Computers in Chemistry – Lecture X

Prof. Dr. Stephan Irle
Quantum Chemistry Group
Nagoya University

Get this lecture online

- Please go to: http://qc.chem.nagoya-u.ac.jp
- Click on "Teaching"
- Click on "PPT" link of "10.1 Lecture X – Linear Algebra in FORTRAN"

  userid: qcguest, password: qcigf!

9.1 Lecture IX - SUBROUTINES and Arrays in FORTRAN (PDF)
9.2 Example programs: temp3.f90, scalarp.f90 (Scalar Product)
10.1 Lecture X - Linear Algebra in FORTRAN (PDF)
10.1 Solutions to Assignment 6: classcode.f90, quadratic2.f90, quadratic3.f90
10.2 Assignment 7 (PDF)
10.3 Example programs: debug.f90, matvec.f90, matmul.f90, degrad.f90

8.1 Arrays

- An array is a **data structure** with elements of same type and same name
- Arrays can be 1-dimensional (vector), 2-dimensional (matrix), or multi-dimensional

```
REAL, DIMENSION(2) :: Vector, Rotated_Vector
REAL, DIMENSION(2,2) :: Rotation_Matrix
```

- Arrays have to be declared in "type statements". These statements specify their dimensions (sizes)
- Arrays are often used in vector calculus, linear algebra, data processing, etc.
- Ideal for computation, especially in combination with counter-controlled DO loops

8.2 Compile-Time Arrays and Run-Time Arrays I

- In the previous example of array type definitions, memory for the 1D arrays "Vector", "Rotated_Vector", and the 2D array "Rotation_Matrix" was allocated at compile time (the dimensions were 2 and 2x2, respectively).
- "Compile-Time Arrays" (CTAs) are older elements of FORTRAN than Run-Time Arrays (RTAs), which were introduced in the FORTRAN 90 standard.
- CTAs are inflexible and may waste computer memory. Nevertheless, they continue to exist in older codes, and are still widely used.
- We will nevertheless first discuss CTAs.

8.2 Compile-Time Arrays and Run-Time Arrays II

- There are two alternative ways to specify CTAs:

```
REAL, DIMENSION(2) :: Vector, Rotated_Vector
REAL, DIMENSION(2,2) :: Rotation_Matrix
```

or

```
REAL :: Vector(2), Rotated_Vector(2)
REAL :: Rotation_Matrix(2,2)
```

- The dimensions can be specified via previously defined parameters, as shown next.

8.2 Compile-Time Arrays and Run-Time Arrays III

- Example for the use of an integer parameter in specifying the dimensions of arrays:

```
INTEGER, PARAMETER :: NDim = 2
REAL, DIMENSION(NDim) :: Vector, Rotated_Vector
REAL, DIMENSION(NDim,NDim) :: Rotation_Matrix
```

or

```
INTEGER, PARAMETER :: NDim = 2
REAL :: Vector(NDim), Rotated_Vector(NDim)
REAL :: Rotation_Matrix(NDim,NDim)
```

- Changing the code from 2D to 3D, for example, is simple, since we only have to change the value in a single place (NDim in the PARAMETER statement).

8.2 Compile-Time Arrays and Run-Time Arrays IV

- Run-Time Arrays (RTAs) require a declaration statement:

```
REAL, DIMENSION (:), ALLOCATABLE :: Vector, Rotated_Vector
REAL, DIMENSION(:,:), ALLOCATABLE  :: Rotation_Matrix
```

and an ALLOCATION statement:

```
READ *, N
ALLOCATE (Vector(N), Rotated_Vector(N), STAT = AllocateStatus)
IF (AllocateStatus /= 0) STOP "*** NOT ENOUGH MEMORY ***"
ALLOCATE (Rotation_Matrix(N,N), STAT = AllocateStatus)
IF (AllocateStatus /= 0) STOP "*** NOT ENOUGH MEMORY ***"
```

8.2 Compile-Time Arrays and Run-Time Arrays V

- In the previous example, N and AllocateStatus are INTEGER variables that have to be declared in the type declaration section:

```
INTEGER :: N, AllocateStatus
```

- If memory is available, the value of "AllocateStatus" will be 0. Otherwise, a non-zero value will be returned.
- RTAs need to be deallocated when no longer needed, using the DEALLOCATE Statement:

```
DEALLOCATE (Vector, Rotated_Vector, STAT = AllocateStatus)
DEALLOCATE (Rotation_Matrix, STAT = AllocateStatus)
```

- AllocateStatus is 0 if successful, otherwise non-zero.

8.2 Working with Arrays I

- Consider the following matrix-vector product:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

- If A is a unitary matrix (AA* = A*A = 1, A* is the conjugate transpose of A), this operation corresponds to a rotation of vector **X** to give vector **Y**.

8.2 Working with Arrays II

- Read Matrix **A** and vector **X**:

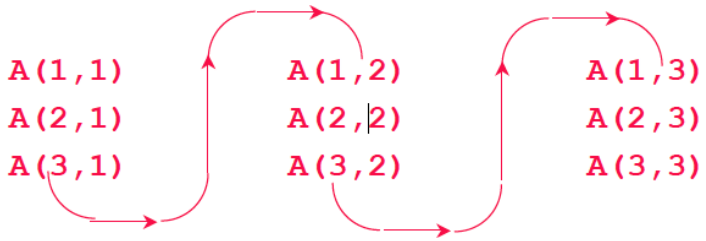
```
INTEGER, PARAMETER :: N = 3      ! Dimension or vector space
INTEGER :: I, J                  ! Counter variables
REAL, DIMENSION (N) :: X, Y      ! Declaration of vectors X, Y
REAL, DIMENSION (N,N) :: A       ! Declaration of matrix X
READ *, A   ! Will read all 9 elements of 2D array A in column order
READ *, X   ! Will read all 3 elements of 1D array X
```

- FORTRAN stores data in 2D arrays by increasing columns first. In the previous example, the following order is read:

  A(1,1), A(2,1), A(3,1), …

8.2 Working with Arrays III

- You can check the order by PRINT *, A



- Memory storage works the same way! If we program, we want to consider this since "jumping" by processing rows first, then columns, decreases the efficiency of the code

8.2 Working with Arrays IV

- To print the matrix in N lines, use:

```
DO I = 1, N
    PRINT *, (A(I,J), J = 1, N)
END DO
```

- In this way, first the rows will be printed, then the columns
- In FORTRAN 90, an array can be processed as a single object!

```
! Print Vector Y
PRINT *, Y
```

8.2 Working with Arrays V

- The matrix-vector product is typically performed as follows:

```
Y = 0. !  Clears accumulator
DO I = 1,3
    DO J = 1,3
        Y(I) = Y(I) + A(I,J)*X(J)
    END DO
END DO
PRINT *, Y
```

- Download program “matvec.f90” from the qc.chem.nagoya-u.ac.jp webpage, compile, and run

8.X Conversion of degree to radians

- FORTRAN’s trigonometric functions (sin, cos, tan) assume that the argument is in units of rad, NOT degree!
- Conversion of angles from degree to radians:

```
PI = 3.141592653589793
RAD = DEG/360.0 * 2*PI
```

- Download program “degrad.f90” from the qc.chem.nagoya-u.ac.jp webpage, compile, and run
- Practice: write a program that can convert radians back to degree!
- This concludes today’s lecture.