n_i+n_j}{2}} \sum_{t=0}^{\frac{n}{2}} \sum_{k=0}^{\frac{n-2t}{2}} A_{n,t,k}\left(l_i, l_j, Z_i, Z_j, Z_C, \alpha_i + \alpha_j\right) \cdot}_{\text{z-component}}$$

$$F_{l+m+n-2(r+s+t)-(i+j+k)}\left(\alpha_i + \alpha_j \left|\vec{P} - \vec{C}\right|^2\right)$$

where

$$A_{l,r,i}\left(l_i, l_j, X_i, X_j, X_C, \gamma\right) = (-1)^i f_l\left(l_i, l_j, \left|X_P - X_i\right|, \left|X_P - X_j\right|\right) \cdot \frac{(-1)^i l! \left|X_P - X_C\right|^{l-2r-2i} \varepsilon^{r+i}}{r!\, i!\, (l-2r-2i)!}$$

and

$$\varepsilon = \frac{1}{4\gamma}$$

and

$$\gamma = \alpha_i + \alpha_j$$

Don't focus too much on this. I just want to show how it's done.

Object-oriented programming

In the earliest programming languages, functionality was captured in functions (makes sense...). In object-oriented programming, we can have another level of abstraction by encapsulating similar functions into a class. The instantiation of that class, is called an object. Here, the *integral evaluation* are encapsulated inside such a class. The user only has to use a simple interface in order to leverage the complex functionality.

<u>TL;DR</u>

I have made complex stuff easy by putting it inside class.

Matrix library (eigen3)

We need a library that defines matrices and is able to do matrix diagonalization. Here, we will use the eigen3 library (open source).

Technical note: There are a plethora of matrix-diagonalization algorithms. They are typically classified by the type of matrix to diagonalize. We will solely diagonalize real (i.e. non-complex) self-adjoint matrices. In other words: symmetric matrices.

C++ code

```
// calculate eigenvalues and eigenvectors
Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> es(A);
Eigen::MatrixXd D = es.eigenvalues().real().asDiagonal();
Eigen::MatrixXd U = es.eigenvectors().real();
```

Matrix library (eigen3)

A very important note which will save you a lot of headache

A matrix diagonalization routine will typically solve this:

$$\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}$$ *(unless specified otherwise)*

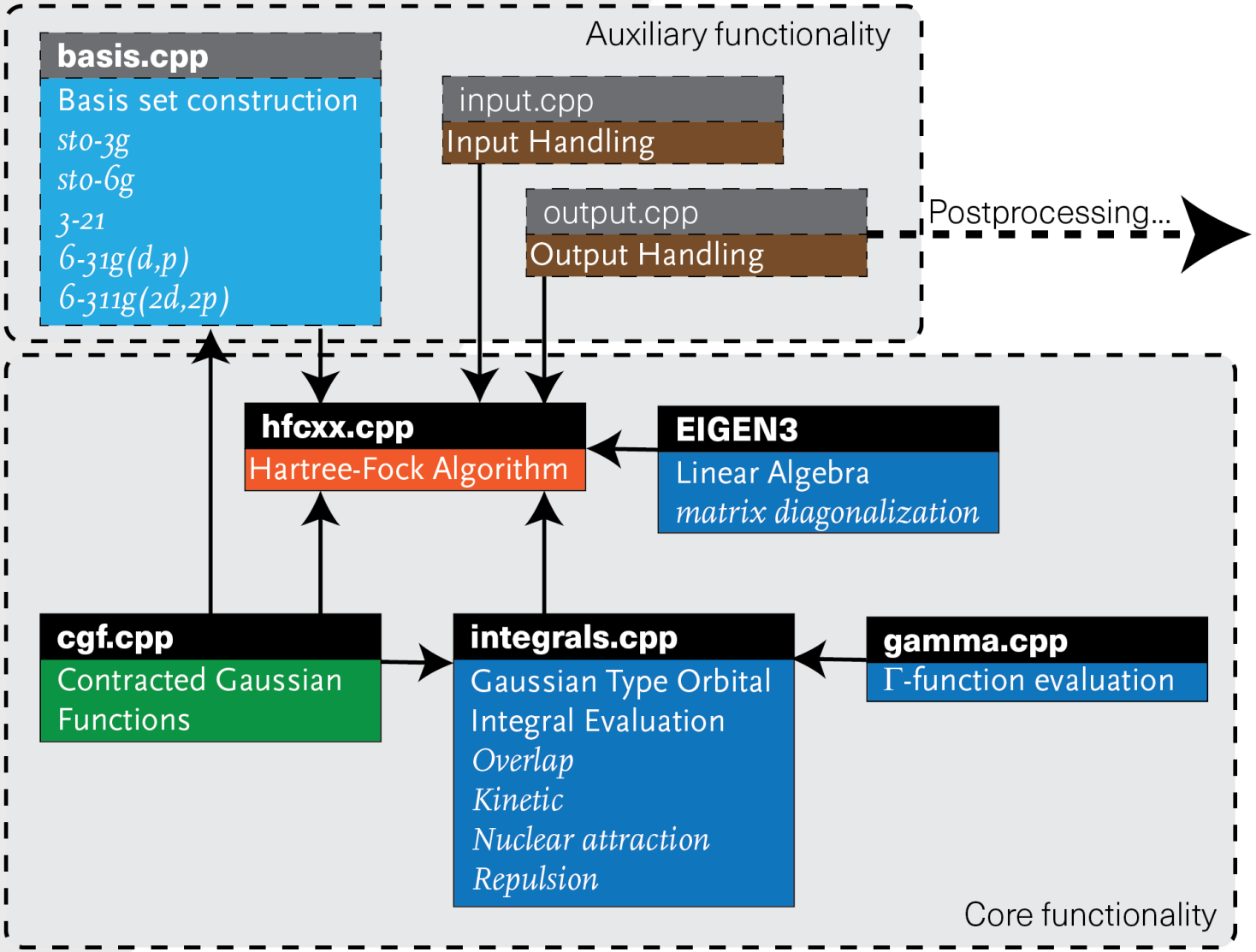
But, you need this for **canonical orthogonalization**:

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^{\dagger}$$ *U is here unitary*

Hence, make absolutely sure that your matrix diagonalization library does this. The default *EigenSolver* from Eigen3 does **not** do this, you need the **SelfAdjointEigenSolver** for this.

So check it! (exercise 4)

Program architecture



Han-sur-Lesse 2017

V. The exercises

Ivo Filot - i.a.w.filot@tue.nl

Inorganic Materials Chemistry

TU/e Technische Universiteit Eindhoven University of Technology

Learning strategy

Assumptions

- This is a very heterogeneous class (some with abundant programming / math skills, others will only basics)
- You are at least comfortable with reading code
- You are proficient with abstraction and *ok* with loosing details to gain conceptual understanding
- You want to be in charge of your own learning trajectory

Learning strategy

- **Collaborative learning**: Not everyone is an expert. Learn from your neighbor. Even if your neighbor is complete and utter expert in the field, you can still help him/her in conveying knowledge.
- **Learn by example**: There are no good programming courses. There are only good programmers and programs. Learn by copying. Next: Improvise. Adapt. Overcome.

Exercises

1. H atom
   Calculate the electronic energy using a basis function
2. He atom
   Explore two-electron integrals
3. H_2 atom
   Construct S, T, V matrices and two-electron integrals using the integral library
4. Transformation matrix
   Calculate the transformation matrix to orthonormalize your basis set
5. Self-consistent field calculation
   Perform a self-consistent field calculation on H_2 and adapt the algorithm to accommodate a larger basis set

Challenge for the fast learners

For the fast learners, there is a bonus exercise:

Adapt the H_2 algorithm to calculate the carbon monoxide (CO) molecule.

<u>However</u>, since the solution for this exercise is given, I challenge you to calculate methane (CH_4) at the STO-3G level of theory.