SAMPLING THE GAUSSIAN DISTRIBUTION WITH THE BOX–MULLER ALGORITHM

GIUSEPPE FORTE
https://www.giuseppeforte.me

LICENSE

1 INTRODUCTION

In this note I discuss how to sample the Gaussian distribution by means of the Box–Muller algorithm. The source codes reported in this notes are used to illustrate the basic usage of vectors and arrays. Moreover, I also present a first example of *dynamical memory allocation*, which may be possibly used with such structures. A simple routine to make the histogram of a 1D random variable is quickly introduced. Such routine, as I discussed in previous notes, can be either written in the same file containing the source code for the main program or, alternatively, in a different file. In this note, I also discuss how to recall a subroutine written in an external file, namely how to ***link two fortran codes***

2 THE BOX–MULLER ALGORITHM

Gaussian Random Variables (RVs) are highly important in Science, because the natural emergence of the Central Limit Theorem[Bouchaud and Georges, 1990; Forte, 2014; Forte et al., 2014].

A (continuous) RV x is distributed according a Gaussian with mean μ and variance σ^2 if the probability density to get the variable within a "volume–element" dx around x is given by

$$P(x) = \left(\frac{1}{2\pi\sigma^2}\right)^{1/2} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \tag{1}$$

When x is distributed according the above density, we write $x \sim \mathcal{N}[\mu, \sigma^2]$. Equivalently, the variable $z = \sigma^{-1}(x - \mu)$ is distributed according a so called standard Gaussian, i. e. $z \sim \mathcal{N}[0, 1]$ and

$$P(z) = \left(\frac{1}{2\pi}\right)^{1/2} \exp\left(-\frac{x^2}{2}\right) \tag{2}$$

A simple algorithm to sample from the Gaussian in Eq. (2) has been developed by Box and Muller [1958]. The algorithm can be easily implemented on a personal laptop. Given two random variables s, t, both uniformly distributed in the unitary interval[1], the random variables

$$Z_c = \sqrt{-2\ln s}\cos(2\pi t) \tag{3}$$
$$Z_s = \sqrt{-2\ln s}\sin(2\pi t) \tag{4}$$

are both distributed according a standard Gaussian, i. e. $Z_{c,s} \sim \mathcal{N}[0, 1]$. As an immediate consequence related to the transformation between RVs [Gardiner, 2012], the RV x defined as

$$x = \mu + \sigma Z_{c,s} \tag{5}$$

will be distributed according a Gaussian $\mathcal{N}[\mu, \sigma^2]$. The derivation of the Box–Muller algorithm follows from the following observation. Two independent standard Gaussian variables Z_c and Z_s are characterized by the joint PDF (Probability Density Function)

$$P(Z_c, Z_s) = \frac{1}{2\pi}\exp\left(-\frac{Z_c^2 + Z_s^2}{2}\right)$$

or, if we define polar coordinates,

$$\begin{cases} Z_c = R\cos\theta \\ Z_s = R\sin\theta \end{cases}$$

the original joint distribution becomes

$$P(R, \theta)dRd\theta = \left(\frac{1}{2\pi}\right) e^{-R^2/2} \quad R \quad dRd\theta$$

We can immediately integrate out the θ contribution from the above expression by integrating over all the possible allowed angles ($\theta \in [0, 2\pi]$). In particular,

$$\frac{1}{2\pi}\int_0^{2\pi} d\theta = 1$$

i. e., $\theta \sim \mathcal{U}[0, 2\pi]$. What remains is the distribution

$$P(R)dR = e^{-R^2/2} \quad R \quad dR$$

or, by introducing the variable transformation $\xi = R^2/2$ ($d\xi = RdR$)

$$P(\xi) = e^{-\xi} \quad d\xi$$

The above PDF is associate to a distribution function

$$F(Q) = \int_0^Q d\xi\, e^{-\xi}$$

1 $s, t \sim \mathcal{U}[0, 1]$

We remember that $F(Q)$ is the probability that $0 \leqslant \xi \leqslant Q$ (i. e. $F(Q) \in [0,1]$). Performing the above integral we get

$$-\ln[1-F(Q)] = Q$$

with $1-F(Q)$ a number between 0 and 1. Thus, if we draw from the uniform distribution between 0 and 1 a RV s, the variable

$$\sqrt{-2\ln s} = R$$

will be distributed according the density $P(R)dR = e^{-R^2/2} \quad R \quad dR$. Drawing $s \sim \mathcal{U}[0,1]$ and $t \sim \mathcal{U}[0,1]$, the variables

$$\begin{cases} Z_c = R\cos\theta \\ Z_s = R\sin\theta \end{cases}$$

with $R = \sqrt{-2\ln s}$ and $\theta = 2\pi t$, by construction, are distributed according two independent Gaussians of 0 mean and unitary variance. At this point, using Eq. (5), we can sample from any kind of Gaussian $\mathcal{N}[\mu, \sigma^2]$

3 FORTRAN CODE

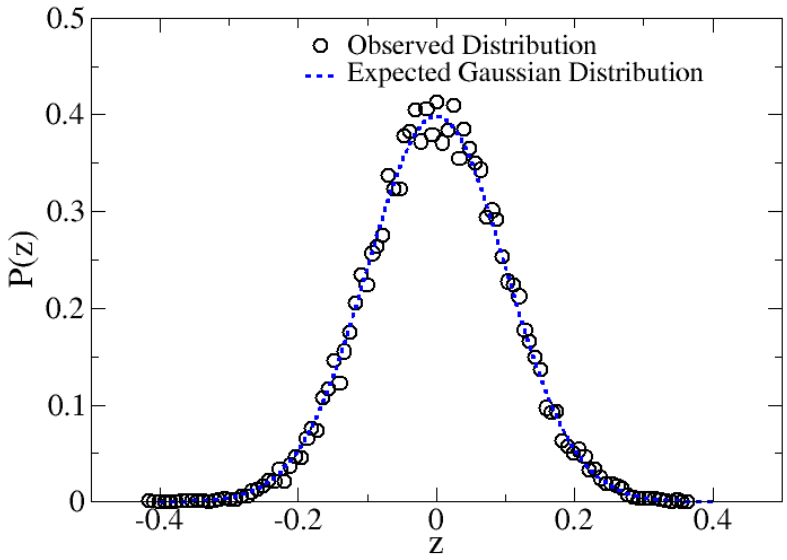

Figure 1: The Listing 1 simulates N_s Independent Gaussian variables with mean $\mu = 1.2$ and standard deviation $\sigma = 10^{-1}$ using the Box–Muller algorithm introduced in Sec. 2. The code in Listing 1 returns a data file "Histo.dat" with three columns. the first one is the abscissa "x", the second one gives the *integer* counts in x and $x + dx$; finally, the last column gives the probability density (normalized to one). Here we show the density of the standardized variable z, obtained by replacing the first column of the file "Histo.dat" with the variable $z = \sigma^{-1}(x - \mu)$. At the same time, the third column, say y, is replaced by $\sigma y \equiv P(z)$.

Listing 1: Fortran code for the implementation of the Box–Muller algorithm introduced in Sec. 2

```
!*****************************************************************
! This program Shows how to generate Gaussian Random Variables using
! the Box - Muller Algorithm. To make the histogram of the output
! data, I also introduce an easy subroutine for 1D histograms.
! The program shows the basic usage of vectors in Fortran.
!*****************************************************************

!*****************************************************************
! Author:        Giuseppe Forte
! First release: 23/09/2017
! e-mail:        giuseppe.forte@giuseppeforte.me
! Website:       https://www.giuseppeforte.me
! License:       Creative Commons Attribution-ShareAlike 4.0
!                International (CC BY-SA 4.0)
!                (https://creativecommons.org/licenses/by-sa/4.0/)
!*****************************************************************


!*****************************************************************
! The following module contains the subroutine to make the Histogram
! of a 1D Random variable.
! Remember to use the current module in the main program by
! "uploading" it with the instruction
!
!       use Histogram
!
! written immediately after the instruction program, i.e.
!
!       program Prog_Name
!--->   use Histogram
!       implicit none
!       [main program body]
!       end program Prog_Name
!*****************************************************************

        module Histogram
        contains

        subroutine MakeHist(Vec,Nbin,Mids,Counts,Density)
        implicit none
        real*8,  dimension(:),intent(in):: Vec ! Vec is a vector
!       variable, its dimension is not specified (otherwise
!       I would have used the instruction dimension(N), with N the
!       predetermined dimension of the vector).
!       That means that a certain memory space will be pre-allocated
!       by fortran, thus the memory is left free and
!       definitively allocated only elsewhere. This procedure is
!       called dynamical memory allocation, in contrast with a
!       fixed allocation, like the vector Counts(Nbin) below, for example,
!       whose dimension is fixed and given by Nbin. In particular,
!       Counts is a set of numbers, Counts(1), Counts(2), ...,
!       Counts(Nbin). Such dimension (Nbin) cannot be changed during
!       the execution of the program. On the other hand,
!       Vec is an array whose dimension is not specified in the
!       declaration section and can be allocated and REALLOCATED many
!       times, as the program develops. Here we shall be concerned
!       with a single allocation of the memory. An example with
!       multiple allocations will be discussed in another note
        integer, intent(in):: Nbin ! Number of bins
        integer, intent(out):: Counts(Nbin) ! counting the number
!       of events within a given bin
        real*8,  intent(out):: Mids(Nbin), Density(Nbin)
!       ! Density = the renormalized counts to make the integral of
!       the density equal to one
```

```
!        Minds = mid points within the range of the random variable
         real*8   xmax,xmin,dx,Pos,Norm,inflim,suplim
         integer  k
         logical,allocatable:: Cond(:) ! this line is completely
!        equivalent to the statement
!        logical, dimension(:):: Cond (Here we are defining a logical
!        vector, whose dimension is not declared, again, dynamical
!        memory allocation)

         ! In order to allocate a vector dynamically allocated, like
!        the vecotor "Vec" (or Cond) we should write an instruction like
!        allocate(Vec(SpecifyDimension)), however, in this case the
!        vector Vec is AUTOMATICALLY allocated. Indeed, Vec is an input
!        argoment of the subroutine, meaning that, once we pass the
!        argument to the subroutine, Vec will be allocated in
!        agreement with the dimension of the passed vector
         xmax = maxval(Vec)
         xmin = minval(Vec)
         dx   = (xmax-xmin)/dble(Nbin)
         Pos  = xmin
         k = 1
         open(unit=15, file="Histo.dat",status="unknown")

         do 108 while(Pos.le.xmax)
           suplim    = Pos + dx
           inflim    = Pos
           Cond      = Vec.lt.suplim ! here, the logical vector "Cond"
!          is AUTOMATICALLY allocated by comparing it with the vector
!          Vec. In order to make the equivalence syntactically correct
!          Cond must be charcaterized by the same dimension of the expression
!          on the right, thus it has to have the same dimension of Vec
           Cond      = Cond.and.(Vec.ge.inflim)
           Counts(k) = count(Cond) !counts the number of true conditions
!          in cond (i.e. the mumber of random variables in the current
!          interval)
           Mids(k)   = inflim + 0.5d0*dx
           Pos       = Pos + dx
           k         = k + 1
108      end do
         Norm    = sum(Counts)*dx
         Density = dble(Counts)/Norm ! sum(Density)/Norm = 1!!!!

         do k=1, Nbin
           write(15,*)real(Mids(k)),Counts(k),real(Density(k))
         end do
         close(15)
         return
         end subroutine MakeHist

         end module Histogram


!*****************************************************************
!        MAIN PROGRAM
!*****************************************************************
         program BoxMULLER
         use Histogram
         implicit none
         integer i !dummy index
         integer Ns, Nbin  ! Ns =Numbers of samples
         parameter(Ns = 10000, Nbin = 100)
!        Nbin = number of bins in the histogram
         real*8 x(Ns)! x is a vector of dimension Ns, i.e.
!        it has got Ns components. We can access the generic
!        component "i" (i=1,2,3,...,Ns) in the vactor x
!        simply by writing x(i)
         real*8 Mids(Nbin), Density(Nbin)
```

```
132           integer Counts(Nbin)
133   !       The abscissa of the histogram of the RV x runs from
134   !       the minimum of x to the maximum of x, divided in
135   !       Nbin intervals. The mid point of such intervals is stored
136   !       in the vector Mids. The number of RVs falling in a given
137   !       interval is stored in the vector Couts. Finally the
138   !       normalized histogram is stored in the vector Density
139           real*8 u1,u2 !convenience variables
140           real*8 Mu, Sigma ! Mu    = mean of the target Gaussian
141                            ! Sigma = Standard Deviation of the target
142                            ! Gaussian
143           parameter(Mu = 1.2d0, Sigma = 0.1d0)
144           real*8 TwoPi ! TwoPi = 2 * Greek Pi = 2 * 3.141593... =
145   !                          = 6.283185...
146           parameter(TwoPi = 8.0d0*atan(1.0d0))
147
148   !       Initialization of the vector "x"
149           x = 0.0d0 ! this instruction puts ALL the components
150   !       of the vector "x" equal to 0
151           !the above instruction is "equivalent" to the set
152   !       of instructions
153           ! do i=1, Ns
154           !    x(i) = 0.0d0
155           ! end do
156   ! Actually, the instruction is not exactly the same. indeed, if we
157   ! use the cycle do i=1...end do, Fortran will execute the instructions
158   ! sequentially on the same processor. If we use the instruction x=0.0d0
159   ! Fortran will run the instruction AUTOMATICALLY on ALL the AVAILABLE
160   ! processors: in other words x=0.0d0 will be parallelized (not bad,
161   ! considering that we are not making explicit use of MPI. This is a
162   ! feauture which has been introduced only in the modern versions
163   ! of Fortran.)
164
165   !       Generation of the Gaussian random variables with mean Mu
166   !       and Standard Deviation Sigma (Variance = Sigma^2)
167           do 100 i=1, Ns
168             call random_number(u1)
169             call random_number(u2)
170             ! Implementation of the Box-Muller Algorithm
171             x(i) = Mu + Sigma*sqrt(-2.0d0*log(u1))*cos(TwoPi*u2)!
172   !         Box-Muller
173   100     end do
174
175   !       Construction of the histogram (frequencies) and
176   !       associated probability density
177           call MakeHist(x,Nbin,Mids,Counts,Density)
178   !      the subroutine MakeHist is contained in the module Histogram
179   !      Such subroutine takes in input the vector "x", i.e. the
180   !      "observed" random values (here we are simulating such values)
181   !      and the number of bins (Nbin). The output are the vectors Mids,
182   !      Counts and Density, TOGETHER with a data file "Histo.dat"
183   !      containing three columns, specifically
184   !
185   !      Mids(k), Counts(k), Density(k) (k=1,2,...,Nbin)
186   !
187
188           end program BoxMULLER
```

A FURTHER COMMENTS AND EASY MODIFICATIONS

Listing 2: external module "Histogram" used by the code in Listing 3

```
1   !*****************************************************************
```

```
! This module contains the subroutine to make the Histogram of a 1D
! Random variable. This is the code for the module ONLY.
! You might thus use it in any other Fortran code, provided that
! you properly recall it and compile all the files in the correct
! way (please, check Listing3.f in order to learn how to compile
! a fortran code which calls a routine written in another
! file)
!**********************************************************************

!**********************************************************************
! Remember to use the current module in the main program by
! "uploading" it with the instruction
!
!       use Histogram
!
! written immediately after the istruction program ....i.e.
!
!       program Prog_Name
!--->   use Histogram
!       implicit none
!       [main program body]
!       end program Prog_Name
!       YOU ALSO NEED TO COMPILE YOUR MAIN PROGRAM CALLING AT
!       THIS MODULE PROPERLY (please, check Listing3.f, which is
!       code calling at this module)
!**********************************************************************

        module Histogram
        contains

        subroutine MakeHist(Vec,Nbin,Mids,Counts,Density)
        implicit none
        real*8,  dimension(:),intent(in):: Vec
        integer, intent(in):: Nbin
        integer, intent(out):: Counts(Nbin)
        real*8,  intent(out):: Mids(Nbin),Density(Nbin)
        real*8   xmax,xmin,dx,Pos,Norm,inflim,suplim
        integer  k
        logical,allocatable:: Cond(:)


        xmax = maxval(Vec)
        xmin = minval(Vec)
        dx   = (xmax-xmin)/dble(Nbin)
        Pos  = xmin
        k = 1
        open(unit=15, file="Histo.dat",status="unknown")

        do 108 while(Pos.le.xmax)
          suplim    = Pos + dx
          inflim    = Pos
          Cond      = Vec.lt.suplim
          Cond      = Cond.and.(Vec.ge.inflim)
          Counts(k) = count(Cond)
          Mids(k)   = inflim + 0.5d0*dx
          Pos       = Pos + dx
          k         = k + 1
108     end do
        Norm    = sum(Counts)*dx
        Density = dble(Counts)/Norm

        do k=1, Nbin
          write(15,*)real(Mids(k)),Counts(k),real(Density(k))
        end do
        close(15)
        return
        end subroutine MakeHist
```

```
69
70          end module Histogram
```

So far, we have used a code to generate Gaussian RVs and make the histogram of such variables. Now, let us assume that we have an experimental sample $x = (x_1, x_2, \cdots, x_{N_s})$ of unspecified size N_s, collected in the file "Experimental_Observations.dat". Our goal is to write a program to determine the size of the sample, i. e. N_s and then make the distribution related to such data. Following the coding technique suggested by Markus and Metcalf [2012] (see also the associated web–site here), we propose the code in Listing 3. Such code uses the external module "Histogram" coded in Listing 2. This is the same module used in Listing 1, however, now such module is written in a file which is external to the main program shown in Listing 3. We thus need to learn how to compile the code in Listing 3 in the correct way, i. e. we need to know how to ***pass*** to the compiler the module in Listing 2. Let us assume that we call the code in Listing 2 with the name "Listing2.f". Now, let us assume that the ***sorce code containing the main program is written in "Listing3.f"***. In order to put everything together we will compile the main program in Listing 3 using the istruction

- gfortran -O3 Listing2.f Listing3.f -o out.x

Listing 3: Fortran Code to make the histogram of an external file of unspecified size

```
1  !************************************************************************
2  ! This program Shows how to make the histogram of a RV contained
3  ! in a given external file, say Experimental_Observations.dat. This
4  ! file might be a file passed to you by anyone of your collaborators.
5  ! The program makes use of an external module, i.e. the Fortran code
6  ! for such an external module is written elsewhere in a
7  ! file DIFFERENT from the current one
8  !************************************************************************
9
10 !************************************************************************
11 ! Author:        Giuseppe Forte
12 ! First release: 23/09/2017
13 ! e-mail:        giuseppe.forte@giuseppeforte.me
14 ! Website:       https://www.giuseppeforte.me
15 ! License:       Creative Commons Attribution-ShareAlike 4.0
16 !                International (CC BY-SA 4.0)
17 !                (https://creativecommons.org/licenses/by-sa/4.0/)
18 !************************************************************************
19
20 ! MAIN PROGRAM !!!!!!
21          program hist
22          use HistoGram ! Look: here we are using a module (HistoGram).
23 !        Such module is not coded in the present file. It is, however,
24 !        contained in another file (Listing2.f). We use it here.
25 !        however, in order to properly recall such module, we need to be
26 !        carefull when we compile the present file (say Listing3.f: the
27 !        present file, THIS FILE)
28 !        IF WE COMPILE THE CURRENT FILE IN THE USUAL WAY, i.e.
29 !        gfortran -O3 Listing3.f -o out.x
30 !        we WILL GET AN ERROR MESSAGE from the compiler. No problem
31 !        In order to compile our codes correctly we just need to perform
32 !        two operations:
33 !        1) Move the file Listing2.f IN THE SAME FOLDER OF THE FILE
34 !           Listing3.f
35 !        2) Compile Listing3.f with the following istruction
36 !           gfortran -O3 Listing2.f Listing3.f -o out.x
37 !        that's it :-)
38 !        typically, if you write
```

```
!           gfortran -O3 Listing3.f Listing2.f -o out.x
!       should be the same, however, some Operative Systems
!       require you to put Listing2.f first
!       some others require you to put Listing3.f first
!       (.....it's a matter of machines...who knows
!       what inside their "mind"? :-))
        implicit none
        integer  Nbin,Ns
        parameter(Nbin = 80)
        real*8, allocatable:: x(:)
        real*8 Density(Nbin),Mids(Nbin)
        integer Counts(Nbin)
        real*8 u
        integer i, nodata, ierr, k


        open(unit=10,file="Experimental_Observations.dat",
     &       status="old",iostat=ierr) ! when we write iostat = ierr
        ! the INTEGER number ierr receives the error status from
        ! an open instruction. This means that the program
        ! executes the open instrunction and, if no errors
        ! happen during such instruction the INTEGER variable
        ! ierr is set equal to 0. Otherwise it will acquire
        ! some positive number. For exemple, if in line 55, we
        ! write file= "Eperimental_Opservation.dat", i.e. the
        ! name of the file does not match the name written in the
        ! instruction, ierr will be assigned a positive number.
        ! If everything is correct, ierr will be assigned 0.

        if(ierr.ne.0)then
          write(*,* )"file Experimental_Observations.dat could not"
          write(*,* )"be opened. Program ended!"
          goto 200 ! because a problem occurred, the programe
                   ! is ended
        end if

        i = 1
        ! if everything went well, at this point we still have
        ! ierr = 0.
        do 100 while(ierr == 0)
          read( 10, *, iostat = ierr )u !here we are using
          ! the read instruction together with the iostat
          ! assignment. What does that mean? After executing
          ! the read instruction in line 79, the fortran
          ! compiler will put an INTEGER number in the variable
          ! ierr. Three different situation might occur
          ! 1) if ierr = 0, the read instruction works in the
          !    usual way (and the do while statement keeps on
          !    going)
          ! 2) if ierr > 0, it means that an error has occurred
          !    during the file reading. For example, if in the file
          !    "Experimental_Observations.dat", at a certain point,
          !    there is an alphanumeric variable, such as NaN,
          !    that would cause the compiler to report an
          !    error. That is because in THIS PROGRAM
          !    , i.e. the CURRENT EXAMPLE, the variable
          !    we would like to read from the file
          !    Experimental_Observations.dat is defined in line 51
          !    as a double precision real variable
          !    and NOT an alphanumeric variable. In this
          !    case the fortran compiler will assign a
          !    positive number to ierr
          ! 3) if ierr < 0, it means that the end of the file
          !    has been reached (the last line has been read)
          if(ierr>0)then
             write(*,*) "Error reading the data!"
             goto 200
```

```
106             elseif(ierr<0)then
107                exit ! Reached the end of the file
108             endif
109             i = i + 1
110   100     end do
111           Ns = i -1 ! Ns = size of the sample
112           close(10)
113           ! Now we know the size of the sample in
114           ! Experimental_Observations.dat, we call such size Ns
115           ! and we are ready to make the histogram
116           open(unit=13,file="Experimental_Observations.dat",
117        &       status="old")
118
119           allocate(x(Ns)) !allocation of the variable x using the
120           ! size Ns
121           do 101 i =1, Ns
122             read(13,*)x(i) !store in x the content of the file
123             ! Experimental_Observations.dat
124   101     end do
125           close(13)
126
127           call MakeHist(x,Nbin,Mids,Counts,Density) ! call the
128           ! histogram routine. Such routine
129           ! IS IN THE MODULE HistoGram contained in an
130           ! external file, for example called Listing2.f
131           deallocate(x) !deallocate the memory reserved for x
132   200     end program hist
133   !! REMEMBER REMEMBER REMEMBER REMEMBER REMEMBER REMEMBER
134   !! REMEMBER REMEMBER REMEMBER REMEMBER REMEMBER REMEMBER
135
136   ! In this example we have
137   ! 1) Listing3.f (the current file)
138   ! 2) Listing3.f DEPENDS ON a routine CONTAINED in
139   !    Listing2.f
140   ! 3) MOVE Listing2.f in the same folder of Listing3.f
141   ! 4) COMPILE WITH THE INSTRUCTION
142   ! 5) gfortran -O3 Listing2.f Listing3.f -o out.x
143   ! HAVE FUN :-) !!!!
```

LICENSE

BIBLIOGRAPHY

Bouchaud, J. P. and A. Georges

1990 "Anomalous diffusion in disordered media: statistical mechanisms, models and physical applications", *Physics reports*, 195, 4–5, pp. 127-293. (Cited on p. 1.)

Box, G. E. P. and M. E. Muller

1958 "A note on the generation of random normal deviates", *The annals of mathematical statistics*, 29, 2, pp. 610-611. (Cited on p. 2.)

Forte, G.

2014 *Diffusive processes in systems with geometrical constraints: from lattice models to continuous channels*, P. A. D I. S., http://padis.uniroma1.it/bitstream/10805/2193/1/Thesis.pdf. (Cited on p. 1.)

Forte, G., F. Cecconi, and A. Vulpiani

2014 "Non–anomalous diffusion is not always Gaussian", *The European Physical Journal B*, 87, 5, pp. 1-9. (Cited on p. 1.)

Gardiner, C. W.

2012 "Handbook of Stochastic Methods". (Cited on p. 2.)

Markus, A. and M. Metcalf

2012 *Modern Fortran in practice*, Cambridge University Press. (Cited on p. 8.)