Outline:
- Fortran modules
- Newton's method example

Reading:
- class notes: Fortran modules
- class notes: Subroutine in a module
- class notes: Fortran example for Newton's method

Fortran modules

General structure of a module:

```
module <MODULE-NAME>
! Declare variables
contains
! Define subroutines or functions
end module <MODULE-NAME>
```

A program/subroutine/function can use this module:

```
program <NAME>
  use <MODULE-NAME>
! Declare variables
! Executable statements
end program <NAME>
```

Can also specify a list of what variables/subroutines/functions from module to be used.

Similar to `from numpy import linspace` rather than `from numpy import *`

```
program <NAME>
  use <MODULE-NAME>, only: <LIST OF SYMBOLS>
! Declare variables
! Executable statements
end program <NAME>
```

Makes it easier to see which variables come from each module.
Fortran module example

```
module subm
contains
    subroutine sub1()
        print *, "In sub1"
    end subroutine sub1
end module subm
```

```
program demo
    use subm, only: sub1
    print *, "In main program"
    call sub1()
end program demo
```

Notes:

Fortran modules

Some uses:

- Can define global variables in modules to be used in several different routines.
  
  In Fortran 77 this had to be done with common blocks — much less elegant.

- Subroutine/function interface information is generated to aid in checking that proper arguments are passed.
  
  It's often best to put all subroutines and functions in modules for this reason.

- Can define new data types to be used in several routines. ("derived types" rather than "intrinsic types")

Notes:

Compiling Fortran modules

If sublm.f90 is a module, then compiling it creates sublm.o and also sublm.mod:

```
$ gfortran -c sublm.f90

$ ls main.f90 sublm.f90 sublm.mod sublm.o
```

the module must be compiled before any subroutine or program that uses it!

```
$ rm -f sublm.mod
$ gfortran main.f90 sublm.f90
main.f90:5.13:
    use sublm
    1
    Fatal Error: Can’t open module file ‘sublm.mod’ for reading at (1): No such file or directory
```
Another module example

```fortran
module circle_mod
  implicit none
  real(kind=8), parameter :: pi = 3.141592653589793d0
contains
  real(kind=8) function area(r)
    real(kind=8), intent(in) :: r
    area = pi * r**2
  end function area
  real(kind=8) function circumference(r)
    real(kind=8), intent(in) :: r
    circumference = 2.0d0 * pi * r
  end function circumference
end module circle_mod
```

Running this gives:

```
pi = 3.14159265358979
area for a circle of radius 2: 12.5663706143592
```

Notes:

```
Module variables

```

```fortran
module circle_mod
  implicit none
  real(kind=8) :: pi save
contains
  real(kind=8) function area(r)
    real(kind=8), intent(in) :: r
    area = pi * r**2
  end function area
  real(kind=8) function circumference(r)
    real(kind=8), intent(in) :: r
    circumference = 2.0d0 * pi * r
  end function circumference
end module circle_mod
```

Notes:
The module variable \( \pi \) should be initialized in a program unit that is called only once.

It can be initialized to full machine precision using

\[ \pi = \arccos(-1) \]
Fortran subroutines

A version that takes an array as input and squares each value:

```fortran
1 ! $UNPSC/codes/fortran/sub2.f90
2 program sub2
3 implicit none
4 real(kind=8), dimension(:) :: y, z
5 integer n
6 y = (/2., 3., 4./)
7 n = size(y)
8 call fsub(y, n, z)
9 print *, "z = " , z
10 end program sub2
11
12 subroutine fsub(y, n, f)
13 ! compute f(x) = x**2 for all elements of the array y
14 ! of length n.
15 implicit none
16 integer, intent(in) :: n
17 real(kind=8), dimension(n), intent(in) :: x
18 real(kind=8), dimension(n), intent(out) :: f
19 f = x**2
20 end subroutine fsub
```

Module version — creates an interface

Now do not need to pass the value n into the subroutine.

```fortran
1 ! $UNPSC/codes/fortran/sub3.f90
2 module subroutine
3 contains
4 subroutine fsub(y, f)
5 ! compute f(x) = x**2 for all elements of the array y.
6 implicit none
7 real(kind=8), dimension(:), intent(in) :: x
8 real(kind=8), dimension(size(x)), intent(out) :: f
9 f = x**2
10 end subroutine fsub
11 end module subroutine
12
13 program sub3
14 use subroutine
15 implicit none
16 real(kind=8), dimension(size(y)) :: y, z
17 y = (/2., 3., 4./)
18 call fsub(y, z)
19 print *, "z = " , z
20 end program sub3
```

Module for Newton’s method

See the class notes: Fortran example for Newton’s method