IOP Concise Physics

Introduction to Computational Physics for Undergraduates

Omair Zubairi and Fridolin Weber

Appendix A

Summary of Fortran features

This overview of Fortran 90/95 features is presented as a series of tables that illustrate the syntax and abilities of Fortran 90/95. Comparisons are made to similar features in the Fortran 77 language. The tables (A.1–A.3) show that Fortran 90/95 has significant improvements over Fortran 77 and matches or exceeds newer software capabilities found in C++ and Matlab for dynamic memory management, user defined data structures, matrix operations, operator definition and overloading,

Table A.1. This table shows array operations in programming constructs. Lower-case letters denote scalar elements or arrays, upper-case letters denote matrices or scalar elements of matrices.

Description	Equation	F90/F95 operation
Scalar plus scalar	$c = a \pm b$	$c = a \pm b$
Element plus scalar	$c_{jk} = a_{jk} \pm b$	$c = a \pm b$
Element plus element	$c_{jk} = a_{jk} \pm b_{jk}$	$c = a \pm b$
Scalar times scalar	$c = a \times b$	$c = a*b$
Element times scalar	$c_{jk} = a_{jk} \times b$	$c = a*b$
Element times element	$c_{jk} = a_{jk} \times b_{jk}$	$c = a*b$
Scalar divide scalar	$c = a/b$	$c = a/b$
Scalar divide element	$c_{jk} = a_{jk}/b$	$c = a/b$
Element divide element	$c_{jk} = a_{jk}/b_{jk}$	$c = a/b$
Scalar power scalar	$c = a^b$	$c = a**b$
Element power scalar	$c_{jk} = a_{jk}^b$	$c = a**b$
Element power element	$c_{jk} = a_{jk}^{b_{jk}}$	$c = a**b$
Matrix transpose	$C_{kj} = A_{jk}$	$C =$ `transpose`(A)
Matrix times matrix	$C_{ij} = \sum_k A_{ik} B_{kj}$	$C =$ `matmul`(A, B)
Vector dot vector	$c = \sum_k A_k B_k$	$c =$ `sum`$(A*B)$
		$c =$ `dot_product`(A, B)

doi:10.1088/978-1-6817-4896-2ch8

Table A.2. Fortran features to include intrinsic data types, relational operators, and flow control statement.

Description	F77	F90/F95
Comment syntax	C,*	!
byte	character	character::
integer	integer	integer::
single precision	real	real::
double precision	double precision	real*8::
complex	complex	complex::
argument	parameter	parameter::
pointer	–	pointer::
structure	–	type::
Equal to	.EQ.	==
Not equal to	.NE.	/=
Less than	.LT.	<
Less or equal	.LE.	<=
Greater than	.GT.	>
Greater or equal	.GE.	>=
Logical NOT	.NOT.	.NOT.
Logical AND	.AND.	.AND.
Logical inclusive OR	.OR.	.OR.
Logical exclusive OR	.XOR.	.XOR.
Logical equivalent	.EQV.	.EQV.
Logical not equivalent	.NEQV.	.NEQV.
Conditionally execute statements	if	if
	end if	end if
Loop a specific number of times	do # k=1,n	do k=1,n
	# continue	end do
Loop an indefinite number of times	–	do while
	–	end do
Terminate and exit loop	go to	exit
Skip a cycle of loop	go to	cycle
Display message and abort	stop	stop
Return to invoking function	return	return
Conditional array action	–	where
Conditional alternative statements	else	else
	elseif	elseif
Conditional array alternatives	–	elsewhere
Conditional case selections	if	select case
	end if	end select

intrinsics for vector and parallel processors, and the basic requirements for object-oriented programming. They are intended to serve as a condense quick reference guide for programming in Fortran 90/95 and for understanding programs developed by others.

Table A.3. Overview of Fortran 90 intrinsic functions. The names of the arguments specify their type (i.e. X = Real, DX = Double precision, IX = Integer, Z = Complex).

F90	Function type	Definition
SQRT(X)	Real	$\sqrt{X}$
DSQR(DX)	Double precision	$\sqrt{DX}$
ABS(X)	Real	$\|X\|$
EXP(X)	Real	e^X
DEXP(DX)	Double precision	e^{DX}
LOG(X)	Real	$\log_e X$
LOG10(X)	Real	$\log_{10} X$
IFIX(X)	Integer	Truncate X to an integer
AINT(X)	Real	Round number
NINT(X)	Real	Round X to an integer
FLOAT(X)	Real	Converts IX to real value
CEILING(X)	Real	Smallest integer $>X$
FLOOR(X)	Real	Largest integer $<X$
MOD(X,Y)	Real	Division remainder
CONJ(Z)	Real	Complex conjugate
IMAG(Z)	Real	Imaginary part
DBLE(X)	Double precision	Convert X to double precision
AMAX1(X,Y,...)	Real	Maximum of (X, Y, ...)
AMAX0(IX,IY,...)	Real	Maximum of (IX, IY, ...)
AMIN0(IX,IY,...)	Real	Minimum of (IX, IY, ...)
AMIN1(X,Y,...)	Real	Minimum of (X, Y, ...)
MIN0(IX,IY,...)	Integer	Minimum of (IX, IY, ...)
SIN(X)	Real	$\sin(X)$
COS(X)	Real	$\cos(X)$
TAN(X)	Real	$\tan(X)$
ASIN(X)	Real	$\arcsin(X)$
ACOS(X)	Real	$\arccos(X)$
ATAN(X)	Real	$\arctan(X)$
SINH(X)	Real	$\sinh(X)$
COSH(X)	Real	$\cosh(X)$
TANH(X)	Real	$\tanh(X)$

IOP Concise Physics

Appendix B

Plotting using Python

There are many plotting programs and software available to use. In this appendix, we show how to plot data using the Python programming language. Plotting data using Python is very useful and versatile and can be performed on multiple platforms The file extension for a Python program is `.py` (i.e. `filename.py`). To compile and run a Python program, simply type

```
> python filename.py                                    ↩
```

Figure B.1 shows a simple graph of some numerical data. The sample Python code which produced figure B.1 is shown below for your reference.

```
#! /usr/bin/python3

import numpy as np
import matplotlib.pyplot as plt

# import data:
fx = np.loadtxt('datafile.dat')

# reading in two columns:
plt.plot(fx[:,0],fx[:,1])

# setting horizontal axis
plt.xlim((0,5))

# setting vertical axis
plt.ylim((0,30))
```

doi:10.1088/978-1-6817-4896-2ch9

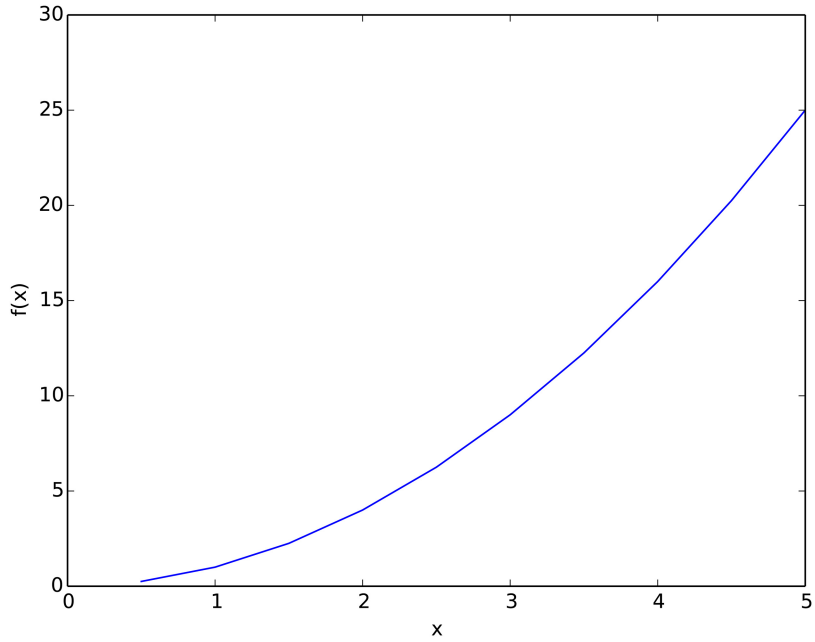

Figure B.1. Plot of the function $f(x) = x^2$.

```
# label axis
plt.xlabel('x')
plt.ylabel('f(x)')

# saving plot into a .pdf format
plt.savefig('graph.pdf')
plt.clf()
```

Note that the symobl '#' is used for comments. The sample code above gives the basics of plotting data; however, much more can be done using Python such as configuring axis, including a title, insert special symbols, etc. For example, in figure B.2, the axes are configured differently with more increments, there is a title for the graph, and we have inserted a special character on the vertical axis.

The sample code below gives a more detailed description on how to reproduce figure B.2.

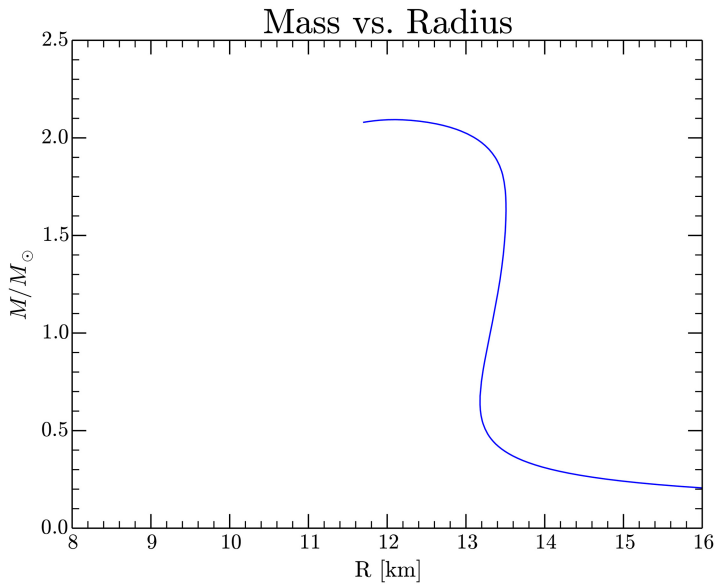

Figure B.2. Mass–radius plot for neutron stars.

```
#! /usr/bin/python

import numpy as np
import matplotlib.pyplot as plt

#import data
mr = np.loadtxt('mass_radius.dat')

#Note: The .dat file has 3 columns -- e_c, mass, radius
#but the columns are e_c = 0, mass = 1, radius = 2

#specify which columns of data are being imported
plt.plot(mr[:,2],mr[:,1])

#Note: ([x],[y])->([radius], [mass]) = ([:,2], [:,1])

#title of plot
plt.title('Mass vs. Radius',fontname='cmr10',fontsize=25)

#configure ticks:
plt.minorticks_on()
plt.tick_params(axis='both',which='minor',length=5,width=0.75,labelsize=15)
plt.tick_params(axis='both',which='major',length=10,width=1.0,labelsize=15)

#configure x-axis
plt.xlim((8,16))                          #range of x-values
plt.xlabel(r'R [km]', fontname='cmr10', fontsize=16)
plt.xticks(fontname='cmr10', fontsize=15, size=15)
```

```
#configure y-axis
plt.ylabel(r'$M/M_\odot$', fontname='cmr10', fontsize=16)
plt.yticks(fontname='cmr10', fontsize=15, size=15)

#save your plot
plt.savefig('mr.pdf')
plt.savefig('mr.eps')
plt.clf()
```

IOP Concise Physics

Omair Zubairi and Fridolin Weber

Appendix C

Fortran 90 sample program illustrating good programming

The following is a sample program which illustrates good Fortran 90 programming.

```
!* module
   module constants
   implicit none
   real, parameter :: pi=acos(-1.)     ! define pi
   real, parameter :: k_B =8.3145      ! k_B in joule/mol/K
   real, parameter :: m_N 2=28.        ! Molecular mass of N2 in g/mol
   real, parameter :: m_02=32.         ! Molecular mass of O2 g/mol
   real, parameter :: m_Ar=40.         ! Molecular mass of Ar g/mol
   end module constants

!* main program
   program boltzmann
!*
! Calculate the fraction of molecules in a thermally equilibrated gas
! (uniform temperature T) whose speed is less than a given speed v,
! given by the expression
!                                    v
! f(v,T) = 4*pi*(m/2/pi/k/T)^1.5 * I dv' exp(-m*v'^2/2/xk/T)*v'^2
!                                    0
!
! Performing the variable transformation v^2/kappa^2=y^2 with
! kappa=(2kT/m)^1/2 leads for f(v,T) to
!                             v/kappa
```

doi:10.1088/978-1-6817-4896-2ch10

```
!  f(v,t) = 4*pi/pi^3/2 * I dy y^2 exp(-y^2)
!                           0
! Given:
!  v    = Speed (m/sec)
!  m_N2 = Mole mass of nitrogen = 28 g/mol
!  m_O2 = Mole mass of oxygen = 32 g/mol
!  m_Ar = Mole mass of argon = 40 g/mol
!        (Air consists of 78%N2, 21% 02, 1% Ar)
!  k_B  = Boltzmann constant = 8.319 Joule/mol/K
!  T    = Temperature (K)

      use constants

      implicit none
      real :: nint, m, Tc, v_max, h_v, T, kappa, y_a, y_b, h_y
      real :: a, b, sum, y_k, f, fvT
      integer :: k, n

                                      ! Standard input from keyboard
      write(*,*) 'Compute fraction of molecules whose speeds v < v_max:'
      write(*,*) 'Enter temperature in Celsius: '
      read(*,*) Tc
      write(*,*) 'Enter upper velocity limit in m/sec: '
      read(*,*) v_max
      write(*,*) 'Enter integration step size in m/sec (e.g. 1.): '
      read(*,*) h_v

      T   = 273.15 + Tc               ! Compute absolute temperature (in K)
      m   = (m_N2*0.78 + m_02*0.21 + m_Ar*0.01) / 1000. ! Masses in kg/mol
      kappa = sqrt(2.*k_B*T/m)

      a=0.
      b=v_max
      y_a  = a/kappa
      y_b  = b/kappa
      h_y  = h_v/kappa
      nint = (y_b-y_a)/h_y

      n   = ifix(nint)


      sum = 0.
Boltzmann_integral: do k=1, n-1
```

```
                    y_k= y_a+h_y*float(k)
                    sum = sum + f(y_k)
                  end do Boltzmann_integral
                  fvT = h_y * (f(y_a)+f(y_b)+2.*sum) / 2.
                  fvT = 4.*pi * fvT / sqrt(pi)**3
! Output the result
   write(*,*) 'Results: '
   write(*,*) '   Average molecular mass (g/mol):  ', m*1000.
   write(*,*) '   Temperatuer of gas (Celsius):    ', Tc
   write(*,*) '   Velocity (m/sec)                 ', v_max
   write (*,20) fvT*100.
20  format('  f(v,T) (in %)                        ',3x,f6.2)


   stop
       end program boltzmann

!* function sub-program
   function f(y)
!*
   implicit none
   real :: f, y, y2

   y2 = y*y
   f = y2*exp(-y2)

   return
     end function f
```