```
!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) Gives as output the (Calpha^ini) for an initial state with
!     energy <Psi0|H|Psi0> in the middle of the spectrum
!  7) Computes the LDOS  --- NOT YET IN THE CODE!!
!  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
!
!
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!            VARIABLES to be used in the whole code
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      module variables

      implicit none
```

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

! Dimension of the matrix is "dim"
      integer (kind=4), PARAMETER :: dim=1000
! 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
! How many times to be used from FILE 'GOETimeIncrements.dat': MAX=5000
      integer (kind=4), PARAMETER :: totTime=5000

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


! 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


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

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


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

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


!-------------------------------------------------------------
! For the OUTPUT files
      character(len=4) dd,aa,ain
      character(len=70) saiE,saiDOS,saiPs,saiPr,saiPRs
      character(len=70) saiCa,saiInf, saiL,saiSP,saiPt

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

      end module

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



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!            Program starts here
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      Program GOEcode

      use variables
      implicit none


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


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! OUTPUT FILES
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
      write(dd,10) dim
      write(aa,10) ave
      write(ain,10) totIni
 10    format(i4.4)

! PLACING THE NAMES OF THE OUTPUT FILES HERE INSTEAD OF
! INSIDE THE SUBROUTINES, AS IT IS DONE NOW,
! IS A GOOD IDEA WHEN DEALING WITH ENSEMBLES OF RANDOM MATRICES
! AND AVERAGES OVER THOSE ENSEMBLES
!
! NOTE: This code is written for a single realization of a
! random matrix. If you plan to deal with an ensemble,
! you will need to modify it to add a loop over random realizations.
!
! Eig     ccccccccccccccccccccccccccccccccccc
!       saiE='Eig_GOE_D'//dd//'AveR'//aa//'.dat'
!       OPEN(unit=40, FILE=saiE,STATUS='UNKNOWN')
! DOS     ccccccccccccccccccccccccccccccccccc
!       saiDOS='DOS_GOE_D'//dd//'AveR'//aa//'.dat'
!       OPEN(unit=42, FILE=saiDOS,STATUS='UNKNOWN')
! P(s)     ccccccccccccccccccccccccccccccccccc
!       saiPs='Ps_GOE_D'//dd//'AveR'//aa//'.dat'
!       OPEN(unit=44, FILE=saiPs,STATUS='UNKNOWN')
! P(r)     ccccccccccccccccccccccccccccccccccc
!       saiPr='rTilde_GOE_D'//dd//'AveR'//aa//'.dat'
!       OPEN(unit=46, FILE=saiPr,STATUS='UNKNOWN')
! PRSh     ccccccccccccccccccccccccccccccccccc
!       saiPRs='PRSh_GOE_D'//dd//'AveR'//aa//'.dat'
!       OPEN(unit=48, FILE=saiPRs,STATUS='UNKNOWN')
! Calpha     ccccccccccccccccccccccccccccccccccc
!       saiCa='Calpha_GOE_D'//dd//'AveR'//aa//'AveI'//ain//'.dat'
!       OPEN(unit=50, FILE=saiCa,STATUS='UNKNOWN')
! LDOS     ccccccccccccccccccccccccccccccccccc
!       saiL='LDOS_GOE_D'//dd//'AveR'//aa//'AveI'//ain//'.dat'
!       OPEN(unit=52, FILE=saiL,STATUS='UNKNOWN')
! SP     ccccccccccccccccccccccccccccccccccc
!       saiSP='SP_GOE_D'//dd//'AveR'//aa//'AveI'//ain//'.dat'
!       OPEN(unit=54, FILE=saiSP,STATUS='UNKNOWN')


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! RANDOM NUMBERS
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      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



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c HAMILTONIAN
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      call HamiltonianGOE()


!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()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c ORDER H(i,i) and then select the INITIAL STATES
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!       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()

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


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

      call CalphaIPRo()



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! EVOLUTION of the INITIAL STATE
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
       call TimeEvolve()
! Subroutine that evolves the SP and the IPR
       call EvolveSPandIPR()


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c END END END END END END END END END END END END END
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!      STOP
      END



!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!ccccccccccccccccc FUNCTIONS FUNCTIONS FUNCTIONS ccccccccccccccccc
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

!  Functions to generate random numbers from uniform and Gaussian distributions
!  using Numerical Recipes "gasdev" routine.
!  rand() 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
      INTEGER (kind=4) :: iset
      REAL (kind=8) :: fac,gset,rsq,v1,v2
      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




!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccc SUBROUTINES SUBROUTINES SUBROUTINES cccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc


!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE to write the GOE HAMILTONIAN
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine HamiltonianGOE()
```

```
      use variables
      implicit none

! ---------------------------------------
! Random numbers from Gaussian distribution
       real (kind=8) :: gasdev
! ---------------------------------------

! Create a matrix M filled with random Gaussian numbers: mean=0, var=1
! Do (M+Transpose[M])/2

! DIAGONAL ELEMENTS
! For diagonal elements, that implies doing
!          Vec(i,i)=2.0d0*gasdev(seed)/2.0d0
! Diagonal elements now have variance=1; <Hii^2>=1
      Do i=1,dim
        Vec(i,i)=gasdev(seed)
        EnIni(i)=dabs(Vec(i,i))
      Enddo

! OFF-DIAGONAL ELEMENTS
! For off-diagonal elements, that implies doing
!          Vec(i,j)= ( gasdev(seed)+gasdev(seed) )/2.0d0
! Off-diagonal elements now have variance=1/2; <Hij^2>=1/2
! that's why we divide the random number by 2
      DO i=1,dim-1
       Do j=i+1,dim
         Vec(i,j) = ( gasdev(seed) + gasdev(seed) )/2.0d0
         Vec(j,i)=Vec(i,j)
       Enddo
      ENDDO
! ATTENTION!!
! We could get the same as above for the off-diagonal elements, by doing
!          Vec(i,j)= gasdev(seed)/sqrt(2.0d0)
! BUT NOT NOT NOT NOT !!
! NOT by doing Vec(i,j)= gasdev(seed)/2.0d0, which gives variance=1/4


!  END of SUBROUTINE that constructs the GOE HAMILTONIAN
      return
      end subroutine HamiltonianGOE
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! SUBROUTINE FOR DIAGONALIZATION
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine Diagonalizing()

      use variables
      implicit none


! Eig     ccccccccccccccccccccccccccccccccc
       saiE='Eig_GOE_D'//dd//'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     ccccccccccccccccccccccccccccccccccc
      saiDOS='DOS_GOE_D'//dd//'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
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



!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), PARAMETER :: percentage = dim/10
! so we start counting eigenvalues at "half" and finish at "dim-half"
      INTEGER (kind=4), PARAMETER :: half = percentage/2
! the total number of spacings that we need is "dim-percentage",
!       INTEGER (kind=4), PARAMETER :: dimSp = dim-percentage
!       real (kind=8), dimension(dimSp) :: spacing
! but I'll just use "dim"
      real (kind=8), dimension(dim) :: spacing
! using "ten" spacings (or "ten"+1 levels) for each block
      INTEGER (kind=4), PARAMETER :: ten = 10
      INTEGER (kind=4), PARAMETER :: dimBloc = (dim-percentage)/ten

      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)     ccccccccccccccccccccccccccccccccc
      saiPs='Ps_GOE_D'//dd//'AveR'//aa//'.dat'
      OPEN(unit=44, FILE=saiPs,STATUS='UNKNOWN')

! ---------------------------------
! 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)

!  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_GOE_D'//dd//'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_GOE_D'//dd//'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
```

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

      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
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc



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

```
      subroutine CalphaIPRo()
      use variables
      implicit none

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


! Calpha     ccccccccccccccccccccccccccccccccccc
      saiCa='Calpha_GOE_D'//dd//'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_GOE_D'//dd//'AveR'//aa//'AveI'//ain//'.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 EVOLVE the INITIAL STATE
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine TimeEvolve()
      use variables
      implicit none
```

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



! SP     ccccccccccccccccccccccccccccccccccccc
      saiSP='SP_GOE_D'//dd//'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 evolves the initial states
     return
     end subroutine TimeEvolve
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc




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

      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_GOE_D'//dd//'AveR'//aa//'AveI'//ain//'.dat'
      OPEN(unit=54, FILE=saiSP,STATUS='UNKNOWN')
! IPR(t)     ccccccccccccccccccccccccccccccccc
      saiPt='IPRt_GOE_D'//dd//'AveR'//aa//'AveI'//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
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc









!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

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