993SM - Laboratory of Computational Physics unit II - Friday October 3, 2025

Maria Peressi
Università degli Studi di Trieste - Dipartimento di Fisica
Sede di Miramare (Strada Costiera 11, Trieste)
e-mail: peressi@units.it
tel.: +39 040 2240242

miscellanea

list of EXERCISES;
more on fortran90,
fit, gnuplot…

LIST OF EXERCISES - Unit II - part (b)

Random numbers with non uniform distributions

1) exponential distribution generated with Inverse Transformation Method

2) another distribution generated with ad-hoc algorithms (compare!), including Inverse Transformation Method

3) gaussian distribution generated with Box-Muller algorithm

4) gaussian distribution generated with the central limit theorem

To do: implementation of the algorithms (or understanding…), histogram, fit…

Ex. 1: exponential deviate

Ex. 1: exponential deviate (λ=1), i.e., we want a distribution p(x)=exp(-λx)

it is reasonable to make a plot for x < 2/λ, therefore:

```
print *," Insert number of bins in the histogram>"
read *, nbin
delta = 2./lambda/nbin
```

$= \dfrac{2}{\lambda * nbin}$ is the bin width

```
do i = 1,n
   call expdev(x)
   ibin = int (x/lambda/delta) + 1   !   x/(lambda*delta)
   if (ibin <= nbin)histo(ibin) = histo(ibin) + 1
end do
```

(neglect other data)

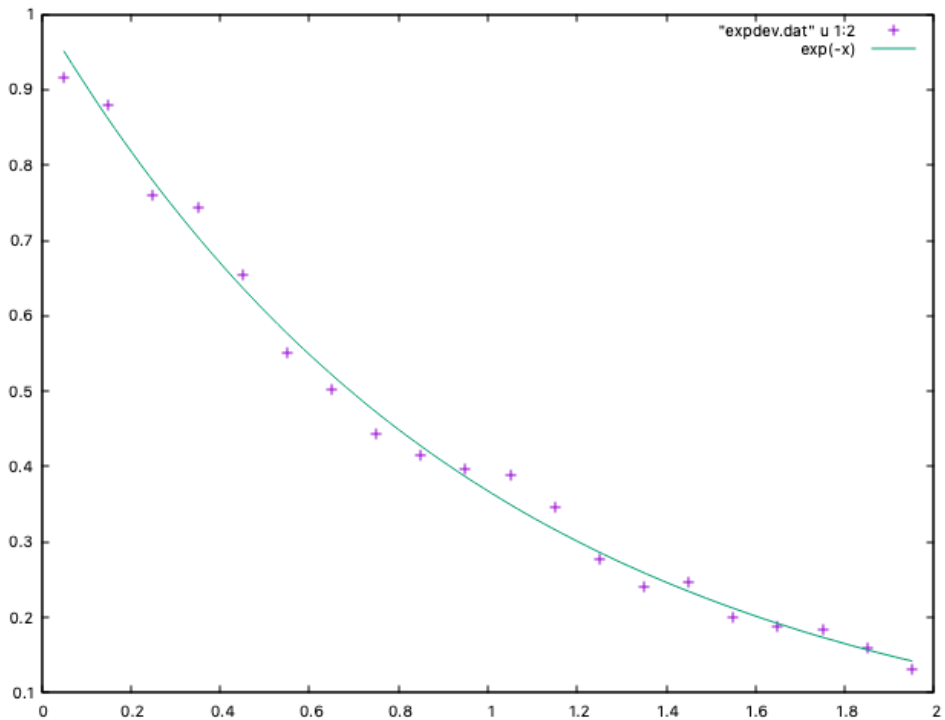
```
print*, ' normalization = ',sum(histo)/float(n)
```

(check normalization includes all the **n** data, i.e. , also those in the neglected tail of the histogram)

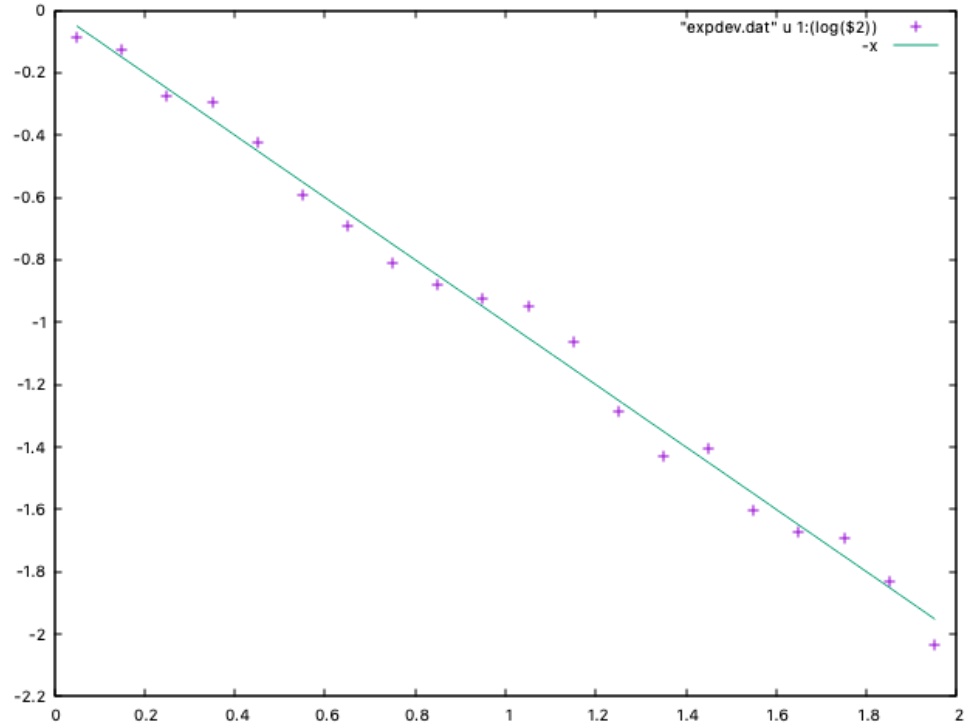
```
do ibin= 1 ,nbin
   write(unit=7,fmt=*)(ibin-0.5)*delta,histo(ibin)/float(n)/delta
end do
```

Making histograms: use int() or similar intrinsic functions

e.g. Ex. 1: exponential deviate (λ=1)



plot raw data, compare with exp(-x)



plot 1:(log($2)), compare with "-x"

Making histograms: use int() or similar intrinsic functions?

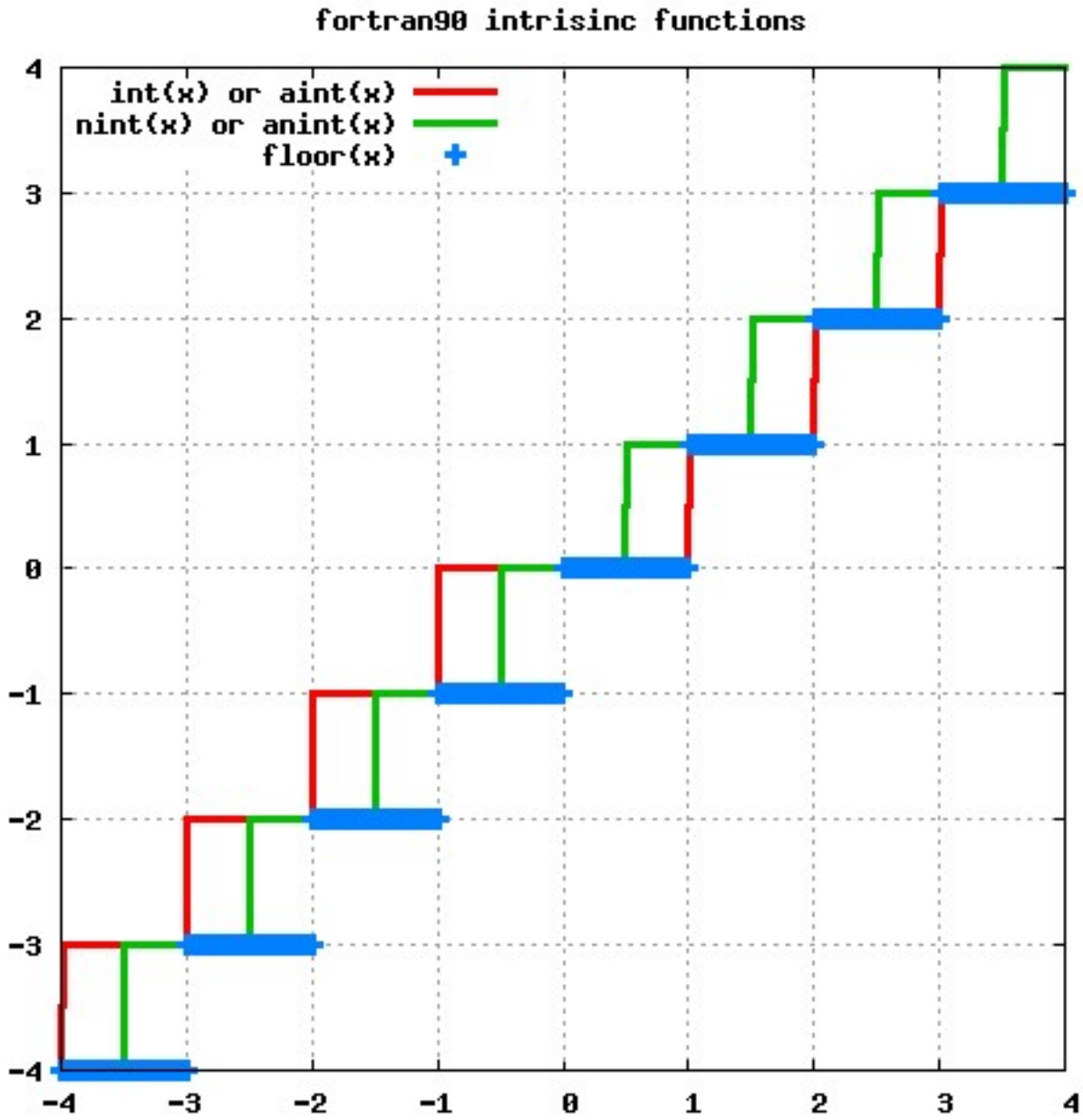
AINT(A[,KIND])
· Real elemental function
· Returns A truncated to a whole number. AINT(A) is the largest integer which is smaller than |A|, with the sign of A. For example, AINT(3.7) is 3.0, and AINT(-3.7) is -3.0.
· Argument A is Real; optional argument KIND is Integer

ANINT(A[,KIND])
· Real elemental function
· Returns the nearest whole number to A. For example, ANINT(3.7) is 4.0, and AINT(-3.7) is -4.0.
· Argument A is Real; optional argument KIND is Integer

FLOOR(A,KIND)
· Integer elemental function
· Returns the largest integer $\leq$ A. For example, FLOOR(3.7) is 3, and FLOOR(-3.7) is -4.
· Argument A is Real of any kind; optional argument KIND is Integer
· Argument KIND is only available in Fortran 95

INT(A[,KIND])
· Integer elemental function
· This function truncates A and converts it into an integer. If A is complex, only the real part is converted. If A is integer, this function changes the kind only.
· A is numeric; optional argument KIND is Integer.

NINT(A[,KIND])
· Integer elemental function
· Returns the nearest integer to the real value A.
· A is Real

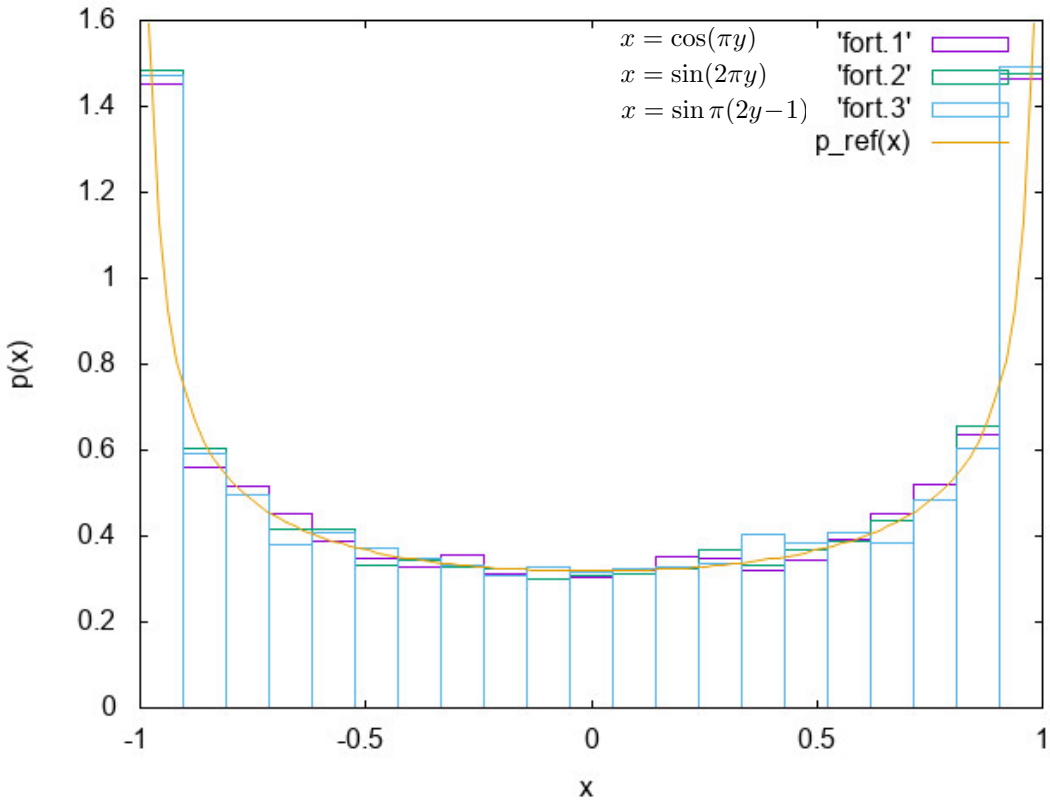

fortran90 intrisinc functions
int(x) or aint(x)
nint(x) or anint(x)
floor(x)
4
3
2
1
0
-1
-2
-3
-4
-4
-3
-2
-1
0
1
2
3
4

Making histograms: use int() or similar intrinsic functions?

e.g. Ex. 2:

Suppose you want to generate a random variate x in (-1,1) with distribution

$$p(x) = \frac{1}{\pi}(1 - x^2)^{-1/2}.$$



$$y = P(x) = \int_{-1}^{x} \frac{1}{\pi}(1 - x^2)^{-1/2}\, dx$$

$$= \frac{1}{\pi} \arcsin(x)\,|_{-1}^{x} = \frac{1}{\pi} \arcsin(x) + \frac{1}{2}$$

and invert…

Here:
different histograms, from distributions generated with different algorithms

=> how to do these histograms?

Making histograms: use int() or similar intrinsic functions?

e.g. Ex. 2:

```
      subroutine p(x)
      real, intent (out) :: x
      real :: z
      call random_number(z)
!     x = cos(pi*z)
!     x=sin(pi*(2*z-1))
      x=sin(2*pi*z)
      end subroutine p
```

Formule parametriche

$$\left.\begin{aligned}\sin\alpha &= \frac{2t}{1+t^2}\\ \cos\alpha &= \frac{1-t^2}{1+t^2}\end{aligned}\right\} \quad \left(t = \tan\frac{\alpha}{2}\right)$$

Making histograms: use int() or similar intrinsic functions?

e.g. Ex. 2:

```
print*, "Inserisci il numero DISPARI di canali nell'istogramma"
read(*,*) nbin
delta = 2./nbin
allocate(histo((-nbin/2):(nbin/2)))  !il canale centrale e' etichettato 0


do i=1,N
call p(x)
ibin = nint(x/delta)
if (abs(ibin) <= nbin/2) histo(ibin) = histo(ibin) + 1 !un check che non servirebbe
end do

do ibin = -nbin/2, nbin/2
write(1,*) ibin*delta, histo(ibin)/float(N)/delta
end do
```

Example: fit using gnuplot - I

Suppose you want to fit your data (say, 'data.dat') with an exponential function.
You have to give: 1) the functional form ; 2) the name of the parameters

gnuplot> f(x) = a * exp (-x*b)

Then we have to recall these informations together with the data we want to fit:
it can be convenient to inizialize the parameters:
gnuplot> a=0. ; b=1. (for example)

gnuplot> fit f(x) 'data.dat' via a,b

On the screen you will have something like:

```
Final set of parameters Asymptotic Standard Error
======================= ==========================

a = 1 +/- 8.276e-08 (8.276e-06%)
b = 10 +/- 1.23e-06 (1.23e-05%)

correlation matrix of the fit parameters:

a b
a 1.000
b 0.671 1.000
```

It's convenient to plot together the original data and the fit:

gnuplot> plot f(x), 'data.dat'

Example: fit using gnuplot - II

If you prefer to use linear regression, **use logarithmic data in the data file**, **or** directly fit the log of the original data using **gnuplot:**

gnuplot> f(x) = a + b*x

Then we have to recall these informations together with the data we want to fit
(in the following example: x=log of the first column; y=log of the second column):

gnuplot> fit f(x) 'data.dat' u (log($1)):(log($2)) via a,b

```
...
   Final set of parameters Asymptotic Standard Error
    ====================== ==========================  (...gnuplot will work for you....)
...
```

Also in this case it will be convenient to plot together the original data and the fit:

gnuplot> plot f(x), 'data.dat' u (log($1)):(log($2))

In case of needs, we can limit the set of data to fit in a certain range **[x_min:x_max]**:

gnuplot> fit [x_min:x_max] f(x) 'data.dat' u ... via ...

A few notes on Fortran

related to the exercises

Intrinsic functions:

LOGARITHM

log returns the natural logarithm

log10 returns the common (base 10) logarithm

(NOTE: also in **gnuplot, log** and **log10** are defined with the same meaning)

INTEGER PART

nint(x) and the others, similar but different (see Lect. II) =>

ex. II requires histogram for negative and positive data values

Arrays:

possible to label the elements from a negative number or 0:

dimension array(-n:m) (e.g., useful for making histograms)

[default in Fortran: n=1; in c and c++: n=0]

Array dimension:

default : dimension array([1:]n)
but also using other dimensions e.g.: dimension array(-n:m)

Important to **check dimensions** of the array when compiling or during execution !
If not done, it is difficult to interpret error messages (typically: "segmentation fault"), or even possible to obtain unpredictable results!

Default in gfortran:
boundaries not checked; use **compiler option**:

gfortran -fcheck=bounds myprogram.f90
(obsolete but still active alternative: -fbounds-check)

Typing (Unix line command):
man gfortran
you can scroll the manual pages and see the possible compilation options

Some Fortran compiler options

-fcheck=bounds enables checking for array subscript expressions

-fbacktrace generate extra information to provide source file traceback at run time
Specify that, when a runtime error is encountered or a deadly signal is emitted (segmentation fault, illegal instruction, bus error or floating-point exception), the Fortran runtime library should output a backtrace of the error. This option only has influence for compilation of the Fortran main program.

-Wall Enables commonly used warning options

…

Structure of a main program with one function or subr.

program name_program (see: expdev.f90 or boxmuller.f90)
  implicit none (*)
 <declaration of variables>
 <executable statements>

contains

```
   subroutine ...  (or function)
    ...
    end subroutine
```

end program

(*) General suggestion for variable declaration:
Use "implicit none" + explicit declaration of variables

See also the use of **module**