# AMath 483/583 — Lecture 10

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>
```

# Fortran modules

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

```
1   ! $UWHPSC/codes/fortran/multifile2/sub1m.f90
2
3   module sub1m
4
5   contains
6
7   subroutine sub1()
8       print *, "In sub1"
9   end subroutine sub1
10
11  end module sub1m
```

```
1   ! $UWHPSC/codes/fortran/multifile2/main.f90
2
3   program demo
4       use sub1m, only: sub1
5       print *, "In main program"
6       call sub1()
7   end program demo
```

# 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")

# Compiling Fortran modules

If `sub1m.f90` is a module, then compiling it creates `sub1m.o` and also `sub1m.mod`:

```
$ gfortran -c sub1m.f90


$ ls
main.f90     sub1m.f90    sub1m.mod    sub1m.o
```

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

```
$ rm -f sub1m.mod
$ gfortran main.f90 sub1m.f90
main.f90:5.13:

    use sub1m
            1
Fatal Error: Can't open module file 'sub1m.mod'
for reading at (1): No such file or directory
```

# Another module example

```
1  ! $UWHPSC/codes/fortran/circles/circle_mod.f90
2
3  module circle_mod
4
5      implicit none
6      real(kind=8), parameter :: pi = 3.1415926535897793d0
7
8  contains
9
10     real(kind=8) function area(r)
11         real(kind=8), intent(in) :: r
12         area = pi * r**2
13     end function area
14
15     real(kind=8) function circumference(r)
16         real(kind=8), intent(in) :: r
17         circumference = 2.d0 * pi * r
18     end function circumference
19
20 end module circle_mod
```

# Another module example

```
1  ! $UWHPSC/codes/fortran/circles/main.f90
2
3  program main
4
5      use circle_mod, only: pi, area
6      implicit none
7      real(kind=8) :: a
8
9      ! print parameter pi defined in module:
10     print *, 'pi = ', pi
11
12     ! test the area function from module:
13     a = area(2.d0)
14     print *, 'area for a circle of radius 2: ', a
15
16 end program main
```

Running this gives:

```
pi =     3.14159265358979
area for a circle of radius 2:     12.5663706143
```

# Module variables

```
1  ! $UWHPSC/codes/fortran/circles/circle_mod.f90
2  ! Version where pi is a module variable.
3
4  module circle_mod
5
6      implicit none
7      real(kind=8) :: pi
8      save
9
10 contains
11
12     real(kind=8) function area(r)
13         real(kind=8), intent(in) :: r
14         area = pi * r**2
15     end function area
16
17     real(kind=8) function circumference(r)
18         real(kind=8), intent(in) :: r
19         circumference = 2.d0 * pi * r
20     end function circumference
21
22 end module circle_mod
```

# Module variables

```
1  ! $UWHPSC/codes/fortran/circles/main.f90
2
3  program main
4
5      use circle_mod, only: pi, area
6      implicit none
7      real(kind=8) :: a
8
9      call initialize()   ! sets pi
10
11     ! print module variable pi:
12     print *, 'pi = ', pi
13
14     ! test the area function from module:
15     a = area(2.d0)
16     print *, 'area for a circle of radius 2: ', a
17
18 end program main
```

# Module variables

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

$$\pi = \arccos(-1)$$

```
1  ! $UWHPSC/codes/fortran/circles/initialize.f90
2
3  subroutine initialize()
4
5      ! Set the value of pi used elsewhere.
6      use circle_mod, only: pi
7      pi = acos(-1.d0)
8
9  end subroutine initialize
```

# Makefile

```
1  # $UWHPSC/codes/fortran/circles2/Makefile
2
3  OBJECTS = circle_mod.o \
4            main.o \
5            initialize.o
6
7  MODULES = circle_mod.mod
8
9  .PHONY: clean
10
11 output.txt: main.exe
12         ./main.exe > output.txt
13
14 main.exe: $(MODULES) $(OBJECTS)
15         gfortran $(OBJECTS) -o main.exe
16
17 %.o: %.f90
18         gfortran -c $<
19
20 %.mod: %.f90
21         gfortran -c $<
22
23 clean:
24         rm -f $(OBJECTS) $(MODULES) main.exe
```

# Fortran subroutines

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

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

# Module version — creates an interface

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

```
1  ! $UWHPSC/codes/fortran/sub3.f90
2
3  module sub3module
4
5  contains
6
7  subroutine fsub(x,f)
8    ! compute f(x) = x**2 for all elements of the array x.
9    implicit none
10   real(kind=8), dimension(:), intent(in) :: x
11   real(kind=8), dimension(size(x)), intent(out) :: f
12   f = x**2
13 end subroutine fsub
14
15 end module sub3module
16
17 !-----------------------------------------------
18
19 program sub3
20     use sub3module
21     implicit none
22     real(kind=8), dimension(3) :: y,z
23
24     y = (/2., 3., 4./)
25     call fsub(y,z)
26     print *, "z = ",z
27 end program sub3
```

# Module for Newton's method

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