CHIRALEX: theory

Emil J. Zak[1,*]

[1] *Center for Free-Electron Laser Science, Deutsches Elektronen-Synchrotron DESY, Notkestraße 85, 22607 Hamburg, Germany*

(Dated: February 3, 2022)

I. INTRODUCTION

Essential parameters are kept in the *params* dictionary, which is passed throughout the program. It is used to construct all types of objects, from grids to the propagator object.

II. BASIS FUNCTIONS AND GRIDS

A. Radial quadratures

B. Radial grid

The radial grid used to solve the TDSE is based on an underlying Gauss-Lobatto (GL) quadrature scheme. First, a generic quadrature grid is created: $\{x_k\}_{k=0,1,...,Nlobs-1} \in [-1,1]$. Its size is determined by the *Nlobs* input parameter (Nl for short notation). Note that the boundary points in the Gauss-Lobatto grid are -1 and $+1$. The GL grid is then scaled to reflect the bin size R_b requested by the user (*binw* input parameter):

$$S_k = \frac{1}{2} R_b \, (x_k + 1) \tag{1}$$

Bin size must be determined empirically to ensure good convergence and stability of the results. Next, the scaled grid is copied and translated by the bin size to generate the full primitive radial grid G_{prim}:

$$r_{ik} = S_k + T_i, \qquad i = 0, 1, ..., Nb-1 \, , k = 0, 1, ..., Nlobs-1 \tag{2}$$

where $T_i = i \cdot R_b + \epsilon$. The constant shift ϵ (rshift keyword) is optional and by default is set to 0. The resulting grid has the size $Nb \times Nlobs$ and contains boundary points $r_{00} = \epsilon$ and $r_{Nb-1,Nlobs-1} = R_{max}$. The last point from bin i and the first point from bin $i+1$ are identical in the primitive grid.

The *coupled grid* G is generated by removing boundary points, plus merging duplicate points at bin boundaries in G_{prim}. Through these operations the size of the coupled grid is $Nr = Nb \times (Nlobs - 1) - 1$. Note that all indices start from 0 for python compatibility. For example $G_{00} = r_{01}$ and $G_{0,Nl-3} = r_{0,Nl-2}$. The boundary of the first bin is at $G_{0,Nl-2} = r_{0,Nl-1} = r_{10}$. The last point in the coupled grid is $G_{Nb-1,Nl-3} = r_{Nb-1,Nl-2}$.

C. Radial basis functions

1. *Interpolated radial functions*

D. Angular basis functions

III. INDEX MAPPING

A natural choice for mapping the basis set indices is to set the grid points as major dimension (changing last). Such a choice offers a simple way of cutting the Hamiltonian matrix at a given grid point by slicing the basis. We name this convention as 'DVR' mapping. Bridge functions, as discussed in II B, are defined for indices $n = Nl - 2$, that is the last index in each bin. A consequence of such a choice is a near block-diagonal structure of the Hamiltonian. Another common choice is to choose bridge functions as first in the bin ($n = 0$), which results in 'arms' reaching out from bin i into bin $i+1$, as shown in **??**. Another option given in the code is to select mapping in which the angular quantum numbers l, m are the major dimension. We name this convention 'SPECTRAL'. The choice for mapping is determined by the 'map_type' = 'DVR,SPECT' keyword. With the 'DVR' convention the map is given as follows:

$$p(i,n,l,m) = (i \cdot (Nl-1) + n) \cdot (l_{max}+1)^2 + l \cdot (l+1) + m \tag{3}$$

with $p(0,0,0,0) = 0$. For $l_{max} = 0$ we have $p(Nb - 1, Nl - 3, 0, 0) = Nb \times (Nl - 1) - 2$, which retrieves the size of the coupled radial grid/basis set. The map is stored in a numpy array ('maparray') and is saved in file. The ranges for the radial indices i and n are the same as for the coupled grid, i.e. $i = 0, 1, 2, ...Nb - 1$ and $n = 0, 1, 2, ..., Nl - 2$ for $i < Nb - 1$ and $n = 0, 1, 2, ..., Nl - 3$ for $i = Nb - 1$.

IV. ROTATED ELECTROSTATIC POTENTIAL

Each molecular-frame orientation gives different electrostatic potential as seen from the laboratory frame, in which the basis set is defined. This means that for each orientation one must generate generally a different Lebedev quadrature grid. Even if a global scheme is used,

* emil.zak@cfel.de

TABLE I. Example 'DVR' coupled basis set index mapping for $Nb = 3$, $Nl = 3$, $lmax = 1$.

i	n	ξ	l	m	p
0	0	0	0	0	0
0	0	0	1	-1	1
0	0	0	1	0	2
0	0	0	1	+1	3
0	1	1	0	0	4
0	1	1	1	-1	5
0	1	1	1	0	6
0	1	1	1	+1	7
...					
2	0	4	1	0	18
2	0	4	1	+1	19

the values of the electrostatic potential will be different at these points for each orientation. This is because the lebedev grid is defined in the laboratory frame. Therefore at this stage, a separate Psi4 calculation must be initialized for each orientation. I do not see a way around it at the moment, other than performing multipole expansion and calculating the angular part (spherical tensor form) of the potential analytically. But finding this expansion is costly, even more than running Psi4 for every orientation.

V. KINETIC ENERGY OPERATOR

VI. POTENTIAL ENERGY OPERATOR

VII. HAMILTONIAN

The current paradigm assumes calculation of the sparse Hamiltonian matrix and feeding this matrix into an eigensolver or propagator routine. In future versions, for better computational performance it may be useful not to calculate the Hamiltonian matrix, but rather code the matrix-vector product only, as part of an interface routine with appropriate iterative eigensolver. Due to high degree of sparsity of the Hamiltonians appearing in our calculations, it is sufficient to keep in memory the whole sparse matrix.

We decided to keep the bound state Hamiltonian H_0 and the initial propagation Hamiltonian H_{init} as separate entities, calculated independently. The computational overhead related to calculating the bound-state part twice is marginal, as we assume that the full propagation basis is much bigger than the basis for the bound Hamiltonian $Nbas \gg Nbas0$. Future releases might recycle the bound state KEO and POT and build only the outside region Hamiltonian.

VIII. WAVEPACKET PROPAGATION

A. time grid

The time grid for the calculation is determined by the input keywords:

TABLE II. Time grid keywords.

keyword	description	type
t0	start time	float
tmax	end time	float
dt	time step	float
time_units	units	string
wfn_saverate	save rate of the wavepacket	float

The generated grid is equidistant given by the following formula:

$$t[i] = t_0 + i \cdot dt;\ i = 0, 1, 2, ..., Ntpts \tag{4}$$

where

$$Ntpts = \left\lceil \frac{t_{max} - t_0}{dt} \right\rceil \tag{5}$$

such that the array of time-points contains elements: $t[0], t[1], ..., t[Ntpts]$. Note that the last point (tmax) is included in the grid.

IX. INPUT FILE

- *molec_name*: name of the molecule (chiralium, h, h2s, etc.)
- *matelem_method*: method of choice for the calculation of the potential energy matrix elements. Availability of methods given below depends on the chosen molecule.
  - *analytic*: use analytic formulas for the matrix elements. When the potential is given as an expansion in the spherical harmonics basis, utilize 3j symbols to calculate the matrix elements.
  - *lebedev*: use own implementation of the spherical Lebedev quadratures with fixed global quadrature level for all grid points defined in *sph_quad_global* = "lebedev_119", "lebedev_131", etc.
  - *lebedev_adaptive*: use own implementation of the spherical Lebedev quadratures with adaptive quadrature level, optimized for each grid point. Stopping condition is based on convergence criteria defined in *sph_quad_tol* tolerance (in a.u.) for the convergence of matrix elements with the maximal values of the l quantum number.

  - *quadpy*: use the Quadpy library to calculate the spherical integrals with Lebedev quadratures.
  - *others*: to be implemented.

- *esp_mode*: method of choice for the calculation of the cationic's core electrostatic potential.
  - *analytic*: available only for the hydrogen atom and some model systems.
  - *psi4*: use psi4 quantum chemistry package for calculate the ESP for all required points.
  - *anton*: only available for the chiralium molecule. Read the precomputed potential kindly provided by A. Artemyev.
  - *psi4_interpolated*: produce an interpolated ESP based on psi4 points.
  - *others*: to be implemented. Any method one can imagine, including the use of other quantum chemistry software.

- *keo_method = klist,slices,blocks*. Method of choice for the calculation of the kinetic energy matrix elements. See **??** for more details. Three main methods are considered so far:
  - *klist* calls *build_keomat_klist*, which uses a precomputed list (*klist*) of all matrix indices for which the matrix elements can be nonzero. It is by far the slowest method, even when Numba jit acceleration is used.
  - *slices* calls *build_keomat_slices*, which utilizes the fact that identical copies of the K_D and K_C matrices are distributed over the KEO matrix. The respective function loops over FEM-DVR bins and in each bin appropriately indexed matrix elements of the KEO matrix have the respective elements of K_D and K_C assigned. The algorithm relies on slicing a scipy.sparse.lil_matrix. The centrifugal energy term is added to the diagonal at the end using *scipy.sparse.diags* method.
  - *blocks* calls *build_keomat_blocks*, which constructs the KEO using *scipy.sparse.block_diag* method. First a block-diagonal matrix formed from inflated K_D matrices is formed. Next the inflated K_C matrices are further inflated to the size of each K_D block and a block-diagonal matrix is formed. The resulting matrix is then stacked with appropriate empty matrices to match the shape of the KEO. Hermitian conjugate of this matrix is calculated. All three matrices are then combined. The centrifugal energy term is added to the diagonal at the end using *scipy.sparse.diags* method.