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

Notes:

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>
```


Notes:

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.

Notes:

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
```


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 `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
```


Notes:

Another module example

```fortran
! $UWHPSC/codes/fortran/circles/circle_mod.f90

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.d0 * pi * r
    end function circumference

end module circle_mod
```


Notes:

Another module example

```fortran
! $UWHPSC/codes/fortran/circles/main.f90

program main

    use circle_mod, only: pi, area
    implicit none
    real(kind=8) :: a

    ! print parameter pi defined in module:
    print *, 'pi = ', pi

    ! test the area function from module:
    a = area(2.d0)
    print *, 'area for a circle of radius 2: ', a

end program main
```

Running this gives:

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


Notes:

Module variables

```fortran
! $UWHPSC/codes/fortran/circles/circle_mod.f90
! Version where pi is a module variable.

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.d0 * pi * r
    end function circumference

end module circle_mod
```


Notes:

Module variables

```
! $UWHPSC/codes/fortran/circles/main.f90

program main

    use circle_mod, only: pi, area
    implicit none
    real(kind=8) :: a

    call initialize()   ! sets pi

    ! print module variable pi:
    print *, 'pi = ', pi

    ! test the area function from module:
    a = area(2.d0)
    print *, 'area for a circle of radius 2: ', a

end program main
```


Notes:

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

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

```
! $UWHPSC/codes/fortran/circles/initialize.f90

subroutine initialize()

    ! Set the value of pi used elsewhere.
    use circle_mod, only: pi
    pi = acos(-1.d0)

end subroutine initialize
```


Notes:

Makefile

```
# $UWHPSC/codes/fortran/circles2/Makefile

OBJECTS = circle_mod.o \
          main.o \
          initialize.o

MODULES = circle_mod.mod

.PHONY: clean

output.txt: main.exe
        ./main.exe > output.txt

main.exe: $(MODULES) $(OBJECTS)
        gfortran $(OBJECTS) -o main.exe

%.o: %.f90
        gfortran -c $<

%.mod: %.f90
        gfortran -c $<

clean:
        rm -f $(OBJECTS) $(MODULES) main.exe
```


Notes:

Fortran subroutines

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

```
! $UWHPSC/codes/fortran/sub2.f90

program sub2
    implicit none
    real(kind=8), dimension(3) :: y,z
    integer n

    y = (/2., 3., 4./)
    n = size(y)
    call fsub(y,n,z)
    print *, "z = ",z
end program sub2

subroutine fsub(x,n,f)
  ! compute f(x) = x**2 for all elements of the array x
  ! of length n.
  implicit none
  integer, intent(in) :: n
  real(kind=8), dimension(n), intent(in) :: x
  real(kind=8), dimension(n), intent(out) :: f
  f = x**2
end subroutine fsub
```


Notes:

Module version — creates an interface

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

```
! $UWHPSC/codes/fortran/sub3.f90

module sub3module

contains

subroutine fsub(x,f)
  ! compute f(x) = x**2 for all elements of the array x.
  implicit none
  real(kind=8), dimension(:), intent(in) :: x
  real(kind=8), dimension(size(x)), intent(out) :: f
  f = x**2
end subroutine fsub

end module sub3module

!-----------------------------------------------

program sub3
    use sub3module
    implicit none
    real(kind=8), dimension(3) :: y,z

    y = (/2., 3., 4./)
    call fsub(y,z)
    print *, "z = ",z
end program sub3
```


Notes:

Module for Newton's method

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

Notes: