A practical guide to FORTAN

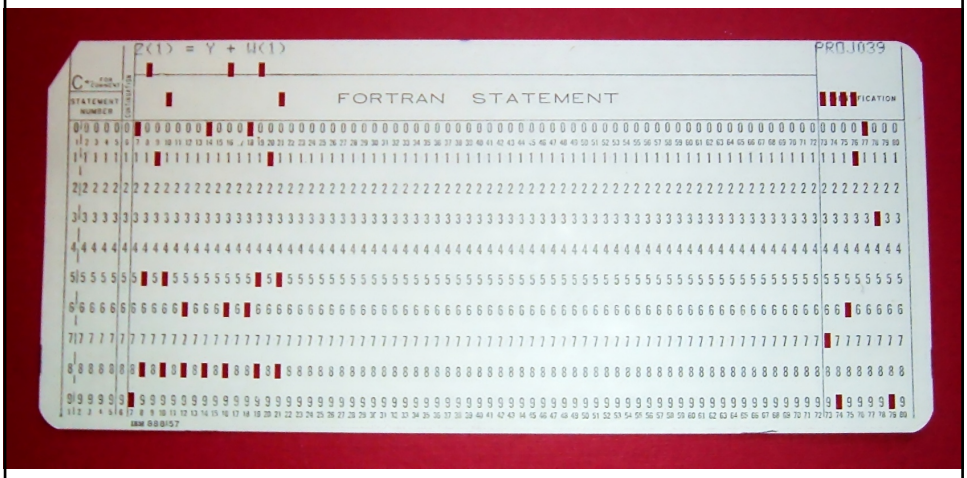
Kenny Wood
kw25@st-andrews.ac.uk

- A practical approach to programming in FORTRAN
- Borrows heavily from many resources, mostly the Stanford tutorial
- Why still using FORTRAN77… because it's readable & what I learned
- Basics of program structure and layout
- Parameter types and declarations
- Arithmetic and intrinsic functions (trig, logs, exponetiation, etc)
- Arrays
- If statements
- Do loops
- Functions and subroutines
- Input/Output
- Compiling, linking, running

Brief history of FORTRAN

- 1953 – proposal to IBM to develop a readable computer language
- Became dominant language for science and engineering applications
- "Experts" predicted demise of fortran, but it is the most enduring language in the history of computing. Most recent releases are high level and designed for massively parallel computing.
- FORTRAN – Formula Translation. Originally all CAPITALS, but modern fortran accepts lower case.
- Originally on punch cards that were 80 characters long, hence the restrictions on line length. One line of code per card.
- FORTRAN I, II, III, IV, FORTRAN66, FORTRAN77
- fortran90, 95, fortran2003, 2008, fortran2018
- This brief intro will show some of what's possible in "fortran77" but you can do much more with newer releases
- Modern fortran is backward compatible… so I really do program in the latest fortran!!

Fortran punchcard



FORTRAN program

Program Structure

- First line – six spaces then the “program” statement
- Second line – six spaces then “implicit none”. This is good programming, makes you declare all variables, zeroth-order debugging
- First column – “c” indicates the rest of line is a comment and is ignored by the computer. In modern fortran can now place comments anywhere (except columns 2-6) using “!” and rest of text on line is ignored by computer.
- Column 6 – a symbol in this column (except 0) tells the computer to continue the statement started on the previous line
- Column 7 – computer reads statements between columns 7 and 80, so each line of programming can have 72 characters
- Body of program contains all the statements, loops, arithmetic, etc
- End your program with “stop” and “end”

Edit, compile, run…frustration...magic...

- Create a file my_program.f
- Compile, link, run… then the magic happens...
- Compile: gfortran my_program.f
- Compiler error messages... de-bugging starts...
- Compile: gfortran my_program.f, produces a.out
- Run the program: ./a.out
- Program crashes, de-bugging continues…

program *name*
declarations

statements

stop
end

- Name the output file: gfortran my_program.f –o my_program.out
- ./my_program.out

Hello World

- Need: text editor; FORTRAN compiler (gfortran)
- Create a file called hello.f … then do some magic...
- Compile it: gfortran hello.f
- Run the executable: ./a.out
- Try to create an executable called hello.out

Note six spaces before text starts on each line!

```
      program hello
      implicit none

      print *, 'Hello World!'

      stop
      end
```

circle.f

```
      program circle
      implicit none

      real r, area

c This program reads a real number r from the
c screen and prints the area of a circle with radius r

      write (*,*) 'Input radius r: '
      read (*,*) r
      area = 3.14159265358  ! Continue on next line
     +         * r * r
      write (*,*) 'Area = ', area

      stop
      end
```

comments

continuation

Variables, types, declarations

- integer *list of variables*
- real *list of variables*
- real*8 *list of variables*

- Integers (32 bits = 4 bytes) have values in the range [-m, m], where m approximately 2 *10^9
- Floating point variables: real and double precision (real*8). Use double precision for calculations that require high accuracy.
- real – seven digits accuracy, range -10^{38} to +10^{38}
- real*8 – fifteen digits, range -10^{308} to +10^{308}

- real: 3.1415926535 becomes 3.141593
- real*8: 3.1415926535d0 stores all the digits

Variables, types, declarations

- parameter statement – fix a value that will not change in the program

- Parameter (*name1 = constant1*, … , *name2 = constant2*)

- Other types: complex, logical, character

- Note – fortran90 onwards just use real which will have at least fifteen digits accuracy, real*8 or double precision not needed

circle.f

```
      program circle
      implicit none

      real r, area, pi
      parameter (pi = 3.14159)

c This program reads a real number r from the
c screen and prints the area of a circle with radius r

      write (*,*) 'Input radius r: '
      read (*,*) r
      area = pi * r * r
      write (*,*) 'Area = ', area

      stop
      end
```

Logical expressions and operators

- In FORTRAN77 and the equivalents in modern fortran

.eq.	means	equal to	==
.ne.		not equal to	/=
.lt.		less than	<
.le.		less than or equal to	<=
.gt.		greater than	>
.ge.		greater than or equal to	>=

- Logical operators: .or. , .and. , .not.
- See also: .true. , .false. , .eqv. , .neqv.

Intrinsic functions

- Arithmetic: + - * /
- Exponentiation: x**b
- Trigonometric functions (x in radians): sin(x), cos(x), tan(x), asin(y), acos(y), atan(y), atan2(y,z)
- Logs, base 10 and natural: log10(a), log(a), see also alog10(a), alog(a)
- Exponential: exp(a)

- See lists of functions including: sqrt, abs, int, dble, real, max, min, mod, sign, …

The if statements

- if (*logical expressions*) *executable statements*
- Can treat single or multiple statements
- Example: Find absolute value of x:

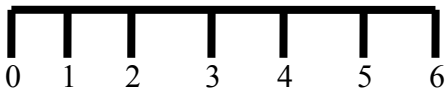
```
if (x .lt. 0) x = -x
```

```
if (logical expressions) then
   statements
endif
```

```
if (logical expressions) then
   statements
elseif (logical expressions) then
   statements
  :
  :
else
   statements
endif
```

Throwing dice…

- Random number generator ran2.f gives a pseudo random number in range (0,1)
- Write a code to generate random dice numbers using the function ran2.f



```
ran = 6. * ran2(iseed)
if (ran .lt. 1.) then
  dice = 1
elseif(ran .lt. 2.) then
  dice = 2
elseif…
    etc, etc
endif
```

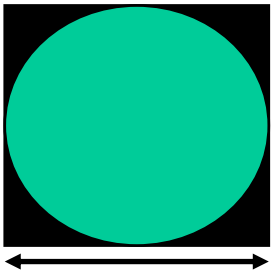
Loops

- Repeated execution of similar things – sums, multiplications, etc
- do-loops, for-loops, while-loops

```
      sum = 0
      do 10 i = 1, N
        sum=sum + i
        write(*,*) 'i = ', i, 'sum = ', sum
10    continue
```

```
      sum = 0
      do i = 1, N
        sum=sum + i
        write(*,*) 'i = ', i, 'sum = ', sum
      end do
```

Your first Monte Carlo code: Estimate π using rejection method



Pick N random positions (x_i, y_i):
x_i in range $[-R, R]$: $x_i = (2\xi - 1)\, R$
y_i in range $[-R, R]$: $y_i = (2\xi - 1)\, R$
Reject (x_i, y_i) if $x_i^2 + y_i^2 > R^2$
Number accepted / $N = \pi R^2 / 4R^2$
$N_A / N = \pi / 4$
Increase accuracy: large N

FORTRAN:

```
do i = 1, N
   x = 2.*ran – 1.
   y = 2.*ran –1.
   if ( (x*x + y*y) .lt. 1. ) NA = NA + 1
end do
pi = 4.*NA / N
```

Arrays

- One dimensional array corresponds to a vector
- Two dimensional array corresponds to a matrix
- Indexes for a p*q array run from 1,2,…p-1, p; 1,2,…q-1, q
- Arrays are declared at start of program

```
integer array1(10), array2(20)
real*8 array3(20,30), array4(100,100,100)
```

```
integer i,j,k,m
parameter(i=10, j=20,k=30,m=100)
integer array1(i), array2(j)
real*8 array3(j,k), array4(m,m,m)
```

- Note that modern fortran allows for dynamically allocated arrays

Tabulated Spectrum

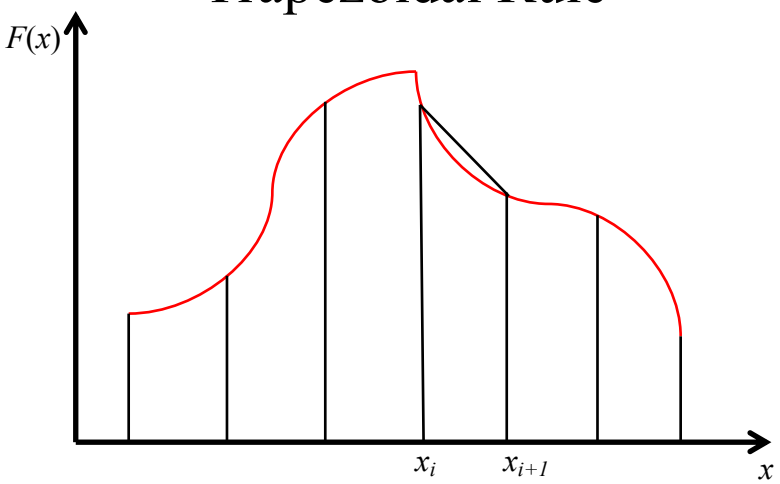
- Solar spectrum: example of using numerical integration
- Given arrays of wavelengths, $\lambda(i)$, and corresponding flux values, $F(i)$, with $i = 1, 2, \ldots N$
- Numerical integration by trapezoidal rule:

$$\int F(\lambda)\,d\lambda \approx \sum_{i=1}^{N-1} \frac{1}{2}\left(F_{i+1} + F_i\right)\left(\lambda_{i+1} - \lambda_i\right)$$

```
sum = 0
do i=1, N-1
  sum = sum + 0.5*[F(i+1)+F(i)]*[lam(i+1)-lam(i)]
end do
```

- Write a program to do trapezoidal integration
- Write a program to integrate a tabulated function using Simpson's Rule

Trapezoidal Rule



Area = $[x_{i+1} - x_i]\,[F(x_i) + F(x_{i+1})] / 2$

Functions and subroutines

Pass variables and a FUNCTION performs maths, algebra, geometry, to return a single result, e.g., random number generators. Need to declare the function type in the main program.

```
program trig
implicit none

real x, cossquared

read(10,*) x
write(10,*) cossquared(x)

stop
end
```

```
function cossquared(x)
implicit none

real x

cossquared = cos(x)*cos(x)

return
end
```

SUBROUTINES are similar, but can return many different results. The order that variables are passed must be the same in the call statement and the subroutine. In main program have the call statement:

```
call my_subroutine(i,j,k,x,y,z)
```

```
subroutine my_subroutine(i,j,k,x,y,z)
implicit none

integer i,j,k
real x,y,z

Math statements, equations, etc

return
end
```

I/O: Input & Output

- Example: read in data from a 2D file and store in an array

```
integer i, j
real*8 array1(20,30)

open(unit=10, file='myfile.dat',status='unknown')
  do i=1,20
    do j=1,30
      read(10,*) array1(i,j)
    end do
  end do
close(10)
```

What's not been covered

- True and false statements
- Character statements
- Common blocks
- Data statements
- Format statements
- Allocatable arrays
- and lots and lots of other stuff…

Exercises

Write your own fortran programs to do the following examples from the notes

- Hello World
- Area of circle
- Dice
- Compute pi
- Numerical integration

Exercises

You can get the ran2.f random number generator code and the Solar spectrum from this website for my Monte Carlo Radiation Transfer course:

http://www-star.st-and.ac.uk/~kw25/teaching/mcrt/mcrt.html

Calling ran2…

This short code shows how to use ran2.f to create two random numbers and print to the screen. Keep ran2.f in a separate file and create the executable “create_ran.out” by typing:

gfortran create_ran.f ran2.f –o create_ran.out

The code will always produce the same two random numbers unless you change the initial value of iseed. Note that for ran2.f, iseed must be set to be a negative integer.

```
program create_ran
implicit none

integer iseed
real random1, random2
real ran2

iseed = -12345

random1 = ran2(iseed)
random2 = ran2(iseed)

print*, random1, random2

stop
end
```