CHEM 555-75 **FINAL PROJECT** **Fall 2021**

Instructor: Prof. Lee M. Thompson E-mail: *lee.thompson.1@louisville.edu* Office: CB 251

Introduction

For the final project, the goal is to code and use your own Hartree-Fock computer program. You should already have MQC installed on the CRC and be able to compile and run programs. Additionally, you should know how to run a Gaussian calculation to obtain a matrix file. Below is an initial code which code which you should use to start your code. Try compiling it before you modify it to check you have compiled it properly. In the code are comments which explain where to add the functions and subroutines listed below. Remember to compile and test your code frequently to make sure you didn't make any mistakes. Once you have a working program, try to run different molecules and check everything still works. Look at how long the calculation takes with different molecules with different sizes of basis sets (given by the nBasis variable) and explore how the calculation scales with the size of nBasis.

Assignment Details

Send your completed code along with a two page report. The two page report should include:

1. Title succinctly explaining the project, name and date
2. Introduction explaining in your own words the Roothan-Hall Hartree-Fock (RHHF) method. Explain:
    - The type of wavefunction the RHHF method is optimizing and what physics is included in the wavefunction
    - The deficiencies of the wavefunction produced by RHHF compared to the exact solution to the Schrödinger equation
    - How the wavefunction in RHHF is represented on a computer and how this representation differs from the original Hartree-Fock formulation
3. Method section explaining in your own words how RHHF works, what are the inputs, what are the outputs, why is the process iterative?
4. Results explaining in your own words some calculations you ran with your code
    - Get a matrix file with 2 equidistant hydrogen atoms in a line, then get the same for 4, 8, 16, 32, 64... hydrogen atoms in a line (keep extanding the size of the system until the calculation takes an appreciable amount of time to rum). Run your code with the maximum iterations set to 1 and time how long the code takes. Make a graph of your results and explain how the RHHF algorithm scales, i.e. if you double the size of the system, how much more time does the calculation take. Can you figure out which part of the RHHF code is the source of the computational cost and explain why?

  - Try some different molecules of your choice (keep them fairly small) and try to get your code to converge (set max_iters to 256 or 512). When does the code converge and when does it not converge? For the cases where the code does not converge, plot the energy with respect to iteration and difference norm of the density matrix with respect to iteration. Do your results indicate that you just need to increase the number of iterations, or is there some other problem occurring?

5. Conclusion section explaining in your own words what you did and what you found. Describe how you might want to improve your code to get more accurate results or better convergence characteristics.

Your project will be graded based on the correctness of your code and the quality of your report. In particular the following criteria will be used with equal weight:

- Code correctness – were you able marry the equations on the lecture slides, the class discussions, the knowledge gained from completing the worksheets through the semester, and the instructions provided, in order to construct a working and usable code?
- Understanding – is it clear that you know why we are bothering to write a RHHF code, do you know what the outputs of the code are and why they are useful, to what extent have you demonstrated an understanding of basic principles of quantum chemistry that would be covered in an undergraduate course, to what extent have you furthered your knowledge this semester to solving the many-electron problem on a computer, have you built a solid foundation for understanding, appreciating and using computational chemistry in the context of all chemistry research?
- Clarity – were you able to follow the instructions provided, can you logically organize your thoughts, do your paragraphs connect to each other, is your spelling and grammar at the level expected of a first semester graduate student, can you succinctly and lucidly explain complicated ideas and concepts in a way that can be described by a general chemistry audience?

Objects and Routines

The following objects contain data from the matrix file, or will contain your data:

- The mqc_wavefunction type contains several kinds of data that can be accessed using a % separator, e.g. mqc_wavefunction%mo_coefficients will access the MO coefficients. The type of the data object within the mqc_wavefunction object are listed below along with that object.

  mqc_wavefunction type
  type(mqc_scf_integral) mo_coefficients ! contains the MO coefficients
  type(mqc_scf_eigenvalues) mo_energies ! contains the MO energies
  type(mqc_scf_integral) core_hamiltonian ! contains the core Hamiltonian
  type(mqc_scf_integral) fock_matrix ! contains the Fock matrix
  type(mqc_scf_integral) density_matrix ! contains the density matrix
  type(mqc_scf_integral) overlap_matrix ! contains the AO overlap matrix
  type(mqc_scalar) nalpha ! number of alpha electrons
  type(mqc_scalar) nbeta ! number of beta electrons

```
type(mqc_scalar) nelectrons ! total number of electrons
type(mqc_scalar) nbasis ! number of basis functions
type(mqc_scalar) charge ! molecule charge (protons-electrons)
type(mqc_scalar) multiplicity ! wavefunction multiplicity
character(len=256) wf_type ! symmetry of the wavefunction (R/U/G)
```

The following subroutines and functions will be required to develop your code. In bold is how the subroutine should be used in your code. The arguments in parenthesis must have the same type (even if the name is different) as indicated in the type declaration below the line in bold. Text after the ! are comments providing information about the argument/subroutine/function.

- You will use MQC SCF integral types widely in your code. The following routines will prove useful:

  call mqc_scf_integral%print(iOut,header)
  ```
     integer(kind=int64),intent(in)::iOut ! Output file
     character(len=*),intent(in)::header ! String that is printed with integral
  ! Prints the integral
  ```

  call mqc_scf_integral%init(nAlpha,nBeta)
  ```
     integer,intent(in)::nAlpha,nBeta ! Dimension of alpha and beta basis functions
  ! Initializes an integral with spin block dimensions set by nAlpha and nBeta
  ```

  call mqc_scf_integral%diag(eVals,eVecs)
  ```
     type(mqc_scf_eigenvalues),optional,intent(inOut)::eVals ! Eigenvalues
     type(mqc_scf_integral),optional,intent(inOut)::eVecs ! Eigenvectors
  ! Diagonalizes an integral and returns eigenvalues and eigenvectors
  ```

  norm = mqc_scf_integral%norm(methodIn)
  ```
     character(len=1),optional,intent(in)::methodIn ! Method for computing norm
  !      'M' - max(abs(A(i,j)))
  !      '1' - one norm
  !      'I' - infinty norm
  !      'F' - Frobenius norm (default)
     type(mqc_scalar)::norm ! Return value of norm
  ! Determines the norm (size) of a matrix
  ```

  integralOut = mqc_scf_integral%orbitals(alphaOrbs,betaOrbs)
  ```
     integer(kind=int64),dimension(:),intent(in)::alphaOrbs,betaOrbs ! list of de-
  sired order orbitals
  !      Example for nAlpha is [(i,i=1,nAlpha)] which gives argument [1,2,3...nAlpha]
  !      which would correspond to the set of alpha occupied orbitals
     type(mqc_scf_integral)::integralOut ! Integral with output orbitals
  ! Select subset and reorder columns in an integral, mainly for MO coefficients
  ```

- MQC SCF eigenvalues will contain MO energies on return from diagonalization. The only routine you require is to print:

  call mqc_scf_eigenvalues%print(iOut,header)

integer(kind=int64),intent(in)::iOut ! Output file
character(len=*),intent(in)::header ! String that is printed with integral
! Prints the eigenvalues

- In addition the following routines are required:

integralOut = matmul(integralA,integralB)
type(mqc_scf_integral),intent(in)::integralA,integralB ! Integrals to multiply
type(mqc_scf_integral)::integralOut ! Output integral
! Multiplies two integrals together

integralOut = transpose(integral)
type(mqc_scf_integral),intent(in)::integral ! Integral to transpose
type(mqc_scf_integral)::integralOut ! Output integral
! Transposes an integral

integralOut = dagger(integral)
type(mqc_scf_integral),intent(in)::integral ! Integral to conjugate transpose
type(mqc_scf_integral)::integralOut ! Output integral
! Conjugate transposes an integral

output = contraction(integral1,integral2)
type(mqc_scf_integral),intent(in)::integral1,integral2 ! Integrals to contract
type(mqc_scalar)::output ! Output value
! Contracts two integrals

output = contraction(eris,integral)
type(mqc_twoERIs),intent(in)::eris ! 2ERIs to contract
type(mqc_scf_integral),intent(in)::integral ! Integral to contract
type(mqc_scalar)::output ! Output value
! Contracts an integral with the last two indices of 2ERIs

call mqc_scf_transformation_matrix(overlap,transform_matrix)
type(mqc_scf_integral),intent(in)::overlap ! Overlap matrix
type(mqc_scf_integral),intent(out)::transform_matrix ! X matrix
! Computes the transformation matrix for converting between AO and
!   orthogonalized basis

- You will need to compute the nuclear-nuclear repulsion energy from mqc_molecule_data:

Vnn = mqc_get_nuclear_repulsion(iOut,mqc_molecule_data)
class(mqc_molecule_data),intent(in)::mqc_molecule_data ! MQC molecule data
type(mqc_scalar)::Vnn ! Total nuclear repulsion
! Computes the nuclear repulsion of a molecule

- The following two lines will be useful for having an mqc_scalar and an integer type version available (the routine ival extracts the value of an mqc_scalar as an integer):

nAlpha = wavefunc%nAlpha%ival()
nBeta = wavefunc%nBeta%ival()

Recommended Strategy

The strategy for developing your code is shown below, where the order provided is the order that the operations should be performed in the code. Below there is an initial code to get you started, where the position of each of the operations below is indicated. Note that computation of nuclear-nuclear repulsion energy and testing convergence procedures are already completed for you in the initial code as an example. The functions and subroutines required to perform the procedures below are given, and more detailed information about the exact nature of the inputs to these functions and subroutines can be found in the section above.

- Compute the nuclear-nuclear repulsion energy.
  - $V_{nn} = \sum_{IJ} \frac{Z_I Z_J}{r_{IJ}}$
  - suggested MQC routines:
    - **Vnn = mqc_get_nuclear_repulsion(iOut,mqc_molecule_data)**
- Initialize density matrix.
  - $\mathbf{P} = \mathbf{0}$
  - suggested MQC routines:
    - **call mqc_scf_integral%init(nBasis,nBasis)**
- Determine AO to OAO transformation matrix.
  - $\mathbf{X} = \mathbf{U}\mathbf{s}^{-1/2}$
  - suggested MQC routines:
    - **call mqc_scf_transformation_matrix(overlap,transform_matrix)**
- Enter iterations.
  - Form G matrix.
    - $\mathbf{G}(\mathbf{P}) = \sum_{\lambda\tau}^{N_{\text{basis}}} P_{\lambda\tau}(\langle\phi_\nu\phi_\lambda|r_{12}^{-1}|\phi_\mu\phi_\tau\rangle - \langle\phi_\nu\phi_\lambda|r_{12}^{-1}|\phi_\tau\phi_\mu\rangle)$
    - suggested MQC routines:
      - **output = contraction(eris,integral)**
  - Form Fock matrix
    - $\mathbf{F} = \mathbf{h} + \mathbf{G}(\mathbf{P})$
    - suggested MQC routines:
      - summation
  - Compute energy at iteration.
    - $E = \frac{1}{2}\sum_{\mu\nu}^{N_{\text{basis}}} P_{\mu\nu}(h_{\mu\nu} + F_{\mu\nu}) + V_{nn}$
    - suggested MQC routines:
      - summation

      - **output = contraction(integralA,integralB)**

- Orthogonalize Fock basis.
  - $\mathbf{F}' = \mathbf{X}^\dagger \mathbf{F} \mathbf{X}$
  - suggested MQC routines:
    - **integralOut = matmul(integralA,integralB)**
    - **integralOut = dagger(integral)**
- Diagonalize Fock matrix.
  - $\mathbf{F}'\mathbf{C}' = \boldsymbol{\varepsilon}\mathbf{C}'$
  - suggested MQC routines:
    - **call mqc_scf_integral%diag(eVals,eVecs)**
- Back-transform MO coefficients.
  - $\mathbf{C} = \mathbf{X}\mathbf{C}'$
  - suggested MQC routines:
    - **integralOut = matmul(integralA,integralB)**
- Form density matrix.
  - $\mathbf{P} = \mathbf{C}_{occ}\mathbf{C}_{occ}^\dagger$
  - remember to save old density matrix
  - suggested MQC routines:
    - **integralOut = mqc_scf_integral%orbitals('occupied',[nAlpha],[nBeta])**
    - **integralOut = matmul(integralA,integralB)**
    - **integralOut = dagger(integral)**
- Test convergence and exit or iterate.
  - norm($\mathbf{P} - \mathbf{P}_{old}$)
  - suggested MQC routines:
    - subtraction
    - **norm = mqc_scf_integral%norm(methodIn)**

Guidelines

To get you started, below is a partially completed code provided with comments to indicate the missing sections. The subroutines and functions you require are referred to in the comment, and the exact format of these functions and subroutines can be found in the section above. For each set of comments, look up the equation in the section above and consider the variables you have available and how the routines can perform the operations in the equation. All the variables you need have already been declared in the code, so you do not need to declare your own variables. You should be able to identify the best variable to use based on the name of the variable. Note that the wavefunc object contains several important variables, and is explained in more detail in the sections above.

You should obtain a matrix file from Gaussian of H_2 with STO-3G basis set (this is a good test model because it has only two basis functions and symmetry so it is easy to check your numbers agree). After compiling, you can run the code with the command "hartreefock.exe -f h2-sto3g.mat". Remember to recompile and test your code frequently to make sure it is doing what you expect. When you are trying to get your code to work, use the density matrix read off the matrix file that is already converged as your initial density matrix. The results in the first iteration should then match the values on the matrix file so you can debug easily. The data from the matrix file is stored in the wavefunc object and in the code below is printed on line 41.

```fortran
 1 program hartreefock
 2
 3 use mqc_gaussian
 4 use iso_fortran_env
 5
 6 implicit none
 7 character(len=:), allocatable :: command,fileName
 8 type(mqc_gaussian_unformatted_matrix_file)::matFile
 9 integer(kind=int64)::iOut=output_unit,iIn=input_unit,iPrint=0,i,j,nElec,nAlpha,nBeta,nBasis, &
10  iter=1,max_iter=256,nBasUse,multi
11 type(mqc_scalar)::conver,thresh
12 type(mqc_wavefunction)::wavefunc
13 type(mqc_molecule_data)::moleculeInfo
14 type(mqc_twoERIs),dimension(1)::eris
15 type(mqc_scalar)::Vnn,Energy,half
16 type(mqc_scf_integral)::Gmat,Fock,Xmat,old_density
17
18 half = 0.5
19 thresh = 1.0e-8
20
21  j = 1
22 do i=1,command_argument_count()
23     if(i.ne.j) cycle
24     call mqc_get_command_argument(i,command)
25     if(command.eq.'-f') then
26 !
27 !*      -f matrix_file          Input matrix file with initial set of molecular orbitals.
28 !*
29     call mqc_get_command_argument(i+1,fileName)
30      j = i+2
```

```fortran
    else
        call mqc_error_A('Unrecognised input flag',6, 'command',command)
    endIf
    deallocate(command)
endDo
!
!      Recover required data from matrix file.
!
call matFile%load(fileName)
call matFile%getESTObj('wavefunction',wavefunc)
call wavefunc%print(iOut,'all')
call matFile%getMolData(moleculeInfo)
call matFile%get2ERIs('regular',eris(1))
call eris(1)%print(iOut,'AO 2ERIs')
nElec = wavefunc%nElectrons%ival()
nAlpha = wavefunc%nAlpha%ival()
nBeta = wavefunc%nBeta%ival()
nBasis = wavefunc%nbasis%ival()
!
!      Compute the nuclear-nuclear repulsion energy.
!
Vnn = mqc_get_nuclear_repulsion(moleculeInfo)
call moleculeInfo%print(iOut)
call Vnn%print(iOut,'Nuclear Repulsion Energy (au)')
!
!       Initialize the density matrix here (to debug you can use the already-converged density
!       matrix from Gaussian stored in wavefunc%density_matrix and check convergence is
!       achieved in a single step. Otherwise initialize the density matrix to the zero matrix
!       using wavefunc%density_matrix%init routine. The wavefunc%density_matrix is the
!       variable to use to store the density matrix.
!
!
!       Get the othogonalization matrix X using the mqc_scf_transformation_matrix routine.
!       You should store the X matrix in the Xmat variable.
!
!
!       Here we start doing the iterations.
!
do while (iter .le. max_iter)
    write(iOut,'(A,I3)') 'Iteration #: ', iter
!
!       Form G matrix using contraction of eris and density matrix using contraction routine.
!       Use the Gmat variable to store the G matrix.
!
!
!       Form the Fock matrix by adding the core Hamitonian to the G matrix. Print it out and
!       if  you are using the density matrix from the Gaussian file,  check it  agrees with the Fock
!       matrix on the Gaussian file  (you can print  it  out using
!       call  wavefunc%fock_matrix%print(iOut,'Fock matrix on Gaussian file') before overwriting
!        it ).  Use the wavefunc%fock_matrix variable to store the Fock matrix.
!
!
!       Compute the energy of the current iteration.  Save the energy in the Energy variable.
!
!
!       Orthogonalize the Fock matrix using matmul and dagger routines.
!
!
!       Diagonalize the Fock matrix using the diag routine and save the MO coefficients into
!       wavefunc%MO_energies and wavefunc%MO_coefficients.
!
!
!        Back-transform the MO coefficients to the nonorthogonal atomic orbital basis
!        using matmul.
!
!
!        Save the old density matrix to old_density and form the new density matrix. You will
!        need to extract just the occupied orbitals from the density matrix, which can be done
!        using wavefunc%MO_coefficients%orbitals('occupied',[nAlpha],[nBeta]). Then use
!        matmul and dagger routines to construct the new density matrix.
!
!
!        Test convergence and exit or  iterate .  Store the convergence in conver variable.  The
!         convergence can be obtained by using the mqc_integral_norm routine on the
!          difference  between the current and old density matrices.  Compare your conver
!          variable  to the thresh  variable  and exit  if  you are comverged (conver.lt.thresh).  If  you
!         have reached the maximum number of iterations, exit the program with an error.
!         Otherwise, increment the iteration  number by one and redo the loop.
!
     conver = mqc_integral_norm((wavefunc%density_matrix-old_density),'F')
      call  conver%print(iOut,'Convergence on density matrix')
      if(conver.le.thresh)  exit
      if( iter .eq.max_iter)  call  mqc_error('Maximum number of iterations reached')
      iter  = iter+1

endDo

call  energy%print(iOut,'Final energy')

end program hartreefock
```