```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!
!
!  This code does the following:
!
!  1) Writes the Hamiltonian and diagonalizes it
!  2) Gives the eigenvalues as output
!  3) Gives the DOS as output
!  4) Gives P(s) as output and rtilde
!  5) Gives PR and Shannon for all eigenstates
!  6) Computes the diagonal and off-diagonal elements of sigma^z_{L/2}
!  7) Gives as output the (Calpha^ini) for an initial state with
!     energy <Psi0|H|Psi0> in the middle of the spectrum
!  8) Computes the survival probability
!
!  ABOUT THE RANDOM NUMBERS
!
! For the SEED:
!VALUE(1):    The year
!VALUE(2):    The month
!VALUE(3):    The day of the month
!VALUE(4):    Time difference with UTC in minutes
!VALUE(5):    The hour of the day
!VALUE(6):    The minutes of the hour
!VALUE(7):    The seconds of the minute
!VALUE(8):    The milliseconds of the second
!
! Random number from uniform distribution in [0,1] is rand()
! Random number from a Gaussian distribution is gasdev
! It has mean=0 and variance=1
! real (kind=8) :: gasdev is WRITTEN in the beginning of
! the subroutines where gasdev is used
! FUNCTION gasdev(seed) is called at the end of the code
!
!
! HAMILTONIANS:
! NN = nearest-neighboring couplings
! NNN = next-nearest-neighboring couplings
! defBorder is used only for the HamXXZ, HamDEF and HamNNN
! open boundaries for the HamXXZ, HamDEF and HamNNN
! closed (periodic) boundaries for the HamDISO
!
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!            VARIABLES to be used in the whole code
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      module variables

      implicit none

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!! PARAMETERS TO CHANGE !!!!!!!!!!!!!!!!!!!!!!!!!!!

! PARAMETERS of the FINAL Hamiltonian
      integer (kind=4), PARAMETER :: ChainSize=12
      integer (kind=4), PARAMETER :: upspins=6
! The dimension is computed below with the function BINOMIAL
      integer (kind=4) :: dim
      real (kind=8), PARAMETER :: Jxy = 1.0d0
! Strength of the Ising interaction
      real (kind=8), PARAMETER :: Jz = 0.48d0
!      real (kind=8), PARAMETER :: Jz = 1.00d0       ! used for HamDISO
! little border defect to break parity and spin reversal
      real (kind=8), PARAMETER :: defBorder = 0.1d0
!      real (kind=8), PARAMETER :: defBorder = 0.0d0 ! used for HamDISO
! ---------------------------------------------------------
! The values of these parameters appear inside
! the subroutine for the Hamiltonian
      real (kind=8) :: defMid,lambda,defOnsite
! defMid placed on site ChainSize/2: --- call HamDEF
!      real (kind=8), PARAMETER :: defMid = 0.9d0
! lambda for couplings between 2nd neighbors: --- call HamNNN
!      real (kind=8), PARAMETER :: lambda = 0.0d0
! defAll for onsite disorder: --- call HamDISO
!      real (kind=8), PARAMETER :: defAll = 0.0d0
! ---------------------------------------------------------
! Average over "ave" number of random realizations
      integer (kind=4), PARAMETER :: ave=1
! How many initial states
      integer (kind=4), PARAMETER :: totIni=100
! How many times to be used from FILE 'TimeIncrements.dat': MAX=5000
      integer (kind=4), PARAMETER :: totTime=5000
! How many states to compute the off-diagonal elements Oab
! I choose totOab=200
! but if int(dim/4)<200, I choose totOab=int(dim/4)
      integer (kind=4), PARAMETER :: TwoHundred=200
      integer (kind=4) :: totOab,inteiro
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


! Integers for do-loops
      integer (kind=4) :: i,j,k,ii,jj,kk


!---------------------------------------------------------------
! Random numbers
!
! SEED according to time
      integer (kind=4), dimension(8) :: val
      integer (kind=4) :: seed


!---------------------------------------------------------------
! SITE-BASIS
      integer (kind=4), dimension(:,:), allocatable :: basis


!---------------------------------------------------------------
! Eigenvalues and eigenvectors of the TOTAL Hamiltonian
      real (kind=8), dimension(:), allocatable ::  Eig
      real (kind=8), dimension(:,:), allocatable :: Vec

!---------------------------------------------------------------
! for the DIAGONALIZATION
      INTEGER (kind=4) :: INFO
      real (kind=8), dimension(:), allocatable :: work


!---------------------------------------------------------------
! for the LDOS and INITIAL STATE
      real (kind=8), dimension(:), allocatable :: EnIni
      integer (kind=4), dimension(totIni) :: IniSt
      real (kind=8), dimension(:,:), allocatable :: Calpha

!---------------------------------------------------------------
! for the EVOLUTION
      INTEGER (kind=4) :: tt
      real (kind=8), dimension(totTime) :: time


!---------------------------------------------------------------
```

```
! For the OUTPUT files
      character(len=2) si,uu
      character(len=4) dd,aa,ain
      character(len=4) dF,zF,lF,hF,bF
      character(len=70) saiE,saiDOS,saiPs,saiPr,saiPRs
      character(len=70) saiCa,saiInf, saiL,saiSP
      character(len=70) saiEEV,saiOFF,saiPt

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      end module

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!           Program starts here
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      Program SpinChain

      use variables
      implicit none


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! DIMENSION of the HAMILTONIAN MATRIX
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      integer (kind=4) :: BINOMIAL
      dim=BINOMIAL(ChainSize, upspins)


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! NUMBER of STATES to COMPUTE Oab
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      inteiro=int(dim/4)
      if(inteiro.lt.TwoHundred) then
        totOab=inteiro
      else
        totOab=TwoHundred
      endif
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! ALLOCATIONS
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      allocate(basis(dim,ChainSize))
      allocate(Vec(dim,dim))
      allocate(Eig(dim))
      allocate(EnIni(dim))
      allocate(Calpha(totIni,dim))



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c READ INSTANTS of TIME
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      OPEN(unit=20, FILE='TimeIncrements.dat',STATUS='OLD')
      Do tt=1,totTime
        read(20,*) time(tt)
      Enddo
      Close(20)



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! RANDOM NUMBERS
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      call date_and_time (VALUES=val)
      seed = val(6)*100000+val(7)*1000+val(8)
      call srand(seed)
! The random numbers from the uniform distribution in [0,1] are rand()
! The random numbers from the Gaussian distribution
! with mean=0 and variance=1 are gasdev(seed)
! real (kind=8) :: gasdev is WRITTEN in the beginning of
! the subroutines where gasdev is used



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c CREATE SITE-BASIS
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      call SiteBasis()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!c HAMILTONIAN
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!       call HamXXZ() ! integrable clean model with NN couplings
!       call HamDEF() ! chaotic single-defect model with NN couplings
       call HamNNN()  ! chaotic clean model with NN couplings
!       call HamDISO()! chaotic onsite disordered model with NN couplings


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! OUTPUT FILES
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      call OutValues()

!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c DIAGONALIZATION of the HAMILTONIAN
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

       call Diagonalizing()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c DENSITY OF STATES: DOS
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      call DOS()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c LEVEL SPACING DISTRIBUTION
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      call LevSpDist()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c RATIO OF CONSECUTIVE LEVELS
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      call RatioLev()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c PR and Shannon Entropy of EIGENSTATES
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
      call VecPRSh()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c EEV: eigenstate expectation values = DIAGONAL ELEMENTS of O
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      call EEV()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c OFF-DIAGONAL ELEMENTS of O
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      call OFFO()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c ORDER H(i,i) and then select the INITIAL STATES
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!       uses MRGRNK to order EnIni(i) = dabs(Vec(i,i)) then
!       i=1 gives the EnIni with absolute value closest to zero
!       i=2 gives the 2nd EnIni with absolute value closest to zero

      call InitialStateE0()

!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c Specific basis-site choice of INITIAL STATE, such as DW=111...000
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!       call InitialStateSite()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c Calpha's for the INITIAL STATE and IPRo
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      call CalphaIPRo()



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! EVOLUTION of the INITIAL STATE
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! Subroutine that evolves only the SP
       call TimeEvolve()
! Subroutine that evolves the SP and the IPR
```

```
!       call EvolveSPandIPR()


!ccccccccccccccccccccccccccccccccccccccccccccccccccc
!c END END END END END END END END END END END END
!ccccccccccccccccccccccccccccccccccccccccccccccccccc
!      STOP
       END




!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccc FUNCTION FUNCTION FUNCTION FUNCTION ccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      FUNCTION BINOMIAL(X,Y)

      implicit none
            INTEGER (kind=4) :: BINOMIAL
            INTEGER (kind=4) :: X,Y,k
            INTEGER (kind=8) :: num,den

            num=1
            den=1

            IF(Y.GT.(X-Y)) then
                  DO k= Y+1, X
                        num=k*num
                  ENDDO
                  DO k= 1, (X-Y)
                        den=k*den
                  ENDDO
                  BINOMIAL=num/den
            ELSE
                  DO k= (X-Y)+1, X
                        num=k*num
                  ENDDO
                  DO k= 1, Y
                        den=k*den
                  ENDDO
                  BINOMIAL=num/den
            ENDIF

            return
```

```
      END FUNCTION BINOMIAL
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!  Functions to generate random numbers from uniform and Gaussian distributions
!  using Numerical Recipes ran1 and gasdev routines.
!  ran1 gives uniform random numbers between 0 and 1
!  gasdev gives random numbers from a Gaussian distribution with zero mean
!  and unit variance

! Gaussian random number
      FUNCTION gasdev(seed)
      INTEGER (kind=4) :: seed
      REAL (kind=8) :: gasdev
!     USES ran1
      INTEGER (kind=4) :: iset
      REAL (kind=8) :: fac,gset,rsq,v1,v2,ran1
      SAVE iset,gset
      DATA iset/0/
      if (iset.eq.0) then
1       v1=2.0d0*rand()-1.0d0
       v2=2.0d0*rand()-1.0d0
       rsq=v1**2+v2**2
       if(rsq.ge.1..or.rsq.eq.0.)goto 1
       fac=sqrt(-2.0d0*log(rsq)/rsq)
       gset=v1*fac
       gasdev=v2*fac
       iset=1
      else
       gasdev=gset
       iset=0
      endif
      return
      END
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!ccccccccccccccccc SUBROUTINES SUBROUTINES SUBROUTINES ccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! ************ WRITING VALUES for the FILES NAMES ******************
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine OutValues()

      use variables
      implicit none


       write(si,9) ChainSize
       write(uu,9) upspins
 9     format(i2.2)
       write(aa,10) ave
       write(ain,10) totIni
 10     format(i4.4)
       write(zF,11) Jz
       write(lF,11) lambda
       write(dF,11) defMid
       write(hF,11) defOnsite
       write(bF,11) defBorder
 11    format(F4.2)

!c END of SUBROUTINE for the VALUES used in the NAMES of the FILES
      return
      end subroutine OutValues



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! *************** WRITING THE SITE-BASIS *************************
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SiteBasis()
      use variables
      implicit none

      INTEGER (kind=4) :: ib,jb

      logical mtc
      integer (kind=4) :: in(ChainSize),m2,h
```

```
      mtc=.false.

!c INITIALIZATION
      Do ib=1,dim
        Do jb=1,ChainSize
          basis(ib,jb)=0
        enddo
      enddo

      ii=1

 71   call nexksb(ChainSize,upspins,in,mtc,m2,h)
      Do jb=1,upspins
        basis(ii,in(jb))=1
      Enddo
      ii=ii+1
      if(mtc) goto 71

!c END of SUBROUTINE for SITE-BASIS
      return
      end subroutine SiteBasis
!ccccccccccccccccccccccccccccccccccccccccccc
!c SUBROUTINE to get the COMBINATIONS
!ccccccccccccccccccccccccccccccccccccccccccc
      subroutine nexksb(n,k,a,mtc,m2,h)
      integer (kind=4) :: n,k,a(n),m2,h,jn
      logical mtc

      if(.not.mtc) then
      m2=0
      h=k
      go to 50
      endif
      if(m2.lt.n-h) h=0
      h=h+1
      m2=a(k+1-h)
   50 do jn=1,h
      a(k+jn-h)=m2+jn
      enddo
      mtc=a(1).ne.n-k+1

      return
      end subroutine nexksb
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to write the XXZ HAMILTONIAN
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine HamXXZ()

      use variables
      implicit none

      INTEGER (kind=4) :: tot,DifferentSite(ChainSize)


! PARAMETERS FOR THE HAMILTONIAN
      defMid = 0.0d0
      lambda = 0.0d0
      defOnsite = 0.0d0


! INITIALIZATION
      Do i=1,dim
        Do j=1,dim
          Vec(i,j)=0.0d0
        Enddo
      Enddo

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! DIAGONAL ELEMENTS ***********************************************************
      Do i=1,dim
! defect at the EDGE
        Vec(i,i)=Vec(i,i)+(defBorder/2.0d0)*( (-1.0d0)**(1+basis(i,1)) )
! NN ISING interaction
        Do j=1,ChainSize-1
        Vec(i,i)=Vec(i,i)+(Jz/4.d0)*(-1.0d0)**(basis(i,j)+basis(i,j+1))
        enddo
        EnIni(i)=dabs(Vec(i,i))
! CLOSING i=1,dim
      enddo
! END of DIAGONAL *************************************************************
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! OFF-DIAGONAL ELEMENTS *******************************************************
```

```
      Do i = 1, dim-1
        Do j = i+1, dim

          tot = 0
          Do k = 1, ChainSize
            DifferentSite(k)=0
          Enddo

          Do k = 1, ChainSize
            If( basis(i,k).ne.basis(j,k) ) then
            tot = tot + 1
            DifferentSite(tot)=k
            Endif
          Enddo


          IF(tot.EQ.2) then
!!!!!!!!! NN flip-flop
          IF((DifferentSite(2) - DifferentSite(1)).EQ.1) then
            Vec(i,j)=Vec(i,j)+Jxy/2.0d0
            Vec(j,i)=Vec(i,j)
          Endif
! CLOSING IF for tot=2
          ENDIF


!c CLOSING Do i=1,dim-1 and Do j=i+1,dim
         enddo
       enddo
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!  END of SUBROUTINE that constructs the XXZ HAMILTONIAN
      return
      end subroutine HamXXZ
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc




!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to write the SINGLE-DEFECT HAMILTONIAN
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
      subroutine HamDEF()

      use variables
      implicit none

      INTEGER (kind=4) :: tot,DifferentSite(ChainSize)


! PARAMETERS FOR THE HAMILTONIAN
      defMid = 0.9d0
      lambda = 0.0d0
      defOnsite = 0.0d0


! INITIALIZATION
      Do i=1,dim
        Do j=1,dim
          Vec(i,j)=0.0d0
        Enddo
      Enddo

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! DIAGONAL ELEMENTS ************************************************************
      Do i=1,dim
! defect in the MIDDLE and at the EDGE
        Vec(i,i)=Vec(i,i)+(defMid/2.0d0)*( (-1.0d0)**(1+basis(i,ChainSize/2)) )
        Vec(i,i)=Vec(i,i)+(defBorder/2.0d0)*( (-1.0d0)**(1+basis(i,1)) )
! NN ISING interaction
        Do j=1,ChainSize-1
        Vec(i,i)=Vec(i,i)+(Jz/4.d0)*(-1.0d0)**(basis(i,j)+basis(i,j+1))
        enddo
        EnIni(i)=dabs(Vec(i,i))
! CLOSING i=1,dim
      enddo
! END of DIAGONAL **************************************************************
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! OFF-DIAGONAL ELEMENTS ********************************************************
      Do i = 1, dim-1
        Do j = i+1, dim

          tot = 0
          Do k = 1, ChainSize
            DifferentSite(k)=0
```

```fortran
        Enddo

        Do k = 1, ChainSize
         If( basis(i,k).ne.basis(j,k) ) then
         tot = tot + 1
         DifferentSite(tot)=k
         Endif
        Enddo

        IF(tot.EQ.2) then
!!!!!!!!! NN flip-flop
        IF((DifferentSite(2) - DifferentSite(1)).EQ.1) then
         Vec(i,j)=Vec(i,j)+Jxy/2.0d0
         Vec(j,i)=Vec(i,j)
        Endif
! CLOSING IF for tot=2
        ENDIF

!c CLOSING Do i=1,dim-1 and Do j=i+1,dim
       enddo
      enddo
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!  END of SUBROUTINE that constructs the SINGLE-DEFECT HAMILTONIAN
      return
      end subroutine HamDEF
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc




!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to write the NNN HAMILTONIAN
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine HamNNN()

      use variables
      implicit none

      INTEGER (kind=4) :: tot,DifferentSite(ChainSize)


! PARAMETERS FOR THE HAMILTONIAN
```

```fortran
      defMid = 0.0d0
      lambda = 1.0d0
      defOnsite = 0.0d0


! INITIALIZATION
      Do i=1,dim
       Do j=1,dim
        Vec(i,j)=0.0d0
       Enddo
      Enddo

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! DIAGONAL ELEMENTS *******************************************************
      Do i=1,dim
! defect at the EDGE
       Vec(i,i)=Vec(i,i)+(defBorder/2.0d0)*( (-1.0d0)**(1+basis(i,1)) )
! NN ISING interaction
       Do j=1,ChainSize-1
       Vec(i,i)=Vec(i,i)+(Jz/4.d0)*(-1.0d0)**(basis(i,j)+basis(i,j+1))
       enddo
! NNN ISING interaction
       Do j=1,ChainSize-2
       Vec(i,i)=Vec(i,i)+lambda*(Jz/4.d0)*(-1.0d0)**(basis(i,j)+basis(i,j+2))
       enddo
! Diagonal elements
       EnIni(i)=dabs(Vec(i,i))
! CLOSING i=1,dim
      enddo
! END of DIAGONAL *******************************************************
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! OFF-DIAGONAL ELEMENTS *************************************************
      Do i = 1, dim-1
       Do j = i+1, dim

        tot = 0
        Do k = 1, ChainSize
         DifferentSite(k)=0
        Enddo

        Do k = 1, ChainSize
         If( basis(i,k).ne.basis(j,k) ) then
         tot = tot + 1
```

```fortran
            DifferentSite(tot)=k
            Endif
          Enddo

          IF(tot.EQ.2) then
!!!!!!!!! NN flip-flop
          IF((DifferentSite(2) - DifferentSite(1)).EQ.1) then
            Vec(i,j)=Vec(i,j)+Jxy/2.0d0
            Vec(j,i)=Vec(i,j)
          Endif
!!!!!!!!! NNN flip-flop
          IF((DifferentSite(2) - DifferentSite(1)).EQ.2) then
            Vec(i,j)=Vec(i,j)+lambda*Jxy/2.0d0
            Vec(j,i)=Vec(i,j)
          Endif
! CLOSING IF for tot=2
          ENDIF

!c CLOSING Do i=1,dim-1 and Do j=i+1,dim
        enddo
      enddo
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!  END of SUBROUTINE that constructs the NNN HAMILTONIAN
      return
      end subroutine HamNNN
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc




!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to write the HAMILTONIAN with ONSITE DISORDER
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine HamDISO()

      use variables
      implicit none

      INTEGER (kind=4) :: tot,DifferentSite(ChainSize)
      real (kind=8), dimension(ChainSize) :: rar
      real (kind=8) :: defAll
```

```
!ccccccccccc
!ccccccccccc  This Hamiltonian has CLOSED BOUNDARIES ccccccccccccccc
!ccccccccccc

! PARAMETERS FOR THE HAMILTONIAN
      defMid = 0.0d0
      lambda = 0.0d0
      defAll = 0.5d0

! LOOP FOR THE VALUES OF THE ONSITE DISORDER: Zeeman splittings
      Do j=1,ChainSize
        rar(j) = defAll*rand()/2.0d0
      Enddo

! INITIALIZATION
      Do i=1,dim
        Do j=1,dim
          Vec(i,j)=0.0d0
        Enddo
      Enddo

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! DIAGONAL ELEMENTS ************************************************************
      Do i=1,dim
        Do j=1,ChainSize-1
! ONSITE DISORDER
        Vec(i,i)=Vec(i,i)+rar(j)*( (-1.0d0)**(1+basis(i,j)) )
! NN ISING interaction
        Vec(i,i)=Vec(i,i)+(Jz/4.d0)*(-1.0d0)**(basis(i,j)+basis(i,j+1))
        enddo
! ONSITE DISORDER for the last site
        Vec(i,i)=Vec(i,i)+rar(ChainSize)*( (-1.0d0)**(1+basis(i,ChainSize)) )
! NN ISING for CLOSED boundaries
        Vec(i,i)=Vec(i,i)+(Jz/4.d0)*(-1.0d0)**(basis(i,ChainSize)+basis(i,1))
! ENERGIES OF THE INITIAL STATES, when they are BASIS VECTORS
        EnIni(i)=dabs(Vec(i,i))
! CLOSING i=1,dim
      enddo
! END of DIAGONAL **************************************************************
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! OFF-DIAGONAL ELEMENTS ********************************************************
```

```
      Do i = 1, dim-1
       Do j = i+1, dim

         tot = 0
         Do k = 1, ChainSize
           DifferentSite(k)=0
         Enddo

         Do k = 1, ChainSize
           If( basis(i,k).ne.basis(j,k) ) then
           tot = tot + 1
           DifferentSite(tot)=k
           Endif
         Enddo

         IF(tot.EQ.2) then
!cccccccccccccccc NN flip-flop ccccccccccccccccccc
         IF((DifferentSite(2) - DifferentSite(1)).EQ.1) then
           Vec(i,j)=Vec(i,j)+Jxy/2.0d0
           Vec(j,i)=Vec(i,j)
         Endif
!ccccccccccccccc CLOSED BOUNDARIES ccccccccccccccccc
         IF((DifferentSite(2) - DifferentSite(1)).EQ.(ChainSize-1)) then
           Vec(i,j)=Vec(i,j)+Jxy/2.0d0
           Vec(j,i)=Vec(i,j)
         Endif
! CLOSING IF for tot=2
         ENDIF

!c CLOSING Do i=1,dim-1 and Do j=i+1,dim
       enddo
      enddo
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


!  END of SUBROUTINE that constructs the HAMILTONIAN with ONSITE DISORDER
      return
      end subroutine HamDISO
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc





!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
! SUBROUTINE FOR DIAGONALIZATION
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine Diagonalizing()

      use variables
      implicit none


! Eig     ccccccccccccccccccccccccccccccccc
       saiE='Eig_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'.dat'
       OPEN(unit=40, FILE=saiE,STATUS='UNKNOWN')


       allocate(work(7*dim))
       CALL DSYEV('V','U',dim,Vec,dim,Eig,WORK,7*dim,INFO)
       deallocate(work)

       Do i=1,dim
        write(40,*) Eig(i)
       Enddo


      close(40)

!  END of SUBROUTINE that diagonalizes the HAMILTONIAN
      return
      end subroutine Diagonalizing
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to write the DOS
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine DOS()

      use variables
      implicit none

       real (kind=8) :: Emin,Emax,bin
       INTEGER (kind=4), PARAMETER :: Nbin = 40
       real (kind=8), dimension(Nbin+1) :: Eint
       INTEGER (kind=4), dimension(Nbin) :: NinBin
```

```
! DOS     ccccccccccccccccccccccccccccccccccccc

saiDOS='DOS_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'.dat'
      OPEN(unit=42, FILE=saiDOS,STATUS='UNKNOWN')



      Emin=Eig(1)
      Emax=Eig(dim)
      bin = (Emax-Emin)/dfloat(Nbin-1)
      Do i=1,Nbin+1
        Eint(i)=Emin-bin-bin/2.0d0 + bin*dfloat(i)
      Enddo

      Do i=1,Nbin
        NinBin(i) = 0
      Enddo

      Do i=1,dim
       do k=1,Nbin
         If(Eig(i) >= Eint(k) .AND. Eig(i) < Eint(k+1)) then
         NinBin(k) = NinBin(k)+1
         Endif
       enddo
      Enddo

! Normalized density of states
       write(42,*) Eint(1),0.0d0
      Do k=1,Nbin
       write(42,*) Eint(k),dfloat(NinBin(k))/(bin*dfloat(dim))
       write(42,*) Eint(k+1),dfloat(NinBin(k))/(bin*dfloat(dim))
      Enddo
       write(42,*) Eint(Nbin+1),0.0d0


     close(42)


!  END of SUBROUTINE that gets the DOS
     return
     end subroutine DOS
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to compute the LEVEL SPACING DISTRIBUTION
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine LevSpDist()

      use variables
      implicit none

! 10% of discarded eigenvalues from the borders
      INTEGER (kind=4) :: percentage
! so we start counting eigenvalues at "half" and finish at "dim-half"
      INTEGER (kind=4) :: half
! the total number of spacings is then
      INTEGER (kind=4) :: dimSp
      real (kind=8), dimension(:), allocatable :: spacing
! using "ten" spacings (or "ten"+1 levels) for each block
      INTEGER (kind=4), PARAMETER :: ten = 10
      INTEGER (kind=4) :: dimBloc

      real (kind=8) :: average
      real (kind=8), PARAMETER :: spcmin = 0.0d0
      real (kind=8), PARAMETER :: spcmax = 8.0d0
      real (kind=8), PARAMETER :: binS = 0.1d0
      INTEGER (kind=4), PARAMETER :: NofBINs = 80
      INTEGER (kind=4), PARAMETER :: NofBINsPlus = 81
      real (kind=8), dimension(NofBINs) :: Nhist
      real (kind=8), dimension(NofBINsPlus) :: SPChist
      real (kind=8) :: normaliza


! P(s)     ccccccccccccccccccccccccccccccccccc
      saiPs='Ps_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'.dat'
      OPEN(unit=44, FILE=saiPs,STATUS='UNKNOWN')


      percentage = dim/10
      half = percentage/2
      dimSp = dim-percentage
      dimBloc = (dim-percentage)/ten
      allocate(spacing(dimSp))

! ---------------------------------
! For the Histogram
```

```
      SPChist(1)=spcmin
      Do i=1,NofBINs
        SPChist(i+1)=SPChist(i)+binS
        Nhist(i)=0.0d0
      Enddo
! ---------------------------------


! ---------------------------------
! Unfolded Spacings
      Do j=1,dimBloc
        average=(Eig(half+10*j)-Eig(half+10*(j-1)))/dfloat(ten)
        Do i=1+10*(j-1),10*j
      spacing(i)=(Eig(half+i)-Eig(half-1+i))/average
        Enddo
      Enddo
! ---------------------------------



! ---------------------------------
! Histogram
      Do k=1,10*dimBloc
        Do j=1,NofBINs
         If(spacing(k) >= SPChist(j) .AND. spacing(k) < SPChist(j+1)) then
           Nhist(j) = Nhist(j) + 1.0d0
         Endif
        Enddo
      Enddo
! ---------------------------------



! ---------------------------------
! Normalization
      normaliza=0.0d0
      Do i=1,NofBINs
        normaliza=normaliza+binS*Nhist(i)
      Enddo
! ---------------------------------



! ---------------------------------
! Output
       write(44,*) SPChist(1),0.0d0
```

```
      Do i=1,NofBINs
       write(44,*) SPChist(i),Nhist(i)/normaliza
       write(44,*) SPChist(i+1),Nhist(i)/normaliza
      Enddo
! ------------------------------------

      close(44)

      deallocate(spacing)

!  END of SUBROUTINE that computes the level spacing distribution
     return
     end subroutine LevSpDist
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc




!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to compute the AVERAGE of the ratio of consecutive levels
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

     subroutine RatioLev()

     use variables
     implicit none

     real (kind=8) :: rtilde,sn,sn1

! P(r)     ccccccccccccccccccccccccccccccccc

saiPr='rTilde_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'.dat'
      OPEN(unit=46, FILE=saiPr,STATUS='UNKNOWN')


     rtilde=0.0d0
     Do i=3,dim
      sn = ( Eig(i)-Eig(i-1) )/( Eig(i-1)-Eig(i-2) )
      sn1 =( Eig(i-1)-Eig(i-2) )/( Eig(i)-Eig(i-1) )
      If( sn.lt.sn1) then
      rtilde = rtilde + sn
      else
      rtilde = rtilde + sn1
      Endif
```

```
      Enddo
      write(46,*) rtilde/(dfloat(dim-2))


     close(46)

!  END of SUBROUTINE that computes the ratio of consecutive levels
      return
      end subroutine RatioLev
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc




!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! *************** PR, SHANNON of EIGENSTATE ************************
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine VecPRSh()
      use variables
      implicit none

      real (kind=8) :: IPR,SH


! PRSh     ccccccccccccccccccccccccccccccccc

saiPRs='PRSh_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'.dat'
      OPEN(unit=48, FILE=saiPRs,STATUS='UNKNOWN')

! COMPUTATIONS
      DO i=1,dim
        IPR=0.0d0
        SH=0.0d0
       Do j=1,dim
         IPR = IPR + Vec(j,i)**4
         SH = SH - (Vec(j,i)**2)*Log(Vec(j,i)**2)
       Enddo
       write(48,*) Eig(i),1.0d0/IPR,SH
      ENDDO

! CLOSE file
      close(48)


!c END of SUBROUTINE for PR and Shannon entropy
```

```
      return
      end subroutine VecPRSh
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc




!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! *************** EEV: DIAGONAL of Obs ***************************
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine EEV()
      use variables
      implicit none

      real (kind=8) :: obz
      integer (kind=4) :: auxl


! EEV     ccccccccccccccccccccccccccccccccc

saiEEV='EEV_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'.dat'
      OPEN(unit=60, FILE=saiEEV,STATUS='UNKNOWN')

! COMPUTATION of <alpha|S^z_{L/2}.S^z_{L/2+1}|alpha>
      DO i=1,dim
        obz=0.0d0
       Do j=1,dim
         auxl=basis(j,ChainSize/2)+basis(j,ChainSize/2+1)
         obz=obz+0.25d0*(Vec(j,i)**2)*( (-1.0d0)**auxl )
       Enddo
       write(60,*) Eig(i),obz
      ENDDO

! CLOSE file
      close(60)



!c END of SUBROUTINE for EEV
      return
      end subroutine EEV
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! ********** OFF-DIAGONAL ETH ***************************
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine OFFO()
      use variables
      implicit none

      real (kind=8) :: obz


! OFF     ccccccccccccccccccccccccccccccccc

saiOFF='Oab_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'.dat'
      OPEN(unit=62, FILE=saiOFF,STATUS='UNKNOWN')

! COMPUTATION of <beta|sigma^z_{L/2}|alpha>
! COMPUTATIONS: i=|alpha>, j=|beta>, k=|Fock>
      DO i = dim/2-(totOab/2), dim/2+(totOab/2)-1
        Do j = i+1, dim/2+(totOab/2)
          obz=0.0d0
          Do k = 1, dim
       obz = obz + Vec(k,i)*Vec(k,j)*( (-1.0d0)**(1+basis(k,ChainSize/2)) )
          Enddo
          write(62,*) obz
        Enddo
      ENDDO

! CLOSE file
      close(62)



!c END of SUBROUTINE for the OFF-DIAGONAL ETH
     return
     end subroutine OFFO
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!
```

```
! SUBROUTINE to obtain the INITIAL STATES with Eini~0
!
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine InitialStateE0()
      use variables
      implicit none

      integer (kind=4) :: auxSub
      real (kind=8), dimension(:), allocatable :: XVALT
      integer (kind=4), dimension(:), allocatable :: IMULT

      allocate(XVALT(dim))
      allocate(IMULT(dim))

      do kk=1,dim
        XVALT(kk)=EnIni(kk)
      enddo

! ORDERING the ABSOLUTE values of H(i,i).
      call MRGRNK (XVALT, IMULT)

! k=1 is the CLOSEST value to ZERO.
      Do kk = 1, totIni
        IniSt(kk) = IMULT(kk)
      Enddo

      deallocate(XVALT)
      deallocate(IMULT)


!  END of SUBROUTINE that finds the initial states
      return
      end subroutine InitialStateE0
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to obtain <Eo>, sig^2, IPRo, PRsat and OUTPUT Calpha
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine CalphaIPRo()
      use variables
      implicit none
```

```
      real (kind=8) :: Eini,Esqu,IPRo,sigSq
      real (kind=8) :: teA,teAsq,teB, IPRsat


! Calpha     ccccccccccccccccccccccccccccccccccc

saiCa='Calpha_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'AveI'/
/ain//'.dat'
      OPEN(unit=50, FILE=saiCa,STATUS='UNKNOWN')

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Calpha's are the ROW of the HAMILTONIAN after DIAGONALIZATION
      Do j=1,totINI
        Calpha(j,:)=Vec(IniSt(j),:)
      Enddo
      Do i=1,dim
        write(50,*) Eig(i), (Calpha(j,i), j=1,totINI)
      Enddo
      close(50)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!



!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! ADITIONAL USEFUL INFORMATION
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Info     ccccccccccccccccccccccccccccccccccc

saiInf='InfoIni_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'AveI'//a
in//'.dat'
      OPEN(unit=51, FILE=saiInf,STATUS='UNKNOWN')


      Do j=1, totINI
       Eini=0.0d0
       Esqu=0.0d0
       IPRo=0.0d0
       teA=0.0d0
       teB=0.0d0
       Do jj=1,dim
         Eini = Eini + (Calpha(j,jj)**2)*Eig(jj)
         Esqu = Esqu + (Calpha(j,jj)**2)*(Eig(jj)**2)
         IPRo = IPRo + (Calpha(j,jj)**4)
       teAsq=0.0d0
         Do kk=1,dim
```

```
        teAsq=teAsq+ (Calpha(j,kk)**2)*(Vec(jj,kk)**2)
        teB=teB+ (Calpha(j,jj)**4)*(Vec(kk,jj)**4)
         Enddo
        teA=teA+teASq**2
        Enddo
        sigSq = Esqu - Eini**2
        IPRsat=2.0d0*teA-teB
! OUTPUT
        write(51,*) '# basis number'
        write(51,*) IniSt(j)
        write(51,*)
        write(51,*) 'Eini from diagonal and Eini computed: they should match'
        write(51,*) EnIni(IniSt(j)),Eini
        write(51,*)
        write(51,*) '# spin configuration'
        write(51,*) (basis(IniSt(j),k),k=1,ChainSize)
        write(51,*)
        write(51,*) 'Gamma^2 '
        write(51,*) sigSq
        write(51,*)
        write(51,*) 'IPRo: infinite-time average of SP(t)'
        write(51,*) IPRo
        write(51,*)
        write(51,*) 'IPRsat: infinite time average of IPR(t)'
        write(51,*) IPRsat
        write(51,*)
       Enddo
       close(51)



!  END of SUBROUTINE that finds <Eo>, sig^2, IPRo and OUTPUT Calpha's
      return
      end subroutine CalphaIPRo
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to COMPUTE SP(t)
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

       subroutine TimeEvolve()
       use variables
       implicit none
```

```
!-------------------------------------------------------------
! for the TIME EVOLUTION
      real (kind=8) :: auxCos,auxSin
      real (kind=8) :: SPave



! SP     ccccccccccccccccccccccccccccccccccccc

saiSP='SP_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'AveI'//ain/
/'.dat'
      OPEN(unit=54, FILE=saiSP,STATUS='UNKNOWN')


!----------
! DYNAMICS
!----------
      DO tt=1,totTime

!-------------------------------------------------------------
! EVOLUTION in the ENERGY EIGENBASIS and SURVIVAL PROBABILITY
!-------------------------------------------------------------
        SPave=0.0d0
        Do j=1,totINI
          auxCos=0.0d0
          auxSin=0.0d0
          Do i=1,dim
        auxCos=auxCos+(Calpha(j,i)**2 )*dcos(time(tt)*Eig(i))
        auxSin=auxSin+(Calpha(j,i)**2 )*dsin(time(tt)*Eig(i))
          Enddo
          SPave = SPave + auxCos**2 + auxSin**2
        Enddo
        write(54,*) time(tt), SPave/DBLE(totINI)

! end the TIME-LOOP
      ENDDO



! CLOSING
      close(54)



!  END of SUBROUTINE that computes SP(t)
```

```
      return
      end subroutine TimeEvolve
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc









!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to COMPUTE SP(t) and IPR(t)
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine EvolveSPandIPR()
      use variables
      implicit none

!---------------------------------------------------------------
! for the TIME EVOLUTION: average over totINI initial states
! for <SP(t)>
      real (kind=8) :: auxCos,auxSin
      real (kind=8) :: SPave
! for <PR(t)> and <Shannon(t)>
      real (kind=8), dimension(:,:), allocatable :: CosE0,SinE0
      real (kind=8), dimension(:,:), allocatable :: sCosE0,sSinE0
      real (kind=8) :: IPRt, auxDyn
      real (kind=8) :: IPRave

! ALLOCATE
      allocate(CosE0(totINI,dim))
      allocate(SinE0(totINI,dim))
      allocate(sCosE0(totINI,dim))
      allocate(sSinE0(totINI,dim))


! SP     ccccccccccccccccccccccccccccccccc

saiSP='SP_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'AveI'//ain/
/'.dat'
      OPEN(unit=54, FILE=saiSP,STATUS='UNKNOWN')
```

```
! IPR(t)     ccccccccccccccccccccccccccccccccccc

saiPt='IPRt_L'//si//'u'//uu//'Jz'//zF//'dF'//dF//'lF'//lF//'hF'//hF//'bF'//bF//'AveR'//aa//'Avel'//ain
//'.dat'
      OPEN(unit=56, FILE=saiPt,STATUS='UNKNOWN')


!----------
! DYNAMICS
!----------
      DO tt=1,totTime

! Initialization for each instant of time
        SPave=0.0d0
        IPRave=0.0d0

!-------------------------------------------------------
! EVOLUTION in the ENERGY EIGENBASIS and SURVIVAL PROBABILITY
!-------------------------------------------------------
        Do j=1,totINI
          auxCos=0.0d0
          auxSin=0.0d0
          Do i=1,dim
! for SP(t)
        auxCos=auxCos+(Calpha(j,i)**2 )*dcos(time(tt)*Eig(i))
        auxSin=auxSin+(Calpha(j,i)**2 )*dsin(time(tt)*Eig(i))
! for IPR(t)
        CosE0(j,i)=   Calpha(j,i)*dcos( time(tt)*Eig(i))
        SinE0(j,i)= - Calpha(j,i)*dsin( time(tt)*Eig(i))
          Enddo
          SPave = SPave + auxCos**2 + auxSin**2
        Enddo
!-------------
! OUTPUT SP(t)
!-------------
        write(54,*) time(tt), SPave/DBLE(totINI)


!-------------------------------------------------------
! BACK TO SITE-BASIS: basis of spin configurations, such as 101100
!-------------------------------------------------------
      Do j=1,totINI
       call dgemv('n',dim,dim,1.0d0,Vec,dim,CosE0(j,:),1,0.0d0,sCosE0(j,:),1)
       call dgemv('n',dim,dim,1.0d0,Vec,dim,SinE0(j,:),1,0.0d0,sSinE0(j,:),1)
      Enddo
```

```
!------------------------------
! IPR(t)
!------------------------------
      Do j=1,totINI ! each initial state
       IPRt=0.0d0
       Do i=1,dim   ! sum over all basis vectors
         auxDyn=(sCosE0(j,i)**2 + sSinE0(j,i)**2) !=|Cn(t)|^2
         IPRt = IPRt + (sCosE0(j,i)**2 + sSinE0(j,i)**2)**2
       Enddo        ! close loop of basis vectors
       IPRave = IPRave + IPRt
      Enddo         ! close loop for initial state
!--------------
! OUTPUT IPR(t)
!--------------
      write(56,*) time(tt), IPRave/DBLE(totINI)



! end the TIME-LOOP
      ENDDO



! CLOSING
      close(54)
      close(56)

! DEALLOCATE
      deallocate(CosE0)
      deallocate(SinE0)
      deallocate(sCosE0)
      deallocate(sSinE0)



!  END of SUBROUTINE that computes SP(t) and IPR(t)
      return
      end subroutine EvolveSPandIPR
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!               EXTERNAL SUBROUTINES
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


      Subroutine mrgrnk (XDONT, IRNGT)
      use variables
      implicit none
! ____________________________________________________________________
!   MRGRNK = Merge-sort ranking of an array
!   For performance reasons, the first 2 passes are taken
!   out of the standard loop, and use dedicated coding.
! ____________________________________________________________________
! ____________________________________________________________________
      Real (kind=8), Dimension (dim), Intent (In) :: XDONT
      Integer (kind=4), Dimension (dim), Intent (Out) :: IRNGT
! ____________________________________________________________________
      Real (kind=8) :: XVALA, XVALB
!
      Integer (kind=4), Dimension (SIZE(IRNGT)) :: JWRKT
      Integer (kind=4) :: LMTNA, LMTNC, IRNG1, IRNG2
      Integer (kind=4) :: NVAL, IIND, IWRKD, IWRK, IWRKF, JINDA, IINDA, IINDB
!
      NVAL = Min (SIZE(XDONT), SIZE(IRNGT))
      Select Case (NVAL)
      Case (:0)
        Return
      Case (1)
        IRNGT (1) = 1
        Return
      Case Default
        Continue
      End Select
!
!  Fill-in the index array, creating ordered couples
!
      Do IIND = 2, NVAL, 2
```

```
        If (XDONT(IIND-1) <= XDONT(IIND)) Then
          IRNGT (IIND-1) = IIND - 1
          IRNGT (IIND) = IIND
        Else
          IRNGT (IIND-1) = IIND
          IRNGT (IIND) = IIND - 1
        End If
      End Do
      If (Modulo(NVAL, 2) /= 0) Then
        IRNGT (NVAL) = NVAL
      End If
!
!  We will now have ordered subsets A - B - A - B - ...
!  and merge A and B couples into     C   -   C   - ...
!
      LMTNA = 2
      LMTNC = 4
!
!  First iteration. The length of the ordered subsets goes from 2 to 4
!
      Do
        If (NVAL <= 2) Exit
!
!   Loop on merges of A and B into C
!
        Do IWRKD = 0, NVAL - 1, 4
          If ((IWRKD+4) > NVAL) Then
            If ((IWRKD+2) >= NVAL) Exit
!
!   1 2 3
!
            If (XDONT(IRNGT(IWRKD+2)) <= XDONT(IRNGT(IWRKD+3))) Exit
!
!   1 3 2
!
            If (XDONT(IRNGT(IWRKD+1)) <= XDONT(IRNGT(IWRKD+3))) Then
              IRNG2 = IRNGT (IWRKD+2)
              IRNGT (IWRKD+2) = IRNGT (IWRKD+3)
              IRNGT (IWRKD+3) = IRNG2
!
!   3 1 2
!
            Else
              IRNG1 = IRNGT (IWRKD+1)
              IRNGT (IWRKD+1) = IRNGT (IWRKD+3)
              IRNGT (IWRKD+3) = IRNGT (IWRKD+2)
```

```
            IRNGT (IWRKD+2) = IRNG1
          End If
          Exit
        End If
!
!  1 2 3 4
!
        If (XDONT(IRNGT(IWRKD+2)) <= XDONT(IRNGT(IWRKD+3))) Cycle
!
!  1 3 x x
!
        If (XDONT(IRNGT(IWRKD+1)) <= XDONT(IRNGT(IWRKD+3))) Then
          IRNG2 = IRNGT (IWRKD+2)
          IRNGT (IWRKD+2) = IRNGT (IWRKD+3)
          If (XDONT(IRNG2) <= XDONT(IRNGT(IWRKD+4))) Then
!  1 3 2 4
            IRNGT (IWRKD+3) = IRNG2
          Else
!  1 3 4 2
            IRNGT (IWRKD+3) = IRNGT (IWRKD+4)
            IRNGT (IWRKD+4) = IRNG2
          End If
!
!  3 x x x
!
        Else
          IRNG1 = IRNGT (IWRKD+1)
          IRNG2 = IRNGT (IWRKD+2)
          IRNGT (IWRKD+1) = IRNGT (IWRKD+3)
          If (XDONT(IRNG1) <= XDONT(IRNGT(IWRKD+4))) Then
            IRNGT (IWRKD+2) = IRNG1
            If (XDONT(IRNG2) <= XDONT(IRNGT(IWRKD+4))) Then
!  3 1 2 4
              IRNGT (IWRKD+3) = IRNG2
            Else
!  3 1 4 2
              IRNGT (IWRKD+3) = IRNGT (IWRKD+4)
              IRNGT (IWRKD+4) = IRNG2
            End If
          Else
!  3 4 1 2
            IRNGT (IWRKD+2) = IRNGT (IWRKD+4)
            IRNGT (IWRKD+3) = IRNG1
            IRNGT (IWRKD+4) = IRNG2
          End If
        End If
```

```
        End Do
!
!  The Cs become As and Bs
!
        LMTNA = 4
        Exit
      End Do
!
!  Iteration loop. Each time, the length of the ordered subsets
!  is doubled.
!
      Do
        If (LMTNA >= NVAL) Exit
        IWRKF = 0
        LMTNC = 2 * LMTNC
!
!   Loop on merges of A and B into C
!
        Do
          IWRK = IWRKF
          IWRKD = IWRKF + 1
          JINDA = IWRKF + LMTNA
          IWRKF = IWRKF + LMTNC
          If (IWRKF >= NVAL) Then
            If (JINDA >= NVAL) Exit
            IWRKF = NVAL
          End If
          IINDA = 1
          IINDB = JINDA + 1
!
!   Shortcut for the case when the max of A is smaller
!   than the min of B. This line may be activated when the
!   initial set is already close to sorted.
!
!         IF (XDONT(IRNGT(JINDA)) <= XDONT(IRNGT(IINDB))) CYCLE
!
!  One steps in the C subset, that we build in the final rank array
!
!  Make a copy of the rank array for the merge iteration
!
          JWRKT (1:LMTNA) = IRNGT (IWRKD:JINDA)
!
          XVALA = XDONT (JWRKT(IINDA))
          XVALB = XDONT (IRNGT(IINDB))
!
          Do
```

```
            IWRK = IWRK + 1
!
!  We still have unprocessed values in both A and B
!
            If (XVALA > XVALB) Then
               IRNGT (IWRK) = IRNGT (IINDB)
               IINDB = IINDB + 1
               If (IINDB > IWRKF) Then
!  Only A still with unprocessed values
                  IRNGT (IWRK+1:IWRKF) = JWRKT (IINDA:LMTNA)
                  Exit
               End If
               XVALB = XDONT (IRNGT(IINDB))
            Else
               IRNGT (IWRK) = JWRKT (IINDA)
               IINDA = IINDA + 1
               If (IINDA > LMTNA) Exit! Only B still with unprocessed values
               XVALA = XDONT (JWRKT(IINDA))
            End If
!
         End Do
      End Do
!
!  The Cs become As and Bs
!
      LMTNA = 2 * LMTNA
   End Do
!
   Return
!
    End Subroutine mrgrnk

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```