CHEM 225 Name: _____

Programming Exercise 3

Complete the following problems.
This assignment is due by 11:59 pm on April 22, 2019.

These programming exercises build on concepts developed in previous programming exercises. You should (always!) feel free to reuse code from your previous work.

1. **PROGRAMMING EXERCISE** Write and compile a program named `pgrm_03_01.exe` that reads a symmetric matrix stored in a user-provided input file and forms the inverse square-root of the matrix returned in full N×N storage. After forming the inverse square-root matrix, the program should form the product of the inverse square-root with itself and the input matrix using the intrinsic function `MatMul` to verify it is correct (which should yield the identity matrix).

   The program should rely on the LAPack routine `SSPEV`. Note that `SSPEV` requires a symmetric packed-storage input matrix. The input matrix may be overwritten during the diagonalization process; therefore, it is important that you copy the matrix to a temporary array before calling `SSPEV`.

   An example main program for this exercise is:

```
      program pgrm_03_01
!
!     This program computes the inverse square-root of a matrix.
!
!     At execution time, the program expects 2 command line arguments: (1) nDim;
!     and (2) the matrix being raised to the (1/2) power.
!
!
      implicit none
      integer,parameter::unitIn=10
      integer::i,iError,nDim,lenSym
      real,dimension(:),allocatable::inputSymMatrix
      real,dimension(:,:),allocatable::inputSqMatrix,invSqrtInputMatrix
      character(len=256)::cmdlineArg
!
!
!     Begin by reading the leading dimension of the matrix and the input file
!     name from the command line. Then, open the file and read the input matrix,
!     inputSymMatrix
!
      call Get_Command_Argument(1,cmdlineArg)
      read(cmdlineArg,'(I)') nDim
      lenSym = (nDim*(nDim+1))/2
      allocate(inputSymMatrix(lenSym),inputSqMatrix(nDim,nDim),  &
        invSqrtInputMatrix(nDim,nDim))
      call Get_Command_Argument(2,cmdlineArg)
      open(Unit=unitIn,File=TRIM(cmdlineArg),Status='OLD',IOStat=iError)
```

```
      if(iError.ne.0) then
        write(*,*)' Error opening input file.'
        STOP
      endIf
      do i = 1,lenSym
        read(unitIn,*) inputSymMatrix(i)
      endDo
      close(Unit=unitIn)
!
!     Form the square-root of inputSymMatrix. The result is loaded into square
!     (full storage) matrix invSqrtInputMatrix.
!
      write(*,*)' The matrix loaded (column) upper-triangle packed:'
      call SymmetricPacked2Matrix_UpperPacked(nDim,inputSymMatrix,  &
        inputSqMatrix)
      write(*,*)' Input Matrix:'
      call Print_Matrix_Full_Real(inputSqMatrix,nDim,nDim)
      call InvSQRT_SymMatrix(nDim,inputSymMatrix,invSqrtInputMatrix)
      write(*,*)' Inverse SQRT Matrix:'
      call Print_Matrix_Full_Real(invSqrtInputMatrix,nDim,nDim)
      write(*,*)' Matrix product that should be the identity.'
      call Print_Matrix_Full_Real(MatMul(MatMul(invSqrtInputMatrix,  &
        invSqrtInputMatrix),inputSqMatrix),nDim,nDim)
!
      end program pgrm_03_01
```

Two example input files (`input_03_1.txt` and `input_03_2.txt`) are provided in your GitHub repositories. Output files, with the `out` extension, are provided in the same directory.

2. **PROGRAMMING EXERCISE** Write and compile a program named `pgrm_03_02.exe` that reads a symmetric square matrix given in full-storage form from a user-provided input file and *symmetrizes* and *linearizes* to upper/column storage form of the matrix. This scheme will begin with a matrix such as

$$\begin{pmatrix} 4 & 2 & 1 \\ 2 & 6 & 4 \\ 1 & 4 & 3 \end{pmatrix}$$

and converts it to a rank-1 array as

$$\begin{pmatrix} 4 \\ 2 \\ 6 \\ 1 \\ 4 \\ 3 \end{pmatrix}$$

An example main program for this exercise is:

```
      program pgrm_03_02
!
!     This program symmetrizes a square matrix and reports the results as a
!     linearized array (vector).
!
!     At execution time, the program expects 2 command line arguments: (1) nDim,
!     the leading dimension of the square matrix; and (2) an input file
!     containing the matrix elements in full storage form.
!
!
      implicit none
      integer,parameter::unitIn=10
      integer::i,j,iError,nDim,lenSym
      real,dimension(:),allocatable::symMatrix
      real,dimension(:,:),allocatable::sqMatrix
      character(len=256)::cmdlineArg
!
!
!     Begin by reading the leading dimension of the matrix and the input file
!     name from the command line. Then, open the file and read the input matrix,
!     sqMatrix
!
      call Get_Command_Argument(1,cmdlineArg)
      read(cmdlineArg,'(I)') nDim
      lenSym = (nDim*(nDim+1))/2
      allocate(sqMatrix(nDim,nDim),symMatrix(lenSym))
!
      call Get_Command_Argument(2,cmdlineArg)
      open(Unit=unitIn,File=TRIM(cmdlineArg),Status='OLD',IOStat=iError)
      if(iError.ne.0) then
        write(*,*)' Error opening input file.'
        STOP
      endIf
      do i = 1,nDim
        do j = 1,nDim
          read(unitIn,*) sqMatrix(i,j)
        endDo
      endDo
      close(Unit=unitIn)
!
!     Print the input matrix.
!
      write(*,*)' Input Matrix:'
      call Print_Matrix_Full_Real(sqMatrix,nDim,nDim)
      call Sq2SymMatrix(nDim,sqMatrix,symMatrix)
      write(*,*)
      write(*,*)' Output Matrix:'
      do i = 1,lenSym
```

```
        write(*,*) symMatrix(i)
      endDo
!
      end program pgrm_03_02
```

Two example input files (`input_02_3.txt` and `input_02_4.txt`) are provided in your GitHub repositories. Output files, with the `out` extension, are provided in the same directory.

3. **PROGRAMMING EXERCISE** Write and compile a program named `pgrm_03_03.exe` that computes the one-electron energy contribution, two-electron energy contribution, and the density matrix from a Hartree-Fock program. As input, the program will be given the number of electrons (the program will assume closed-shell and compute the number of occupied molecular orbitals), number of basis functions, and names for files that provide the core Hamiltonian, overlap, and Fock matrices from a converged calculation. These files will be provided by the instructor (*vide infra*).

   After reading in the provided data, the program should form the inverse-square-root of the overlap matrix, $\mathbf{S}^{-1/2}$, and transformed Fock matrix based on the symmetric decomposition of the atomic orbital basis, $\widetilde{\mathbf{F}} = \mathbf{S}^{-1/2}\mathbf{F}\mathbf{S}^{-1/2}$.

   Using LAPack routine `SSPEV`, the eigenvectors of $\widetilde{\mathbf{F}}$ can be found and then transformed to the canonical MO coefficient matrix.

   Once the canonical MOs are known, the program print the MO energies and then form the density matrix in the atomic orbital basis and print it.

   Then, the program should evaluate and print the one-electron, $\langle \mathbf{PH} \rangle$, and two-electron, $\frac{1}{2}\langle \mathbf{PG} \rangle$, contributions to the total electronic energy. Note that $\mathbf{G} = \mathbf{F} - \mathbf{H}$.

   Finally, the program should compute and print the contraction of the density and overlap matrices, $\langle \mathbf{PS} \rangle$. Note that this result should equal the number of electrons in the system.

   Two sets of input files (`input_03_5_*.txt` and `input_03_6_*.txt`, where the *'s are h, s, and f for the three input matrices and *=ABOUT has text giving information about the input jobs include number of electrons and number of basis functions) are provided in your GitHub repositories. Output files, with the `out` extension, are provided in the same directory.

   To assist with your writing of this program, here are a few code snippets from a working code.

   The top of a working version of this program could look like:

```
      program pgrm_03_03
!
!     This program computes components of the Hartree-Fock energy and the number
!     of electrons using contraction of various matrices loaded from
!     user-provided files (that presumably come from Gaussian calculations).
!
!     At run time, the program expects 5 command line arguments:
!       (1) the number of electrons;
!       (2) the number of atomic orbital basis functions;
!       (3) an input file containing core-Hamiltonian matrix elements (in
```

```
!           symmetric upper/column storage form);
!       (4) an input file containing the overlap matrix elements (in
!           symmetric upper/column storage form); and
!       (5) an input file containing Fock matrix elements (in symmetric
!           upper/column storage form).
!
!     At run time, the program outputs 6 items:
!       (1) the MO energies;
!       (2) the MO coefficients;
!       (3) the density matrix;
!       (4) the one-electron contribution to the total electronic energy;
!       (5) the two-electron contribution to the total electronic energy; and
!       (6) the tr(PS), which should equal the number of electrons.
!
!
!
      implicit none
      integer,parameter::unitIn=10
      integer::i,iError,nElectrons,nOcc,nBasis,lenSym
      real::oneElectronEnergy,twoElectronEnergy,tracePS
      real,dimension(:),allocatable::symFock,symCoreHamiltonian,symOverlap,  &
        moEnergies,tempSymMatrix
      real,dimension(:,:),allocatable::sqFock,fockTilde,sqCoreHamiltonian,  &
        InvSqrtOverlap,moCoefficients,densityMatrix,tempSqMatrix,  &
        SSPEV_Scratch
      character(len=256)::cmdlineArg
!
!
!     Begin by reading the number of basis functions, allocating array memory,
!     and loading the symmetric Fock and overlap matrices from input files
!     provided on the command line.
!
      call Get_Command_Argument(1,cmdlineArg)
      read(cmdlineArg,'(I)') nElectrons
      nOcc = nElectrons/2
!
      call Get_Command_Argument(2,cmdlineArg)
      read(cmdlineArg,'(I)') nBasis
      lenSym = (nBasis*(nBasis+1))/2
      allocate(symFock(lenSym),symCoreHamiltonian(lenSym),  &
        symOverlap(lenSym),tempSymMatrix(lenSym))
      allocate(moEnergies(nBasis))
      allocate(sqFock(nBasis,nBasis),fockTilde(nBasis,nBasis),  &
        sqCoreHamiltonian(nBasis,nBasis),invSqrtOverlap(nBasis,nBasis),  &
        moCoefficients(nBasis,nBasis),densityMatrix(nBasis,nBasis),  &
        tempSqMatrix(nBasis,nBasis),SSPEV_Scratch(nBasis,3))
```

Using routines written earlier, the following is a bit of code that can form $\mathbf{S}^{-1/2}$ and $\widetilde{\mathbf{F}}$. This

code snippet also demonstrates the use of the fortran intrinsic function `Reshape`, which can be used to have vectors treated as a single column or row in a matrix, etc.

```
!
!     Form the square-root of the overlap matrix.
!
      call InvSQRT_SymMatrix(nBasis,symOverlap,invSqrtOverlap)
!
!     Form fTilde and solve for the MO energies and coefficients.
!
      call SymmetricPacked2Matrix_UpperPacked(nBasis,symFock,sqFock)
      tempSqMatrix = MatMul(invSqrtOverlap,sqFock)
      fockTilde = MatMul(tempSqMatrix,invSqrtOverlap)
      call Sq2SymMatrix(nBasis,fockTilde,tempSymMatrix)
      call SSPEV('V','U',nBasis,tempSymMatrix,moEnergies,  &
        moCoefficients,nBasis,SSPEV_Scratch,iError)
      If(iError.ne.0) then
        write(*,*)' Failure in SSPEV.'
        STOP
      endIf
      write(*,*)' MO Energies:'
      call Print_Matrix_Full_Real(Reshape(moEnergies,[nBasis,1]),nBasis,1)
      tempSqMatrix = moCoefficients
      moCoefficients = MatMul(invSqrtOverlap,tempSqMatrix)
      write(*,*)' MO Coefficients:'
      call Print_Matrix_Full_Real(moCoefficients,nBasis,nBasis)
```

Running the program with the command

```
./pgrm_03_03.exe 2 2 input_03_4_h.txt input_03_4_s.txt input_03_4_f.txt
```

should yield the output:

```
 The matrix loaded (column) upper-triangle packed:
 Input Matrix:
                1            2            3
     1      4.000000     2.000000     1.000000
     2      2.000000     6.000000     4.000000
     3      1.000000     4.000000     3.000000
 Inverse SQRT Matrix:
                1            2            3
     1      0.546412    -0.164918     0.086995
     2     -0.164918     1.002827    -0.894685
     3      0.086995    -0.894685     1.589114
 Matrix product that should be the identity.
                1            2            3
     1      0.999999     0.000001     0.000000
     2      0.000000     0.999996     0.000002
     3     -0.000001     0.000003     0.999995
```