```
! ============================================================
! Programa ARM - [A]mazonas [R]odrigo [M]elissa
! Analise estatistica da conformacao em cadeias oligomericas
!
! Versao 8 / set.2010 (under dev.)
! Versao 1  : Jarlesson Amazonas
! Versao 2-7: Rodrigo Ramos e Melissa F. S. Pinto
! Versao 8  : Rodrigo Ramos e Jarlesson Amazonas
! Versao 9  : Rodrigo Ramos
! Versao 10 : Rodrigo Ramos
! Versao 11 : J. Amazonas & R. Ramos.
! ============================================================
!
! notas:
! * http://www.megasolutions.net/fortran/Epsilon,-Precision-or-Tiny_-64109.aspx

PROGRAM ang_anel

USE vectors
IMPLICIT NONE

integer::i,j,k,l,n,m,o
integer::atm1,atm2,atm5
integer::count

integer::nat,natol,ncadeias
integer::natmon,naneisolig

integer::uni,unit,unid,unida

integer, PARAMETER::natCARBmon = 8

REAL(8),dimension(:,:),allocatable::xm, ym, zm
REAL(8),dimension(:,:,:,:),allocatable:: R
real(8)::xmv2, ymv2, zmv2
real(8)::xmv4, ymv4, zmv4
real(8)::xmv6, ymv6, zmv6

TYPE (vector), DIMENSION (:,:),allocatable:: v_normal, v_eixo, vnn
TYPE (vector), DIMENSION (:,:,:,:),allocatable:: vrr
TYPE (vector), DIMENSION (:,:),allocatable::vdesl
TYPE (vector)::v1,v2

character(len=25)::nimp1, nimp2, nimp3, nimp4, nimp5, nimp6, nimp10
character(len=45)::arquivo,arquivo1,arquivo2

real(8), PARAMETER::PI= 3.141592654
REAL(8), DIMENSION (3) :: array_out


integer,dimension(:,:),allocatable::num
integer,dimension(:,:),allocatable::atom
character(len=10):: chainfile
real(8),dimension(:,:),allocatable::cx,cy,cz

REAL(8):: DELTAxm, DELTAym, DELTAzm

real(8):: alpha, beta, gamma, theta1,theta2, phi, delta, tau, lambda
```

```
! tratamento das condicoes periodicas

real(8) A(3,3),B(3,3)        !matriz de vetores da celula unitaria e inversa
integer(8) IPBX,IPBY,IPBZ

real(8),dimension(:,:),    allocatable:: sx,sy,sz          ! ...para posicoes
real(8):: ssx,ssy,ssz                                      ! ...para distancias
real(8):: xxm,yym,zzm

real(8),dimension(:,:),    allocatable:: vnnx,vnny,vnnz

double precision:: dotp,dotp1,dotp2

TYPE (vector) anelanel
TYPE (vector) v_comp, v_plano
real(8):: Lplanar, Lp
real(8):: Estlm, Estno

                                !mapa de rede topologica
INTEGER,dimension(:,:),allocatable:: sitio
INTEGER,dimension(:,:),allocatable:: conjuga
integer conjlgth,mant

real(8) limphi, limtheta,limR,limhpp,limtau,limdelta,limlambda
integer flgphi,flgtheta1,flgtheta2,flgR,flghpp,flgtau,flgdelta,flglambda

REAL(8) HOPPING,hpp,SITEENERGY

print *, '========================================='
print *, 'Statistical Analisis'
print *, 'ARM pack- [A]mazonas, [R]amos, [M]elissa'
print *, 'Version 10 / may.2011'
print *, '========================================='
print *, ''

print *, 'Reading info.stat'
open(10,file='info.stat')

read(10,*) natol          !numero de atomos de um oligomero
read(10,*) ncadeias       !numero de cadeias
read(10,*) naneisolig     !numero de aneis por oligomero
! cuttoffs para conexao (hopping) entre sitios
read(10,*) limR           ! raio de corte para avaliar angulos entre-aneis e hopping e etc... inter-cadeia
read(10,*) limhpp         ! corte no hopping (eV)// 8meV coerente c/ 5.5A

!read(10,*) Lplanar        !numero de aneis por oligomero

!limR     = 5.5   ! raio de corte para avaliar angulos entre-aneis e hopping e etc... inter-cadeia
!limhpp   = 0.008 ! corte no hopping (eV)// 8meV coerente c/ 5.5A

Lp = 6.667

if (naneisolig.eq.1) then

Lplanar = 2.80

else
```

```
Lplanar = Lp*(naneisolig - 1)

end if

print *, 'Lp=', Lp, 'Lplanar=', Lplanar

close(10)

!print*, LPlanar

!stop

print *,'\/\ ok'

open(12,file='MAT.in')

!lendo matriz de vetores da celula unitaria (A) - sim eh transposto assim mesmo =P

read(12,*) A(1,1),A(2,1),A(3,1)
read(12,*) A(1,2),A(2,2),A(3,2)
read(12,*) A(1,3),A(2,3),A(3,3)

close(12)

! invertendo matriz vetores da celula unitaria (B)

do i = 1, 3
   do j =1,3
      B(i,j)=A(i,j)
   end do
end do

call INV(B,3)

open(14,file='MAT.inv')

write(14,*) B(1,1),B(1,2),B(1,3)
write(14,*) B(2,1),B(2,2),B(2,3)
write(14,*) B(3,1),B(3,2),B(3,3)

close(14)

allocate(cx(ncadeias,natol))
allocate(cy(ncadeias,natol))
allocate(cz(ncadeias,natol))

cx = 0.0
cy = 0.0
cz = 0.0

allocate(atom(ncadeias,natol))   !tipo de atomo
allocate(num(ncadeias,natol))    !numero do atomo no filme

num  = 0.0

!READING
```

```
nat = 0
open(15,file='chains.in')   ! colocar arquivos do dir ./chains/
                        ! na lista 'chains.in'

!WARNING: isso pode ser arbitrario jah que o pdb pode ter campos extras
!         rode cautelosamente os scripts de separacao

do i=1,ncadeias

  read(15,'(a7)') chainfile   ! formato NNN.pdb
  open(16, file='./chainsout/'//chainfile)

  do j=1,natol
    read(16,*) nimp1, atom(i,j), nimp2, nimp2, cx(i,j), cy(i,j), cz(i,j), nimp4, nimp5, nimp6
    nat = nat + 1
    num(i,j) = nat
  end do

  close(16)

end do

close(15)

allocate(v_normal(ncadeias,naneisolig))
allocate(v_eixo(ncadeias,naneisolig))
allocate(vnn(ncadeias,naneisolig))
allocate(vdesl(ncadeias,naneisolig))

allocate(xm(ncadeias,naneisolig))
allocate(ym(ncadeias,naneisolig))
allocate(zm(ncadeias,naneisolig))

xm = 0.0
ym = 0.0
zm = 0.0

allocate(R(ncadeias,naneisolig,ncadeias,naneisolig))

R = 0.0

allocate(vnnx(ncadeias,naneisolig))
allocate(vnny(ncadeias,naneisolig))
allocate(vnnz(ncadeias,naneisolig))

vnnx = 0.0
vnny = 0.0
vnnz = 0.0

allocate(sitio(ncadeias,naneisolig))
allocate(conjuga(ncadeias,naneisolig))


!==================================
!Structural information
!==================================

print *, 'Colecting Structural information'
```

```
!print *, ''


print *, '-------------------------------------------------------'
print *, 'Notes:'
print *, '--> PPV phenil capped structure'
print *, '--> Each ring followed by vinil structures in pdb file'
print *, '--> 1st ring *without* its vinil structure'
print *, '--> hydrogen atoms excluded'
print *, '-------------------------------------------------------'

!calculating centers and normal vectors
!checado (vpython) / ok.
open(18,file='all.vec.rg.dat')                !coordenadas dos centros
do l = 1, ncadeias
  atm2 = 3
  atm5 = 6
  do m = 1, naneisolig

    xm(l,m)=( cx(l,atm2)+cx(l,atm5) )/2   !\vec centro
    ym(l,m)=( cy(l,atm2)+cy(l,atm5) )/2
    zm(l,m)=( cz(l,atm2)+cz(l,atm5) )/2

    xmv4=xm(l,m)-cx(l,atm2-1)       !\vec centro para 4 (at)
    ymv4=ym(l,m)-cy(l,atm2-1)
    zmv4=zm(l,m)-cz(l,atm2-1)

    xmv6=xm(l,m)-cx(l,atm5-2)       !\vec centro para 6 (at)
    ymv6=ym(l,m)-cy(l,atm5-2)
    zmv6=zm(l,m)-cz(l,atm5-2)

    v1 = (/ xmv4, ymv4, zmv4 /)
    v2 = (/ xmv6, ymv6, zmv6 /)

    v_normal(l,m) = ( v1*v2 )
    v_normal(l,m) = v_normal(l,m)/sqrt( v_normal(l,m) .DOT. v_normal(l,m) )


! diversas possibilidades de definir o vetor do eixo do anel.
! *nota: a opcao usando os vetores 4 e 6 tem a vantagem de preservar c/ mais rigor a ortogonalidade
!       com o vetor normal. >> melhor para determinar as rotacoes // precisa avaliar a imprecisao.

!     xmv2=( cx(l,atm2)-cx(l,atm5) )   !do atm5 para o atm2
!     ymv2=( cy(l,atm2)-cy(l,atm5) )
!     zmv2=( cz(l,atm2)-cz(l,atm5) )

!     usando os vetores que sao perpend. a normal.
!
!     v_eixo(l,m)= v1+v2              ! usando a simetria dos vetores 4 e 6
!

    xmv2=cx(l,atm5)-xm(l,m)         !do centro para o carbono alpha (atm5)
    ymv2=cy(l,atm5)-ym(l,m)
    zmv2=cz(l,atm5)-zm(l,m)

    v_eixo(l,m) = (/ xmv2, ymv2, zmv2 /)

    v_eixo(l,m) = v_eixo(l,m)/sqrt( v_eixo(l,m).DOT.v_eixo(l,m) )
```

```
! impondo a orientacao simetrica das normais, pode pois: [0,180] = [0,90]

      dotp = v_normal(l,m-1).DOT.v_normal(l,m)
      if (dotp <= 0.0d0) then
        v_normal(l,m) = real_times_vector(-1.0d00,v_normal(l,m))
      end if

      write(18,'(2i3,9f10.5)')l,m, xm(l,m), ym(l,m), zm(l,m),v_normal(l,m),v_eixo(l,m)

!
!WARNING: Isso (prox. linhas) pode ser arbitrario
!        dependendo da numeracao da cadeia.
!        A estrutura phenil capped tem esse problema de nao ter o
!        primeiro monomero completo.

        atm2 =  atm2   +  natCARBmon
        atm5 =  atm5   +  natCARBmon
   end do
end do
close(18)

open(18,file='intra.vec.1nb.dat')

do l=1, ncadeias
   do m = 1, naneisolig-1

      v1 = (/ xm(l,m)  , ym(l,m)  , zm(l,m)   /)
      v2 = (/ xm(l,m+1), ym(l,m+1), zm(l,m+1) /)

      vnn(l,m) = v2 - v1

      write(18,'(2i3, 3f10.5)') l,m, vnn(l,m)

   end do
   write(18,'(2i3, 3f10.5)') l,m, 0.0,0.0,0.0
end do
close(18)

!=======================================
! Calculating statistical distributions
!=======================================

!-------------------
!A. INTRACADEIAS
!-------------------
print *, 'intrachain DATA'

!A.1. Distancia entre centros de aneis intra (todos com todos)
!obs: nao estah usando o corte diagonal // opt. para a prox. versao.

do l=1, ncadeias
   do m = 1, naneisolig-1
      do o = m+1, naneisolig

        v1 = (/ xm(l,m), ym(l,m), zm(l,m) /)
        v2 = (/ xm(l,o), ym(l,o), zm(l,o) /)
```

```
        R(l,m,l,o) = sqrt((v2 - v1).DOT.(v2 - v1))

        R(l,o,l,m) = R(l,m,l,o)

      end do
    end do
end do


!A.2. Calculando "linearidade" end-to-end (relativa ao tam. planar)
!(*Kuhn factor)

open(18, file='intra.lin-ete.dat')

!print*, Lplanar

do l = 1, ncadeias
   write(18,*) l, R(l,1,l,naneisolig)/Lplanar

end do

close(18)

!A.4. Vetores unitarios entre aneis. -> anisotropia.

open(18,file='intra.anis.dat')

do l=1, ncadeias
   do m = 1, naneisolig-1

      vnn(l,m) = vnn(l,m)/sqrt( vnn(l,m) .DOT. vnn(l,m) )
      write(18,*) l,m, vnn(l,m)
      write(18,*) l,m,  real_times_vector (-1.0d0, vnn(l,m))

   end do
end do
close(18)

!A.5-B Angulo entre dois segmentos de cadeia adjacentes // 3 aneis
open(18,file='intra.ang-sg.dat')

do l=1, ncadeias
   do m = 1, naneisolig-2

      gamma = acos( (vnn(l,m) .DOT. vnn(l,m+1)) )*(180.0/PI)
      write(18,'(3i5, 1f10.3)') l, m, m+1, gamma

   end do
end do
close(18)


!A.6. Angulos entre aneis // reformulacao introduzida na versao. 10.
! delta: tilt angle
! tau  : torsion angle

limtau   = 60.0
limlambda= 40.0
```

```
open(18,file='intra.ang-rg.dat')
open(19,file='intra.vec.ang-rg.dat')

count = 1                       !mapping conjugation breaK
do l=1,ncadeias
  sitio(l,1) = count
  conjlgth = 1
  mant = 1
! write(18,'(2i5, 2a10, 1i5)') l, 1, '-', '-', sitio(l,1)
  do m = 2, naneisolig

! v_comp eh o completamento ortogonal da base v_normal-v_eixo {(z)-(y)}
    v_comp = v_eixo(l,m-1)*v_normal(l,m-1)

    write(19,*)v_normal(l,m-1) !R
    write(19,*)v_eixo(l,m-1)   !G
    write(19,*)v_comp          !B

! TAU // angulo de torsao (angulo c/ "normal de ref" (m-1) da projecao da "normal" (m) no plano [normal - comp])

    v_plano= v_normal(l,m)-real_times_vector(v_normal(l,m).DOT.v_eixo(l,m-1), v_eixo(l,m-1))
    v_plano= v_plano/sqrt(v_plano.DOT.v_plano)

    write(19,*)v_plano         !RB

    dotp = v_plano .DOT. v_normal(l,m-1)
    dotp = dotp*0.9999999d0 ! gambiarra para instabilidade do acos. nota (*)
    tau  = dacos( dotp )*(180.d0/PI)

! LAMBDA // angulo entre eixos de aneis

    dotp  = v_eixo(l,m) .DOT. v_eixo(l,m-1)
    dotp  = dotp*0.9999999d0 ! gambiarra para instabilidade do acos. nota (*)
    lambda= dacos( dotp )*(180.d0/PI)

! vinculos fisicos para avaliar conjugacao

    write(19,*)v_normal(l,m)   !W

    flgtau   = 1 ! flg: flags - 0: falso
    flglambda= 1

    if ((tau >= limtau).and.(tau <= (180-limtau)))            flgtau=0
    if ((lambda >= limlambda).and.(lambda <= (180-limlambda))) flglambda=0

    if((flgtau == 0).or.(flglambda == 0)) then ! quebrou a conjugacao
      do j=mant,m-1
        conjuga(l,j) = conjlgth
      end do
      mant = m
      count=count+1
      conjlgth = 0
    end if
    sitio(l,m) = count
    conjlgth = conjlgth + 1
    if (m .eq. naneisolig) then
      do j=mant,m
```

```
            conjuga(l,j) = conjlgth
          end do
        end if

        write(18,'(2i5, 2f10.3, 1i5)') l, m, tau, lambda, sitio(l,m)

      end do
      count=count+1

end do
close(18)
close(19)

open(18,file='intra.conj.dat')
do l=1,ncadeias
  do m = 1, naneisolig
    write(18,*) l,m,sitio(l,m),conjuga(l,m)
  end do
end do
close(18)

print*, '\/\/ok'

! talvez seja bom desalocar a memoria que nao for mais ser usada.
! como tratar as imagens periodicas considerando cadeias distintas?

!----------------------
!B. INTER-CADEIAS
!----------------------
print *, 'interchain DATA'

!inserting periodic boundary condictions
!(important to relevant to inter chain distances).

allocate(sx(ncadeias,naneisolig))
allocate(sy(ncadeias,naneisolig))
allocate(sz(ncadeias,naneisolig))

sx = 0.0
sy = 0.0
sz = 0.0
open(18,file='all.vec.rg-pbc.dat')              !coord. centros pbc
do l = 1, ncadeias
  do m = 1, naneisolig

sx(l,m) = -1.0
sy(l,m) = -1.0
sz(l,m) = -1.0

! estah com essa gambiarra para colocar todo mundo na celula
! [0,a1]x[0,a2]x[0,a3] // dah p/ fazer a coisa mais bonitinha com algebra melhor
! sem loop, mas ficou assim por ora.

do while ((sx(l,m) < 0.0).or.(sy(l,m)<0.0).or.(sz(l,m)<0.0))

sx(l,m) = B(1,1)*xm(l,m)+B(1,2)*ym(l,m)+B(1,3)*zm(l,m)
sy(l,m) = B(2,1)*xm(l,m)+B(2,2)*ym(l,m)+B(2,3)*zm(l,m)
sz(l,m) = B(3,1)*xm(l,m)+B(3,2)*ym(l,m)+B(3,3)*zm(l,m)
```

```
IPBX = dint(sx(l,m)+1.d0)-1
IPBY = dint(sy(l,m)+1.d0)-1
IPBZ = dint(sz(l,m)+1.d0)-1

sx(l,m)= sx(l,m) - IPBX
sy(l,m)= sy(l,m) - IPBY
sz(l,m)= sz(l,m) - IPBZ

xm(l,m) = A(1,1)*sx(l,m)+A(1,2)*sy(l,m)+A(1,3)*sz(l,m)
ym(l,m) = A(2,1)*sx(l,m)+A(2,2)*sy(l,m)+A(2,3)*sz(l,m)
zm(l,m) = A(3,1)*sx(l,m)+A(3,2)*sy(l,m)+A(3,3)*sz(l,m)

end do

write(18,'(2i3,9f10.5)')l,m, xm(l,m), ym(l,m), zm(l,m),v_normal(l,m)

  end do
end do

close(18)


allocate(vrr(ncadeias,naneisolig,ncadeias,naneisolig))

!B.1. DISTANCIA ENTRE OS CENTROS DOS ANEIS

open(18,file='dist-rg.dat')

! isso aqui eh a parte mais lenta...
! talvez convenha partir para lista de vizinhos...

print *, 'entrando na etapa mais lenta... / inter.dist-rg'

! refazendo inclusive intracadeias c/ pbc: estava bugado antes!
! estah com redundancias nas diagonais (comentarios abaixo)

do l=1,ncadeias
  do m = 1, naneisolig
    do n = 1, ncadeias
      do o = 1, naneisolig

ssx = sx(n,o) - sx(l,m)
ssx = ssx  - (dint((2.d0*ssx +3.d0)/2.d0)-1.d0)
ssy = sy(n,o) - sy(l,m)
ssy = ssy  - (dint((2.d0*ssy +3.d0)/2.d0)-1.d0)
ssz = sz(n,o) - sz(l,m)
ssz = ssz  - (dint((2.d0*ssz +3.d0)/2.d0)-1.d0)

xxm  = A(1,1)*ssx +A(1,2)*ssy +A(1,3)*ssz
yym  = A(2,1)*ssx +A(2,2)*ssy +A(2,3)*ssz
zzm  = A(3,1)*ssx +A(3,2)*ssy +A(3,3)*ssz

vrr(l,m,n,o) = (/ xxm, yym, zzm /)
!vrr(n,o,l,m) = vrr(l,m,n,o) - 2.0d0*vrr(l,m,n,o)

R(l,m,n,o)= sqrt( vrr(l,m,n,o).DOT.vrr(l,m,n,o) )
!R(n,o,l,m) = R(l,m,n,o)
```

```
if ((R(l,m,n,o) .le. 10.00) .and. (R(l,m,n,o) .gt. 0.00001)) write(18,*) l,m,n,o, R(l,m,n,o)

        end do
      end do
   end do
end do

close(18)

! tudo checado ateh aqui, comparando com RDF (cerius) --> ok. Amazonas & Rodrigo // 4.out.2011

print *, 'mapeando angulos e topologia...'

deallocate(sx)
deallocate(sy)
deallocate(sz)

!B.2. Angulos inter_cadeia dentro de um raio de corte
!Estabelecendo relacao com orientacao relativa dos aneis
!e gerando topologia com _multiplas ligacoes_ entre sitios:
!ATENCAO: retirar multiplicidade com ajmobmax.py!!

open(18,file='inter.site.dat')              !coordenadas
open(19,file='siteenergies.dat')
open(20,file='mobili.input')                !coordenadas
open(21,file='anelanel.dat')

do l=1,ncadeias
   do m = 1, naneisolig
      do n = 1, ncadeias              ! geral
         do o = 1, naneisolig

            if (sitio(l,m).ne.sitio(n,o))then

            anelanel = vrr(l,m,n,o)/R(l,m,n,o)

            ! angulo normal//normal
            dotp = v_normal(l,m).DOT.v_normal(n,o)
            dotp = dotp*0.9999999d0
            phi  = dacos( dotp )*(180.d0/PI)

            !angulo normal//v-[anel/anel]  // 1

            dotp1  = v_normal(l,m) .DOT. anelanel
            dotp1  = dotp1*0.9999999d0
            theta1 =  dacos( dotp1 ) *(180.d0/PI)

            !angulo normal//v-[anel/anel] // 2

            dotp2  = v_normal(n,o) .DOT. anelanel
            dotp2  = dotp2*0.9999999d0
            theta2 =  dacos( dotp2 ) *(180.d0/PI)

            if ( R(l,m,n,o) <= limR ) write(21,'(4f10.3)') R(l,m,n,o),phi,theta1,theta2

            ! flg: flags - 0: falso
            flgR      = 1
```

```
          flghpp    = 1

          if ( R(l,m,n,o) >= limR ) then
            flgR = 0
            hpp = 0.0
          else
            hpp = HOPPING( R(l,m,n,o),dotp,dotp1,dotp2) !HOPPING: funcao tipo-dipolo (abr.2011)
          end if

          if (hpp <= limhpp) flghpp = 0

          if((flghpp == 1).and.(flgR == 1)) then

            Estno = SITEENERGY(conjuga(n,o))
            Estlm = SITEENERGY(conjuga(l,m))

            write(18,'(6i5, 4f10.3)') l,m,sitio(l,m),n,o,sitio(n,o),R(l,m,n,o), phi, theta1,theta2

            write(19,'(i5, f10.3,i5, f10.3)') sitio(l,m),Estlm, sitio(n,o), Estno

            write(20,'(2i5, 8f10.3)') sitio(l,m), sitio(n,o), R(l,m,n,o), anelanel, hpp, Estno - Estlm

          end if

          end if
        end do
      end do
    end do
  end do

close(18)
close(19)
close(20)
close(21)

print *, 'concluido!'

print *, '\/\/\/\ ok'

print *, ''
print *, 'Successful termination. ;^)'
print *, ''

END PROGRAM ANG_ANEL

!-----------------------------------------
! ---------- UNDER CONSTRUCTION / START
!-----------------------------------------

! expressao para calculo do hopping na rede complexa
! R.Ramos & Amazonas -  v10, aug.2011
! obs.: falta o pi-stacking, usa apenas expressoes do HB.

REAL(8) FUNCTION HOPPING(R,dot,dot1,dot2)
REAL(8):: R,dot,dot1,dot2
REAL(8):: A,B,C,D

real(8), PARAMETER::PI= 3.141592654
```

```
! Parametros // ajustaveis DFT por contas do J.G.
! modelo "tipo-dipolo"

 A = 106.493
 B = 1.74800
 C = 1.35843
 D = 0.04446

HOPPING = A*exp(-B*R)*( C*abs(dot)+D*abs(dot1*dot2) )
!write(99,*) R,HOPPING
END FUNCTION HOPPING

! expressao para calculo da energia de sitio
! v.11 - reparametrizado c/ GW - Amazonas, sep.2011

REAL(8) FUNCTION SITEENERGY(conjug)
INTEGER conjug
REAL(8) A,B,conjuglgth

!A = 19.42
!B = 5.53

!A = 5.5008    !GW   (antigo ver)
!B = 6.6240

!A = 10.135
!B = 5.736

A = 14.2084
B = 5.70539
 conjuglgth = conjug*6.66

!A = 4.089     !AM1
!B = 7.898

!A= 0.671072
!B= 0.388162
C= 0.836435
!D= 0.284468

SITEENERGY = (A/conjuglgth) + B

!SITEENERGY =A*exp(-B*conjug)+ C*exp(-D*conjug**2)

!SITEENERGY =A*exp(-B*conjug)

END FUNCTION SITEENERGY


!-------------------------------------------
! ---------- UNDER CONSTRUCTION // END
!-------------------------------------------

SUBROUTINE INV(ARRAY,N)
            implicit double precision (a-h,o-z)
       dimension ARRAY(N,N)
                        !AMAX(n,n),SAV
```

```
!  troquei os dimensions ik e jk de 4 p/ 16 .
      DIMENSION IK(N),JK(N)
11     DO 100 K=1,N
      AMAX=0.0d0
21     DO 30 I=K,N
      DO 30 J=K,N
23     IF(dABS(AMAX)-dABS(ARRAY(I,J)))24,24,30
24     AMAX=ARRAY(I,J)
      IK(K)=I
      JK(K)=J
30     CONTINUE
41     I=IK(K)
      IF(I-K)21,51,43
43     DO 50 J=1,N
      SAV=ARRAY(K,J)
      ARRAY(K,J)=ARRAY(I,J)
50     ARRAY(I,J)=-SAV
51     J=JK(K)
      IF(J-K)21,61,53
53     DO 60 I=1,N
      SAV=ARRAY(I,K)
      ARRAY(I,K)=ARRAY(I,J)
60     ARRAY(I,J)=-SAV
61     DO 70 I=1,N
      IF(I-K)63,70,63
63     ARRAY(I,K)=-ARRAY(I,K)/AMAX
70     CONTINUE
71     DO 80 I=1,N
      DO 80 J=1,N
      IF(I-K)74,80,74
74     IF(J-K)75,80,75
75     ARRAY(I,J)=ARRAY(I,J)+ARRAY(I,K)*ARRAY(K,J)
80     CONTINUE
81     DO 90 J=1,N
      IF(J-K)83,90,83
83     ARRAY(K,J)=ARRAY(K,J)/AMAX
90     CONTINUE
      ARRAY(K,K)=1./AMAX
100    CONTINUE
101    DO 130 L=1,N
      K=N-L+1
      J=IK(K)
      IF(J-K)111,111,105
105    DO 110 I=1,N
      SAV=ARRAY(I,K)
      ARRAY(I,K)=-ARRAY(I,J)
110    ARRAY(I,J)=SAV
111    I=JK(K)
      IF(I-K)130,130,113
113    DO 120 J=1,N
      SAV=ARRAY(K,J)
      ARRAY(K,J)=-ARRAY(I,J)
120    ARRAY(I,J)=SAV
130    CONTINUE
140    RETURN
      END

! pacotes de suporte dos tipos vectors usados para simplificar o calculo vetorial
```

```
! J.G. Amazonas / v1

MODULE vectors

IMPLICIT NONE

!Declare vector data type:
TYPE :: vector
REAL(8) :: x
REAL(8) :: y
REAL(8) :: z
END TYPE

!Declare interface operators
INTERFACE ASSIGNMENT (=)
      MODULE PROCEDURE array_to_vector
      MODULE PROCEDURE vector_to_array
END INTERFACE

INTERFACE OPERATOR (+)
      MODULE PROCEDURE vector_add
END INTERFACE

INTERFACE OPERATOR (-)
      MODULE PROCEDURE vector_subtract
END INTERFACE

INTERFACE OPERATOR (*)
      MODULE PROCEDURE vector_times_real
      MODULE PROCEDURE real_times_vector
      MODULE PROCEDURE vector_times_int
      MODULE PROCEDURE int_times_vector
      MODULE PROCEDURE cross_product
END INTERFACE

INTERFACE OPERATOR (/)
      MODULE PROCEDURE vector_div_real
      MODULE PROCEDURE vector_div_int
END INTERFACE

INTERFACE OPERATOR (.DOT.)
      MODULE PROCEDURE dot_product
END INTERFACE

!     Now define the implementing functions.

CONTAINS
      SUBROUTINE array_to_vector (vec_result, array)
      TYPE (vector), INTENT (OUT) :: vec_result
      REAL(8), DIMENSION (3), INTENT (IN) :: array
      vec_result%x = array (1)
      vec_result%y = array (2)
      vec_result%z = array (3)
      END SUBROUTINE array_to_vector

      SUBROUTINE vector_to_array(array_result, vec_1)
      REAL(8), DIMENSION (3), INTENT (OUT) :: array_result
      TYPE (vector), INTENT (IN) :: vec_1
```

```
array_result (1) = vec_1%x
array_result (2) = vec_1%y
array_result (3) = vec_1%z
END SUBROUTINE vector_to_array

FUNCTION vector_add (vec_1, vec_2)
TYPE (vector) :: vector_add
TYPE (vector), INTENT (IN) :: vec_1, vec_2
vector_add%x = vec_1%x + vec_2%x
vector_add%y = vec_1%y + vec_2%y
vector_add%z = vec_1%z + vec_2%z
END  FUNCTION vector_add

FUNCTION vector_subtract (vec_1, vec_2)
TYPE (vector) :: vector_subtract
TYPE (vector), INTENT (IN) :: vec_1, vec_2
vector_subtract%x = vec_1%x - vec_2%x
vector_subtract%y = vec_1%y - vec_2%y
vector_subtract%z = vec_1%z - vec_2%z
END  FUNCTION vector_subtract

FUNCTION vector_times_real (vec_1, real_2)
TYPE (vector) ::  vector_times_real
TYPE (vector), INTENT (IN) :: vec_1
REAL(8), INTENT (IN) :: real_2
vector_times_real%x = vec_1%x * real_2
vector_times_real%y = vec_1%y * real_2
vector_times_real%z = vec_1%z * real_2
END  FUNCTION  vector_times_real

FUNCTION real_times_vector (real_1, vec_2)
TYPE (vector) ::  real_times_vector
REAL(8), INTENT (IN) :: real_1
TYPE (vector), INTENT (IN) :: vec_2
real_times_vector%x = real_1 * vec_2%x
real_times_vector%y = real_1 * vec_2%y
real_times_vector%z = real_1 * vec_2%z
END  FUNCTION real_times_vector

FUNCTION vector_times_int (vec_1, int_2)
TYPE (vector) :: vector_times_int
TYPE (vector), INTENT (IN) :: vec_1
INTEGER, INTENT (IN) :: int_2
vector_times_int%x = vec_1%x * REAL(int_2)
vector_times_int%y = vec_1%y * REAL(int_2)
vector_times_int%z = vec_1%z * REAL(int_2)
END  FUNCTION vector_times_int

FUNCTION int_times_vector (int_1, vec_2)
TYPE (vector) :: int_times_vector
INTEGER, INTENT (IN) :: int_1
TYPE (vector), INTENT (IN) :: vec_2
int_times_vector%x = REAL(int_1) * vec_2%x
int_times_vector%y = REAL(int_1) * vec_2%y
int_times_vector%z = REAL(int_1) * vec_2%z
END  FUNCTION  int_times_vector

FUNCTION vector_div_real(vec_1, real_2)
```

```
    TYPE (vector) ::  vector_div_real
    TYPE (vector), INTENT(IN) :: vec_1
    REAL(8), INTENT(IN) :: real_2
    vector_div_real%x = vec_1%x / real_2
    vector_div_real%y = vec_1%y / real_2
    vector_div_real%z = vec_1%z / real_2
    END  FUNCTION  vector_div_real

    FUNCTION vector_div_int(vec_1, int_2)
    TYPE (vector) :: vector_div_int
    TYPE (vector), INTENT(IN) :: vec_1
    INTEGER, INTENT(IN) :: int_2
    vector_div_int%x = vec_1%x / REAL(int_2)
    vector_div_int%y = vec_1%y / REAL(int_2)
    vector_div_int%z = vec_1%z / REAL(int_2)
    END FUNCTION vector_div_int

    FUNCTION dot_product (vec_1, vec_2)
    REAL(8) :: dot_product
    TYPE (vector), INTENT (IN) :: vec_1, vec_2
    dot_product = vec_1%x*vec_2%x + vec_1%y*vec_2%y + vec_1%z*vec_2%z
    END FUNCTION dot_product

    FUNCTION cross_product (vec_1, vec_2)
        TYPE (vector) :: cross_product
        TYPE (vector), INTENT (IN) :: vec_1, vec_2
        cross_product%x = vec_1%y*vec_2%z - vec_1%z*vec_2%y
        cross_product%y = vec_1%z*vec_2%x - vec_1%x*vec_2%z
        cross_product%z = vec_1%x*vec_2%y - vec_1%y*vec_2%x
    END FUNCTION cross_product

END MODULE vectors
```