**Example, Nov. 7, 2016**

This example code calculates the diabatic and adiabatic potential energy matrix elements of two geometries -- the equilibrium geometries of the ground and the first excited state of thioanisole -- using the PES subroutine.

Compilation and execution:

\* Edit makefile to choose suitable compiler and math library, if necessary.

\* Compile the example code by running "make" in a Linux shell.

\* Execute the code by runing "./example.exe" in a Linux shell.

\* After execution, the following results should be printed to standard output.

Diabatic potential energy matrix elements (U) and adiabatic potential energies (V) in eV:

Geom # U(1,1) U(2,2) U(3,3) U(1,2) U(1,3) U(2,3) V(1) V(2) V(3)

1 0.202586 4.759661 5.217826 -0.008163 0.000000 0.000000 0.202571 4.759676 5.217826

2 0.419966 4.557015 5.187150 -0.008163 -0.000020 -0.000001 0.419950 4.557031 5.187150

\* For example, from the results of the equilibrium geometry of the ground state (Geom # 1), the vertical excitation energies of the first and second excited states can be calculated as V(2)-V(1) and V(3)-V(1), which are 4.56 eV and 5.02 eV respectively, as shown in Table III in the paper.

**makefile**

fc = gfortran

lib = -llapack

libpath =

example.exe: main.o phsch3\_aprp.o

$(fc) $(libpath) $(lib) -o example.exe main.o phsch3\_aprp.o

main.o: main.f90

$(fc) -c main.f90

phsch3\_aprp.o: ../phsch3\_aprp.f90

$(fc) -c ../phsch3\_aprp.f90

clean:

rm -f \*.o \*.exe

**main.f90**

program phsch3\_aprp

implicit none

double precision, parameter :: ang\_bohr=0.529177249d0 &

, eV\_hartree=27.211386d0

integer,parameter :: ifin=100, natom=16, nstate=3

integer :: i, ngeom, igeom, j, k, l

integer :: igrad, repflag

double precision :: xx(3,natom), uu(nstate,nstate), vv(nstate) &

, guu(3,natom,nstate,nstate), gvv(3,natom,nstate) &

, dvec(3,natom,nstate,nstate), cc(nstate,nstate)

double precision :: qps(3)

character\*2 :: dummy

!--- for numerical gradient

double precision :: gvvn(3,natom,nstate), guun(3,natom,nstate,nstate), step

double precision :: xx2(3,natom), uu2(nstate,nstate), vv2(nstate) &

, guu2(3,natom,nstate,nstate), gvv2(3,natom,nstate) &

, dvec2(3,natom,nstate,nstate), cc2(nstate,nstate) &

, vv3(nstate), uu3(nstate,nstate)

integer :: ifrandcoord

logical :: numergrad

open(ifin, file='example\_input.txt', action='read')

read(ifin, \*) ngeom

write(\*, \*) 'Diabatic potential energy matrix elements (U) and adiabatic potential energies (V) in eV:'

write(\*, '(10A12)') 'Geom #', 'U(1,1)', 'U(2,2)', 'U(3,3)', 'U(1,2)', 'U(1,3)', 'U(2,3)', 'V(1)', 'V(2)', 'V(3)'

do igeom = 1, ngeom

read(ifin, \*)

do i = 1, natom

read(ifin, \*) dummy, xx(1,i), xx(2,i), xx(3,i)

end do

xx = xx/ang\_bohr

call pot(igrad,xx,uu,guu,vv,gvv,dvec,cc,repflag)

uu = uu\*eV\_hartree

vv = vv\*eV\_hartree

!--- print PES

write(\*,'(1I12,9F12.6)') igeom, uu(1,1), uu(2,2), uu(3,3), uu(1,2), uu(1,3), uu(2,3), vv(1), vv(2), vv(3)

end do

end program phsch3\_aprp

**example input.txt**

2

C -2.028196E+00 -1.365191E+00 1.198265E-04

C -6.334476E-01 -1.275127E+00 2.887931E-04

C -1.970402E-03 -1.239798E-02 3.800463E-04

C -7.904528E-01 1.152476E+00 2.994067E-04

C -2.191705E+00 1.049899E+00 1.295453E-04

C -2.817480E+00 -2.024427E-01 3.893622E-05

H -2.500655E+00 -2.347475E+00 5.094673E-05

H -2.557632E-02 -2.181092E+00 3.508996E-04

H -3.311684E-01 2.138783E+00 3.666255E-04

H -2.790948E+00 1.960398E+00 6.848879E-05

H -3.904058E+00 -2.750174E-01 -9.291101E-05

S 1.763029E+00 -1.397770E-02 5.915949E-04

C 2.171676E+00 1.767232E+00 6.623258E-04

H 1.785575E+00 2.263195E+00 -8.984234E-01

H 1.785362E+00 2.263173E+00 8.996684E-01

H 3.266366E+00 1.820795E+00 7.931073E-04

C -1.955163E+00 -1.313180E+00 4.824102E-05

C -5.035406E-01 -1.183817E+00 1.223748E-04

C 2.493909E-02 1.465541E-01 1.834997E-04

C -7.579716E-01 1.317161E+00 1.762369E-04

C -2.204812E+00 1.163231E+00 1.013187E-04

C -2.752254E+00 -1.538375E-01 3.983480E-05

H -2.417619E+00 -2.301320E+00 -3.276419E-07

H 1.561820E-01 -2.052752E+00 1.312981E-04

H -3.075016E-01 2.309726E+00 2.245031E-04

H -2.853296E+00 2.039342E+00 9.280545E-05

H -3.842451E+00 -2.664812E-01 -1.636414E-05

S 1.782848E+00 1.641957E-01 2.703702E-04

C 2.295577E+00 1.899351E+00 3.484481E-04

H 1.915770E+00 2.402473E+00 -9.011012E-01

H 1.915688E+00 2.402416E+00 9.017951E-01

H 3.393570E+00 1.896810E+00 3.987264E-04

**phsch3\_aprp.f90**

!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

! Shaohong L. Li, Oct. 2016

!

! Reference for this potential energy surface:

! "Full-dimensional ground- and excited-state potential energy

! surfaces and state couplings for photodissociation of

! thioanisole"

! by Shaohong L. Li and Donald G. Truhlar, J. Chem. Phys.,

! submitted Nov. 2016.

! Coordinates and diabatic surfaces are identical to those in

! this reference, but the numbering of atoms is different

! (see "Important notes" below).

!

! This PES subroutine has been tested with the following compiler

! and software:

! \* Intel ifort 13.1.3 with MKL

! \* GCC gfortran 4.8.0 with LAPACK

! \* ANT 2014-2

! \* POLYRATE 2010-A

!

!\*this\*line\*is\*66\*characters\*long\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

!=================================================================

! Important notes:

! \* Variable 'debug' should be set to .false. for production runs.

! \* Internal working units (for all subroutines except pot):

! Angstrom, radian, hartree.

! \* LAPACK routine dsyev is needed for diagonalization.

! \* Internal numbering of atoms:

!

! H10 H9 H14,15

! \ / \

! C5---C4 C13---H16

! / \ /

! H11--C6 C3---S12

! \ /

! C1---C2

! / \

! H7 H8

!=================================================================

!=================================================================

! \*def pot

! Main PES subroutine for calculating potential energies and

! gradients of thioanisole. Designed to interface with ANT.

! Input:

! igrad, repflag: dummy

! xx: cart. coord. of atoms (in bohrs);

! first index from 1 to 3 representing x, y, z,

! second index from 1 to # of atoms (16).

! The order of atoms must be consistent with the internal

! numbering (see "Important notes" above).

! Output: (output units: bohr, radian, hartree)

! uu: 3\*3 diabatic matrix (in hartrees)

! guu: diabat. grad. (in hartrees/bohr);

! guu(i,j,k,l) = derivative of uu(k,l) w.r.t. coord. i

! of atom j

! vv: adiabatic energies (in hatrees);

! vv[i] = pot. energy of adiab. state i

! gvv: adiabat. grad. (in hartrees/bohr);

! gvv(i,j,k) = derivative of vv(k) w.r.t. coord. i

! of atom j

! dvec: nonadiab.coupl. (in bohr\*\*(-1));

! dvec(i,j,k,l) = component of coupling between adiabatic

! states k & l corresp. to coord. i of atom j

! cc: 3\*3 diab. to adiab. matrix (unitless)

!

!=================================================================

subroutine pot(igrad,xx,uu,guu,vv,gvv,dvec,cc,repflag)

implicit none

!--- {'global' constants

integer, parameter :: natom=16, nstate=3, ntermmax=200 &

, ntermtype=4, napmax=10, nqtpset=2, nlcmax=6

double precision, parameter :: pi=3.14159265358979d0

!--- convertion factor: 0.53 Angstrom = 1 bohr

double precision, parameter :: ang\_bohr=0.529177249d0

!--- }'global' constants

!--- {arguments

integer :: igrad, repflag

double precision :: xx(3,natom), uu(nstate,nstate), vv(nstate) &

, guu(3,natom,nstate,nstate), gvv(3,natom,nstate) &

, dvec(3,natom,nstate,nstate), cc(nstate,nstate)

!--- }arguments

integer :: i, j, k, l, ist, jst

double precision :: x(3,natom), vt(nstate,nstate) &

, gvt(3,natom,nstate,nstate), tmpmat33(nstate,nstate)

!--- internals for prim.&sec. pot, tert. pot., tert. coupl.

!--- qtp has two sets, for bonded & dissoc. S-Me

double precision :: qps(3), qtp(ntermmax,nqtpset) &

, qtc(ntermmax) &

!--- attributes

, bmatqtc(ntermmax,natom\*3)

!--- number/type of terms and atom index lists for qtp

integer :: nqtp(nqtpset), itermqtptype(ntermmax,nqtpset) &

, iqtplist(4,ntermmax,nqtpset) &

, nqtc, qtcsym(ntermmax), qtctyp(ntermmax)

!--- for diagonalization of prim+sec U

double precision :: DTAmat(nstate,nstate), Vtmp(nstate)

logical :: debug, debug2

!--- zero of energy

double precision :: Ezero

!--- debug: input cartesians in Angstroms; otherwise in bohr

debug = .false.

!--- debug2: use only 1 anchor point

debug2 = .false.

!--- convert Cartesians from bohr to Angstrom

if(.not.debug) then

x(:,:) = xx(:,:)\*ang\_bohr

else

x(:,:) = xx(:,:)

end if

!--- calculate primary and secondary internals

call calcqps(qps, x)

!--- calculate redundant tertiary internals for tert. diab. pot.

call calcqtp(qtp, nqtp, itermqtptype, iqtplist, x)

!--- calculate nonred. tert. internals for diab. coupl.

call calcqtc(qtc, nqtc, qtcsym, qtctyp, bmatqtc, x)

uu = 0d0; guu = 0d0

!--- calculate prim.&sec. U elements and their grad. w.r.t. x

call calcu1\_ps(uu, guu, x, qps)

call calcu2\_ps(uu, guu, x, qps)

call calcu3\_ps(uu, guu, x, qps)

call calcu12\_ps(uu, guu, x, qps)

call calcu13\_ps(uu, guu, x, qps)

call calcu23\_ps(uu, guu, x, qps)

!--- calculate tert. potential and its grad.

call calcuii\_t(vt, gvt, x, qps, qtp, nqtp,&

itermqtptype, iqtplist)

!== {assuming tert. pot. are diabatic

uu = uu+vt

guu = guu+gvt

!== }assuming tert. pot. are diabatic

!--- calculate tert. diab. coupl.

call calcuij\_t(uu, guu, x, qps, qtc, nqtc, qtcsym, qtctyp,&

bmatqtc)

!--- calculate Born-Mayer potential and add it to diabats

call calcbornmayer(uu, guu, x)

!--- setting zero of energy as that of V1 at equilibrium by XQDPT

Ezero = -668.474438125

do i = 1, 3

uu(i,i) = uu(i,i)-Ezero

end do

!--- calculate adiab. V, DTA matrix cc, dV, NACME F

dvec = 0d0

call diagonalize(vv, cc, uu, nstate)

do i = 1, 3

do j = 1, natom

!--- C^T\*dU\*C

tmpmat33(:,:) = guu(i,j,:,:)

tmpmat33 = matmul(transpose(cc), matmul(tmpmat33, cc))

!--- dV\_i = (C^T\*dU\*C)\_ii

do ist = 1, nstate

gvv(i,j,ist) = tmpmat33(ist,ist)

end do

!--- F\_ij = (C^T\*dU\*C)\_ij / (V\_j-V\_i)

do ist = 1, nstate-1

do jst = ist+1, nstate

dvec(i,j,ist,jst) = &

tmpmat33(ist,jst)/(vv(jst)-vv(ist))

dvec(i,j,jst,ist) = -dvec(i,j,ist,jst)

end do

end do

end do

end do

!--- convert gradients from Ang-1 to bohr-1

if(.not.debug) then

guu = guu\*ang\_bohr

gvv = gvv\*ang\_bohr

dvec = dvec\*ang\_bohr

end if

contains

!=================================================================

! \*def calcqps

! Calculate primary & secondary internal coordinates.

! Input:

! x: Cartesian coordinates in Angstroms

! Output:

! qps: values of primary & secondary internal coordinates

!=================================================================

subroutine calcqps(qps, x)

implicit none

double precision :: qps(3), x(3,natom)

qps(1) = evalBL(x, 12, 13)

!--- the CSCC torsion is not a good prim. coordinate;

!--- instead, use the "special" coordinate phi

qps(2) = evalSP1(x, 2, 3, 4, 12, 13)

! qps(2) = evalTO(x, 4, 3, 12, 13)

!--- qps(3) is obsolete, keep it just in case

qps(3) = evalBA(x, 3, 12, 13)

end subroutine calcqps

!=================================================================

! \*def calcqtp

! Calculate tertiary int. coord. as defined by QFF for potential.

! Input:

! x: Cartesian coordinates in Angstroms

! Output:

! qtp: values of tertiary internal coordinates for potential

! nterm: number of terms in each qtp set

! itypeqtp: type of each term (and coord.)

! iqtplist: atom index list array

!=================================================================

subroutine calcqtp(qtp, nterm, itypeqtp, iqtplist, x)

implicit none

double precision :: qtp(:,:), x(3,natom)

integer :: nterm(nqtpset), itypeqtp(ntermmax,nqtpset) &

, iqtplist(4,ntermmax,nqtpset)

integer :: iterm, iset, ii, jj, kk, ll

!--- assign attributes of tert. int. for potentials

call assignqtp(nterm, itypeqtp, iqtplist)

do iset = 1, nqtpset

do iterm = 1, nterm(iset)

ii = iqtplist(1,iterm,iset)

jj = iqtplist(2,iterm,iset)

kk = iqtplist(3,iterm,iset)

ll = iqtplist(4,iterm,iset)

select case(itypeqtp(iterm,iset))

!--- bond length

case(1)

qtp(iterm,iset) = evalBL(x, ii, jj)

!--- bond angle

case(2)

qtp(iterm,iset) = evalBA(x, ii, jj, kk)

!--- torsion

case(3)

qtp(iterm,iset) = evalTO(x, ii, jj, kk, ll)

!--- oop distance

case(5)

qtp(iterm,iset) = evalOD(x, ii, jj, kk, ll)

end select

end do

end do

end subroutine calcqtp

!=================================================================

! \*def calcqtc

! Calculate nonred. tert. int. coord. for diab. coupl.

! Input:

! x: Cartesian coordinates in Angstroms

! Output:

! qtc: values of tert. internals

! nqtc: number of qtc

! qtcsym: symmetry (a' or a") of qtc

! qtctyp: type of qtc (CSC bend or other)

! bmatqtc: B matrix, B(i, j) = dqtc(i)/dx(j)

!=================================================================

subroutine calcqtc(qtc, nqtc, qtcsym, qtctyp, bmatqtc, x)

implicit none

integer :: qtcsym(ntermmax), nqtc

double precision :: qtc(ntermmax), x(3,natom) &

, bmatqtc(ntermmax,natom\*3)

double precision :: qr(ntermmax), lccoef(nlcmax,ntermmax) &

, lcqr(nlcmax), bmatrow(natom\*3), bmatr(ntermmax,natom\*3)

integer :: nqr, itypeqr(ntermmax), iqrlist(10,ntermmax) &

, nqnr, lcindex(nlcmax,ntermmax), nlc(ntermmax) &

, qnrindex(ntermmax), qtctyp(ntermmax) &

, iqtc, iqnr, iqr, ilc

!--- calculate redundant q

call calcqr(qr, x, nqr, itypeqr, iqrlist)

!--- attributes of nonredund. q (qnr),

!--- esp. their relation to qr

call assignqnr(nqnr, lcindex, lccoef, nlc)

!--- attributes of qtc,

!--- esp. their relation to qnr

call assignqtc(nqtc, qtcsym, qnrindex, qtctyp)

!--- value of qtc

!--- loop over qtc index

do iqtc = 1, nqtc

!--- corresponding qnr index

iqnr = qnrindex(iqtc)

!--- value of qr involved in qtc(i)

lcqr(1:nlcmax) = qr(lcindex(1:nlcmax,iqnr))

!--- take lin.comb. of qr to get qtc

qtc(iqtc) = dot\_product(lccoef(:,iqnr),lcqr)

end do

!--- B matrix for qr

!--- loop over qr index

do iqr = 1, nqr

call calcbmat(bmatrow, x, iqrlist(:,iqr), itypeqr(iqr))

bmatr(iqr, :) = bmatrow(:)

end do

!--- B matrix for qtc from lin.comb. of bmatr

bmatqtc = 0d0

!--- loop over qtc

do iqtc = 1, nqtc

!--- corresponding qnr index

iqnr = qnrindex(iqtc)

!--- loop over qr involved in the lin.comb.

do ilc = 1, nlc(iqnr)

!--- lin.comb. of bmatr rows

bmatqtc(iqtc,:) = bmatqtc(iqtc,:) &

+lccoef(ilc,iqnr)\*bmatr(lcindex(ilc,iqnr),:)

end do

end do

end subroutine calcqtc

!=================================================================

! \*def calcqr

! Calculate redundant q (qr) for later use of qtc

! Input:

! x: Cartesian coordinates in Angstroms

! nqr: number of qr

! itypeqr: type of each qr

! iqrlist: atom index list array

! Output:

! qr: values of qr

!=================================================================

subroutine calcqr(qr, x, nqr, itypeqr, iqrlist)

implicit none

double precision :: qr(ntermmax), x(3,natom)

integer :: nqr, itypeqr(ntermmax), iqrlist(10,ntermmax)

integer :: iterm, ii, jj, kk, ll

!--- attributes of redundant q (qr)

call assignqr(nqr, itypeqr, iqrlist)

do iterm = 1, nqr

ii = iqrlist(1,iterm)

jj = iqrlist(2,iterm)

kk = iqrlist(3,iterm)

ll = iqrlist(4,iterm)

select case(itypeqr(iterm))

!--- bond length

case(1)

qr(iterm) = evalBL(x, ii, jj)

!--- bond angle

case(2)

qr(iterm) = evalBA(x, ii, jj, kk)

!--- torsion

case(3)

qr(iterm) = evalTO(x, ii, jj, kk, ll)

!--- oop bend

case(4)

qr(iterm) = evalOB(x, ii, jj, kk, ll)

end select

end do

end subroutine calcqr

!=================================================================

! \*def calu1\_ps

! Calculate prim.+sec. U1 and its grad. w.r.t. Cartesians.

! Input:

! uu: primary+secondary U matrix in hartrees

! guu: gradient array (in hatrees/Angstrom)

! x: Cartesian coordinates in Angstroms

! qps: values of primary & secondary internal coordinates

! Output:

! uu: primary+secondary U w/ calculated U1\_PS added

! guu: grad. array w/ calculated dU1/dx added to guu(:,:,1,1)

!=================================================================

subroutine calcu1\_ps(uu, guu, x, qps)

implicit none

double precision :: uu(nstate,nstate) &

, guu(3,natom,nstate,nstate), x(3,natom), qps(:)

double precision :: R, phi, theta, gradu1(3)

double precision :: De, b, Re, A, BB, alpha1, Rw &

, B0(2:4), alpha(2:4), R0(2:4) &

, B01, alpha01, R01, B02, alpha02, R02

double precision :: W1, k(2:4), theta0, dtheta0dR, dkdR, u1

integer :: i

double precision :: bmatrow(3\*natom), bmat(3,3\*natom)

integer :: iatomlist(10)

!--- param. set 6

De = 0.137295383087261

b = 2.29327756747856

Re = 1.82748498740852

A = -668.338016424618

BB = 9.635683829111455d-002

alpha1 = 0.711348372403729

Rw = 1.45278649582810

!--- The following are unused after treating theta as tertiary;

!--- keep them just to avoid messing up other parts

B0(2) = 0.220483011622790d0

alpha(2)= 0.187499658992014d0

R0(2) = 1.82443674739262d0

B0(3) = 0.475903698493558d0

alpha(3)= 0.242387888035320d0

R0(3) = 2.33364647705814d0

B0(4) = 0.514926437655652d0

alpha(4)= 0.219406435333100d0

R0(4) = 1.72584530006004d0

B01 = 0.413985475374836d0

alpha01 = 0.271134056977830d0

R01 = -0.280509801889082d0

B02 = 1.68618904454606d0

alpha02 = 0.050592957575334d0

R02 = 2.46799725550623d0

R = qps(1); phi = qps(2); theta = qps(3)

W1 = BB\*exp(-alpha1\*(R-Rw)\*\*2)

do i = 2, 4

k(i) = B0(i)\*exp(-alpha(i)\*(R-R0(i))\*\*2)

end do

theta0 = B01\*exp(-alpha01\*(R-R01)\*\*2) &

+ B02\*exp(-alpha02\*(R-R02)\*\*2)

u1 = 0d0

!--- primary U1

u1 = u1 + A-De\*(1+b\*(R-Re))\*exp(-b\*(R-Re)) &

!--- primary U1(R, phi)

+ W1\*(1-cos(2\*phi))

!--- secondary U1(R, theta)

uu(1,1) = uu(1,1)+u1

!--- (d U1)/(d R)

gradu1(1) = De\*b\*\*2\*(R-Re)\*exp(-b\*(R-Re)) &

- 2\*alpha1\*(R-Rw)\*W1\*(1-cos(2\*phi))

dtheta0dR = -2\*alpha01\*(R-R01)\*B01\*exp(-alpha01\*(R-R01)\*\*2) &

- 2\*alpha02\*(R-R02)\*B02\*exp(-alpha02\*(R-R02)\*\*2)

!--- (d U1)/(d phi)

gradu1(2) = 2\*W1\*sin(2\*phi)

!--- (d U1)/(d theta)

gradu1(3) = 0d0

!--- use B matrix to convert dU1/dq to dU1/dx

iatomlist(1:4) = (/12,13,0,0/)

call calcbmat(bmatrow, x, iatomlist, 1)

bmat(1,:) = bmatrow(:)

!--- special coordinate in place of CCSC torsion

iatomlist(1:5) = (/2,3,4,12,13/)

call calcbmat(bmatrow, x, iatomlist, 6)

bmat(2,:) = bmatrow(:)

guu(:,:,1,1) = guu(:,:,1,1) &

+reshape(matmul(gradu1, bmat), (/3,natom/))

end subroutine calcu1\_ps

!=================================================================

! \*def calu2\_ps

! Calculate prim.+sec. U2 and its grad. w.r.t. Cartesians.

! Input:

! uu: primary+secondary U matrix in hartrees

! guu: gradient array (in hatrees/Angstrom)

! x: Cartesian coordinates in Angstroms

! qps: values of primary & secondary internal coordinates

! Output:

! uu: primary+secondary U w/ calculated U2\_PS added

! guu: grad. array w/ calculated dU2/dx added to guu(:,:,2,2)

!=================================================================

subroutine calcu2\_ps(uu, guu, x, qps)

implicit none

double precision :: uu(nstate,nstate) &

, guu(3,natom,nstate,nstate), x(3,natom), qps(:)

double precision :: R, phi, theta, gradu2(3)

double precision :: De, b, Re, A, BB, alpha1, Rw &

, B0(2:4), alpha(2:4), R0(2:4) &

, B01, alpha01, R01, B02, alpha02, R02

double precision :: W1, k(2:4), theta0, dtheta0dR, dkdR, u2

integer :: i

double precision :: bmatrow(3\*natom), bmat(3,3\*natom)

integer :: iatomlist(10)

!--- param. set 3

De = 9.093579980902658d-002

b = 1.95603566794475

Re = 1.80932281976175

A = -668.305130416298

BB = 8.161902457843210d-003

alpha1 = 1.41618582563090

Rw = 1.08429805788656

!--- The following are unused after treating theta as tertiary;

!--- keep them just to avoid messing up other parts

B0(2) = 0.295303558215292d0

alpha(2) = 6.983752180574522d-2

R0(2) = 0.677613151501115d0

B0(3) = 0.242598194722343d0

alpha(3) = 5.810399423252850d-2

R0(3) = 3.04835360232912d0

B01 = 1.87588273113145d0

alpha01 = 6.815218991924723d-3

R = qps(1); phi = qps(2); theta = qps(3)

W1 = BB\*exp(-alpha1\*(R-Rw)\*\*2)

do i = 2, 3

k(i) = B0(i)\*exp(-alpha(i)\*(R-R0(i))\*\*2)

end do

theta0 = B01\*exp(-alpha01\*R\*\*2)

u2 = 0d0

!--- primary U2

u2 = u2 + A+De\*(1d0-exp(-b\*(R-Re)))\*\*2 &

!--- primary U2(R, phi)

+ W1\*(1-cos(2\*phi))

!--- secondary U2(R, theta)

uu(2,2) = uu(2,2)+u2

!--- (d U2)/(d R)

gradu2(1) = 2\*De\*(1d0-exp(-b\*(R-Re)))\*b\*exp(-b\*(R-Re)) &

- 2\*alpha1\*(R-Rw)\*W1\*(1d0-cos(2\*phi))

!--- (d U2)/(d phi)

gradu2(2) = 2\*W1\*sin(2\*phi)

!--- (d U2)/(d theta)

gradu2(3) = 0d0

!--- use B matrix to convert dU2/dq to dU2/dx

iatomlist(1:4) = (/12,13,0,0/)

call calcbmat(bmatrow, x, iatomlist, 1)

bmat(1,:) = bmatrow(:)

!--- special coordinate in place of CCSC torsion

iatomlist(1:5) = (/2,3,4,12,13/)

call calcbmat(bmatrow, x, iatomlist, 6)

bmat(2,:) = bmatrow(:)

guu(:,:,2,2) = guu(:,:,2,2) &

+reshape(matmul(gradu2, bmat), (/3,natom/))

end subroutine calcu2\_ps

!=================================================================

! \*def calu3\_ps

! Calculate prim.+sec. U3 and its grad. w.r.t. Cartesians.

! Input:

! uu: primary+secondary U matrix in hartrees

! guu: gradient array (in hatrees/Angstrom)

! x: Cartesian coordinates in Angstroms

! qps: values of primary & secondary internal coordinates

! Output:

! uu: primary+secondary U w/ calculated U3\_PS added

! guu: grad. array w/ calculated dU3/dx added to guu(:,:,3,3)

!=================================================================

subroutine calcu3\_ps(uu, guu, x, qps)

implicit none

double precision :: uu(nstate,nstate) &

, guu(3,natom,nstate,nstate), x(3,natom), qps(:)

double precision :: R, phi, theta, gradu3(3)

double precision :: De, b, Re, A, BB, alpha1, Rw &

, B0(2:4), alpha(2:4), R0(2:4) &

, B01, alpha01, R01, B02, alpha02, R02 &

, CC, e1a, e4a

double precision :: W1, k(2:4), theta0, dtheta0dR, dkdR, u3

integer :: i

double precision :: bmatrow(3\*natom), bmat(3,3\*natom)

integer :: iatomlist(10)

!--- param. set 5

De = 1.65138486208663

b = 1.77311772459465

A = -668.352732063236

BB = -5.913514866085083d-002

alpha1 = 0.721710258187793

Rw = 1.63756083461647

CC = -3.832768992221822d-002

!--- The following are unused after treating theta as tertiary;

!--- keep them just to avoid messing up other parts

B0(2) = 0.474611475756769

alpha(2)= 0.474097840746512

R0(2) = 0.759193306575120

B0(3) = 0.551515405129791

alpha(3)= 0.649583151192963

R0(3) = 1.37421784973235

B01 = 2.26427273199978

alpha01 = 2.524238127368299d-3

R01 = 12.1853085944417

R = qps(1); phi = qps(2); theta = qps(3)

e1a = exp(-alpha1\*(R-Rw)\*\*2)

e4a = exp(-4\*alpha1\*(R-Rw)\*\*2)

W1 = BB\*e1a + CC\*e4a

do i = 2, 3

k(i) = B0(i)\*exp(-alpha(i)\*(R-R0(i))\*\*2)

end do

theta0 = B01\*exp(-alpha01\*(R-R01)\*\*2)

u3 = 0d0

!--- primary U3

u3 = u3 + A+De\*exp(-b\*R) &

!--- secondary U3(R, phi)

+ W1\*(1-cos(2\*phi))

!--- secondary U3(R, theta)

uu(3,3) = uu(3,3)+u3

!--- (d U3)/(d R)

gradu3(1) = -b\*De\*exp(-b\*R) &

- (2\*BB\*e1a+8\*CC\*e4a)\*alpha1\*(R-Rw)\*(1d0-cos(2\*phi))

!--- (d U3)/(d phi)

gradu3(2) = 2\*W1\*sin(2\*phi)

!--- (d U3)/(d theta)

gradu3(3) = 0d0

!--- use B matrix to convert dU3/dq to dU3/dx

iatomlist(1:4) = (/12,13,0,0/)

call calcbmat(bmatrow, x, iatomlist, 1)

bmat(1,:) = bmatrow(:)

!--- special coordinate in place of CCSC torsion

iatomlist(1:5) = (/2,3,4,12,13/)

call calcbmat(bmatrow, x, iatomlist, 6)

bmat(2,:) = bmatrow(:)

guu(:,:,3,3) = guu(:,:,3,3) &

+reshape(matmul(gradu3, bmat), (/3,natom/))

end subroutine calcu3\_ps

!=================================================================

! \*def calu12\_ps

! Calculate prim.+sec. U12 and its grad. w.r.t. Cartesians.

! Input:

! uu: primary+secondary U matrix in hartrees

! guu: gradient array (in hatrees/Angstrom)

! x: Cartesian coordinates in Angstroms

! qps: values of primary & secondary internal coordinates

! Output:

! uu: primary+secondary U w/ calculated U12\_PS added

! guu: grad. array w/ calculated dU12/dx added to guu(:,:,1,2)

!=================================================================

subroutine calcu12\_ps(uu, guu, x, qps)

implicit none

double precision :: uu(nstate,nstate) &

, guu(3,natom,nstate,nstate), x(3,natom), qps(:)

double precision :: R, phi, theta, gradu12(3) &

, B(0:4), alpha(2:4), k(2:4), R0(2:4), u12, dkdR

integer :: i, iatomlist(10)

double precision :: bmatrow(3\*natom), bmat(3,3\*natom)

!--- B(0) is average of intercept over all R

B(0) = -3d-4

B(2) = 0.01911d0

alpha(2) = 2.185135878d0

R0(2) = 1.96523d0

B(4) = -0.0185d0

alpha(4) = 2.332225304d0

R0(4) = 2.0054d0

R = qps(1); phi = qps(2); theta = qps(3)

!--- U12

u12 = B(0)

do i = 2, 4, 2

k(i) = B(i)\*exp(-alpha(i)\*(R-R0(i))\*\*2)

u12 = u12+k(i)\*sin(phi)\*\*i

end do

uu(1,2) = uu(1,2)+u12; uu(2,1) = uu(1,2)

!--- dU12

gradu12(:) = 0d0

do i = 2, 4, 2

!--- (d U12)/(d R)

dkdR = -2\*alpha(i)\*(R-R0(i))\*k(i)

gradu12(1) = gradu12(1)+dkdR\*sin(phi)\*\*i

!--- (d U12)/(d phi)

gradu12(2) = gradu12(2)+i\*k(i)\*sin(phi)\*\*(i-1)\*cos(phi)

!--- (d U12)/(d theta) has been set to zero

end do

!--- use B matrix to convert dU12/dq to dU12/dx

iatomlist(1:4) = (/12,13,0,0/)

call calcbmat(bmatrow, x, iatomlist, 1)

bmat(1,:) = bmatrow(:)

!--- special coordinate in place of CCSC torsion

iatomlist(1:5) = (/2,3,4,12,13/)

call calcbmat(bmatrow, x, iatomlist, 6)

bmat(2,:) = bmatrow(:)

iatomlist(1:4) = (/3,12,13,0/)

call calcbmat(bmatrow, x, iatomlist, 2)

bmat(3,:) = bmatrow(:)

guu(:,:,1,2) = guu(:,:,1,2) &

+reshape(matmul(gradu12, bmat), (/3,natom/))

guu(:,:,2,1) = guu(:,:,1,2)

end subroutine calcu12\_ps

!=================================================================

! \*def calu13\_ps

! Calculate prim.+sec. U13 and its grad. w.r.t. Cartesians.

! Input:

! uu: primary+secondary U matrix in hartrees

! guu: gradient array (in hatrees/Angstrom)

! x: Cartesian coordinates in Angstroms

! qps: values of primary & secondary internal coordinates

! Output:

! uu: primary+secondary U w/ calculated U13\_PS added

! guu: grad. array w/ calculated dU13/dx added to guu(:,:,1,3)

!=================================================================

subroutine calcu13\_ps(uu, guu, x, qps)

implicit none

double precision :: uu(nstate,nstate) &

, guu(3,natom,nstate,nstate), x(3,natom), qps(:)

double precision :: R, phi, theta, gradu13(3), u13 &

, B2, alpha2, R2, c0, c1, alpha4, k(2:4), dkdR(2:4)

integer :: i, iatomlist(10)

double precision :: bmatrow(3\*natom), bmat(3,3\*natom)

B2 = -0.09737d0

alpha2 = 0.848925978d0

R2 = 1.507d0

c0 = -6.89806d0

c1 = 3.90215d0

alpha4 = 2.86681d0

R = qps(1); phi = qps(2); theta = qps(3)

!--- U13

u13 = 0d0

k(2) = B2\*exp(-alpha2\*(R-R2)\*\*2)

k(4) = (c0+c1\*R)\*exp(-alpha4\*R)

do i = 2, 4, 2

u13 = u13+k(i)\*sin(i\*phi)

end do

uu(1,3) = uu(1,3)+u13; uu(3,1) = uu(1,3)

!--- dU13

gradu13(:) = 0d0

dkdR(2) = -2\*alpha2\*(R-R2)\*k(2)

dkdR(4) = (-alpha4\*(c0+c1\*R)+c1)\*exp(-alpha4\*R)

do i = 2, 4, 2

!--- (d U13)/(d R)

gradu13(1) = gradu13(1)+dkdR(i)\*sin(i\*phi)

!--- (d U13)/(d phi)

gradu13(2) = gradu13(2)+i\*k(i)\*cos(i\*phi)

!--- (d U13)/(d theta) has been set to zero

end do

!--- use B matrix to convert dU13/dq to dU13/dx

iatomlist(1:4) = (/12,13,0,0/)

call calcbmat(bmatrow, x, iatomlist, 1)

bmat(1,:) = bmatrow(:)

!--- special coordinate in place of CCSC torsion

iatomlist(1:5) = (/2,3,4,12,13/)

call calcbmat(bmatrow, x, iatomlist, 6)

bmat(2,:) = bmatrow(:)

iatomlist(1:4) = (/3,12,13,0/)

call calcbmat(bmatrow, x, iatomlist, 2)

bmat(3,:) = bmatrow(:)

guu(:,:,1,3) = guu(:,:,1,3) &

+reshape(matmul(gradu13, bmat), (/3,natom/))

guu(:,:,3,1) = guu(:,:,1,3)

end subroutine calcu13\_ps

!=================================================================

! \*def calu23\_ps

! Calculate prim.+sec. U23 and its grad. w.r.t. Cartesians.

! Input:

! uu: primary+secondary U matrix in hartrees

! guu: gradient array (in hatrees/Angstrom)

! x: Cartesian coordinates in Angstroms

! qps: values of primary & secondary internal coordinates

! Output:

! uu: primary+secondary U w/ calculated U23\_PS added

! guu: grad. array w/ calculated dU23/dx added to guu(:,:,2,3)

!=================================================================

subroutine calcu23\_ps(uu, guu, x, qps)

implicit none

double precision :: uu(nstate,nstate) &

, guu(3,natom,nstate,nstate), x(3,natom), qps(:)

double precision :: R, phi, theta, gradu23(3), u23 &

, B(4:6), alpha(4:6), R0(4:6), k(4:6), dkdR

integer :: i, iatomlist(10)

double precision :: bmatrow(3\*natom), bmat(3,3\*natom)

B(4) = -0.00485d0

alpha(4) = 9.143137602d0

R0(4) = 1.98608d0

B(6) = 0.00114d0

alpha(6) = 8.827060236d0

R0(6) = 2.12743d0

R = qps(1); phi = qps(2); theta = qps(3)

!--- U23

u23 = 0d0

do i = 4, 6, 2

k(i) = B(i)\*exp(-alpha(i)\*(R-R0(i))\*\*2)

u23 = u23+k(i)\*sin(i\*phi)

end do

uu(2,3) = uu(2,3)+u23; uu(3,2) = uu(2,3)

!--- dU23

gradu23(:) = 0d0

do i = 4, 6, 2

!--- (d U23)/(d R)

dkdR = -2\*alpha(i)\*(R-R0(i))\*k(i)

gradu23(1) = gradu23(1)+dkdR\*sin(i\*phi)

!--- (d U23)/(d phi)

gradu23(2) = gradu23(2)+i\*k(i)\*cos(i\*phi)

!--- (d U23)/(d theta) has been set to zero

end do

!--- use B matrix to convert dU23/dq to dU23/dx

iatomlist(1:4) = (/12,13,0,0/)

call calcbmat(bmatrow, x, iatomlist, 1)

bmat(1,:) = bmatrow(:)

!--- special coordinate in place of CCSC torsion

iatomlist(1:5) = (/2,3,4,12,13/)

call calcbmat(bmatrow, x, iatomlist, 6)

bmat(2,:) = bmatrow(:)

iatomlist(1:4) = (/3,12,13,0/)

call calcbmat(bmatrow, x, iatomlist, 2)

bmat(3,:) = bmatrow(:)

guu(:,:,2,3) = guu(:,:,2,3) &

+reshape(matmul(gradu23, bmat), (/3,natom/))

guu(:,:,3,2) = guu(:,:,2,3)

end subroutine calcu23\_ps

!=================================================================

! \*def calcuii\_t

! Calculate tert. diab. pot. and its grad. w.r.t. Cartesians.

! Input:

! x: Cartesian coordinates in Angstroms

! qps: values of primary & secondary internal coordinates

! qtp: values of tert. redund. internals

! nterm: number of terms in each qtp set

! itypeqtp: type of each term (and coord.)

! iqtplist: atom index list array

! Output:

! vt: tertiary Uii in hartrees

! gvt: grad. array w/ calculated dU/dx

!=================================================================

subroutine calcuii\_t(vt, gvt, x, qps, qtp, nterm, itypeqtp, &

iqtplist)

implicit none

double precision :: vt(nstate,nstate) &

, gvt(3,natom,nstate,nstate), x(3,natom), qps(:), qtp(:,:)

integer :: nterm(nqtpset), itypeqtp(ntermmax,nqtpset) &

, iqtplist(4,ntermmax,nqtpset)

integer :: i, j, imin

double precision :: vmin, vtmp, gvtmp(3,natom)

vt = 0d0; gvt = 0d0

!--- calculate vt and gvt in diabatic order

!--- uii\_t is the actual routine

do i = 1, nstate

call uii\_t(vt, gvt, x, qps, qtp, nterm, itypeqtp, &

iqtplist, i)

end do

end subroutine calcuii\_t

!--- Actual core routine for calcuii\_t

!--- Parameters have the same meaning as in calcuii\_t

subroutine uii\_t(vt, gvt, x, qps, qtp, nterm, itypeqtp, &

iqtplist, istate)

implicit none

double precision :: vt(nstate,nstate) &

, gvt(3,natom,nstate,nstate), x(3,natom), qps(:), qtp(:,:)

integer :: nterm(nqtpset), itypeqtp(ntermmax,nqtpset) &

, iqtplist(4,ntermmax,nqtpset)

integer :: istate

double precision :: k(ntermmax,napmax) &

, q0(ntermmax,napmax) &

, R0(napmax), phi0(napmax), theta0(napmax) &

, eterm, gterm, energy(0:napmax)

integer :: iap, iterm, nap, n(ntermmax,napmax), ntermap &

, iset, iatomlist(10), iqtpset(napmax)

double precision :: bmatrow(3\*natom), gradc(3,natom,0:napmax)

double precision,allocatable :: bmat(:,:), gg(:)

double precision :: tent(napmax), dtent(0:napmax), R

call assignparamuii\_t(k, q0, n, istate)

!--- set up state-dependent variables

!--- Here 'istate' is diabatic state at phi = 0

energy = 0d0

select case(istate)

case(1)

!--- # of anchor points

nap = 4

!--- anchor points 1-3 use internals set 1

!--- anchor point 4 uses set 2

iqtpset(1:3) = 1

iqtpset(4) = 2

!--- prim&sec coord. of each anchor point

if(debug2) then

R0(1:4) = (/1.8d0,100d0,200d0,300d0/)

else

R0(1:4) = (/1.8d0,2.2d0,3.2d0,6.0d0/)

end if

phi0(1:4) = 0d0

!--- const. part of tert. potential

!--- which is relaxed minus unrelaxed energy at APs

energy(1:4) = (/&

-0.000583020570508334,&

-0.002661649641063303,&

-0.011739718798774815,&

-0.0097357454764833776/)

case(2)

!--- # of anchor points

nap = 2

!--- anchor points 1-2 use internals set 1

iqtpset(1:2) = 1

!--- prim&sec coord. of each anchor point

if(debug2) then

R0(1:2) = (/1.8d0,100d0/)

else

R0(1:2) = (/1.8d0,2.2d0/)

end if

phi0(1:2) = 0d0

!--- const. part of tert. potential

!--- which is relaxed minus unrelaxed energy at APs

energy(1:2) = (/&

-0.0093642039986545408,&

-0.01134068516458372 /)

case(3)

!--- # of anchor points

nap = 4

!--- anchor points 1-3 uses internals set 1

!--- anchor point 4 uses set 2

iqtpset(1:3) = 1

iqtpset(4) = 2

!--- prim&sec coord. of each anchor point

if(debug2) then

R0(1:4) = (/1.9d0,100d0,200d0,300d0/)

else

R0(1:4) = (/1.9d0,2.2d0,3.2d0,6.0d0/)

end if

phi0(1:4) = 0d0

!--- const. part of tert. potential

!--- which is relaxed minus unrelaxed energy at APs

energy(1:4) = (/&

-0.0070903858216569311,&

-0.0061801905494431415,&

-0.022128914781961655 ,&

-0.0089441460466859052/)

end select

!--- loop over anchor points

do iap = 1, nap

ntermap = nterm(iqtpset(iap))

iset = iqtpset(iap)

allocate(bmat(ntermap,3\*natom),gg(ntermap))

do iterm = 1, ntermap

!--- energy and grad. w.r.t. internals

call calcFFterm(eterm, gterm, k(iterm,iap), &

q0(iterm,iap), n(iterm,iap), &

qtp(iterm,iset), &

itypeqtp(iterm, iset))

energy(iap) = energy(iap)+eterm

gg(iterm) = gterm

!--- calc. B matrix row

iatomlist(1:4) = iqtplist(:,iterm,iqtpset(iap))

call calcbmat(bmatrow, x, iatomlist, &

itypeqtp(iterm, iset))

bmat(iterm,:) = bmatrow(:)

end do

!--- use B matrix to convert to grad. w.r.t. Cartesians

gradc(:,:,iap) = reshape(matmul(gg, bmat), (/3,natom/))

deallocate(bmat,gg)

end do

!--- calculate tent func. and dtent/dR

R = qps(1)

do iap = 1, nap

call calctent1(tent(iap), dtent(iap), R, R0, &

iap-1, iap, iap+1, nap)

end do

energy(0) = 0d0; gradc(:,:,0) = 0d0; dtent(0) = 0d0

do iap = 1, nap

!--- interpolate anchor point values using tent func.

energy(0) = energy(0)+energy(iap)\*tent(iap)

gradc(:,:,0) = gradc(:,:,0)+gradc(:,:,iap)\*tent(iap)

!--- interpolate dtent/dR to get dV/dR

dtent(0) = dtent(0)+dtent(iap)\*energy(iap)

end do

!--- use B matrix to convert dV/dR to dV/dx

iatomlist(1:4) = (/12,13,0,0/)

call calcbmat(bmatrow, x, iatomlist, 1)

gradc(:,:,0) = gradc(:,:,0) &

+reshape(dtent(0)\*bmatrow, (/3,natom/))

!--- collect results

vt(istate,istate) = energy(0)

gvt(:,:,istate,istate) = gradc(:,:,0)

if(debug2) then

vt(istate,istate) = energy(1)

gvt(:,:,istate,istate) = gradc(:,:,1)

end if

end subroutine uii\_t

!=================================================================

! \*def calcuij\_t

! Calculate tert. diab. coupl. and its grad. w.r.t. Cart.

! Input:

! uu: diab. matrix (in hartrees)

! guu: gradient array (in hatrees/Angstrom)

! x: Cartesian coordinates in Angstroms

! qps: values of primary & secondary internal coordinates

! qtc: values of tert. internals for couplings

! nqtc: number of qtc

! qtcsym: symmetry (a' or a") of qtc

! qtctyp: type of qtc (CSC bend or other)

! bmatqtc: B matrix, B(i, j) = dqtc(i)/dx(j)

! Output:

! uu: diab. matrix w/ calculated uij added

! guu: grad. array w/ calculated duij/dx added

!=================================================================

subroutine calcuij\_t(uu, guu, x, qps, qtc, nqtc, &

qtcsym, qtctyp, bmatqtc)

implicit none

double precision :: uu(nstate,nstate), qps(:), qtc(:) &

, guu(3,natom,nstate,nstate), x(3,natom) &

, bmatqtc(ntermmax,natom\*3)

integer :: nqtc, qtcsym(ntermmax), qtctyp(ntermmax)

double precision :: uij, duij(3,natom)

call uij\_t(uij, duij, x, qps, qtc, &

nqtc, qtcsym, qtctyp, bmatqtc, 12)

uu(1,2) = uu(1,2)+uij; uu(2,1) = uu(1,2)

guu(:,:,1,2) = guu(:,:,1,2)+duij(:,:)

guu(:,:,2,1) = guu(:,:,1,2)

call uij\_t(uij, duij, x, qps, qtc, &

nqtc, qtcsym, qtctyp, bmatqtc, 13)

uu(1,3) = uu(1,3)+uij; uu(3,1) = uu(1,3)

guu(:,:,1,3) = guu(:,:,1,3)+duij(:,:)

guu(:,:,3,1) = guu(:,:,1,3)

call uij\_t(uij, duij, x, qps, qtc, &

nqtc, qtcsym, qtctyp, bmatqtc, 23)

uu(2,3) = uu(2,3)+uij; uu(3,2) = uu(2,3)

guu(:,:,2,3) = guu(:,:,2,3)+duij(:,:)

guu(:,:,3,2) = guu(:,:,2,3)

end subroutine calcuij\_t

!--- Actual core routine of calcuij\_t

!--- Parameters have the same meaning as in calcuij\_t

subroutine uij\_t(uij, duij, x, qps, qtc, nqtc, &

qtcsym, qtctyp, bmatqtc, istate)

implicit none

double precision :: uij, duij(3,natom), x(3,natom), qps(:) &

, qtc(:), bmatqtc(ntermmax,natom\*3)

integer :: nqtc, qtcsym(ntermmax), qtctyp(ntermmax), istate

integer,parameter :: nR0=2, nphi0=5, nk=2

integer :: iR0, iphi0, iqtc, iatomlist(10)

double precision, allocatable :: gg(:), bmat(:,:)

double precision :: ee(nR0,nphi0), e, g &

, k(nk,nR0,nphi0,nqtc) &

, gradc(3,natom,nR0,nphi0), R, phi, qtc0(nqtc) &

, tentR(0:nR0), dtentR(0:nR0), R0(nR0), phi0(nphi0) &

, tentphi(0:nphi0), dtentphi(0:nphi0), cos2phi0(nphi0) &

, qtcv(nqtc), bmatrow(3\*natom) &

, sij(nphi0), dsij(nphi0)

!--- initialization

!--- prim&sec coord.

R = qps(1)

phi = qps(2)

qtcv(1:nqtc) = qtc(1:nqtc)

!--- R and phi values at anchor points

R0(:) = (/1.97d0,3.5d0/)

phi0(:) = (/0d0,10d0,45d0,80d0,90d0/)

phi0(:) = phi0(:)/180d0\*pi

do iphi0 = 1, nphi0

cos2phi0(iphi0) = -cos(2\*phi0(iphi0))

end do

!--- state-dependent params

call assignparamuij\_t(k, qtc0, nqtc, nk, nR0, nphi0, istate)

ee = 0d0

allocate(gg(nqtc),bmat(nqtc,3\*natom))

bmat(1:nqtc,:) = bmatqtc(1:nqtc,:)

!--- loop over anchor points (each with 2 subscripts iR0,iphi0)

do iR0 = 1, nR0

do iphi0 = 1, nphi0

do iqtc = 1, nqtc

!--- Calculate uij and duij/dqtc at anchor points:

!--- the working unit for calcTDCterm is degree;

!--- this is an exception due to the parameterization

call calcTDCterm(e, g, &

k(:,iR0,iphi0,iqtc), nk, qtc0(iqtc), &

qtcv(iqtc)/pi\*180d0, qtctyp(iqtc))

if(iqtc.eq.2 .or. iqtc.eq.8 .or. iqtc.eq.11) then

e = 0d0; g = 0d0

end if

if (iphi0.eq.1 .or. iphi0.eq.nphi0) then

e = 0d0; g = 0d0

end if

!--- convert gg from deg^-1 to rad^-1

g = g\*180d0/pi

ee(iR0,iphi0) = ee(iR0,iphi0)+e

gg(iqtc) = g

end do

!--- use B matrix to convert to grad. w.r.t. Cartesians

gradc(:,:,iR0,iphi0) = &

reshape(matmul(gg, bmat), (/3,natom/))

end do

end do

deallocate(gg,bmat)

!--- calculate tent func., sij, and dtentR/dR & dtentphi/dphi

!--- & dsij/dphi

do iR0 = 1, nR0

call calctent1(tentR(iR0), dtentR(iR0), R, R0, &

iR0-1, iR0, iR0+1, nR0)

end do

do iphi0 = 1, nphi0

call calctent1(tentphi(iphi0), dtentphi(iphi0), &

-cos(2\*phi), cos2phi0, &

iphi0-1, iphi0, iphi0+1, nphi0)

!--- argument of tent func. is actually -cos(2\*phi)

!--- dtentphi/dphi = dtentphi/d-cos2phi \* d-cos2phi/dphi

dtentphi(iphi0) = dtentphi(iphi0)\*(2\*sin(2\*phi))

if (istate.eq.12) then

sij(iphi0) = 1d0

dsij(iphi0) = 0d0

else

sij(iphi0) = sign(1d0, sin(2\*phi))

dsij(iphi0) = 0d0

end if

end do

uij = 0d0; duij(:,:) = 0d0

dtentR(0) = 0d0; dtentphi(0) = 0d0

do iR0 = 1, nR0

do iphi0 = 1, nphi0

!--- interpolate anchor point values using tent func.

!--- to get uij and duij/dqtc

uij = uij+ &

ee(iR0,iphi0)\*tentR(iR0)\*tentphi(iphi0)&

\*sij(iphi0)

duij(:,:) = duij(:,:) &

+gradc(:,:,iR0,iphi0)\*tentR(iR0)\*tentphi(iphi0)&

\*sij(iphi0)

!--- interpolate dtentq/dq to get duij/dq (q=R, phi)

!--- (dtentq(0) is duij/dq)

dtentR(0) = dtentR(0) &

+ee(iR0,iphi0)\*dtentR(iR0)\*tentphi(iphi0)&

\*sij(iphi0)

dtentphi(0) = dtentphi(0) &

+ee(iR0,iphi0)\*tentR(iR0)&

\*(dtentphi(iphi0)\*sij(iphi0)&

+tentphi(iphi0)\*dsij(iphi0))

end do

end do

!--- use B matrix to convert duij/dq to duij/dx

iatomlist(1:4) = (/12,13,0,0/)

call calcbmat(bmatrow, x, iatomlist, 1)

duij(:,:) = duij(:,:) &

+reshape(dtentR(0)\*bmatrow, (/3,natom/))

!--- special coordinate in place of CCSC torsion

iatomlist(1:5) = (/2,3,4,12,13/)

call calcbmat(bmatrow, x, iatomlist, 6)

duij(:,:) = duij(:,:) &

+reshape(dtentphi(0)\*bmatrow, (/3,natom/))

if(debug2) then

uij = ee(1,1)

duij(:,:) = gradc(:,:,1,1)

end if

end subroutine uij\_t

!=================================================================

! \*def calcbornmayer

! Calculate Born-Mayer potential for nonbonded para C atoms

! Input:

! uu: diab. matrix (in hartrees)

! guu: gradient array (in hatrees/Angstrom)

! x: Cartesian coordinates in Angstroms

! Output:

! uu: diab. matrix w/ calculated B-M potential added

! guu: grad. array w/ calculated grad. of B-M pot. added

!=================================================================

subroutine calcbornmayer(uu, guu, x)

implicit none

double precision :: uu(nstate,nstate) &

, guu(3,natom,nstate,nstate), x(3,natom)

double precision :: dist(3), BB, alpha, bmatrow(natom\*3), Vbm &

, dVdr

integer :: ic, is, iatomlist(10,3)

dist(1) = evalBL(x, 1, 4); iatomlist(1:4,1) = (/1,4,0,0/)

dist(2) = evalBL(x, 2, 5); iatomlist(1:4,2) = (/2,5,0,0/)

dist(3) = evalBL(x, 3, 6); iatomlist(1:4,3) = (/3,6,0,0/)

!--- parameters: BB=42000kcal=66.931hartree, alpha=3.58 A-1

BB = 66.931; alpha = 3.58

do ic = 1, 3

Vbm = BB\*exp(-alpha\*dist(ic))

dVdr = -alpha\*Vbm

call calcbmat(bmatrow, x, iatomlist(:,ic), 1)

do is = 1, nstate

uu(is,is) = uu(is,is)+Vbm

guu(:,:,is,is) = guu(:,:,is,is) &

+reshape(dVdr\*bmatrow, (/3,natom/))

end do

end do

end subroutine calcbornmayer

!=================================================================

! \*def calcFFterm

! Calculate one FF term and its gradient.

! Input:

! k: force constant

! q0: rest value

! n: multiplicity (for torsion) or dummy (for others)

! q: internal coordinate

! itype: type of term

! Output:

! ee: energy

! gg: gradient w.r.t. the internal coordinate

!=================================================================

subroutine calcFFterm(ee, gg, k, q0, n, q, itype)

implicit none

double precision :: ee, gg, k, q0, q

integer :: n, itype

select case (itype)

!--- bond length

case(1)

ee = 0.5d0\*k\*(1d0-q0/q)\*\*2

gg = k\*(1d0-q0/q)\*q0/q\*\*2

!--- bond angle

case(2)

ee = 0.5d0\*k\*(cos(q)-cos(q0))\*\*2

gg = -k\*(cos(q)-cos(q0))\*sin(q)

!--- torsion

case(3)

ee = 0.5d0\*k\*(1d0-cos(n\*(q-q0)))

gg = 0.5d0\*n\*k\*sin(n\*(q-q0))

!--- oop distance

case(5)

ee = 0.5d0\*k\*(q-q0)\*\*2

gg = k\*(q-q0)

end select

end subroutine calcFFterm

!=================================================================

! \*def calcTDCterm

! Calculate tert. diab. coupl. term and its grad.,

! which is a nk-degree polynomial (w/ intercept=0).

! (Unlike most internal routines, the input and output unit

! here for angles is degree due to parametrization.)

! Input:

! k: coefficients of polynomial

! nk: number of degree of polynomial

! q0: center of expansion

! q: value of internal coordinate

! ityp: type of TDC term; see below

! Output:

! ee: diab. coupling

! gg: gradient w.r.t. the internal coordinate

!=================================================================

subroutine calcTDCterm(ee, gg, k, nk, q0, q, ityp)

implicit none

integer :: nk, ityp

double precision :: ee, gg, k(nk), q0, q

integer :: i

double precision :: dq, cc, expo, qq, qq0

select case(ityp)

!--- ityp 1: use the coordinate as variable

case(1)

dq = q-q0

ee = 0d0; gg = 0d0

do i = 1, nk

ee = ee+k(i)\*dq\*\*i

gg = gg+i\*k(i)\*dq\*\*(i-1)

end do

!--- damp the couplings at large dq

cc = 100d0

expo = exp(-dq\*\*2/cc\*\*2)

gg = gg\*expo+ee\*(-2\*dq/cc\*\*2)\*expo

ee = ee\*expo

!--- ityp 2: use cosine term as variable

!--- for bends involving a breaking bond

!--- Note that q needs deg->rad first,

!--- and gg needs rad-1 -> deg-1 last

case(2)

qq = q/180d0\*pi; qq0 = q0/180d0\*pi

dq = cos(qq)-cos(qq0)

ee = 0d0; gg = 0d0

do i = 1, nk

ee = ee+k(i)\*dq\*\*i

gg = gg+i\*k(i)\*dq\*\*(i-1)

end do

gg = -gg\*sin(qq) /180d0\*pi

end select

end subroutine calcTDCterm

!=================================================================

! \*def calctent1

! Calculate tent func. and its grad.

! (for interpolating R)

! Input:

! q: internal coordinate as argument

! q0: ref. q values at anchor points

! i, j, k: anchor point indexes

! nq0: number of anchor points

! Output:

! t: tent function value (unitless)

! dtdq: dt/dq

!=================================================================

subroutine calctent1(t, dtdq, q, q0, i, j, k, nq0)

implicit none

double precision :: t, dtdq, q, q0(:)

integer :: i, j, k, nq0

!--- if j is the 1st anchor point

if(i.eq.0) then

if(q<q0(j)) then

t = 1d0; dtdq = 0d0

else if(q>q0(k)) then

t = 0d0; dtdq = 0d0

else

call tent1(t, dtdq, q, q0(k), q0(j))

end if

!--- if j is the last anchor point

else if(k>nq0) then

if(q>q0(j)) then

t = 1d0; dtdq = 0d0

else if(q<q0(i)) then

t = 0d0; dtdq = 0d0

else

call tent1(t, dtdq, q, q0(i), q0(j))

end if

else

if(q<q0(i).or.q>q0(k)) then

t = 0d0; dtdq = 0d0

else if(q<q0(j)) then

call tent1(t, dtdq, q, q0(i), q0(j))

else

call tent1(t, dtdq, q, q0(k), q0(j))

end if

end if

end subroutine calctent1

!--- called by calctent1

!--- tent(q1) = 0, tent(q2) = 1

subroutine tent1(t, dtdq, q, q1, q2)

implicit none

double precision :: t, dtdq, q, q1, q2

double precision :: dq1, dq2

dq1 = q-q1

dq2 = q-q2

t = dq1\*\*4/(dq1\*\*4+dq2\*\*4)

dtdq = ( 4\*dq1\*\*3\*(dq1\*\*4+dq2\*\*4)-4\*dq1\*\*4\*(dq1\*\*3+dq2\*\*3) ) &

/ (dq1\*\*4+dq2\*\*4)\*\*2

end subroutine tent1

!=================================================================

! \*def evalBL

! Get bond length of atoms i and j.

! Input:

! x: Cartesian coordinates in Angstroms

! i, j: atom indices

! Return:

! bond length in Angstroms

!=================================================================

double precision function evalBL(x, i, j)

implicit none

integer :: i, j

double precision :: x(3,natom)

double precision :: r(3)

r(:) = x(:,i)-x(:,j)

evalBL = sqrt(dot\_product(r, r))

end function evalBL

!=================================================================

! \*def evalBA

! Get bond angle of atoms i ,j, k.

! Input:

! x: Cartesian coordinates in Angstroms

! i, j, k: atom indices

! Return:

! bond angle in radians

!=================================================================

double precision function evalBA(x, i, j, k)

implicit none

integer :: i, j, k

double precision :: x(3,natom)

double precision :: eji(3), ejk(3), cosBA

eji(:) = (x(:,i)-x(:,j))/evalBL(x, i, j)

ejk(:) = (x(:,k)-x(:,j))/evalBL(x, j, k)

cosBA = dot\_product(eji, ejk)

!--- avoid numerical problem

if(cosBA>1) then

evalBA = 0d0

else if(cosBA<-1) then

evalBA = pi

else

evalBA = acos(cosBA)

end if

end function evalBA

!=================================================================

! \*def evalTO

! Get torsion of atoms i ,j, k, l.

! Input:

! x: Cartesian coordinates in Angstroms

! i, j, k, l: atom indices

! Return:

! torsion in radians

!=================================================================

double precision function evalTO(x, i, j, k, l)

implicit none

integer :: i, j, k, l

double precision :: x(3,natom)

integer :: sign

double precision :: eij(3), ejk(3), ekl(3) &

, sinijk, sinjkl, cosTO &

, cross1(3), cross2(3), cross3(3)

eij(:) = (x(:,j)-x(:,i))/evalBL(x, i, j)

ejk(:) = (x(:,k)-x(:,j))/evalBL(x, j, k)

ekl(:) = (x(:,l)-x(:,k))/evalBL(x, k, l)

sinijk = sin(evalBA(x, i, j, k))

sinjkl = sin(evalBA(x, j, k, l))

call xprod(cross1, eij, ejk)

call xprod(cross2, ejk, ekl)

cosTO = dot\_product(cross1, cross2)/sinijk/sinjkl

!--- determine sign of torsion

call xprod(cross1, eij, ejk)

call xprod(cross2, ejk, ekl)

call xprod(cross3, cross1, cross2)

if(dot\_product(cross3, ejk)>0) then

sign = 1

else

sign = -1

end if

!--- avoid numerical problem

if(cosTO>1) then

evalTO = 0d0

else if(cosTO<-1) then

evalTO = pi

else

evalTO = sign\*acos(cosTO)

end if

end function evalTO

!=================================================================

! \*def evalOB

! Get out-of-plane bend of atoms i ,j, k, l.

! Input:

! x: Cartesian coordinates in Angstroms

! i, j, k, l: atom indices

! Return:

! oop bend in radians

!=================================================================

double precision function evalOB(x, i, j, k, l)

implicit none

integer :: i, j, k, l

double precision :: x(3,natom)

double precision :: eli(3), elj(3), elk(3), sinjlk, sinOB &

, cross1(3)

eli(:) = (x(:,i)-x(:,l))/evalBL(x, l, i)

elj(:) = (x(:,j)-x(:,l))/evalBL(x, l, j)

elk(:) = (x(:,k)-x(:,l))/evalBL(x, l, k)

sinjlk = sin(evalBA(x, j, l, k))

call xprod(cross1, elj, elk)

sinOB = dot\_product(cross1, eli)/sinjlk

!--- avoid numerical problem

if(sinOB>1) then

evalOB = pi/2

else if(sinOB<-1) then

evalOB = -pi/2

else

evalOB = asin(sinOB)

end if

end function evalOB

!=================================================================

! \*def evalOD

! Get out-of-plane distance of atoms i ,j, k, l.

! Input:

! x: Cartesian coordinates in Angstroms

! i, j, k, l: atom indices

! Return:

! oop distance in Angstroms

!=================================================================

double precision function evalOD(x, i, j, k, l)

implicit none

integer :: i, j, k, l

double precision :: x(3,natom)

double precision :: eij(3), eik(3), eijxeik(3), ril(3), sinjik

ril(:) = x(:,l)-x(:,i)

eij(:) = (x(:,j)-x(:,i))/evalBL(x, i, j)

eik(:) = (x(:,k)-x(:,i))/evalBL(x, i, k)

call xprod(eijxeik, eij, eik)

sinjik = sin(evalBA(x, j, i, k))

evalOD = abs(dot\_product(ril, eijxeik)/sinjik)

end function evalOD

!=================================================================

! \*def evalSP1

! Get a special coordinate to replace torsion CCSC

! Input:

! x: Cartesian coordinates in Angstroms

! i, j, k, l, m: atom indices

! Return:

! The coordinate

!=================================================================

double precision function evalSP1(x, i, j, k, l, m)

implicit none

integer :: i, j, k, l, m

double precision :: x(3,natom)

double precision :: e12(3), e23(3), e13(3), eZ(3), eX(3), ep(3) &

, e45(3), e24(3), etmp(3), etmp2(3), etmp3(3) &

, sin123, sin13Z, cos13Z, e12xe23(3), cosSP1, sgn

e12(:) = (x(:,j)-x(:,i))/evalBL(x, i, j)

e23(:) = (x(:,k)-x(:,j))/evalBL(x, j, k)

e13(:) = (x(:,k)-x(:,i))/evalBL(x, i, k)

e45(:) = (x(:,m)-x(:,l))/evalBL(x, l, m)

e24(:) = (x(:,l)-x(:,j))/evalBL(x, j, l)

call xprod(etmp, e12, e23)

eZ(:) = etmp(:)/sqrt(dot\_product(etmp, etmp))

call xprod(etmp, e13, eZ)

eX(:) = etmp(:)/sqrt(dot\_product(etmp, etmp))

etmp(:) = e45(:)-eX(:)\*dot\_product(e45, eX)

ep(:) = etmp(:)/sqrt(dot\_product(etmp, etmp))

cosSP1 = dot\_product(ep, e13)

!--- determine sign

call xprod(etmp, e13, ep)

if(dot\_product(etmp, eX)>0) then

sgn = 1

else

sgn = -1

end if

!--- avoid numerical problem

if(cosSP1>1) then

evalSP1 = 0d0

else if(cosSP1<-1) then

evalSP1 = pi

else

evalSP1 = sgn\*acos(cosSP1)

end if

end function evalSP1

!=================================================================

! \*def xprod

! Cross product of two 3D vectors.

! Input:

! a(3), b(3): two vectors

! Return:

! axb(3): = a x b

!=================================================================

subroutine xprod(axb, a, b)

implicit none

double precision :: axb(3), a(3), b(3)

axb(1) = a(2)\*b(3)-a(3)\*b(2)

axb(2) = a(3)\*b(1)-a(1)\*b(3)

axb(3) = a(1)\*b(2)-a(2)\*b(1)

end subroutine xprod

!=================================================================

! \*def diagonalize

! Diagonalize a matrix and get all eigen-values and -vectors.

! Input:

! mat: ndim\*ndim matrix to be diagonalized

! ndim: dimension of matrix

! Output:

! eigval: ndim vector containing eigenvalues

! eigvec: ndim\*ndim matrix containing eigenvectors

!=================================================================

subroutine diagonalize(eigval, eigvec, mat, ndim)

implicit none

double precision :: eigval(ndim), eigvec(ndim,ndim) &

, mat(ndim,ndim)

integer :: ndim

double precision, allocatable :: work(:)

integer :: lwork, info

eigvec = mat

lwork = -1

allocate(work(1))

call dsyev('V', 'U', ndim, eigvec, ndim, eigval, work, lwork, info)

lwork = int(work(1))

deallocate(work)

allocate(work(lwork))

call dsyev('V', 'U', ndim, eigvec, ndim, eigval, work, lwork, info)

deallocate(work)

if (info.ne.0 ) then

write(\*,\*) "dsyev exits abnormally in diagonalization."

stop

end if

end subroutine diagonalize

!=================================================================

! \*def calcbmat

! Calculate a row of B matrix for an internal coordinate.

! Input:

! x: Cartesian coordinates in Angstroms

! iatomlist: indexes of atoms involved in the internal coord.

! inttype: type of the internal coordinate

! Output:

! bmatrow: row of B matrix

!=================================================================

subroutine calcbmat(bmatrow, x, iatomlist, inttype)

implicit none

double precision :: bmatrow(natom\*3), x(3,natom)

integer :: inttype, iatomlist(10)

double precision :: rij, rij1, rij2, rij3, rjk1, rjk2, rjk3, rjk, pijk

double precision :: cijk, sijk, pxmg, pymg, pzmg, sjkl, sijkl

double precision :: rkl, rkl1, rkl2, rkl3, cjkl, cijkl, pjkl

integer :: i, j, ijk, im, k, l, m, ijkl, ilisttmp(4), ii, iii, jj

double precision :: A243,S243,C243,E41(3),E42(3),E43(3) &

,R41,R42,R43,OOPB,COOPB,TOOPB,TMPV(3),BMV1(3),BMV2(3),BMV3(3) &

,BMV4(3),e12(3),e13(3),v14(3),r12,r13,theta,rtmp &

,tmpv2(3),tmpv3(3),rowtmp(3\*natom),xtmp(3,natom) &

,coord1,coord2,step

bmatrow(:) = 0d0

select case(inttype)

!--- bond length; adapted from polyrate

case(1)

I=iatomlist(1)

J=iatomlist(2)

RIJ1=X(1,J)-X(1,I)

RIJ2=x(2,J)-x(2,I)

RIJ3=x(3,J)-x(3,I)

RIJ=evalBL(x, i, j)

BMATROW(3\*I-2)=-RIJ1/RIJ

BMATROW(3\*I-1)=-RIJ2/RIJ

BMATROW(3\*I)=-RIJ3/RIJ

BMATROW(3\*J-2)=RIJ1/RIJ

BMATROW(3\*J-1)=RIJ2/RIJ

BMATROW(3\*J)=RIJ3/RIJ

!--- bending; adapted from polyrate

case(2)

I=iatomlist(1)

J=iatomlist(2)

K=iatomlist(3)

RIJ1=X(1,J)-X(1,I)

RIJ2=X(2,J)-X(2,I)

RIJ3=x(3,J)-x(3,I)

RJK1=X(1,K)-X(1,J)

RJK2=X(2,K)-X(2,J)

RJK3=x(3,K)-x(3,J)

RIJ=evalBL(x, I, J)

RJK=evalBL(x, J, K)

PIJK=evalBA(x, I, J, K)

CIJK=cos(PIJK)

SIJK=sin(PIJK)

BMATROW(3\*I-2)=(-CIJK\*RIJ1\*RJK-RIJ\*RJK1)/(SIJK\*RIJ\*\*2\*RJK)

BMATROW(3\*I-1)=(-CIJK\*RIJ2\*RJK-RIJ\*RJK2)/(SIJK\*RIJ\*\*2\*RJK)

BMATROW(3\*I)=(-CIJK\*RIJ3\*RJK-RIJ\*RJK3)/(SIJK\*RIJ\*\*2\*RJK)

BMATROW(3\*J-2)= (-RIJ\*RIJ1\*RJK+CIJK\*RIJ1\*RJK\*\*2-CIJK\*RIJ\*\*2\*RJK1+RIJ\*RJK\*RJK1) &

/(SIJK\*RIJ\*\*2\*RJK\*\*2)

BMATROW(3\*J-1)= (-RIJ\*RIJ2\*RJK+CIJK\*RIJ2\*RJK\*\*2-CIJK\*RIJ\*\*2\*RJK2+RIJ\*RJK\*RJK2) &

/(SIJK\*RIJ\*\*2\*RJK\*\*2)

BMATROW(3\*J)= (-RIJ\*RIJ3\*RJK+CIJK\*RIJ3\*RJK\*\*2-CIJK\*RIJ\*\*2\*RJK3+RIJ\*RJK\*RJK3) &

/(SIJK\*RIJ\*\*2\*RJK\*\*2)

BMATROW(3\*K-2)=(RIJ1\*RJK+CIJK\*RIJ\*RJK1)/(SIJK\*RIJ\*RJK\*\*2)

BMATROW(3\*K-1)=(RIJ2\*RJK+CIJK\*RIJ\*RJK2)/(SIJK\*RIJ\*RJK\*\*2)

BMATROW(3\*K)=(RIJ3\*RJK+CIJK\*RIJ\*RJK3)/(SIJK\*RIJ\*RJK\*\*2)

!--- torsion; adpated from polyrate

case(3)

I=IATOMLIST(1)

J=IATOMLIST(2)

K=IATOMLIST(3)

L=IATOMLIST(4)

RIJ1=X(1,J)-X(1,I)

RIJ2=X(2,J)-X(2,I)

RIJ3=X(3,J)-X(3,I)

RJK1=X(1,K)-X(1,J)

RJK2=X(2,K)-X(2,J)

RJK3=X(3,K)-X(3,J)

RKL1=X(1,L)-X(1,K)

RKL2=X(2,L)-X(2,K)

RKL3=X(3,L)-X(3,K)

RIJ=evalBL(x, I,J)

RJK=evalBL(x, J,K)

RKL=evalBL(x, K,L)

PIJK=evalBA(x, I,J,K)

CIJK=COS(PIJK)

SIJK=SIN(PIJK)

PJKL=evalBA(x, J,K,L)

CJKL=COS(PJKL)

SJKL=SIN(PJKL)

CIJKL=((-RIJ2\*RJK1+RIJ1\*RJK2)\*(-RJK2\*RKL1+RJK1\*RKL2)+ &

(RIJ3\*RJK1-RIJ1\*RJK3)\*(RJK3\*RKL1-RJK1\*RKL3)+ &

(-RIJ3\*RJK2+RIJ2\*RJK3)\*(-RJK3\*RKL2+RJK2\*RKL3))/ &

(SIJK\*SJKL\*RIJ\*RJK\*RJK\*RKL)

SIJKL=((-RIJ3\*RJK2+RIJ2\*RJK3)\*RKL1+(RIJ3\*RJK1-RIJ1\*RJK3)\*RKL2+ &

(-(RIJ2\*RJK1)+RIJ1\*RJK2)\*RKL3)/(RIJ\*RJK\*RKL\*SIJK\*SJKL)

BMATROW(3\*I-2)=SIJKL\*RIJ1/(CIJKL\*SIJK\*\*2\*RIJ\*\*2)+ &

CIJK\*SIJKL\*RJK1/(CIJKL\*SIJK\*\*2\*RIJ\*RJK)+ &

(RJK3\*RKL2-RJK2\*RKL3)/(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)

BMATROW(3\*I-1)=SIJKL\*RIJ2/(CIJKL\*SIJK\*\*2\*RIJ\*\*2)+ &

CIJK\*SIJKL\*RJK2/(CIJKL\*SIJK\*\*2\*RIJ\*RJK)+ &

(-RJK3\*RKL1+RJK1\*RKL3)/(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)

BMATROW(3\*I)=SIJKL\*RIJ3/(CIJKL\*SIJK\*\*2\*RIJ\*\*2)+ &

CIJK\*SIJKL\*RJK3/(CIJKL\*SIJK\*\*2\*RIJ\*RJK)+ &

(RJK2\*RKL1-RJK1\*RKL2)/(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)

BMATROW(3\*J-2)=-(SIJKL\*RIJ1/(CIJKL\*SIJK\*\*2\*RIJ\*\*2))+ &

CIJK\*SIJKL\*(RIJ1-RJK1)/(CIJKL\*SIJK\*\*2\*RIJ\*RJK)- &

SIJKL\*RJK1/(CIJKL\*RJK\*\*2)+SIJKL\*RJK1/(CIJKL\*SIJK\*\*2\*RJK\*\*2)+ &

SIJKL\*RJK1/(CIJKL\*SJKL\*\*2\*RJK\*\*2)+ &

CJKL\*SIJKL\*RKL1/(CIJKL\*SJKL\*\*2\*RJK\*RKL)+ &

(-RIJ3\*RKL2-RJK3\*RKL2+RIJ2\*RKL3+RJK2\*RKL3)/ &

(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)

BMATROW(3\*J-1)=-SIJKL\*RIJ2/(CIJKL\*SIJK\*\*2\*RIJ\*\*2)+ &

CIJK\*SIJKL\*(RIJ2-RJK2)/(CIJKL\*SIJK\*\*2\*RIJ\*RJK)- &

SIJKL\*RJK2/(CIJKL\*RJK\*\*2)+SIJKL\*RJK2/(CIJKL\*SIJK\*\*2\*RJK\*\*2)+ &

SIJKL\*RJK2/(CIJKL\*SJKL\*\*2\*RJK\*\*2)+ &

CJKL\*SIJKL\*RKL2/(CIJKL\*SJKL\*\*2\*RJK\*RKL)+ &

(RIJ3\*RKL1+RJK3\*RKL1-RIJ1\*RKL3-RJK1\*RKL3)/ &

(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)

BMATROW(3\*J)=-SIJKL\*RIJ3/(CIJKL\*SIJK\*\*2\*RIJ\*\*2)+ &

CIJK\*SIJKL\*(RIJ3-RJK3)/(CIJKL\*SIJK\*\*2\*RIJ\*RJK)- &

SIJKL\*RJK3/(CIJKL\*RJK\*\*2)+SIJKL\*RJK3/(CIJKL\*SIJK\*\*2\*RJK\*\*2)+ &

SIJKL\*RJK3/(CIJKL\*SJKL\*\*2\*RJK\*\*2)+ &

(-RIJ2\*RKL1-RJK2\*RKL1+RIJ1\*RKL2+RJK1\*RKL2)/ &

(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)+ &

CJKL\*SIJKL\*RKL3/(CIJKL\*SJKL\*\*2\*RJK\*RKL)

BMATROW(3\*K-2)=-CIJK\*SIJKL\*RIJ1/(CIJKL\*SIJK\*\*2\*RIJ\*RJK)+ &

SIJKL\*RJK1/(CIJKL\*RJK\*\*2)-SIJKL\*RJK1/(CIJKL\*SIJK\*\*2\*RJK\*\*2)- &

SIJKL\*RJK1/(CIJKL\*SJKL\*\*2\*RJK\*\*2)+ &

CJKL\*SIJKL\*(RJK1-RKL1)/(CIJKL\*SJKL\*\*2\*RJK\*RKL)+ &

SIJKL\*RKL1/(CIJKL\*SJKL\*\*2\*RKL\*\*2)+ &

(RIJ3\*RJK2-RIJ2\*RJK3+RIJ3\*RKL2-RIJ2\*RKL3)/ &

(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)

BMATROW(3\*K-1)=-CIJK\*SIJKL\*RIJ2/(CIJKL\*SIJK\*\*2\*RIJ\*RJK)+ &

SIJKL\*RJK2/(CIJKL\*RJK\*\*2)-SIJKL\*RJK2/(CIJKL\*SIJK\*\*2\*RJK\*\*2)- &

SIJKL\*RJK2/(CIJKL\*SJKL\*\*2\*RJK\*\*2)+ &

CJKL\*SIJKL\*(RJK2-RKL2)/(CIJKL\*SJKL\*\*2\*RJK\*RKL)+ &

SIJKL\*RKL2/(CIJKL\*SJKL\*\*2\*RKL\*\*2)+ &

(-RIJ3\*RJK1+RIJ1\*RJK3-RIJ3\*RKL1+RIJ1\*RKL3)/ &

(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)

BMATROW(3\*K)=-CIJK\*SIJKL\*RIJ3/(CIJKL\*SIJK\*\*2\*RIJ\*RJK)+ &

SIJKL\*RJK3/(CIJKL\*RJK\*\*2)-SIJKL\*RJK3/(CIJKL\*SIJK\*\*2\*RJK\*\*2)- &

SIJKL\*RJK3/(CIJKL\*SJKL\*\*2\*RJK\*\*2)+ &

(RIJ2\*RJK1-RIJ1\*RJK2+RIJ2\*RKL1-RIJ1\*RKL2)/ &

(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)+ &

CJKL\*SIJKL\*(RJK3-RKL3)/(CIJKL\*SJKL\*\*2\*RJK\*RKL)+ &

SIJKL\*RKL3/(CIJKL\*SJKL\*\*2\*RKL\*\*2)

BMATROW(3\*L-2)=-CJKL\*SIJKL\*RJK1/(CIJKL\*SJKL\*\*2\*RJK\*RKL)+ &

(-RIJ3\*RJK2+RIJ2\*RJK3)/(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)- &

SIJKL\*RKL1/(CIJKL\*SJKL\*\*2\*RKL\*\*2)

BMATROW(3\*L-1)=-CJKL\*SIJKL\*RJK2/(CIJKL\*SJKL\*\*2\*RJK\*RKL)+ &

(RIJ3\*RJK1-RIJ1\*RJK3)/(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)- &

SIJKL\*RKL2/(CIJKL\*SJKL\*\*2\*RKL\*\*2)

BMATROW(3\*L)=(-RIJ2\*RJK1+RIJ1\*RJK2)/ &

(CIJKL\*SIJK\*SJKL\*RIJ\*RJK\*RKL)- &

CJKL\*SIJKL\*RJK3/(CIJKL\*SJKL\*\*2\*RJK\*RKL)- &

SIJKL\*RKL3/(CIJKL\*SJKL\*\*2\*RKL\*\*2)

!--- out-of-plane bend

case(4)

I=IATOMLIST(1)

J=IATOMLIST(2)

K=IATOMLIST(3)

L=IATOMLIST(4)

e41(1)=X(1,I)-X(1,L)

e41(2)=X(2,I)-X(2,L)

e41(3)=X(3,I)-X(3,L)

e42(1)=X(1,j)-X(1,L)

e42(2)=X(2,j)-X(2,L)

e42(3)=X(3,j)-X(3,L)

e43(1)=X(1,k)-X(1,L)

e43(2)=X(2,k)-X(2,L)

e43(3)=X(3,k)-X(3,L)

R41=evalBL(x, I,L)

R42=evalBL(x, J,L)

R43=evalBL(x, K,L)

e41=e41/R41

e42=e42/R42

e43=e43/r43

a243=evalBA(x, j,l,k)

c243=cos(a243)

s243=sin(a243)

call xprod(tmpv,e42,e43)

oopb=asin(dot\_product(tmpv,e41)/s243)

coopb=cos(oopb)

toopb=tan(oopb)

call xprod(tmpv,e42,e43)

bmv1=(tmpv/coopb/s243-toopb\*e41)/r41

bmatrow(3\*I-2)=bmv1(1)

bmatrow(3\*i-1)=bmv1(2)

bmatrow(3\*i)=bmv1(3)

call xprod(tmpv,e43,e41)

bmv2=(tmpv/coopb/s243-toopb/s243\*\*2\*(e42-e43\*c243))/r42

bmatrow(3\*j-2)=bmv2(1)

bmatrow(3\*j-1)=bmv2(2)

bmatrow(3\*j)=bmv2(3)

call xprod(tmpv,e41,e42)

bmv3=(tmpv/coopb/s243-toopb/s243\*\*2\*(e43-e42\*c243))/r43

bmatrow(3\*k-2)=bmv3(1)

bmatrow(3\*k-1)=bmv3(2)

bmatrow(3\*k)=bmv3(3)

bmv4=-bmv1-bmv2-bmv3

bmatrow(3\*l-2)=bmv4(1)

bmatrow(3\*l-1)=bmv4(2)

bmatrow(3\*l)=bmv4(3)

!--- out-of-plane distance (aka pyramid height)

case(5)

I=IATOMLIST(1)

J=IATOMLIST(2)

K=IATOMLIST(3)

L=IATOMLIST(4)

e12(1)=X(1,j)-X(1,i)

e12(2)=X(2,j)-X(2,i)

e12(3)=X(3,j)-X(3,i)

e13(1)=X(1,k)-X(1,i)

e13(2)=X(2,k)-X(2,i)

e13(3)=X(3,k)-X(3,i)

v14(1)=X(1,l)-X(1,i)

v14(2)=X(2,l)-X(2,i)

v14(3)=X(3,l)-X(3,i)

r12=evalBL(x, i,j)

r13=evalBL(x, i,k)

e12(:)=e12(:)/r12

e13(:)=e13(:)/r13

theta=evalBA(x, j, i, k)

call xprod(tmpv2, e12, e13)

rtmp=dot\_product(v14, tmpv2)

call xprod(tmpv, e13, v14)

tmpv3(:)=(cos(theta)\*e12(:)-e13(:))/r12/sin(theta)

bmv2 = (tmpv-rtmp\*e12)/sin(theta)/r12 &

- cos(theta)\*rtmp/sin(theta)\*\*2\*tmpv3

call xprod(tmpv, v14, e12)

tmpv3(:)=(cos(theta)\*e13(:)-e12(:))/r13/sin(theta)

bmv3 = (tmpv-rtmp\*e13)/sin(theta)/r13 &

- cos(theta)\*rtmp/sin(theta)\*\*2\*tmpv3

call xprod(tmpv, e12, e13)

bmv4 = tmpv/sin(theta)

bmv1 = -bmv2-bmv3-bmv4

if(rtmp/sin(theta)<0) then

bmv1 = -bmv1

bmv2 = -bmv2

bmv3 = -bmv3

bmv4 = -bmv4

end if

bmatrow(3\*I-2)=bmv1(1)

bmatrow(3\*i-1)=bmv1(2)

bmatrow(3\*i)=bmv1(3)

bmatrow(3\*j-2)=bmv2(1)

bmatrow(3\*j-1)=bmv2(2)

bmatrow(3\*j)=bmv2(3)

bmatrow(3\*k-2)=bmv3(1)

bmatrow(3\*k-1)=bmv3(2)

bmatrow(3\*k)=bmv3(3)

bmatrow(3\*l-2)=bmv4(1)

bmatrow(3\*l-1)=bmv4(2)

bmatrow(3\*l)=bmv4(3)

!--- special coordinate replacing torsionCCSC

!--- numerical gradient

case(6)

step = 1d-5

i = iatomlist(1)

j = iatomlist(2)

k = iatomlist(3)

l = iatomlist(4)

m = iatomlist(5)

!--- loop over atoms

do ii = 1, 5

iii = iatomlist(ii)

!--- loop over x,y,z

do jj = 1, 3

xtmp = x

xtmp(jj, iii) = x(jj, iii)+step

coord1 = evalSP1(xtmp, i, j, k, l, m)

xtmp(jj, iii) = x(jj, iii)-step

coord2 = evalSP1(xtmp, i, j, k, l, m)

bmatrow(3\*iii-3+jj) = (coord1-coord2)/step/2

end do

end do

end select

end subroutine calcbmat

!=================================================================

! \*def assignqtp

! Assign attributes of tert. int. for potentials

! Input:

! NA

! Output:

! nterm: number of terms in each qtp set

! itypeqtp: type of each term (and coord.)

! iqtplist: atom index list array

!

! ~200 lines of params

!=================================================================

subroutine assignqtp(nterm, itypeqtp, iqtplist)

implicit none

integer :: nterm(nqtpset), itypeqtp(ntermmax,nqtpset) &

, iqtplist(4,ntermmax,nqtpset)

nterm(1:2) = (/73,67/)

!--- type of each FF term of each set

!--- 1=BL(BondLen), 2=BA(BondAng), 3=TO(Torsion), 5=OD(OOPDist)

!----- set 1, for short R, corresp. to atom connectivity

!----- of Ph-S-CH3

itypeqtp( 1:15,1) = 1

itypeqtp(16:40,1) = 2

itypeqtp(41:67,1) = 3

itypeqtp(68:73,1) = 5

!----- set 2, for long R, corresp. to atom connectivity

!----- of Ph-S + CH3

itypeqtp( 1:15,2) = 1

itypeqtp(16:36,2) = 2

itypeqtp(37:60,2) = 3

itypeqtp(61:67,2) = 5

iqtplist = 0

!--- first set (for short R)

!--- BEGIN generated by 'python gen\_qtp.py qr\_tert\_pot\_1.txt'

iqtplist(1:2, 1,1) = (/ 1, 2/)

iqtplist(1:2, 2,1) = (/ 1, 6/)

iqtplist(1:2, 3,1) = (/ 1, 7/)

iqtplist(1:2, 4,1) = (/ 2, 3/)

iqtplist(1:2, 5,1) = (/ 2, 8/)

iqtplist(1:2, 6,1) = (/13,14/)

iqtplist(1:2, 7,1) = (/13,15/)

iqtplist(1:2, 8,1) = (/13,16/)

iqtplist(1:2, 9,1) = (/ 3, 4/)

iqtplist(1:2, 10,1) = (/ 3,12/)

iqtplist(1:2, 11,1) = (/ 4, 5/)

iqtplist(1:2, 12,1) = (/ 4, 9/)

iqtplist(1:2, 13,1) = (/ 5, 6/)

iqtplist(1:2, 14,1) = (/ 5,10/)

iqtplist(1:2, 15,1) = (/ 6,11/)

iqtplist(1:3, 16,1) = (/ 1, 2, 3/)

iqtplist(1:3, 17,1) = (/ 1, 2, 8/)

iqtplist(1:3, 18,1) = (/ 1, 6, 5/)

iqtplist(1:3, 19,1) = (/ 1, 6,11/)

iqtplist(1:3, 20,1) = (/ 2, 1, 6/)

iqtplist(1:3, 21,1) = (/ 2, 1, 7/)

iqtplist(1:3, 22,1) = (/ 2, 3, 4/)

iqtplist(1:3, 23,1) = (/ 2, 3,12/)

iqtplist(1:3, 24,1) = (/13,12, 3/)

iqtplist(1:3, 25,1) = (/ 3, 2, 8/)

iqtplist(1:3, 26,1) = (/ 3, 4, 5/)

iqtplist(1:3, 27,1) = (/ 3, 4, 9/)

iqtplist(1:3, 28,1) = (/ 4, 3,12/)

iqtplist(1:3, 29,1) = (/ 4, 5, 6/)

iqtplist(1:3, 30,1) = (/ 4, 5,10/)

iqtplist(1:3, 31,1) = (/ 5, 4, 9/)

iqtplist(1:3, 32,1) = (/ 5, 6,11/)

iqtplist(1:3, 33,1) = (/ 6, 1, 7/)

iqtplist(1:3, 34,1) = (/ 6, 5,10/)

iqtplist(1:3, 35,1) = (/14,13,15/)

iqtplist(1:3, 36,1) = (/14,13,16/)

iqtplist(1:3, 37,1) = (/14,13,12/)

iqtplist(1:3, 38,1) = (/15,13,16/)

iqtplist(1:3, 39,1) = (/15,13,12/)

iqtplist(1:3, 40,1) = (/16,13,12/)

iqtplist(1:4, 41,1) = (/ 1, 2, 3, 4/)

iqtplist(1:4, 42,1) = (/ 1, 2, 3,12/)

iqtplist(1:4, 43,1) = (/ 1, 6, 5, 4/)

iqtplist(1:4, 44,1) = (/ 1, 6, 5,10/)

iqtplist(1:4, 45,1) = (/ 2, 1, 6, 5/)

iqtplist(1:4, 46,1) = (/ 2, 1, 6,11/)

iqtplist(1:4, 47,1) = (/ 2, 3, 4, 5/)

iqtplist(1:4, 48,1) = (/ 2, 3, 4, 9/)

iqtplist(1:4, 49,1) = (/ 3, 2, 1, 6/)

iqtplist(1:4, 50,1) = (/ 3, 2, 1, 7/)

iqtplist(1:4, 51,1) = (/ 3, 4, 5, 6/)

iqtplist(1:4, 52,1) = (/ 3, 4, 5,10/)

iqtplist(1:4, 53,1) = (/ 3,12,13,14/)

iqtplist(1:4, 54,1) = (/ 3,12,13,15/)

iqtplist(1:4, 55,1) = (/ 3,12,13,16/)

iqtplist(1:4, 56,1) = (/ 4, 3, 2, 8/)

iqtplist(1:4, 57,1) = (/ 4, 5, 6,11/)

iqtplist(1:4, 58,1) = (/ 5, 4, 3,12/)

iqtplist(1:4, 59,1) = (/ 5, 6, 1, 7/)

iqtplist(1:4, 60,1) = (/ 6, 1, 2, 8/)

iqtplist(1:4, 61,1) = (/ 6, 5, 4, 9/)

iqtplist(1:4, 62,1) = (/11, 6, 1, 7/)

iqtplist(1:4, 63,1) = (/11, 6, 5,10/)

iqtplist(1:4, 64,1) = (/ 7, 1, 2, 8/)

iqtplist(1:4, 65,1) = (/ 8, 2, 3,12/)

iqtplist(1:4, 66,1) = (/ 9, 4, 3,12/)

iqtplist(1:4, 67,1) = (/ 9, 4, 5,10/)

iqtplist(1:4, 68,1) = (/ 1, 3, 8, 2/)

iqtplist(1:4, 69,1) = (/ 1, 5,11, 6/)

iqtplist(1:4, 70,1) = (/ 2, 4,12, 3/)

iqtplist(1:4, 71,1) = (/ 2, 6, 7, 1/)

iqtplist(1:4, 72,1) = (/ 3, 5, 9, 4/)

iqtplist(1:4, 73,1) = (/ 4, 6,10, 5/)

!--- END generated by 'python gen\_qtp.py qr\_tert\_pot\_1.txt'

!--- second set (for long R)

!--- BEGIN generated by 'python gen\_qtp.py qr\_tert\_pot\_2.txt'

iqtplist(1:2, 1,2) = (/ 1, 2/)

iqtplist(1:2, 2,2) = (/ 1, 6/)

iqtplist(1:2, 3,2) = (/ 1, 7/)

iqtplist(1:2, 4,2) = (/ 2, 3/)

iqtplist(1:2, 5,2) = (/ 2, 8/)

iqtplist(1:2, 6,2) = (/13,14/)

iqtplist(1:2, 7,2) = (/13,15/)

iqtplist(1:2, 8,2) = (/13,16/)

iqtplist(1:2, 9,2) = (/ 3, 4/)

iqtplist(1:2, 10,2) = (/ 3,12/)

iqtplist(1:2, 11,2) = (/ 4, 5/)

iqtplist(1:2, 12,2) = (/ 4, 9/)

iqtplist(1:2, 13,2) = (/ 5, 6/)

iqtplist(1:2, 14,2) = (/ 5,10/)

iqtplist(1:2, 15,2) = (/ 6,11/)

iqtplist(1:3, 16,2) = (/ 1, 2, 3/)

iqtplist(1:3, 17,2) = (/ 1, 2, 8/)

iqtplist(1:3, 18,2) = (/ 1, 6, 5/)

iqtplist(1:3, 19,2) = (/ 1, 6,11/)

iqtplist(1:3, 20,2) = (/ 2, 1, 6/)

iqtplist(1:3, 21,2) = (/ 2, 1, 7/)

iqtplist(1:3, 22,2) = (/ 2, 3, 4/)

iqtplist(1:3, 23,2) = (/ 2, 3,12/)

iqtplist(1:3, 24,2) = (/ 3, 2, 8/)

iqtplist(1:3, 25,2) = (/ 3, 4, 5/)

iqtplist(1:3, 26,2) = (/ 3, 4, 9/)

iqtplist(1:3, 27,2) = (/ 4, 3,12/)

iqtplist(1:3, 28,2) = (/ 4, 5, 6/)

iqtplist(1:3, 29,2) = (/ 4, 5,10/)

iqtplist(1:3, 30,2) = (/ 5, 4, 9/)

iqtplist(1:3, 31,2) = (/ 5, 6,11/)

iqtplist(1:3, 32,2) = (/ 6, 1, 7/)

iqtplist(1:3, 33,2) = (/ 6, 5,10/)

iqtplist(1:3, 34,2) = (/14,13,15/)

iqtplist(1:3, 35,2) = (/14,13,16/)

iqtplist(1:3, 36,2) = (/15,13,16/)

iqtplist(1:4, 37,2) = (/ 1, 2, 3, 4/)

iqtplist(1:4, 38,2) = (/ 1, 2, 3,12/)

iqtplist(1:4, 39,2) = (/ 1, 6, 5, 4/)

iqtplist(1:4, 40,2) = (/ 1, 6, 5,10/)

iqtplist(1:4, 41,2) = (/ 2, 1, 6, 5/)

iqtplist(1:4, 42,2) = (/ 2, 1, 6,11/)

iqtplist(1:4, 43,2) = (/ 2, 3, 4, 5/)

iqtplist(1:4, 44,2) = (/ 2, 3, 4, 9/)

iqtplist(1:4, 45,2) = (/ 3, 2, 1, 6/)

iqtplist(1:4, 46,2) = (/ 3, 2, 1, 7/)

iqtplist(1:4, 47,2) = (/ 3, 4, 5, 6/)

iqtplist(1:4, 48,2) = (/ 3, 4, 5,10/)

iqtplist(1:4, 49,2) = (/ 4, 3, 2, 8/)

iqtplist(1:4, 50,2) = (/ 4, 5, 6,11/)

iqtplist(1:4, 51,2) = (/ 5, 4, 3,12/)

iqtplist(1:4, 52,2) = (/ 5, 6, 1, 7/)

iqtplist(1:4, 53,2) = (/ 6, 1, 2, 8/)

iqtplist(1:4, 54,2) = (/ 6, 5, 4, 9/)

iqtplist(1:4, 55,2) = (/11, 6, 1, 7/)

iqtplist(1:4, 56,2) = (/11, 6, 5,10/)

iqtplist(1:4, 57,2) = (/ 7, 1, 2, 8/)

iqtplist(1:4, 58,2) = (/ 8, 2, 3,12/)

iqtplist(1:4, 59,2) = (/ 9, 4, 3,12/)

iqtplist(1:4, 60,2) = (/ 9, 4, 5,10/)

iqtplist(1:4, 61,2) = (/ 1, 3, 8, 2/)

iqtplist(1:4, 62,2) = (/ 1, 5,11, 6/)

iqtplist(1:4, 63,2) = (/ 2, 4,12, 3/)

iqtplist(1:4, 64,2) = (/ 2, 6, 7, 1/)

iqtplist(1:4, 65,2) = (/ 3, 5, 9, 4/)

iqtplist(1:4, 66,2) = (/ 4, 6,10, 5/)

iqtplist(1:4, 67,2) = (/14,15,16,13/)

!--- END generated by 'python gen\_qtp.py qr\_tert\_pot\_2.txt'

end subroutine assignqtp

!=================================================================

! \*def assignqr

! Assign attributes of redundant q (qr) for later use of qtc

! Input:

! NA

! Output:

! nqr: number of qr

! itypeqr: type of each qr

! iqrlist: atom index list array

!

! ~50 lines of params

!=================================================================

subroutine assignqr(nqr, itypeqr, iqrlist)

implicit none

integer :: nqr, itypeqr(ntermmax) &

, iqrlist(10,ntermmax)

nqr = 55

!--- type of qr

!--- 1=BL(BondLen), 2=BA(BondAng), 3=TO(Torsion), 4=OB(OOPBend)

itypeqr( 1:16) = 1

itypeqr(17:41) = 2

itypeqr(42:49) = 3

itypeqr(50:55) = 4

!--- BEGIN generated by 'python gen\_qr\_forUij-t.py rdef0.txt'

!--- bond length

iqrlist(1:2, 1) = (/ 2, 1/)

iqrlist(1:2, 2) = (/ 3, 2/)

iqrlist(1:2, 3) = (/ 4, 3/)

iqrlist(1:2, 4) = (/ 5, 4/)

iqrlist(1:2, 5) = (/ 6, 5/)

iqrlist(1:2, 6) = (/ 1, 6/)

iqrlist(1:2, 7) = (/ 7, 1/)

iqrlist(1:2, 8) = (/ 8, 2/)

iqrlist(1:2, 9) = (/ 9, 4/)

iqrlist(1:2,10) = (/10, 5/)

iqrlist(1:2,11) = (/11, 6/)

iqrlist(1:2,12) = (/12, 3/)

iqrlist(1:2,13) = (/13,12/)

iqrlist(1:2,14) = (/14,13/)

iqrlist(1:2,15) = (/15,13/)

iqrlist(1:2,16) = (/16,13/)

!--- bend

iqrlist(1:3,17) = (/ 3, 2, 1/)

iqrlist(1:3,18) = (/ 4, 3, 2/)

iqrlist(1:3,19) = (/ 5, 4, 3/)

iqrlist(1:3,20) = (/ 6, 5, 4/)

iqrlist(1:3,21) = (/ 1, 6, 5/)

iqrlist(1:3,22) = (/ 2, 1, 6/)

iqrlist(1:3,23) = (/ 7, 1, 6/)

iqrlist(1:3,24) = (/ 7, 1, 2/)

iqrlist(1:3,25) = (/ 8, 2, 1/)

iqrlist(1:3,26) = (/ 8, 2, 3/)

iqrlist(1:3,27) = (/ 9, 4, 3/)

iqrlist(1:3,28) = (/ 9, 4, 5/)

iqrlist(1:3,29) = (/10, 5, 4/)

iqrlist(1:3,30) = (/10, 5, 6/)

iqrlist(1:3,31) = (/11, 6, 5/)

iqrlist(1:3,32) = (/11, 6, 1/)

iqrlist(1:3,33) = (/12, 3, 2/)

iqrlist(1:3,34) = (/12, 3, 4/)

iqrlist(1:3,35) = (/13,12, 3/)

iqrlist(1:3,36) = (/14,13,15/)

iqrlist(1:3,37) = (/15,13,16/)

iqrlist(1:3,38) = (/14,13,16/)

iqrlist(1:3,39) = (/16,13,12/)

iqrlist(1:3,40) = (/14,13,12/)

iqrlist(1:3,41) = (/15,13,12/)

!--- torsion

iqrlist(1:4,42) = (/ 4, 3, 2, 1/)

iqrlist(1:4,43) = (/ 5, 4, 3, 2/)

iqrlist(1:4,44) = (/ 6, 5, 4, 3/)

iqrlist(1:4,45) = (/ 1, 6, 5, 4/)

iqrlist(1:4,46) = (/ 2, 1, 6, 5/)

iqrlist(1:4,47) = (/ 3, 2, 1, 6/)

iqrlist(1:4,48) = (/13,12, 3, 4/)

iqrlist(1:4,49) = (/16,13,12, 3/)

!--- oop bend

iqrlist(1:4,50) = (/ 7, 2, 6, 1/)

iqrlist(1:4,51) = (/ 8, 3, 1, 2/)

iqrlist(1:4,52) = (/ 9, 5, 3, 4/)

iqrlist(1:4,53) = (/10, 6, 4, 5/)

iqrlist(1:4,54) = (/11, 1, 5, 6/)

iqrlist(1:4,55) = (/12, 4, 2, 3/)

!--- END generated by 'python gen\_qr\_forUij-t.py rdef0.txt'

end subroutine assignqr

!=================================================================

! \*def assignqnr

! Assign attributes of nonredund. q (qnr) for later use of qtc

! Input:

! NA

! Output:

! nqnr: number of qr

! lcindex: index of qr involved in lin.comb. to get each qnr

! lccoef: coefficients of lin.comb. to get each qnr

! nlc: number of terms in lin.comb. to get each qnr

!

! ~200 lines of params

!=================================================================

subroutine assignqnr(nqnr, lcindex, lccoef, nlc)

implicit none

integer :: nqnr, lcindex(nlcmax,ntermmax), nlc(ntermmax)

double precision :: lccoef(nlcmax,ntermmax)

nqnr = 42

!--- BEGIN generated by python gen\_qnr\_forUij-t.py intnrdef.txt

nlc ( 1) = 1

lcindex(1:nlc( 1), 1) = &

(/ 1/)

lccoef (1:nlc( 1), 1) = &

(/ 1.00000/)

nlc ( 2) = 1

lcindex(1:nlc( 2), 2) = &

(/ 2/)

lccoef (1:nlc( 2), 2) = &

(/ 1.00000/)

nlc ( 3) = 1

lcindex(1:nlc( 3), 3) = &

(/ 3/)

lccoef (1:nlc( 3), 3) = &

(/ 1.00000/)

nlc ( 4) = 1

lcindex(1:nlc( 4), 4) = &

(/ 4/)

lccoef (1:nlc( 4), 4) = &

(/ 1.00000/)

nlc ( 5) = 1

lcindex(1:nlc( 5), 5) = &

(/ 5/)

lccoef (1:nlc( 5), 5) = &

(/ 1.00000/)

nlc ( 6) = 1

lcindex(1:nlc( 6), 6) = &

(/ 6/)

lccoef (1:nlc( 6), 6) = &

(/ 1.00000/)

nlc ( 7) = 1

lcindex(1:nlc( 7), 7) = &

(/ 7/)

lccoef (1:nlc( 7), 7) = &

(/ 1.00000/)

nlc ( 8) = 1

lcindex(1:nlc( 8), 8) = &

(/ 8/)

lccoef (1:nlc( 8), 8) = &

(/ 1.00000/)

nlc ( 9) = 1

lcindex(1:nlc( 9), 9) = &

(/ 9/)

lccoef (1:nlc( 9), 9) = &

(/ 1.00000/)

nlc ( 10) = 1

lcindex(1:nlc(10),10) = &

(/ 10/)

lccoef (1:nlc(10),10) = &

(/ 1.00000/)

nlc ( 11) = 1

lcindex(1:nlc(11),11) = &

(/ 11/)

lccoef (1:nlc(11),11) = &

(/ 1.00000/)

nlc ( 12) = 1

lcindex(1:nlc(12),12) = &

(/ 12/)

lccoef (1:nlc(12),12) = &

(/ 1.00000/)

nlc ( 13) = 1

lcindex(1:nlc(13),13) = &

(/ 13/)

lccoef (1:nlc(13),13) = &

(/ 1.00000/)

nlc ( 14) = 3

lcindex(1:nlc(14),14) = &

(/ 14, 15, 16/)

lccoef (1:nlc(14),14) = &

(/ 0.57735, 0.57735, 0.57735/)

nlc ( 15) = 3

lcindex(1:nlc(15),15) = &

(/ 14, 15, 16/)

lccoef (1:nlc(15),15) = &

(/ -0.40825, -0.40825, 0.81650/)

nlc ( 16) = 3

lcindex(1:nlc(16),16) = &

(/ 14, 15, 16/)

lccoef (1:nlc(16),16) = &

(/ 0.70711, -0.70711, 0.00000/)

nlc ( 17) = 6

lcindex(1:nlc(17),17) = &

(/ 17, 18, 19, 20, 21, 22/)

lccoef (1:nlc(17),17) = &

(/ 0.40825, -0.40825, 0.40825, -0.40825, 0.40825, -0.40825/)

nlc ( 18) = 6

lcindex(1:nlc(18),18) = &

(/ 17, 18, 19, 20, 21, 22/)

lccoef (1:nlc(18),18) = &

(/ 0.57735, -0.28868, -0.28868, 0.57735, -0.28868, -0.28868/)

nlc ( 19) = 6

lcindex(1:nlc(19),19) = &

(/ 17, 18, 19, 20, 21, 22/)

lccoef (1:nlc(19),19) = &

(/ 0.00000, 0.50000, -0.50000, 0.00000, 0.50000, -0.50000/)

nlc ( 20) = 2

lcindex(1:nlc(20),20) = &

(/ 23, 24/)

lccoef (1:nlc(20),20) = &

(/ 0.70711, -0.70711/)

nlc ( 21) = 2

lcindex(1:nlc(21),21) = &

(/ 25, 26/)

lccoef (1:nlc(21),21) = &

(/ 0.70711, -0.70711/)

nlc ( 22) = 2

lcindex(1:nlc(22),22) = &

(/ 27, 28/)

lccoef (1:nlc(22),22) = &

(/ 0.70711, -0.70711/)

nlc ( 23) = 2

lcindex(1:nlc(23),23) = &

(/ 29, 30/)

lccoef (1:nlc(23),23) = &

(/ 0.70711, -0.70711/)

nlc ( 24) = 2

lcindex(1:nlc(24),24) = &

(/ 31, 32/)

lccoef (1:nlc(24),24) = &

(/ 0.70711, -0.70711/)

nlc ( 25) = 2

lcindex(1:nlc(25),25) = &

(/ 33, 34/)

lccoef (1:nlc(25),25) = &

(/ 0.70711, -0.70711/)

nlc ( 26) = 1

lcindex(1:nlc(26),26) = &

(/ 35/)

lccoef (1:nlc(26),26) = &

(/ 1.00000/)

nlc ( 27) = 6

lcindex(1:nlc(27),27) = &

(/ 36, 37, 38, 39, 40, 41/)

lccoef (1:nlc(27),27) = &

(/ 0.40825, 0.40825, 0.40825, -0.40825, -0.40825, -0.40825/)

nlc ( 28) = 3

lcindex(1:nlc(28),28) = &

(/ 36, 37, 38/)

lccoef (1:nlc(28),28) = &

(/ 0.81650, -0.40825, -0.40825/)

nlc ( 29) = 2

lcindex(1:nlc(29),29) = &

(/ 37, 38/)

lccoef (1:nlc(29),29) = &

(/ 0.70711, -0.70711/)

nlc ( 30) = 3

lcindex(1:nlc(30),30) = &

(/ 39, 40, 41/)

lccoef (1:nlc(30),30) = &

(/ 0.81650, -0.40825, -0.40825/)

nlc ( 31) = 2

lcindex(1:nlc(31),31) = &

(/ 40, 41/)

lccoef (1:nlc(31),31) = &

(/ 0.70711, -0.70711/)

nlc ( 32) = 6

lcindex(1:nlc(32),32) = &

(/ 42, 43, 44, 45, 46, 47/)

lccoef (1:nlc(32),32) = &

(/ 0.40825, -0.40825, 0.40825, -0.40825, 0.40825, -0.40825/)

nlc ( 33) = 6

lcindex(1:nlc(33),33) = &

(/ 42, 43, 44, 45, 46, 47/)

lccoef (1:nlc(33),33) = &

(/ 0.50000, 0.00000, -0.50000, 0.50000, 0.00000, -0.50000/)

nlc ( 34) = 6

lcindex(1:nlc(34),34) = &

(/ 42, 43, 44, 45, 46, 47/)

lccoef (1:nlc(34),34) = &

(/ -0.28868, 0.57735, -0.28868, -0.28868, 0.57735, -0.28868/)

nlc ( 35) = 1

lcindex(1:nlc(35),35) = &

(/ 48/)

lccoef (1:nlc(35),35) = &

(/ 1.00000/)

nlc ( 36) = 1

lcindex(1:nlc(36),36) = &

(/ 49/)

lccoef (1:nlc(36),36) = &

(/ 1.00000/)

nlc ( 37) = 1

lcindex(1:nlc(37),37) = &

(/ 50/)

lccoef (1:nlc(37),37) = &

(/ 1.00000/)

nlc ( 38) = 1

lcindex(1:nlc(38),38) = &

(/ 51/)

lccoef (1:nlc(38),38) = &

(/ 1.00000/)

nlc ( 39) = 1

lcindex(1:nlc(39),39) = &

(/ 52/)

lccoef (1:nlc(39),39) = &

(/ 1.00000/)

nlc ( 40) = 1

lcindex(1:nlc(40),40) = &

(/ 53/)

lccoef (1:nlc(40),40) = &

(/ 1.00000/)

nlc ( 41) = 1

lcindex(1:nlc(41),41) = &

(/ 54/)

lccoef (1:nlc(41),41) = &

(/ 1.00000/)

nlc ( 42) = 1

lcindex(1:nlc(42),42) = &

(/ 55/)

lccoef (1:nlc(42),42) = &

(/ 1.00000/)

!--- END generated by python gen\_qnr\_forUij-t.py intnrdef.txt

end subroutine assignqnr

!=================================================================

! \*def assignqtc

! Assign attributes of internal coord. for tert. coupl. (qtc)

! Input:

! NA

! Output:

! nqtc: number of qtc

! qtcsym: symmetry (a' or a") of qtc

! qnrindex: qnr index to which each qtc corresponds

! qtctyp: type of qtc; used for choosing form of Uij[3] term

!=================================================================

subroutine assignqtc(nqtc, qtcsym, qnrindex, qtctyp)

implicit none

integer :: nqtc, qtcsym(ntermmax), qnrindex(ntermmax) &

, qtctyp(ntermmax)

nqtc = 11

!--- correspondence of qtc to qnr

qnrindex(1:nqtc) = (/42,31,32,34, 18,19,25,30, 33, 26, 38/)

!--- type of qtc: 1=normal; 2=angle involving a breaking bond

qtctyp (1:nqtc) = (/ 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1/)

!--- symm. of qtc at Cs geom.

!--- 1=a', 2=a"

qtcsym(1:nqtc) = (/2,2,2,2, 1,1,1,1, 2, 1, 2/)

end subroutine assignqtc

!=================================================================

! \*def assignparamuii\_t

! Assign tert. FF parameters to param. arrays

! Input:

! istate: electronic state of interest

! Output:

! k: force constants

! q0: rest values

! n: multiplicities (for torsion) or dummy (for others)

!

! ~2200 lines of params

!=================================================================

subroutine assignparamuii\_t(k, q0, n, istate)

implicit none

double precision :: k(ntermmax,napmax), q0(ntermmax,napmax)

integer :: n(ntermmax,napmax), istate

!--- unit of parameters have been coverted to

!--- hartree, Angstrom, rad

select case(istate)

case(1)

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 1, anchor point 1)

k ( 1, 1) = 2.619244e+00

q0( 1, 1) = 1.390000e+00

n ( 1, 1) = 0

k ( 2, 1) = 2.492554e+00

q0( 2, 1) = 1.397000e+00

n ( 2, 1) = 0

k ( 3, 1) = 1.545696e+00

q0( 3, 1) = 1.090000e+00

n ( 3, 1) = 0

k ( 4, 1) = 2.337722e+00

q0( 4, 1) = 1.403000e+00

n ( 4, 1) = 0

k ( 5, 1) = 1.534213e+00

q0( 5, 1) = 1.091000e+00

n ( 5, 1) = 0

k ( 6, 1) = 1.426792e+00

q0( 6, 1) = 1.097000e+00

n ( 6, 1) = 0

k ( 7, 1) = 1.426793e+00

q0( 7, 1) = 1.097000e+00

n ( 7, 1) = 0

k ( 8, 1) = 1.451096e+00

q0( 8, 1) = 1.096000e+00

n ( 8, 1) = 0

k ( 9, 1) = 2.377239e+00

q0( 9, 1) = 1.398000e+00

n ( 9, 1) = 0

k ( 10, 1) = 2.216034e+00

q0( 10, 1) = 1.765000e+00

n ( 10, 1) = 0

k ( 11, 1) = 2.519101e+00

q0( 11, 1) = 1.397000e+00

n ( 11, 1) = 0

k ( 12, 1) = 1.551794e+00

q0( 12, 1) = 1.088000e+00

n ( 12, 1) = 0

k ( 13, 1) = 2.560072e+00

q0( 13, 1) = 1.392000e+00

n ( 13, 1) = 0

k ( 14, 1) = 1.544277e+00

q0( 14, 1) = 1.090000e+00

n ( 14, 1) = 0

k ( 15, 1) = 1.553667e+00

q0( 15, 1) = 1.089000e+00

n ( 15, 1) = 0

k ( 16, 1) = 2.382446e-01

q0( 16, 1) = 2.100172e+00

n ( 16, 1) = 0

k ( 17, 1) = 1.833758e-01

q0( 17, 1) = 2.096943e+00

n ( 17, 1) = 0

k ( 18, 1) = 2.370064e-01

q0( 18, 1) = 2.080293e+00

n ( 18, 1) = 0

k ( 19, 1) = 1.710962e-01

q0( 19, 1) = 2.101481e+00

n ( 19, 1) = 0

k ( 20, 1) = 2.150197e-01

q0( 20, 1) = 2.103034e+00

n ( 20, 1) = 0

k ( 21, 1) = 1.641261e-01

q0( 21, 1) = 2.083626e+00

n ( 21, 1) = 0

k ( 22, 1) = 1.087538e-01

q0( 22, 1) = 2.081776e+00

n ( 22, 1) = 0

k ( 23, 1) = 2.963787e-01

q0( 23, 1) = 2.034460e+00

n ( 23, 1) = 0

k ( 24, 1) = 3.029554e-01

q0( 24, 1) = 1.795420e+00

n ( 24, 1) = 0

k ( 25, 1) = 1.503844e-01

q0( 25, 1) = 2.086489e+00

n ( 25, 1) = 0

k ( 26, 1) = 2.380949e-01

q0( 26, 1) = 2.093435e+00

n ( 26, 1) = 0

k ( 27, 1) = 1.498012e-01

q0( 27, 1) = 2.110679e+00

n ( 27, 1) = 0

k ( 28, 1) = 3.251847e-01

q0( 28, 1) = 2.167629e+00

n ( 28, 1) = 0

k ( 29, 1) = 2.146442e-01

q0( 29, 1) = 2.108620e+00

n ( 29, 1) = 0

k ( 30, 1) = 1.620271e-01

q0( 30, 1) = 2.079315e+00

n ( 30, 1) = 0

k ( 31, 1) = 1.791926e-01

q0( 31, 1) = 2.079612e+00

n ( 31, 1) = 0

k ( 32, 1) = 1.709755e-01

q0( 32, 1) = 2.101900e+00

n ( 32, 1) = 0

k ( 33, 1) = 1.652768e-01

q0( 33, 1) = 2.096943e+00

n ( 33, 1) = 0

k ( 34, 1) = 1.678104e-01

q0( 34, 1) = 2.095634e+00

n ( 34, 1) = 0

k ( 35, 1) = 8.883264e-02

q0( 35, 1) = 1.928344e+00

n ( 35, 1) = 0

k ( 36, 1) = 9.599204e-02

q0( 36, 1) = 1.902845e+00

n ( 36, 1) = 0

k ( 37, 1) = 1.180519e-01

q0( 37, 1) = 1.942743e+00

n ( 37, 1) = 0

k ( 38, 1) = 9.599242e-02

q0( 38, 1) = 1.902845e+00

n ( 38, 1) = 0

k ( 39, 1) = 1.180515e-01

q0( 39, 1) = 1.942743e+00

n ( 39, 1) = 0

k ( 40, 1) = 1.218393e-01

q0( 40, 1) = 1.842370e+00

n ( 40, 1) = 0

k ( 41, 1) = 2.088609e-18

q0( 41, 1) = 0.000000e+00

n ( 41, 1) = 2

k ( 42, 1) = 5.749321e-03

q0( 42, 1) = 0.000000e+00

n ( 42, 1) = 2

k ( 43, 1) = 1.958462e-02

q0( 43, 1) = 0.000000e+00

n ( 43, 1) = 2

k ( 44, 1) = 1.239744e-02

q0( 44, 1) = 0.000000e+00

n ( 44, 1) = 2

k ( 45, 1) = 5.303858e-03

q0( 45, 1) = 0.000000e+00

n ( 45, 1) = 2

k ( 46, 1) = 1.150228e-02

q0( 46, 1) = 0.000000e+00

n ( 46, 1) = 2

k ( 47, 1) = 1.573999e-02

q0( 47, 1) = 0.000000e+00

n ( 47, 1) = 2

k ( 48, 1) = 8.958560e-03

q0( 48, 1) = 0.000000e+00

n ( 48, 1) = 2

k ( 49, 1) = 1.798006e-02

q0( 49, 1) = 0.000000e+00

n ( 49, 1) = 2

k ( 50, 1) = 1.261305e-02

q0( 50, 1) = 0.000000e+00

n ( 50, 1) = 2

k ( 51, 1) = 4.521086e-03

q0( 51, 1) = 0.000000e+00

n ( 51, 1) = 2

k ( 52, 1) = 1.239651e-02

q0( 52, 1) = 0.000000e+00

n ( 52, 1) = 2

k ( 53, 1) = 5.795252e-03

q0( 53, 1) = 1.047198e+00

n ( 53, 1) = 3

k ( 54, 1) = 5.795341e-03

q0( 54, 1) = 1.047198e+00

n ( 54, 1) = 3

k ( 55, 1) = 4.122935e-03

q0( 55, 1) = 1.047198e+00

n ( 55, 1) = 3

k ( 56, 1) = 9.876327e-03

q0( 56, 1) = 0.000000e+00

n ( 56, 1) = 2

k ( 57, 1) = 1.224966e-02

q0( 57, 1) = 0.000000e+00

n ( 57, 1) = 2

k ( 58, 1) = 1.043591e-02

q0( 58, 1) = 0.000000e+00

n ( 58, 1) = 2

k ( 59, 1) = 1.141268e-02

q0( 59, 1) = 0.000000e+00

n ( 59, 1) = 2

k ( 60, 1) = 1.257323e-02

q0( 60, 1) = 0.000000e+00

n ( 60, 1) = 2

k ( 61, 1) = 1.250853e-02

q0( 61, 1) = 0.000000e+00

n ( 61, 1) = 2

k ( 62, 1) = 4.548687e-03

q0( 62, 1) = 0.000000e+00

n ( 62, 1) = 2

k ( 63, 1) = 5.334954e-03

q0( 63, 1) = 0.000000e+00

n ( 63, 1) = 2

k ( 64, 1) = 5.444707e-03

q0( 64, 1) = 0.000000e+00

n ( 64, 1) = 2

k ( 65, 1) = 2.744636e-03

q0( 65, 1) = 0.000000e+00

n ( 65, 1) = 2

k ( 66, 1) = 2.243665e-03

q0( 66, 1) = 0.000000e+00

n ( 66, 1) = 2

k ( 67, 1) = 5.217540e-03

q0( 67, 1) = 0.000000e+00

n ( 67, 1) = 2

k ( 68, 1) = 1.320081e-01

q0( 68, 1) = 0.000000e+00

n ( 68, 1) = 0

k ( 69, 1) = 9.511945e-02

q0( 69, 1) = 0.000000e+00

n ( 69, 1) = 0

k ( 70, 1) = 1.780134e-01

q0( 70, 1) = 0.000000e+00

n ( 70, 1) = 0

k ( 71, 1) = 1.315198e-01

q0( 71, 1) = 0.000000e+00

n ( 71, 1) = 0

k ( 72, 1) = 1.285078e-01

q0( 72, 1) = 0.000000e+00

n ( 72, 1) = 0

k ( 73, 1) = 1.150246e-01

q0( 73, 1) = 0.000000e+00

n ( 73, 1) = 0

!--- END generated by 'gen\_FFparam.py'

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 1, anchor point 2)

k ( 1, 2) = 2.586008e+00

q0( 1, 2) = 1.392000e+00

n ( 1, 2) = 0

k ( 2, 2) = 2.494312e+00

q0( 2, 2) = 1.396000e+00

n ( 2, 2) = 0

k ( 3, 2) = 1.539699e+00

q0( 3, 2) = 1.090000e+00

n ( 3, 2) = 0

k ( 4, 2) = 2.273210e+00

q0( 4, 2) = 1.407000e+00

n ( 4, 2) = 0

k ( 5, 2) = 1.527902e+00

q0( 5, 2) = 1.091000e+00

n ( 5, 2) = 0

k ( 6, 2) = 1.509773e+00

q0( 6, 2) = 1.091000e+00

n ( 6, 2) = 0

k ( 7, 2) = 1.509773e+00

q0( 7, 2) = 1.091000e+00

n ( 7, 2) = 0

k ( 8, 2) = 1.512585e+00

q0( 8, 2) = 1.091000e+00

n ( 8, 2) = 0

k ( 9, 2) = 2.365982e+00

q0( 9, 2) = 1.398000e+00

n ( 9, 2) = 0

k ( 10, 2) = 2.165702e+00

q0( 10, 2) = 1.770000e+00

n ( 10, 2) = 0

k ( 11, 2) = 2.518431e+00

q0( 11, 2) = 1.396000e+00

n ( 11, 2) = 0

k ( 12, 2) = 1.558609e+00

q0( 12, 2) = 1.087000e+00

n ( 12, 2) = 0

k ( 13, 2) = 2.541240e+00

q0( 13, 2) = 1.393000e+00

n ( 13, 2) = 0

k ( 14, 2) = 1.538749e+00

q0( 14, 2) = 1.090000e+00

n ( 14, 2) = 0

k ( 15, 2) = 1.549891e+00

q0( 15, 2) = 1.089000e+00

n ( 15, 2) = 0

k ( 16, 2) = 2.432243e-01

q0( 16, 2) = 2.099893e+00

n ( 16, 2) = 0

k ( 17, 2) = 1.842465e-01

q0( 17, 2) = 2.090782e+00

n ( 17, 2) = 0

k ( 18, 2) = 2.357369e-01

q0( 18, 2) = 2.078705e+00

n ( 18, 2) = 0

k ( 19, 2) = 1.710109e-01

q0( 19, 2) = 2.103174e+00

n ( 19, 2) = 0

k ( 20, 2) = 2.129420e-01

q0( 20, 2) = 2.103663e+00

n ( 20, 2) = 0

k ( 21, 2) = 1.636367e-01

q0( 21, 2) = 2.085442e+00

n ( 21, 2) = 0

k ( 22, 2) = 9.800080e-02

q0( 22, 2) = 2.074498e+00

n ( 22, 2) = 0

k ( 23, 2) = 3.048183e-01

q0( 23, 2) = 2.037480e+00

n ( 23, 2) = 0

k ( 24, 2) = 2.269325e-01

q0( 24, 2) = 1.787793e+00

n ( 24, 2) = 0

k ( 25, 2) = 1.474177e-01

q0( 25, 2) = 2.093802e+00

n ( 25, 2) = 0

k ( 26, 2) = 2.414756e-01

q0( 26, 2) = 2.099439e+00

n ( 26, 2) = 0

k ( 27, 2) = 1.490913e-01

q0( 27, 2) = 2.100033e+00

n ( 27, 2) = 0

k ( 28, 2) = 3.040409e-01

q0( 28, 2) = 2.173476e+00

n ( 28, 2) = 0

k ( 29, 2) = 2.153762e-01

q0( 29, 2) = 2.110854e+00

n ( 29, 2) = 0

k ( 30, 2) = 1.628635e-01

q0( 30, 2) = 2.076610e+00

n ( 30, 2) = 0

k ( 31, 2) = 1.774977e-01

q0( 31, 2) = 2.082038e+00

n ( 31, 2) = 0

k ( 32, 2) = 1.709660e-01

q0( 32, 2) = 2.101795e+00

n ( 32, 2) = 0

k ( 33, 2) = 1.649907e-01

q0( 33, 2) = 2.095163e+00

n ( 33, 2) = 0

k ( 34, 2) = 1.664442e-01

q0( 34, 2) = 2.094168e+00

n ( 34, 2) = 0

k ( 35, 2) = 1.060995e-01

q0( 35, 2) = 2.011544e+00

n ( 35, 2) = 0

k ( 36, 2) = 1.101787e-01

q0( 36, 2) = 1.985801e+00

n ( 36, 2) = 0

k ( 37, 2) = 9.485893e-02

q0( 37, 2) = 1.841881e+00

n ( 37, 2) = 0

k ( 38, 2) = 1.101791e-01

q0( 38, 2) = 1.985801e+00

n ( 38, 2) = 0

k ( 39, 2) = 9.485816e-02

q0( 39, 2) = 1.841881e+00

n ( 39, 2) = 0

k ( 40, 2) = 1.099867e-01

q0( 40, 2) = 1.772138e+00

n ( 40, 2) = 0

k ( 41, 2) = 4.272806e-18

q0( 41, 2) = 0.000000e+00

n ( 41, 2) = 2

k ( 42, 2) = 7.822627e-03

q0( 42, 2) = 0.000000e+00

n ( 42, 2) = 2

k ( 43, 2) = 1.976277e-02

q0( 43, 2) = 0.000000e+00

n ( 43, 2) = 2

k ( 44, 2) = 1.275336e-02

q0( 44, 2) = 0.000000e+00

n ( 44, 2) = 2

k ( 45, 2) = 5.714018e-03

q0( 45, 2) = 0.000000e+00

n ( 45, 2) = 2

k ( 46, 2) = 1.164570e-02

q0( 46, 2) = 0.000000e+00

n ( 46, 2) = 2

k ( 47, 2) = 1.552873e-02

q0( 47, 2) = 0.000000e+00

n ( 47, 2) = 2

k ( 48, 2) = 8.696028e-03

q0( 48, 2) = 0.000000e+00

n ( 48, 2) = 2

k ( 49, 2) = 1.820143e-02

q0( 49, 2) = 0.000000e+00

n ( 49, 2) = 2

k ( 50, 2) = 1.233255e-02

q0( 50, 2) = 0.000000e+00

n ( 50, 2) = 2

k ( 51, 2) = 4.511505e-03

q0( 51, 2) = 0.000000e+00

n ( 51, 2) = 2

k ( 52, 2) = 1.175369e-02

q0( 52, 2) = 0.000000e+00

n ( 52, 2) = 2

k ( 53, 2) = 1.725200e-03

q0( 53, 2) = 1.047198e+00

n ( 53, 2) = 3

k ( 54, 2) = 1.725198e-03

q0( 54, 2) = 1.047198e+00

n ( 54, 2) = 3

k ( 55, 2) = -4.529911e-19

q0( 55, 2) = 1.047198e+00

n ( 55, 2) = 3

k ( 56, 2) = 1.004664e-02

q0( 56, 2) = 0.000000e+00

n ( 56, 2) = 2

k ( 57, 2) = 1.199563e-02

q0( 57, 2) = 0.000000e+00

n ( 57, 2) = 2

k ( 58, 2) = 1.262188e-02

q0( 58, 2) = 0.000000e+00

n ( 58, 2) = 2

k ( 59, 2) = 1.171924e-02

q0( 59, 2) = 0.000000e+00

n ( 59, 2) = 2

k ( 60, 2) = 1.254759e-02

q0( 60, 2) = 0.000000e+00

n ( 60, 2) = 2

k ( 61, 2) = 1.274894e-02

q0( 61, 2) = 0.000000e+00

n ( 61, 2) = 2

k ( 62, 2) = 5.001135e-03

q0( 62, 2) = 0.000000e+00

n ( 62, 2) = 2

k ( 63, 2) = 5.629632e-03

q0( 63, 2) = 0.000000e+00

n ( 63, 2) = 2

k ( 64, 2) = 5.644706e-03

q0( 64, 2) = 0.000000e+00

n ( 64, 2) = 2

k ( 65, 2) = 3.448609e-03

q0( 65, 2) = 0.000000e+00

n ( 65, 2) = 2

k ( 66, 2) = 2.542402e-03

q0( 66, 2) = 0.000000e+00

n ( 66, 2) = 2

k ( 67, 2) = 5.294531e-03

q0( 67, 2) = 0.000000e+00

n ( 67, 2) = 2

k ( 68, 2) = 1.223143e-01

q0( 68, 2) = 0.000000e+00

n ( 68, 2) = 0

k ( 69, 2) = 8.630360e-02

q0( 69, 2) = 0.000000e+00

n ( 69, 2) = 0

k ( 70, 2) = 1.966613e-01

q0( 70, 2) = 0.000000e+00

n ( 70, 2) = 0

k ( 71, 2) = 1.220161e-01

q0( 71, 2) = 0.000000e+00

n ( 71, 2) = 0

k ( 72, 2) = 1.226224e-01

q0( 72, 2) = 0.000000e+00

n ( 72, 2) = 0

k ( 73, 2) = 1.110292e-01

q0( 73, 2) = 0.000000e+00

n ( 73, 2) = 0

!--- END generated by 'gen\_FFparam.py'

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 1, anchor point 3)

k ( 1, 3) = 2.560516e+00

q0( 1, 3) = 1.393000e+00

n ( 1, 3) = 0

k ( 2, 3) = 2.501888e+00

q0( 2, 3) = 1.395000e+00

n ( 2, 3) = 0

k ( 3, 3) = 1.537737e+00

q0( 3, 3) = 1.091000e+00

n ( 3, 3) = 0

k ( 4, 3) = 2.239875e+00

q0( 4, 3) = 1.408000e+00

n ( 4, 3) = 0

k ( 5, 3) = 1.533643e+00

q0( 5, 3) = 1.090000e+00

n ( 5, 3) = 0

k ( 6, 3) = 1.560748e+00

q0( 6, 3) = 1.088000e+00

n ( 6, 3) = 0

k ( 7, 3) = 1.560748e+00

q0( 7, 3) = 1.088000e+00

n ( 7, 3) = 0

k ( 8, 3) = 1.565978e+00

q0( 8, 3) = 1.088000e+00

n ( 8, 3) = 0

k ( 9, 3) = 2.314755e+00

q0( 9, 3) = 1.401000e+00

n ( 9, 3) = 0

k ( 10, 3) = 2.169444e+00

q0( 10, 3) = 1.758000e+00

n ( 10, 3) = 0

k ( 11, 3) = 2.552078e+00

q0( 11, 3) = 1.394000e+00

n ( 11, 3) = 0

k ( 12, 3) = 1.543869e+00

q0( 12, 3) = 1.088000e+00

n ( 12, 3) = 0

k ( 13, 3) = 2.521473e+00

q0( 13, 3) = 1.394000e+00

n ( 13, 3) = 0

k ( 14, 3) = 1.536166e+00

q0( 14, 3) = 1.090000e+00

n ( 14, 3) = 0

k ( 15, 3) = 1.548182e+00

q0( 15, 3) = 1.089000e+00

n ( 15, 3) = 0

k ( 16, 3) = 2.578413e-01

q0( 16, 3) = 2.100626e+00

n ( 16, 3) = 0

k ( 17, 3) = 1.822046e-01

q0( 17, 3) = 2.090922e+00

n ( 17, 3) = 0

k ( 18, 3) = 2.340991e-01

q0( 18, 3) = 2.076697e+00

n ( 18, 3) = 0

k ( 19, 3) = 1.735537e-01

q0( 19, 3) = 2.103645e+00

n ( 19, 3) = 0

k ( 20, 3) = 2.035320e-01

q0( 20, 3) = 2.107886e+00

n ( 20, 3) = 0

k ( 21, 3) = 1.605101e-01

q0( 21, 3) = 2.080555e+00

n ( 21, 3) = 0

k ( 22, 3) = 8.282731e-02

q0( 22, 3) = 2.068128e+00

n ( 22, 3) = 0

k ( 23, 3) = 2.794139e-01

q0( 23, 3) = 2.058825e+00

n ( 23, 3) = 0

k ( 24, 3) = 0.110980e+00 !<- modified to that determined by ab initio

q0( 24, 3) = 1.755714e+00

n ( 24, 3) = 0

k ( 25, 3) = 1.429100e-01

q0( 25, 3) = 2.092388e+00

n ( 25, 3) = 0

k ( 26, 3) = 2.464998e-01

q0( 26, 3) = 2.105321e+00

n ( 26, 3) = 0

k ( 27, 3) = 1.384849e-01

q0( 27, 3) = 2.078390e+00

n ( 27, 3) = 0

k ( 28, 3) = 2.913622e-01

q0( 28, 3) = 2.158257e+00

n ( 28, 3) = 0

k ( 29, 3) = 2.125451e-01

q0( 29, 3) = 2.108532e+00

n ( 29, 3) = 0

k ( 30, 3) = 1.634257e-01

q0( 30, 3) = 2.079001e+00

n ( 30, 3) = 0

k ( 31, 3) = 1.793233e-01

q0( 31, 3) = 2.098846e+00

n ( 31, 3) = 0

k ( 32, 3) = 1.698797e-01

q0( 32, 3) = 2.103331e+00

n ( 32, 3) = 0

k ( 33, 3) = 1.664545e-01

q0( 33, 3) = 2.095669e+00

n ( 33, 3) = 0

k ( 34, 3) = 1.668216e-01

q0( 34, 3) = 2.094866e+00

n ( 34, 3) = 0

k ( 35, 3) = 1.128791e-01

q0( 35, 3) = 2.093034e+00

n ( 35, 3) = 0

k ( 36, 3) = 1.095914e-01

q0( 36, 3) = 2.080206e+00

n ( 36, 3) = 0

k ( 37, 3) = 7.410439e-02

q0( 37, 3) = 1.675429e+00

n ( 37, 3) = 0

k ( 38, 3) = 1.095914e-01

q0( 38, 3) = 2.080206e+00

n ( 38, 3) = 0

k ( 39, 3) = 7.410439e-02

q0( 39, 3) = 1.675429e+00

n ( 39, 3) = 0

k ( 40, 3) = 7.933426e-02

q0( 40, 3) = 1.626996e+00

n ( 40, 3) = 0

k ( 41, 3) = 7.685329e-03

q0( 41, 3) = 0.000000e+00

n ( 41, 3) = 2

k ( 42, 3) = 4.701895e-03

q0( 42, 3) = 0.000000e+00

n ( 42, 3) = 2

k ( 43, 3) = 1.443515e-02

q0( 43, 3) = 0.000000e+00

n ( 43, 3) = 2

k ( 44, 3) = 1.335221e-02

q0( 44, 3) = 0.000000e+00

n ( 44, 3) = 2

k ( 45, 3) = 1.217395e-02

q0( 45, 3) = 0.000000e+00

n ( 45, 3) = 2

k ( 46, 3) = 1.134732e-02

q0( 46, 3) = 0.000000e+00

n ( 46, 3) = 2

k ( 47, 3) = 8.573103e-03

q0( 47, 3) = 0.000000e+00

n ( 47, 3) = 2

k ( 48, 3) = 7.218820e-03

q0( 48, 3) = 0.000000e+00

n ( 48, 3) = 2

k ( 49, 3) = 1.144454e-02

q0( 49, 3) = 0.000000e+00

n ( 49, 3) = 2

k ( 50, 3) = 1.221598e-02

q0( 50, 3) = 0.000000e+00

n ( 50, 3) = 2

k ( 51, 3) = 1.015292e-02

q0( 51, 3) = 0.000000e+00

n ( 51, 3) = 2

k ( 52, 3) = 1.082467e-02

q0( 52, 3) = 0.000000e+00

n ( 52, 3) = 2

k ( 53, 3) = 2.037170e-19

q0( 53, 3) = 1.047198e+00

n ( 53, 3) = 3

k ( 54, 3) = -2.114968e-19

q0( 54, 3) = 1.047198e+00

n ( 54, 3) = 3

k ( 55, 3) = 2.117121e-18

q0( 55, 3) = 1.047198e+00

n ( 55, 3) = 3

k ( 56, 3) = 1.051896e-02

q0( 56, 3) = 0.000000e+00

n ( 56, 3) = 2

k ( 57, 3) = 1.195683e-02

q0( 57, 3) = 0.000000e+00

n ( 57, 3) = 2

k ( 58, 3) = 1.250556e-02

q0( 58, 3) = 0.000000e+00

n ( 58, 3) = 2

k ( 59, 3) = 1.182710e-02

q0( 59, 3) = 0.000000e+00

n ( 59, 3) = 2

k ( 60, 3) = 1.215134e-02

q0( 60, 3) = 0.000000e+00

n ( 60, 3) = 2

k ( 61, 3) = 1.352822e-02

q0( 61, 3) = 0.000000e+00

n ( 61, 3) = 2

k ( 62, 3) = 5.187953e-03

q0( 62, 3) = 0.000000e+00

n ( 62, 3) = 2

k ( 63, 3) = 5.858189e-03

q0( 63, 3) = 0.000000e+00

n ( 63, 3) = 2

k ( 64, 3) = 6.068200e-03

q0( 64, 3) = 0.000000e+00

n ( 64, 3) = 2

k ( 65, 3) = 4.091956e-03

q0( 65, 3) = 0.000000e+00

n ( 65, 3) = 2

k ( 66, 3) = 1.609711e-03

q0( 66, 3) = 0.000000e+00

n ( 66, 3) = 2

k ( 67, 3) = 5.318200e-03

q0( 67, 3) = 0.000000e+00

n ( 67, 3) = 2

k ( 68, 3) = 1.084807e-01

q0( 68, 3) = 0.000000e+00

n ( 68, 3) = 0

k ( 69, 3) = 7.573418e-02

q0( 69, 3) = 0.000000e+00

n ( 69, 3) = 0

k ( 70, 3) = 2.138858e-01

q0( 70, 3) = 0.000000e+00

n ( 70, 3) = 0

k ( 71, 3) = 1.169523e-01

q0( 71, 3) = 0.000000e+00

n ( 71, 3) = 0

k ( 72, 3) = 1.257060e-01

q0( 72, 3) = 0.000000e+00

n ( 72, 3) = 0

k ( 73, 3) = 1.056851e-01

q0( 73, 3) = 0.000000e+00

n ( 73, 3) = 0

!--- END generated by 'gen\_FFparam.py'

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 1, anchor point 4)

k ( 1, 4) = 2.502030e+00

q0( 1, 4) = 1.397000e+00

n ( 1, 4) = 0

k ( 2, 4) = 2.409036e+00

q0( 2, 4) = 1.402000e+00

n ( 2, 4) = 0

k ( 3, 4) = 1.504721e+00

q0( 3, 4) = 1.094000e+00

n ( 3, 4) = 0

k ( 4, 4) = 2.159074e+00

q0( 4, 4) = 1.414000e+00

n ( 4, 4) = 0

k ( 5, 4) = 1.504317e+00

q0( 5, 4) = 1.093000e+00

n ( 5, 4) = 0

k ( 6, 4) = 1.545522e+00

q0( 6, 4) = 1.086000e+00

n ( 6, 4) = 0

k ( 7, 4) = 1.545476e+00

q0( 7, 4) = 1.086000e+00

n ( 7, 4) = 0

k ( 8, 4) = 1.545476e+00

q0( 8, 4) = 1.086000e+00

n ( 8, 4) = 0

k ( 9, 4) = 2.159100e+00

q0( 9, 4) = 1.414000e+00

n ( 9, 4) = 0

k ( 10, 4) = 2.043999e+00

q0( 10, 4) = 1.747000e+00

n ( 10, 4) = 0

k ( 11, 4) = 2.502011e+00

q0( 11, 4) = 1.397000e+00

n ( 11, 4) = 0

k ( 12, 4) = 1.504319e+00

q0( 12, 4) = 1.093000e+00

n ( 12, 4) = 0

k ( 13, 4) = 2.409125e+00

q0( 13, 4) = 1.402000e+00

n ( 13, 4) = 0

k ( 14, 4) = 1.504723e+00

q0( 14, 4) = 1.094000e+00

n ( 14, 4) = 0

k ( 15, 4) = 1.513360e+00

q0( 15, 4) = 1.092000e+00

n ( 15, 4) = 0

k ( 16, 4) = 2.573168e-01

q0( 16, 4) = 2.091812e+00

n ( 16, 4) = 0

k ( 17, 4) = 1.741128e-01

q0( 17, 4) = 2.096332e+00

n ( 17, 4) = 0

k ( 18, 4) = 2.347779e-01

q0( 18, 4) = 2.080729e+00

n ( 18, 4) = 0

k ( 19, 4) = 1.658820e-01

q0( 19, 4) = 2.101464e+00

n ( 19, 4) = 0

k ( 20, 4) = 1.958142e-01

q0( 20, 4) = 2.109702e+00

n ( 20, 4) = 0

k ( 21, 4) = 1.493316e-01

q0( 21, 4) = 2.078076e+00

n ( 21, 4) = 0

k ( 22, 4) = 1.065801e-01

q0( 22, 4) = 2.083818e+00

n ( 22, 4) = 0

k ( 23, 4) = 2.061597e-01

q0( 23, 4) = 2.100259e+00

n ( 23, 4) = 0

k ( 24, 4) = 1.361638e-01

q0( 24, 4) = 2.095512e+00

n ( 24, 4) = 0

k ( 25, 4) = 2.573286e-01

q0( 25, 4) = 2.091812e+00

n ( 25, 4) = 0

k ( 26, 4) = 1.361680e-01

q0( 26, 4) = 2.095512e+00

n ( 26, 4) = 0

k ( 27, 4) = 2.061071e-01

q0( 27, 4) = 2.100277e+00

n ( 27, 4) = 0

k ( 28, 4) = 1.958077e-01

q0( 28, 4) = 2.109702e+00

n ( 28, 4) = 0

k ( 29, 4) = 1.493255e-01

q0( 29, 4) = 2.078076e+00

n ( 29, 4) = 0

k ( 30, 4) = 1.741121e-01

q0( 30, 4) = 2.096332e+00

n ( 30, 4) = 0

k ( 31, 4) = 1.658816e-01

q0( 31, 4) = 2.101464e+00

n ( 31, 4) = 0

k ( 32, 4) = 1.594352e-01

q0( 32, 4) = 2.095861e+00

n ( 32, 4) = 0

k ( 33, 4) = 1.594284e-01

q0( 33, 4) = 2.095861e+00

n ( 33, 4) = 0

k ( 34, 4) = 1.035906e-01

q0( 34, 4) = 2.094517e+00

n ( 34, 4) = 0

k ( 35, 4) = 1.035906e-01

q0( 35, 4) = 2.094517e+00

n ( 35, 4) = 0

k ( 36, 4) = 1.035906e-01

q0( 36, 4) = 2.094517e+00

n ( 36, 4) = 0

k ( 37, 4) = 6.104468e-04

q0( 37, 4) = 0.000000e+00

n ( 37, 4) = 2

k ( 38, 4) = 1.074035e-02

q0( 38, 4) = 0.000000e+00

n ( 38, 4) = 2

k ( 39, 4) = 1.621593e-02

q0( 39, 4) = 0.000000e+00

n ( 39, 4) = 2

k ( 40, 4) = 1.216098e-02

q0( 40, 4) = 0.000000e+00

n ( 40, 4) = 2

k ( 41, 4) = 5.417335e-03

q0( 41, 4) = 0.000000e+00

n ( 41, 4) = 2

k ( 42, 4) = 1.090588e-02

q0( 42, 4) = 0.000000e+00

n ( 42, 4) = 2

k ( 43, 4) = 1.159637e-02

q0( 43, 4) = 0.000000e+00

n ( 43, 4) = 2

k ( 44, 4) = 8.506780e-03

q0( 44, 4) = 0.000000e+00

n ( 44, 4) = 2

k ( 45, 4) = 1.780232e-02

q0( 45, 4) = 0.000000e+00

n ( 45, 4) = 2

k ( 46, 4) = 1.139541e-02

q0( 46, 4) = 0.000000e+00

n ( 46, 4) = 2

k ( 47, 4) = 7.035355e-03

q0( 47, 4) = 0.000000e+00

n ( 47, 4) = 2

k ( 48, 4) = 1.139430e-02

q0( 48, 4) = 0.000000e+00

n ( 48, 4) = 2

k ( 49, 4) = 8.504789e-03

q0( 49, 4) = 0.000000e+00

n ( 49, 4) = 2

k ( 50, 4) = 1.090593e-02

q0( 50, 4) = 0.000000e+00

n ( 50, 4) = 2

k ( 51, 4) = 1.073720e-02

q0( 51, 4) = 0.000000e+00

n ( 51, 4) = 2

k ( 52, 4) = 1.216066e-02

q0( 52, 4) = 0.000000e+00

n ( 52, 4) = 2

k ( 53, 4) = 1.308835e-02

q0( 53, 4) = 0.000000e+00

n ( 53, 4) = 2

k ( 54, 4) = 1.308665e-02

q0( 54, 4) = 0.000000e+00

n ( 54, 4) = 2

k ( 55, 4) = 4.533650e-03

q0( 55, 4) = 0.000000e+00

n ( 55, 4) = 2

k ( 56, 4) = 4.533081e-03

q0( 56, 4) = 0.000000e+00

n ( 56, 4) = 2

k ( 57, 4) = 4.578897e-03

q0( 57, 4) = 0.000000e+00

n ( 57, 4) = 2

k ( 58, 4) = 2.232449e-03

q0( 58, 4) = 0.000000e+00

n ( 58, 4) = 2

k ( 59, 4) = 2.231001e-03

q0( 59, 4) = 0.000000e+00

n ( 59, 4) = 2

k ( 60, 4) = 4.576198e-03

q0( 60, 4) = 0.000000e+00

n ( 60, 4) = 2

k ( 61, 4) = 1.121196e-01

q0( 61, 4) = 0.000000e+00

n ( 61, 4) = 0

k ( 62, 4) = 8.981265e-02

q0( 62, 4) = 0.000000e+00

n ( 62, 4) = 0

k ( 63, 4) = 2.236630e-01

q0( 63, 4) = 0.000000e+00

n ( 63, 4) = 0

k ( 64, 4) = 1.151899e-01

q0( 64, 4) = 0.000000e+00

n ( 64, 4) = 0

k ( 65, 4) = 1.121871e-01

q0( 65, 4) = 0.000000e+00

n ( 65, 4) = 0

k ( 66, 4) = 1.152280e-01

q0( 66, 4) = 0.000000e+00

n ( 66, 4) = 0

k ( 67, 4) = 4.859191e-02

q0( 67, 4) = 0.000000e+00

n ( 67, 4) = 0

!--- END generated by 'gen\_FFparam.py'

case(2)

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 2, anchor point 1)

k ( 1, 1) = 2.099256e+00

q0( 1, 1) = 1.448000e+00

n ( 1, 1) = 0

k ( 2, 1) = 2.332055e+00

q0( 2, 1) = 1.400000e+00

n ( 2, 1) = 0

k ( 3, 1) = 1.524362e+00

q0( 3, 1) = 1.091000e+00

n ( 3, 1) = 0

k ( 4, 1) = 1.676149e+00

q0( 4, 1) = 1.421000e+00

n ( 4, 1) = 0

k ( 5, 1) = 1.545961e+00

q0( 5, 1) = 1.091000e+00

n ( 5, 1) = 0

k ( 6, 1) = 1.388955e+00

q0( 6, 1) = 1.100000e+00

n ( 6, 1) = 0

k ( 7, 1) = 1.388952e+00

q0( 7, 1) = 1.100000e+00

n ( 7, 1) = 0

k ( 8, 1) = 1.423946e+00

q0( 8, 1) = 1.098000e+00

n ( 8, 1) = 0

k ( 9, 1) = 2.034759e+00

q0( 9, 1) = 1.400000e+00

n ( 9, 1) = 0

k ( 10, 1) = 1.861592e+00

q0( 10, 1) = 1.758000e+00

n ( 10, 1) = 0

k ( 11, 1) = 2.197888e+00

q0( 11, 1) = 1.446000e+00

n ( 11, 1) = 0

k ( 12, 1) = 1.536207e+00

q0( 12, 1) = 1.090000e+00

n ( 12, 1) = 0

k ( 13, 1) = 2.036112e+00

q0( 13, 1) = 1.418000e+00

n ( 13, 1) = 0

k ( 14, 1) = 1.530491e+00

q0( 14, 1) = 1.090000e+00

n ( 14, 1) = 0

k ( 15, 1) = 1.475892e+00

q0( 15, 1) = 1.096000e+00

n ( 15, 1) = 0

k ( 16, 1) = 1.315937e-01

q0( 16, 1) = 2.032436e+00

n ( 16, 1) = 0

k ( 17, 1) = 1.807470e-01

q0( 17, 1) = 2.133595e+00

n ( 17, 1) = 0

k ( 18, 1) = 3.722420e-01

q0( 18, 1) = 2.152078e+00

n ( 18, 1) = 0

k ( 19, 1) = 1.184643e-01

q0( 19, 1) = 2.066871e+00

n ( 19, 1) = 0

k ( 20, 1) = 5.047498e-02

q0( 20, 1) = 2.081986e+00

n ( 20, 1) = 0

k ( 21, 1) = 1.793240e-01

q0( 21, 1) = 2.098532e+00

n ( 21, 1) = 0

k ( 22, 1) = 1.671987e-01

q0( 22, 1) = 2.184716e+00

n ( 22, 1) = 0

k ( 23, 1) = 1.926083e-01

q0( 23, 1) = 1.956532e+00

n ( 23, 1) = 0

k ( 24, 1) = 4.050324e-01

q0( 24, 1) = 1.868148e+00

n ( 24, 1) = 0

k ( 25, 1) = 1.136161e-01

q0( 25, 1) = 2.117660e+00

n ( 25, 1) = 0

k ( 26, 1) = 1.315137e-01

q0( 26, 1) = 2.050081e+00

n ( 26, 1) = 0

k ( 27, 1) = 1.199475e-01

q0( 27, 1) = 2.129040e+00

n ( 27, 1) = 0

k ( 28, 1) = 8.090805e-02

q0( 28, 1) = 2.143160e+00

n ( 28, 1) = 0

k ( 29, 1) = 1.123646e-01

q0( 29, 1) = 2.066871e+00

n ( 29, 1) = 0

k ( 30, 1) = 1.423482e-01

q0( 30, 1) = 2.104151e+00

n ( 30, 1) = 0

k ( 31, 1) = 1.874988e-01

q0( 31, 1) = 2.104693e+00

n ( 31, 1) = 0

k ( 32, 1) = 1.359300e-01

q0( 32, 1) = 2.064829e+00

n ( 32, 1) = 0

k ( 33, 1) = 1.224407e-01

q0( 33, 1) = 2.103209e+00

n ( 33, 1) = 0

k ( 34, 1) = 1.382518e-01

q0( 34, 1) = 2.112721e+00

n ( 34, 1) = 0

k ( 35, 1) = 8.629370e-02

q0( 35, 1) = 1.921660e+00

n ( 35, 1) = 0

k ( 36, 1) = 1.004670e-01

q0( 36, 1) = 1.924400e+00

n ( 36, 1) = 0

k ( 37, 1) = 1.016203e-01

q0( 37, 1) = 1.918728e+00

n ( 37, 1) = 0

k ( 38, 1) = 1.004689e-01

q0( 38, 1) = 1.924400e+00

n ( 38, 1) = 0

k ( 39, 1) = 1.016195e-01

q0( 39, 1) = 1.918728e+00

n ( 39, 1) = 0

k ( 40, 1) = 1.190151e-01

q0( 40, 1) = 1.855721e+00

n ( 40, 1) = 0

k ( 41, 1) = 1.196000e-02

q0( 41, 1) = 0.000000e+00

n ( 41, 1) = 2

k ( 42, 1) = 4.528281e-03

q0( 42, 1) = 0.000000e+00

n ( 42, 1) = 2

k ( 43, 1) = 7.046882e-03

q0( 43, 1) = 0.000000e+00

n ( 43, 1) = 2

k ( 44, 1) = 5.387371e-03

q0( 44, 1) = 0.000000e+00

n ( 44, 1) = 2

k ( 45, 1) = 1.420963e-02

q0( 45, 1) = 0.000000e+00

n ( 45, 1) = 2

k ( 46, 1) = 1.126947e-02

q0( 46, 1) = 0.000000e+00

n ( 46, 1) = 2

k ( 47, 1) = 1.620348e-02

q0( 47, 1) = 0.000000e+00

n ( 47, 1) = 2

k ( 48, 1) = 7.125802e-03

q0( 48, 1) = 0.000000e+00

n ( 48, 1) = 2

k ( 49, 1) = 1.972603e-04

q0( 49, 1) = 0.000000e+00

n ( 49, 1) = 2

k ( 50, 1) = 5.037631e-03

q0( 50, 1) = 0.000000e+00

n ( 50, 1) = 2

k ( 51, 1) = 4.592675e-04

q0( 51, 1) = 0.000000e+00

n ( 51, 1) = 2

k ( 52, 1) = 3.954652e-03

q0( 52, 1) = 0.000000e+00

n ( 52, 1) = 2

k ( 53, 1) = 8.470852e-04

q0( 53, 1) = 1.047198e+00

n ( 53, 1) = 3

k ( 54, 1) = 8.464375e-04

q0( 54, 1) = 1.047198e+00

n ( 54, 1) = 3

k ( 55, 1) = 3.480857e-03

q0( 55, 1) = 1.047198e+00

n ( 55, 1) = 3

k ( 56, 1) = 5.766086e-03

q0( 56, 1) = 0.000000e+00

n ( 56, 1) = 2

k ( 57, 1) = 7.487901e-03

q0( 57, 1) = 0.000000e+00

n ( 57, 1) = 2

k ( 58, 1) = 5.492509e-03

q0( 58, 1) = 0.000000e+00

n ( 58, 1) = 2

k ( 59, 1) = 8.722326e-03

q0( 59, 1) = 0.000000e+00

n ( 59, 1) = 2

k ( 60, 1) = 1.920246e-03

q0( 60, 1) = 0.000000e+00

n ( 60, 1) = 2

k ( 61, 1) = 3.148223e-03

q0( 61, 1) = 0.000000e+00

n ( 61, 1) = 2

k ( 62, 1) = 5.201883e-03

q0( 62, 1) = 0.000000e+00

n ( 62, 1) = 2

k ( 63, 1) = 2.595324e-03

q0( 63, 1) = 0.000000e+00

n ( 63, 1) = 2

k ( 64, 1) = 1.898873e-03

q0( 64, 1) = 0.000000e+00

n ( 64, 1) = 2

k ( 65, 1) = -1.363518e-18

q0( 65, 1) = 0.000000e+00

n ( 65, 1) = 2

k ( 66, 1) = 1.315343e-04

q0( 66, 1) = 0.000000e+00

n ( 66, 1) = 2

k ( 67, 1) = 1.936263e-03

q0( 67, 1) = 0.000000e+00

n ( 67, 1) = 2

k ( 68, 1) = 0.000000e+00

q0( 68, 1) = 0.000000e+00

n ( 68, 1) = 0

k ( 69, 1) = 1.826796e-01

q0( 69, 1) = 0.000000e+00

n ( 69, 1) = 0

k ( 70, 1) = 6.632873e-02

q0( 70, 1) = 0.000000e+00

n ( 70, 1) = 0

k ( 71, 1) = 1.152733e-02

q0( 71, 1) = 0.000000e+00

n ( 71, 1) = 0

k ( 72, 1) = 5.891947e-02

q0( 72, 1) = 0.000000e+00

n ( 72, 1) = 0

k ( 73, 1) = 0.000000e+00

q0( 73, 1) = 0.000000e+00

n ( 73, 1) = 0

!--- END generated by 'gen\_FFparam.py'

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 2, anchor point 2)

k ( 1, 2) = 2.114727e+00

q0( 1, 2) = 1.449000e+00

n ( 1, 2) = 0

k ( 2, 2) = 2.273469e+00

q0( 2, 2) = 1.403000e+00

n ( 2, 2) = 0

k ( 3, 2) = 1.524679e+00

q0( 3, 2) = 1.092000e+00

n ( 3, 2) = 0

k ( 4, 2) = 1.658834e+00

q0( 4, 2) = 1.424000e+00

n ( 4, 2) = 0

k ( 5, 2) = 1.540762e+00

q0( 5, 2) = 1.092000e+00

n ( 5, 2) = 0

k ( 6, 2) = 1.486884e+00

q0( 6, 2) = 1.093000e+00

n ( 6, 2) = 0

k ( 7, 2) = 1.486886e+00

q0( 7, 2) = 1.093000e+00

n ( 7, 2) = 0

k ( 8, 2) = 1.479232e+00

q0( 8, 2) = 1.094000e+00

n ( 8, 2) = 0

k ( 9, 2) = 1.952021e+00

q0( 9, 2) = 1.404000e+00

n ( 9, 2) = 0

k ( 10, 2) = 1.928628e+00

q0( 10, 2) = 1.754000e+00

n ( 10, 2) = 0

k ( 11, 2) = 2.213432e+00

q0( 11, 2) = 1.446000e+00

n ( 11, 2) = 0

k ( 12, 2) = 1.548743e+00

q0( 12, 2) = 1.089000e+00

n ( 12, 2) = 0

k ( 13, 2) = 2.088041e+00

q0( 13, 2) = 1.415000e+00

n ( 13, 2) = 0

k ( 14, 2) = 1.528886e+00

q0( 14, 2) = 1.090000e+00

n ( 14, 2) = 0

k ( 15, 2) = 1.474695e+00

q0( 15, 2) = 1.096000e+00

n ( 15, 2) = 0

k ( 16, 2) = 1.296070e-01

q0( 16, 2) = 2.036904e+00

n ( 16, 2) = 0

k ( 17, 2) = 1.803101e-01

q0( 17, 2) = 2.126422e+00

n ( 17, 2) = 0

k ( 18, 2) = 3.781834e-01

q0( 18, 2) = 2.151851e+00

n ( 18, 2) = 0

k ( 19, 2) = 1.223211e-01

q0( 19, 2) = 2.067255e+00

n ( 19, 2) = 0

k ( 20, 2) = 5.433101e-02

q0( 20, 2) = 2.078513e+00

n ( 20, 2) = 0

k ( 21, 2) = 1.721094e-01

q0( 21, 2) = 2.101952e+00

n ( 21, 2) = 0

k ( 22, 2) = 1.808456e-01

q0( 22, 2) = 2.176129e+00

n ( 22, 2) = 0

k ( 23, 2) = 1.480839e-01

q0( 23, 2) = 1.965398e+00

n ( 23, 2) = 0

k ( 24, 2) = 3.401620e-01

q0( 24, 2) = 1.856245e+00

n ( 24, 2) = 0

k ( 25, 2) = 1.167652e-01

q0( 25, 2) = 2.120854e+00

n ( 25, 2) = 0

k ( 26, 2) = 1.291621e-01

q0( 26, 2) = 2.052158e+00

n ( 26, 2) = 0

k ( 27, 2) = 1.180560e-01

q0( 27, 2) = 2.117416e+00

n ( 27, 2) = 0

k ( 28, 2) = 7.312172e-02

q0( 28, 2) = 2.144294e+00

n ( 28, 2) = 0

k ( 29, 2) = 9.702423e-02

q0( 29, 2) = 2.072666e+00

n ( 29, 2) = 0

k ( 30, 2) = 1.513126e-01

q0( 30, 2) = 2.101673e+00

n ( 30, 2) = 0

k ( 31, 2) = 1.810125e-01

q0( 31, 2) = 2.112477e+00

n ( 31, 2) = 0

k ( 32, 2) = 1.317696e-01

q0( 32, 2) = 2.064864e+00

n ( 32, 2) = 0

k ( 33, 2) = 1.238119e-01

q0( 33, 2) = 2.103645e+00

n ( 33, 2) = 0

k ( 34, 2) = 1.343242e-01

q0( 34, 2) = 2.108026e+00

n ( 34, 2) = 0

k ( 35, 2) = 1.013640e-01

q0( 35, 2) = 2.014442e+00

n ( 35, 2) = 0

k ( 36, 2) = 1.055022e-01

q0( 36, 2) = 2.002277e+00

n ( 36, 2) = 0

k ( 37, 2) = 9.307031e-02

q0( 37, 2) = 1.819855e+00

n ( 37, 2) = 0

k ( 38, 2) = 1.055026e-01

q0( 38, 2) = 2.002277e+00

n ( 38, 2) = 0

k ( 39, 2) = 9.307146e-02

q0( 39, 2) = 1.819855e+00

n ( 39, 2) = 0

k ( 40, 2) = 1.080583e-01

q0( 40, 2) = 1.769083e+00

n ( 40, 2) = 0

k ( 41, 2) = 1.403860e-02

q0( 41, 2) = 0.000000e+00

n ( 41, 2) = 2

k ( 42, 2) = 1.326022e-02

q0( 42, 2) = 0.000000e+00

n ( 42, 2) = 2

k ( 43, 2) = 8.001698e-03

q0( 43, 2) = 0.000000e+00

n ( 43, 2) = 2

k ( 44, 2) = 6.270384e-03

q0( 44, 2) = 0.000000e+00

n ( 44, 2) = 2

k ( 45, 2) = 1.271884e-02

q0( 45, 2) = 0.000000e+00

n ( 45, 2) = 2

k ( 46, 2) = 1.025594e-02

q0( 46, 2) = 0.000000e+00

n ( 46, 2) = 2

k ( 47, 2) = 1.671992e-02

q0( 47, 2) = 0.000000e+00

n ( 47, 2) = 2

k ( 48, 2) = 5.561355e-03

q0( 48, 2) = 0.000000e+00

n ( 48, 2) = 2

k ( 49, 2) = -1.941309e-19

q0( 49, 2) = 0.000000e+00

n ( 49, 2) = 2

k ( 50, 2) = 4.518538e-03

q0( 50, 2) = 0.000000e+00

n ( 50, 2) = 2

k ( 51, 2) = 1.625108e-20

q0( 51, 2) = 0.000000e+00

n ( 51, 2) = 2

k ( 52, 2) = 3.442974e-03

q0( 52, 2) = 0.000000e+00

n ( 52, 2) = 2

k ( 53, 2) = 7.948978e-04

q0( 53, 2) = 1.047198e+00

n ( 53, 2) = 3

k ( 54, 2) = 7.945585e-04

q0( 54, 2) = 1.047198e+00

n ( 54, 2) = 3

k ( 55, 2) = 1.295192e-03

q0( 55, 2) = 1.047198e+00

n ( 55, 2) = 3

k ( 56, 2) = 5.439138e-03

q0( 56, 2) = 0.000000e+00

n ( 56, 2) = 2

k ( 57, 2) = 7.174384e-03

q0( 57, 2) = 0.000000e+00

n ( 57, 2) = 2

k ( 58, 2) = 1.168332e-02

q0( 58, 2) = 0.000000e+00

n ( 58, 2) = 2

k ( 59, 2) = 8.218264e-03

q0( 59, 2) = 0.000000e+00

n ( 59, 2) = 2

k ( 60, 2) = 2.651560e-03

q0( 60, 2) = 0.000000e+00

n ( 60, 2) = 2

k ( 61, 2) = 3.729236e-03

q0( 61, 2) = 0.000000e+00

n ( 61, 2) = 2

k ( 62, 2) = 5.217169e-03

q0( 62, 2) = 0.000000e+00

n ( 62, 2) = 2

k ( 63, 2) = 3.355673e-03

q0( 63, 2) = 0.000000e+00

n ( 63, 2) = 2

k ( 64, 2) = 1.747159e-03

q0( 64, 2) = 0.000000e+00

n ( 64, 2) = 2

k ( 65, 2) = 2.357344e-04

q0( 65, 2) = 0.000000e+00

n ( 65, 2) = 2

k ( 66, 2) = 3.322309e-19

q0( 66, 2) = 0.000000e+00

n ( 66, 2) = 2

k ( 67, 2) = 1.747420e-03

q0( 67, 2) = 0.000000e+00

n ( 67, 2) = 2

k ( 68, 2) = -0.000000e+00

q0( 68, 2) = 0.000000e+00

n ( 68, 2) = 0

k ( 69, 2) = 1.832707e-01

q0( 69, 2) = 0.000000e+00

n ( 69, 2) = 0

k ( 70, 2) = 3.061437e-02

q0( 70, 2) = 0.000000e+00

n ( 70, 2) = 0

k ( 71, 2) = 2.906114e-04

q0( 71, 2) = 0.000000e+00

n ( 71, 2) = 0

k ( 72, 2) = 4.363514e-02

q0( 72, 2) = 0.000000e+00

n ( 72, 2) = 0

k ( 73, 2) = 0.000000e+00

q0( 73, 2) = 0.000000e+00

n ( 73, 2) = 0

!--- END generated by 'gen\_FFparam.py'

case(3)

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 3, anchor point 1)

k ( 1, 1) = 2.373294e+00

q0( 1, 1) = 1.440000e+00

n ( 1, 1) = 0

k ( 2, 1) = 2.177843e+00

q0( 2, 1) = 1.405000e+00

n ( 2, 1) = 0

k ( 3, 1) = 1.509142e+00

q0( 3, 1) = 1.091000e+00

n ( 3, 1) = 0

k ( 4, 1) = 1.154301e+00

q0( 4, 1) = 1.442000e+00

n ( 4, 1) = 0

k ( 5, 1) = 1.535268e+00

q0( 5, 1) = 1.094000e+00

n ( 5, 1) = 0

k ( 6, 1) = 1.416883e+00

q0( 6, 1) = 1.096000e+00

n ( 6, 1) = 0

k ( 7, 1) = 1.416884e+00

q0( 7, 1) = 1.096000e+00

n ( 7, 1) = 0

k ( 8, 1) = 1.403014e+00

q0( 8, 1) = 1.101000e+00

n ( 8, 1) = 0

k ( 9, 1) = 1.529880e+00

q0( 9, 1) = 1.409000e+00

n ( 9, 1) = 0

k ( 10, 1) = 2.117134e+00

q0( 10, 1) = 1.743000e+00

n ( 10, 1) = 0

k ( 11, 1) = 2.514729e+00

q0( 11, 1) = 1.437000e+00

n ( 11, 1) = 0

k ( 12, 1) = 1.557646e+00

q0( 12, 1) = 1.088000e+00

n ( 12, 1) = 0

k ( 13, 1) = 1.876803e+00

q0( 13, 1) = 1.424000e+00

n ( 13, 1) = 0

k ( 14, 1) = 1.525333e+00

q0( 14, 1) = 1.091000e+00

n ( 14, 1) = 0

k ( 15, 1) = 1.509592e+00

q0( 15, 1) = 1.092000e+00

n ( 15, 1) = 0

k ( 16, 1) = 4.205182e-02

q0( 16, 1) = 2.083120e+00

n ( 16, 1) = 0

k ( 17, 1) = 1.205805e-01

q0( 17, 1) = 2.120959e+00

n ( 17, 1) = 0

k ( 18, 1) = 3.783742e-01

q0( 18, 1) = 2.109475e+00

n ( 18, 1) = 0

k ( 19, 1) = 9.620915e-02

q0( 19, 1) = 2.085773e+00

n ( 19, 1) = 0

k ( 20, 1) = -0.000000e+00

q0( 20, 1) = 2.074970e+00

n ( 20, 1) = 0

k ( 21, 1) = 2.055118e-01

q0( 21, 1) = 2.109841e+00

n ( 21, 1) = 0

k ( 22, 1) = 2.457110e-01

q0( 22, 1) = 2.125165e+00

n ( 22, 1) = 0

k ( 23, 1) = 1.410898e-01

q0( 23, 1) = 1.963426e+00

n ( 23, 1) = 0

k ( 24, 1) = 3.675180e-01

q0( 24, 1) = 1.807288e+00

n ( 24, 1) = 0

k ( 25, 1) = 1.674981e-01

q0( 25, 1) = 2.099736e+00

n ( 25, 1) = 0

k ( 26, 1) = -0.000000e+00

q0( 26, 1) = 2.089927e+00

n ( 26, 1) = 0

k ( 27, 1) = 1.327572e-01

q0( 27, 1) = 2.118830e+00

n ( 27, 1) = 0

k ( 28, 1) = 5.140471e-02

q0( 28, 1) = 2.155813e+00

n ( 28, 1) = 0

k ( 29, 1) = 6.438814e-02

q0( 29, 1) = 2.087484e+00

n ( 29, 1) = 0

k ( 30, 1) = 1.297719e-01

q0( 30, 1) = 2.094866e+00

n ( 30, 1) = 0

k ( 31, 1) = 1.817491e-01

q0( 31, 1) = 2.082928e+00

n ( 31, 1) = 0

k ( 32, 1) = 1.378603e-01

q0( 32, 1) = 2.077989e+00

n ( 32, 1) = 0

k ( 33, 1) = 1.153952e-01

q0( 33, 1) = 2.110016e+00

n ( 33, 1) = 0

k ( 34, 1) = 1.465455e-01

q0( 34, 1) = 2.113419e+00

n ( 34, 1) = 0

k ( 35, 1) = 7.546414e-02

q0( 35, 1) = 1.991665e+00

n ( 35, 1) = 0

k ( 36, 1) = 8.991739e-02

q0( 36, 1) = 1.927943e+00

n ( 36, 1) = 0

k ( 37, 1) = 1.274981e-01

q0( 37, 1) = 1.881797e+00

n ( 37, 1) = 0

k ( 38, 1) = 8.991929e-02

q0( 38, 1) = 1.927943e+00

n ( 38, 1) = 0

k ( 39, 1) = 1.274996e-01

q0( 39, 1) = 1.881797e+00

n ( 39, 1) = 0

k ( 40, 1) = 1.086064e-01

q0( 40, 1) = 1.853854e+00

n ( 40, 1) = 0

k ( 41, 1) = 1.047469e-02

q0( 41, 1) = 0.000000e+00

n ( 41, 1) = 2

k ( 42, 1) = 1.572206e-02

q0( 42, 1) = 0.000000e+00

n ( 42, 1) = 2

k ( 43, 1) = 4.691529e-03

q0( 43, 1) = 0.000000e+00

n ( 43, 1) = 2

k ( 44, 1) = 7.708206e-04

q0( 44, 1) = 0.000000e+00

n ( 44, 1) = 2

k ( 45, 1) = 1.212452e-02

q0( 45, 1) = 0.000000e+00

n ( 45, 1) = 2

k ( 46, 1) = 7.812745e-03

q0( 46, 1) = 0.000000e+00

n ( 46, 1) = 2

k ( 47, 1) = 2.113265e-02

q0( 47, 1) = 0.000000e+00

n ( 47, 1) = 2

k ( 48, 1) = 1.133672e-03

q0( 48, 1) = 0.000000e+00

n ( 48, 1) = 2

k ( 49, 1) = -5.200703e-18

q0( 49, 1) = 0.000000e+00

n ( 49, 1) = 2

k ( 50, 1) = 2.922068e-03

q0( 50, 1) = 0.000000e+00

n ( 50, 1) = 2

k ( 51, 1) = 6.892048e-18

q0( 51, 1) = 0.000000e+00

n ( 51, 1) = 2

k ( 52, 1) = 9.637625e-03

q0( 52, 1) = 0.000000e+00

n ( 52, 1) = 2

k ( 53, 1) = 5.763774e-03

q0( 53, 1) = 1.047198e+00

n ( 53, 1) = 3

k ( 54, 1) = 5.763681e-03

q0( 54, 1) = 1.047198e+00

n ( 54, 1) = 3

k ( 55, 1) = 4.194053e-03

q0( 55, 1) = 1.047198e+00

n ( 55, 1) = 3

k ( 56, 1) = 2.181215e-03

q0( 56, 1) = 0.000000e+00

n ( 56, 1) = 2

k ( 57, 1) = 2.666180e-03

q0( 57, 1) = 0.000000e+00

n ( 57, 1) = 2

k ( 58, 1) = 9.700088e-03

q0( 58, 1) = 0.000000e+00

n ( 58, 1) = 2

k ( 59, 1) = 7.602363e-03

q0( 59, 1) = 0.000000e+00

n ( 59, 1) = 2

k ( 60, 1) = 4.103400e-03

q0( 60, 1) = 0.000000e+00

n ( 60, 1) = 2

k ( 61, 1) = 6.551103e-03

q0( 61, 1) = 0.000000e+00

n ( 61, 1) = 2

k ( 62, 1) = 4.802784e-03

q0( 62, 1) = 0.000000e+00

n ( 62, 1) = 2

k ( 63, 1) = 9.354648e-04

q0( 63, 1) = 0.000000e+00

n ( 63, 1) = 2

k ( 64, 1) = 1.598692e-03

q0( 64, 1) = 0.000000e+00

n ( 64, 1) = 2

k ( 65, 1) = 5.542229e-18

q0( 65, 1) = 0.000000e+00

n ( 65, 1) = 2

k ( 66, 1) = -7.681318e-19

q0( 66, 1) = 0.000000e+00

n ( 66, 1) = 2

k ( 67, 1) = 3.090533e-03

q0( 67, 1) = 0.000000e+00

n ( 67, 1) = 2

k ( 68, 1) = 0.000000e+00

q0( 68, 1) = 0.000000e+00

n ( 68, 1) = 0

k ( 69, 1) = 2.290205e-01

q0( 69, 1) = 0.000000e+00

n ( 69, 1) = 0

k ( 70, 1) = -0.000000e+00

q0( 70, 1) = 0.000000e+00

n ( 70, 1) = 0

k ( 71, 1) = 2.056219e-02

q0( 71, 1) = 0.000000e+00

n ( 71, 1) = 0

k ( 72, 1) = 5.071417e-02

q0( 72, 1) = 0.000000e+00

n ( 72, 1) = 0

k ( 73, 1) = 0.000000e+00

q0( 73, 1) = 0.000000e+00

n ( 73, 1) = 0

!--- END generated by 'gen\_FFparam.py'

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 3, anchor point 2)

k ( 1, 2) = 2.597274e+00

q0( 1, 2) = 1.385000e+00

n ( 1, 2) = 0

k ( 2, 2) = 2.217378e+00

q0( 2, 2) = 1.410000e+00

n ( 2, 2) = 0

k ( 3, 2) = 1.513056e+00

q0( 3, 2) = 1.093000e+00

n ( 3, 2) = 0

k ( 4, 2) = 1.978833e+00

q0( 4, 2) = 1.440000e+00

n ( 4, 2) = 0

k ( 5, 2) = 1.535447e+00

q0( 5, 2) = 1.092000e+00

n ( 5, 2) = 0

k ( 6, 2) = 1.406883e+00

q0( 6, 2) = 1.101000e+00

n ( 6, 2) = 0

k ( 7, 2) = 1.406879e+00

q0( 7, 2) = 1.101000e+00

n ( 7, 2) = 0

k ( 8, 2) = 1.447261e+00

q0( 8, 2) = 1.097000e+00

n ( 8, 2) = 0

k ( 9, 2) = 1.896865e+00

q0( 9, 2) = 1.440000e+00

n ( 9, 2) = 0

k ( 10, 2) = 2.365632e+00

q0( 10, 2) = 1.700000e+00

n ( 10, 2) = 0

k ( 11, 2) = 2.598387e+00

q0( 11, 2) = 1.387000e+00

n ( 11, 2) = 0

k ( 12, 2) = 1.554059e+00

q0( 12, 2) = 1.089000e+00

n ( 12, 2) = 0

k ( 13, 2) = 2.222899e+00

q0( 13, 2) = 1.411000e+00

n ( 13, 2) = 0

k ( 14, 2) = 1.508677e+00

q0( 14, 2) = 1.093000e+00

n ( 14, 2) = 0

k ( 15, 2) = 1.520533e+00

q0( 15, 2) = 1.093000e+00

n ( 15, 2) = 0

k ( 16, 2) = 2.421399e-01

q0( 16, 2) = 2.118079e+00

n ( 16, 2) = 0

k ( 17, 2) = 1.924206e-01

q0( 17, 2) = 2.116718e+00

n ( 17, 2) = 0

k ( 18, 2) = 2.096623e-01

q0( 18, 2) = 2.095914e+00

n ( 18, 2) = 0

k ( 19, 2) = 1.743943e-01

q0( 19, 2) = 2.094360e+00

n ( 19, 2) = 0

k ( 20, 2) = 2.161719e-01

q0( 20, 2) = 2.095058e+00

n ( 20, 2) = 0

k ( 21, 2) = 1.448772e-01

q0( 21, 2) = 2.092807e+00

n ( 21, 2) = 0

k ( 22, 2) = 5.533501e-02

q0( 22, 2) = 2.050151e+00

n ( 22, 2) = 0

k ( 23, 2) = 2.609554e-01

q0( 23, 2) = 2.063677e+00

n ( 23, 2) = 0

k ( 24, 2) = 6.084291e-02

q0( 24, 2) = 1.719097e+00

n ( 24, 2) = 0

k ( 25, 2) = 9.994101e-02

q0( 25, 2) = 2.048144e+00

n ( 25, 2) = 0

k ( 26, 2) = 2.504176e-01

q0( 26, 2) = 2.104919e+00

n ( 26, 2) = 0

k ( 27, 2) = 1.118664e-01

q0( 27, 2) = 2.082317e+00

n ( 27, 2) = 0

k ( 28, 2) = 3.157335e-01

q0( 28, 2) = 2.169479e+00

n ( 28, 2) = 0

k ( 29, 2) = 2.111850e-01

q0( 29, 2) = 2.103576e+00

n ( 29, 2) = 0

k ( 30, 2) = 1.439041e-01

q0( 30, 2) = 2.092039e+00

n ( 30, 2) = 0

k ( 31, 2) = 1.877049e-01

q0( 31, 2) = 2.097798e+00

n ( 31, 2) = 0

k ( 32, 2) = 1.769565e-01

q0( 32, 2) = 2.093261e+00

n ( 32, 2) = 0

k ( 33, 2) = 1.522713e-01

q0( 33, 2) = 2.095163e+00

n ( 33, 2) = 0

k ( 34, 2) = 1.523391e-01

q0( 34, 2) = 2.089019e+00

n ( 34, 2) = 0

k ( 35, 2) = 9.501813e-02

q0( 35, 2) = 1.981612e+00

n ( 35, 2) = 0

k ( 36, 2) = 9.449595e-02

q0( 36, 2) = 1.956846e+00

n ( 36, 2) = 0

k ( 37, 2) = 5.735824e-02

q0( 37, 2) = 1.840223e+00

n ( 37, 2) = 0

k ( 38, 2) = 9.449480e-02

q0( 38, 2) = 1.956846e+00

n ( 38, 2) = 0

k ( 39, 2) = 5.735824e-02

q0( 39, 2) = 1.840223e+00

n ( 39, 2) = 0

k ( 40, 2) = 5.735824e-02

q0( 40, 2) = 1.840223e+00

n ( 40, 2) = 0

k ( 41, 2) = 7.234870e-03

q0( 41, 2) = 0.000000e+00

n ( 41, 2) = 2

k ( 42, 2) = 3.086217e-03

q0( 42, 2) = 0.000000e+00

n ( 42, 2) = 2

k ( 43, 2) = 2.389716e-03

q0( 43, 2) = 0.000000e+00

n ( 43, 2) = 2

k ( 44, 2) = 7.655210e-03

q0( 44, 2) = 0.000000e+00

n ( 44, 2) = 2

k ( 45, 2) = 1.248072e-02

q0( 45, 2) = 0.000000e+00

n ( 45, 2) = 2

k ( 46, 2) = 9.453230e-03

q0( 46, 2) = 0.000000e+00

n ( 46, 2) = 2

k ( 47, 2) = 3.242337e-04

q0( 47, 2) = 0.000000e+00

n ( 47, 2) = 2

k ( 48, 2) = 7.212680e-03

q0( 48, 2) = 0.000000e+00

n ( 48, 2) = 2

k ( 49, 2) = 9.569813e-03

q0( 49, 2) = 0.000000e+00

n ( 49, 2) = 2

k ( 50, 2) = 1.464638e-02

q0( 50, 2) = 0.000000e+00

n ( 50, 2) = 2

k ( 51, 2) = 1.847392e-02

q0( 51, 2) = 0.000000e+00

n ( 51, 2) = 2

k ( 52, 2) = 1.679505e-02

q0( 52, 2) = 0.000000e+00

n ( 52, 2) = 2

k ( 53, 2) = 3.251311e-03

q0( 53, 2) = 1.047198e+00

n ( 53, 2) = 3

k ( 54, 2) = 3.251267e-03

q0( 54, 2) = 1.047198e+00

n ( 54, 2) = 3

k ( 55, 2) = 3.720008e-03

q0( 55, 2) = 1.047198e+00

n ( 55, 2) = 3

k ( 56, 2) = 6.190699e-03

q0( 56, 2) = 0.000000e+00

n ( 56, 2) = 2

k ( 57, 2) = 8.298659e-03

q0( 57, 2) = 0.000000e+00

n ( 57, 2) = 2

k ( 58, 2) = -2.921170e-19

q0( 58, 2) = 0.000000e+00

n ( 58, 2) = 2

k ( 59, 2) = 9.439310e-03

q0( 59, 2) = 0.000000e+00

n ( 59, 2) = 2

k ( 60, 2) = 1.437363e-02

q0( 60, 2) = 0.000000e+00

n ( 60, 2) = 2

k ( 61, 2) = 1.364788e-02

q0( 61, 2) = 0.000000e+00

n ( 61, 2) = 2

k ( 62, 2) = 3.645499e-03

q0( 62, 2) = 0.000000e+00

n ( 62, 2) = 2

k ( 63, 2) = 2.821118e-03

q0( 63, 2) = 0.000000e+00

n ( 63, 2) = 2

k ( 64, 2) = 7.117884e-03

q0( 64, 2) = 0.000000e+00

n ( 64, 2) = 2

k ( 65, 2) = -5.278213e-20

q0( 65, 2) = 0.000000e+00

n ( 65, 2) = 2

k ( 66, 2) = 1.551261e-03

q0( 66, 2) = 0.000000e+00

n ( 66, 2) = 2

k ( 67, 2) = 7.582990e-03

q0( 67, 2) = 0.000000e+00

n ( 67, 2) = 2

k ( 68, 2) = 1.477277e-01

q0( 68, 2) = 0.000000e+00

n ( 68, 2) = 0

k ( 69, 2) = 1.866521e-01

q0( 69, 2) = 0.000000e+00

n ( 69, 2) = 0

k ( 70, 2) = 2.170837e-01

q0( 70, 2) = 0.000000e+00

n ( 70, 2) = 0

k ( 71, 2) = 9.500137e-02

q0( 71, 2) = 0.000000e+00

n ( 71, 2) = 0

k ( 72, 2) = 1.383067e-01

q0( 72, 2) = 0.000000e+00

n ( 72, 2) = 0

k ( 73, 2) = 1.087188e-01

q0( 73, 2) = 0.000000e+00

n ( 73, 2) = 0

!--- END generated by 'gen\_FFparam.py'

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 3, anchor point 3)

k ( 1, 3) = 2.632633e+00

q0( 1, 3) = 1.386000e+00

n ( 1, 3) = 0

k ( 2, 3) = 2.214016e+00

q0( 2, 3) = 1.413000e+00

n ( 2, 3) = 0

k ( 3, 3) = 1.510205e+00

q0( 3, 3) = 1.093000e+00

n ( 3, 3) = 0

k ( 4, 3) = 2.053144e+00

q0( 4, 3) = 1.437000e+00

n ( 4, 3) = 0

k ( 5, 3) = 1.517029e+00

q0( 5, 3) = 1.093000e+00

n ( 5, 3) = 0

k ( 6, 3) = 1.557611e+00

q0( 6, 3) = 1.089000e+00

n ( 6, 3) = 0

k ( 7, 3) = 1.557610e+00

q0( 7, 3) = 1.089000e+00

n ( 7, 3) = 0

k ( 8, 3) = 1.529096e+00

q0( 8, 3) = 1.088000e+00

n ( 8, 3) = 0

k ( 9, 3) = 2.155382e+00

q0( 9, 3) = 1.437000e+00

n ( 9, 3) = 0

k ( 10, 3) = 2.878127e+00

q0( 10, 3) = 1.703000e+00

n ( 10, 3) = 0

k ( 11, 3) = 2.586510e+00

q0( 11, 3) = 1.388000e+00

n ( 11, 3) = 0

k ( 12, 3) = 1.529718e+00

q0( 12, 3) = 1.091000e+00

n ( 12, 3) = 0

k ( 13, 3) = 2.227572e+00

q0( 13, 3) = 1.411000e+00

n ( 13, 3) = 0

k ( 14, 3) = 1.509522e+00

q0( 14, 3) = 1.093000e+00

n ( 14, 3) = 0

k ( 15, 3) = 1.518047e+00

q0( 15, 3) = 1.093000e+00

n ( 15, 3) = 0

k ( 16, 3) = 2.867249e-01

q0( 16, 3) = 2.113698e+00

n ( 16, 3) = 0

k ( 17, 3) = 1.710422e-01

q0( 17, 3) = 2.113698e+00

n ( 17, 3) = 0

k ( 18, 3) = 2.073553e-01

q0( 18, 3) = 2.099910e+00

n ( 18, 3) = 0

k ( 19, 3) = 1.752993e-01

q0( 19, 3) = 2.091777e+00

n ( 19, 3) = 0

k ( 20, 3) = 2.160439e-01

q0( 20, 3) = 2.094133e+00

n ( 20, 3) = 0

k ( 21, 3) = 1.501977e-01

q0( 21, 3) = 2.100102e+00

n ( 21, 3) = 0

k ( 22, 3) = 4.848717e-02

q0( 22, 3) = 2.052123e+00

n ( 22, 3) = 0

k ( 23, 3) = 2.425299e-01

q0( 23, 3) = 2.108742e+00

n ( 23, 3) = 0

k ( 24, 3) = 0.139860e+00 !<- modified to that determined by ab initio

q0( 24, 3) = 1.927786e+00

n ( 24, 3) = 0

k ( 25, 3) = 1.585421e-01

q0( 25, 3) = 2.056242e+00

n ( 25, 3) = 0

k ( 26, 3) = 2.382976e-01

q0( 26, 3) = 2.111639e+00

n ( 26, 3) = 0

k ( 27, 3) = 1.265356e-01

q0( 27, 3) = 2.058825e+00

n ( 27, 3) = 0

k ( 28, 3) = 2.976711e-01

q0( 28, 3) = 2.123350e+00

n ( 28, 3) = 0

k ( 29, 3) = 2.244366e-01

q0( 29, 3) = 2.096018e+00

n ( 29, 3) = 0

k ( 30, 3) = 1.526316e-01

q0( 30, 3) = 2.097659e+00

n ( 30, 3) = 0

k ( 31, 3) = 1.725318e-01

q0( 31, 3) = 2.113035e+00

n ( 31, 3) = 0

k ( 32, 3) = 1.714729e-01

q0( 32, 3) = 2.092039e+00

n ( 32, 3) = 0

k ( 33, 3) = 1.499944e-01

q0( 33, 3) = 2.089369e+00

n ( 33, 3) = 0

k ( 34, 3) = 1.491080e-01

q0( 34, 3) = 2.089735e+00

n ( 34, 3) = 0

k ( 35, 3) = 1.646620e-02

q0( 35, 3) = 2.099963e+00

n ( 35, 3) = 0

k ( 36, 3) = 9.248795e-02

q0( 36, 3) = 2.090486e+00

n ( 36, 3) = 0

k ( 37, 3) = 1.282602e-01

q0( 37, 3) = 1.656073e+00

n ( 37, 3) = 0

k ( 38, 3) = 9.248757e-02

q0( 38, 3) = 2.090486e+00

n ( 38, 3) = 0

k ( 39, 3) = 1.282598e-01

q0( 39, 3) = 1.656073e+00

n ( 39, 3) = 0

k ( 40, 3) = 1.282598e-01

q0( 40, 3) = 1.656073e+00

n ( 40, 3) = 0

k ( 41, 3) = 1.904720e-16

q0( 41, 3) = 0.000000e+00

n ( 41, 3) = 2

k ( 42, 3) = 7.617600e-02

q0( 42, 3) = 0.000000e+00

n ( 42, 3) = 2

k ( 43, 3) = 2.797215e-17

q0( 43, 3) = 0.000000e+00

n ( 43, 3) = 2

k ( 44, 3) = 1.430453e-02

q0( 44, 3) = 0.000000e+00

n ( 44, 3) = 2

k ( 45, 3) = 2.907223e-02

q0( 45, 3) = 0.000000e+00

n ( 45, 3) = 2

k ( 46, 3) = 6.191780e-02

q0( 46, 3) = 0.000000e+00

n ( 46, 3) = 2

k ( 47, 3) = -1.563365e-17

q0( 47, 3) = 0.000000e+00

n ( 47, 3) = 2

k ( 48, 3) = 7.617600e-02

q0( 48, 3) = 0.000000e+00

n ( 48, 3) = 2

k ( 49, 3) = 1.896072e-02

q0( 49, 3) = 0.000000e+00

n ( 49, 3) = 2

k ( 50, 3) = -4.746710e-17

q0( 50, 3) = 0.000000e+00

n ( 50, 3) = 2

k ( 51, 3) = 2.885312e-02

q0( 51, 3) = 0.000000e+00

n ( 51, 3) = 2

k ( 52, 3) = 5.852025e-17

q0( 52, 3) = 0.000000e+00

n ( 52, 3) = 2

k ( 53, 3) = 2.914115e-02

q0( 53, 3) = 1.047198e+00

n ( 53, 3) = 3

k ( 54, 3) = 2.914115e-02

q0( 54, 3) = 1.047198e+00

n ( 54, 3) = 3

k ( 55, 3) = 2.805188e-17

q0( 55, 3) = 1.047198e+00

n ( 55, 3) = 3

k ( 56, 3) = 7.617600e-02

q0( 56, 3) = 0.000000e+00

n ( 56, 3) = 2

k ( 57, 3) = 1.363430e-03

q0( 57, 3) = 0.000000e+00

n ( 57, 3) = 2

k ( 58, 3) = 4.243584e-17

q0( 58, 3) = 0.000000e+00

n ( 58, 3) = 2

k ( 59, 3) = 1.017938e-02

q0( 59, 3) = 0.000000e+00

n ( 59, 3) = 2

k ( 60, 3) = 9.139687e-17

q0( 60, 3) = 0.000000e+00

n ( 60, 3) = 2

k ( 61, 3) = 5.799873e-18

q0( 61, 3) = 0.000000e+00

n ( 61, 3) = 2

k ( 62, 3) = 6.645935e-03

q0( 62, 3) = 0.000000e+00

n ( 62, 3) = 2

k ( 63, 3) = 4.455693e-17

q0( 63, 3) = 0.000000e+00

n ( 63, 3) = 2

k ( 64, 3) = -1.003487e-16

q0( 64, 3) = 0.000000e+00

n ( 64, 3) = 2

k ( 65, 3) = -1.053017e-17

q0( 65, 3) = 0.000000e+00

n ( 65, 3) = 2

k ( 66, 3) = 5.604263e-02

q0( 66, 3) = 0.000000e+00

n ( 66, 3) = 2

k ( 67, 3) = 3.804124e-18

q0( 67, 3) = 0.000000e+00

n ( 67, 3) = 2

k ( 68, 3) = 3.921190e-01

q0( 68, 3) = 0.000000e+00

n ( 68, 3) = 0

k ( 69, 3) = 0.000000e+00

q0( 69, 3) = 0.000000e+00

n ( 69, 3) = 0

k ( 70, 3) = 0.000000e+00

q0( 70, 3) = 0.000000e+00

n ( 70, 3) = 0

k ( 71, 3) = 1.006875e-01

q0( 71, 3) = 0.000000e+00

n ( 71, 3) = 0

k ( 72, 3) = -0.000000e+00

q0( 72, 3) = 0.000000e+00

n ( 72, 3) = 0

k ( 73, 3) = 2.298139e-01

q0( 73, 3) = 0.000000e+00

n ( 73, 3) = 0

!--- END generated by 'gen\_FFparam.py'

!--- BEGIN generated by 'gen\_FFparam.py'

!--- (state 3, anchor point 4)

k ( 1, 4) = 2.561712e+00

q0( 1, 4) = 1.393000e+00

n ( 1, 4) = 0

k ( 2, 4) = 2.519063e+00

q0( 2, 4) = 1.395000e+00

n ( 2, 4) = 0

k ( 3, 4) = 1.548283e+00

q0( 3, 4) = 1.090000e+00

n ( 3, 4) = 0

k ( 4, 4) = 2.328953e+00

q0( 4, 4) = 1.401000e+00

n ( 4, 4) = 0

k ( 5, 4) = 1.541133e+00

q0( 5, 4) = 1.090000e+00

n ( 5, 4) = 0

k ( 6, 4) = 1.545522e+00

q0( 6, 4) = 1.086000e+00

n ( 6, 4) = 0

k ( 7, 4) = 1.545476e+00

q0( 7, 4) = 1.086000e+00

n ( 7, 4) = 0

k ( 8, 4) = 1.545476e+00

q0( 8, 4) = 1.086000e+00

n ( 8, 4) = 0

k ( 9, 4) = 2.330569e+00

q0( 9, 4) = 1.401000e+00

n ( 9, 4) = 0

k ( 10, 4) = 2.177974e+00

q0( 10, 4) = 1.755000e+00

n ( 10, 4) = 0

k ( 11, 4) = 2.559662e+00

q0( 11, 4) = 1.393000e+00

n ( 11, 4) = 0

k ( 12, 4) = 1.541109e+00

q0( 12, 4) = 1.090000e+00

n ( 12, 4) = 0

k ( 13, 4) = 2.521272e+00

q0( 13, 4) = 1.395000e+00

n ( 13, 4) = 0

k ( 14, 4) = 1.548297e+00

q0( 14, 4) = 1.090000e+00

n ( 14, 4) = 0

k ( 15, 4) = 1.556223e+00

q0( 15, 4) = 1.089000e+00

n ( 15, 4) = 0

k ( 16, 4) = 2.589180e-01

q0( 16, 4) = 2.088583e+00

n ( 16, 4) = 0

k ( 17, 4) = 1.806598e-01

q0( 17, 4) = 2.098881e+00

n ( 17, 4) = 0

k ( 18, 4) = 2.458881e-01

q0( 18, 4) = 2.083190e+00

n ( 18, 4) = 0

k ( 19, 4) = 1.715335e-01

q0( 19, 4) = 2.100137e+00

n ( 19, 4) = 0

k ( 20, 4) = 2.104061e-01

q0( 20, 4) = 2.106752e+00

n ( 20, 4) = 0

k ( 21, 4) = 1.600991e-01

q0( 21, 4) = 2.080310e+00

n ( 21, 4) = 0

k ( 22, 4) = 1.298786e-01

q0( 22, 4) = 2.093470e+00

n ( 22, 4) = 0

k ( 23, 4) = 2.318169e-01

q0( 23, 4) = 2.095338e+00

n ( 23, 4) = 0

k ( 24, 4) = 1.464541e-01

q0( 24, 4) = 2.096140e+00

n ( 24, 4) = 0

k ( 25, 4) = 2.589207e-01

q0( 25, 4) = 2.088653e+00

n ( 25, 4) = 0

k ( 26, 4) = 1.464529e-01

q0( 26, 4) = 2.096228e+00

n ( 26, 4) = 0

k ( 27, 4) = 2.317472e-01

q0( 27, 4) = 2.095320e+00

n ( 27, 4) = 0

k ( 28, 4) = 2.103737e-01

q0( 28, 4) = 2.106700e+00

n ( 28, 4) = 0

k ( 29, 4) = 1.600774e-01

q0( 29, 4) = 2.080171e+00

n ( 29, 4) = 0

k ( 30, 4) = 1.806697e-01

q0( 30, 4) = 2.098776e+00

n ( 30, 4) = 0

k ( 31, 4) = 1.715727e-01

q0( 31, 4) = 2.100294e+00

n ( 31, 4) = 0

k ( 32, 4) = 1.672383e-01

q0( 32, 4) = 2.096577e+00

n ( 32, 4) = 0

k ( 33, 4) = 1.672661e-01

q0( 33, 4) = 2.096716e+00

n ( 33, 4) = 0

k ( 34, 4) = 1.035906e-01

q0( 34, 4) = 2.094517e+00

n ( 34, 4) = 0

k ( 35, 4) = 1.035906e-01

q0( 35, 4) = 2.094517e+00

n ( 35, 4) = 0

k ( 36, 4) = 1.035906e-01

q0( 36, 4) = 2.094517e+00

n ( 36, 4) = 0

k ( 37, 4) = 7.344867e-03

q0( 37, 4) = 0.000000e+00

n ( 37, 4) = 2

k ( 38, 4) = 1.150083e-02

q0( 38, 4) = 0.000000e+00

n ( 38, 4) = 2

k ( 39, 4) = 1.277474e-02

q0( 39, 4) = 0.000000e+00

n ( 39, 4) = 2

k ( 40, 4) = 1.242807e-02

q0( 40, 4) = 0.000000e+00

n ( 40, 4) = 2

k ( 41, 4) = 1.136981e-02

q0( 41, 4) = 0.000000e+00

n ( 41, 4) = 2

k ( 42, 4) = 1.169830e-02

q0( 42, 4) = 0.000000e+00

n ( 42, 4) = 2

k ( 43, 4) = 8.753866e-03

q0( 43, 4) = 0.000000e+00

n ( 43, 4) = 2

k ( 44, 4) = 9.481992e-03

q0( 44, 4) = 0.000000e+00

n ( 44, 4) = 2

k ( 45, 4) = 1.330871e-02

q0( 45, 4) = 0.000000e+00

n ( 45, 4) = 2

k ( 46, 4) = 1.173337e-02

q0( 46, 4) = 0.000000e+00

n ( 46, 4) = 2

k ( 47, 4) = 1.190960e-02

q0( 47, 4) = 0.000000e+00

n ( 47, 4) = 2

k ( 48, 4) = 1.171697e-02

q0( 48, 4) = 0.000000e+00

n ( 48, 4) = 2

k ( 49, 4) = 9.468767e-03

q0( 49, 4) = 0.000000e+00

n ( 49, 4) = 2

k ( 50, 4) = 1.171277e-02

q0( 50, 4) = 0.000000e+00

n ( 50, 4) = 2

k ( 51, 4) = 1.154456e-02

q0( 51, 4) = 0.000000e+00

n ( 51, 4) = 2

k ( 52, 4) = 1.241307e-02

q0( 52, 4) = 0.000000e+00

n ( 52, 4) = 2

k ( 53, 4) = 1.274807e-02

q0( 53, 4) = 0.000000e+00

n ( 53, 4) = 2

k ( 54, 4) = 1.273423e-02

q0( 54, 4) = 0.000000e+00

n ( 54, 4) = 2

k ( 55, 4) = 5.355534e-03

q0( 55, 4) = 0.000000e+00

n ( 55, 4) = 2

k ( 56, 4) = 5.358725e-03

q0( 56, 4) = 0.000000e+00

n ( 56, 4) = 2

k ( 57, 4) = 5.148179e-03

q0( 57, 4) = 0.000000e+00

n ( 57, 4) = 2

k ( 58, 4) = 3.337459e-03

q0( 58, 4) = 0.000000e+00

n ( 58, 4) = 2

k ( 59, 4) = 3.356269e-03

q0( 59, 4) = 0.000000e+00

n ( 59, 4) = 2

k ( 60, 4) = 5.141157e-03

q0( 60, 4) = 0.000000e+00

n ( 60, 4) = 2

k ( 61, 4) = 1.181836e-01

q0( 61, 4) = 0.000000e+00

n ( 61, 4) = 0

k ( 62, 4) = 9.210745e-02

q0( 62, 4) = 0.000000e+00

n ( 62, 4) = 0

k ( 63, 4) = 1.953495e-01

q0( 63, 4) = 0.000000e+00

n ( 63, 4) = 0

k ( 64, 4) = 1.192638e-01

q0( 64, 4) = 0.000000e+00

n ( 64, 4) = 0

k ( 65, 4) = 1.180206e-01

q0( 65, 4) = 0.000000e+00

n ( 65, 4) = 0

k ( 66, 4) = 1.193255e-01

q0( 66, 4) = 0.000000e+00

n ( 66, 4) = 0

k ( 67, 4) = 4.859191e-02

q0( 67, 4) = 0.000000e+00

n ( 67, 4) = 0

!--- END generated by 'gen\_FFparam.py'

end select

end subroutine assignparamuii\_t

!=================================================================

! \*def assignparamuij\_t

! Assign param. for tert. couplings

! Input:

! nqtc: number of tert. coord. for coupling

! nk: number of parameters for each anchor point

! nR0: number of R-type anchor points

! nphi0: number of phi-type anchor points

! istate: the 'ij' value of uij of interest

! Output:

! k: force coefficients

! qtc0: rest values

!

! ~400 lines of params

!=================================================================

!\*\*\*

subroutine assignparamuij\_t(k, qtc0, nqtc, nk, nR0, nphi0, istate)

implicit none

integer :: nqtc, nk, nR0, nphi0, istate

double precision :: k(nk,nR0,nphi0,nqtc), qtc0(nqtc)

!--- rest values

!--- BEGIN generated by "python gen\_uijparam.py u12-t\_param.txt 2"

qtc0( 1) = 0.000000e+00

qtc0( 2) = 0.000000e+00

qtc0( 3) = 0.000000e+00

qtc0( 4) = 0.000000e+00

qtc0( 5) = 1.007440e+00

qtc0( 6) = -1.055167e+00

qtc0( 7) = -5.647235e+00

qtc0( 8) = -4.539317e+00

qtc0( 9) = 0.000000e+00

qtc0(10) = 1.028800e+02

qtc0(11) = 0.000000e+00

!--- END generated by "python gen\_uijparam.py u12-t\_param.txt 2"

!--- ver8.0: anchor points at phi = 0 and 90 deg removed.

!--- New iphi0 (3rd dim of k) 1,2,3 corresp. to

!--- old iphi0 2,3,4 in uij-t\_param.txt (ij = 12,13,23)

!--- ver8.1b: anchor points at phi = 0 and 90 deg added back.

!--- i.e. Params same as ver7.2. BUT the total force field

!--- for phi = 0 and 90 deg are set to zero

!--- no matter what the params are.

select case(istate)

case(12)

!--- k(K1orK2, iR0, iphi0, iAP)

!--- BEGIN generated by "python gen\_uijparam.py u12-t\_param.txt 1"

k(:, 1, 1, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 1) = (/-3.761870e-05, 0.000000e+00/)

k(:, 1, 3, 1) = (/-1.356040e-04,-2.289310e-06/)

k(:, 1, 4, 1) = (/ 2.566430e-04, 0.000000e+00/)

k(:, 1, 5, 1) = (/ 2.566430e-04, 0.000000e+00/)

k(:, 2, 1, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 2) = (/ 0.000000e+00,-2.506840e-08/)

k(:, 1, 2, 2) = (/-8.741780e-06,-2.506840e-07/)

k(:, 1, 3, 2) = (/-2.871230e-05, 9.292700e-11/)

k(:, 1, 4, 2) = (/-3.277990e-05,-1.629990e-06/)

k(:, 1, 5, 2) = (/ 0.000000e+00,-1.629990e-06/)

k(:, 2, 1, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 3) = (/ 0.000000e+00, 1.621010e-08/)

k(:, 1, 2, 3) = (/ 4.570820e-07, 1.621010e-07/)

k(:, 1, 3, 3) = (/ 1.298620e-05, 8.125900e-07/)

k(:, 1, 4, 3) = (/-6.459140e-05,-1.069700e-06/)

k(:, 1, 5, 3) = (/ 0.000000e+00,-1.069700e-06/)

k(:, 2, 1, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 4) = (/ 0.000000e+00, 5.321210e-08/)

k(:, 1, 2, 4) = (/-2.436850e-06, 5.321210e-07/)

k(:, 1, 3, 4) = (/-3.571280e-05, 1.220330e-06/)

k(:, 1, 4, 4) = (/-2.547150e-05, 3.035980e-06/)

k(:, 1, 5, 4) = (/-2.547150e-05, 3.035980e-06/)

k(:, 2, 1, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 5) = (/-4.219240e-06, 8.579970e-08/)

k(:, 1, 2, 5) = (/-4.219240e-05, 8.579970e-07/)

k(:, 1, 3, 5) = (/ 8.694530e-05, 1.328920e-07/)

k(:, 1, 4, 5) = (/ 2.021970e-04,-3.878100e-06/)

k(:, 1, 5, 5) = (/ 2.021970e-04,-3.878100e-06/)

k(:, 2, 1, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 6) = (/-1.625200e-07,-4.013710e-08/)

k(:, 1, 2, 6) = (/-1.625200e-06,-4.013710e-07/)

k(:, 1, 3, 6) = (/ 1.111190e-04,-1.825150e-06/)

k(:, 1, 4, 6) = (/ 1.983250e-04, 2.991040e-06/)

k(:, 1, 5, 6) = (/ 1.983250e-04, 2.991040e-06/)

k(:, 2, 1, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 7) = (/-1.269050e-05, 1.795710e-08/)

k(:, 1, 2, 7) = (/-1.269050e-04, 1.795710e-07/)

k(:, 1, 3, 7) = (/-1.568250e-04, 1.466680e-06/)

k(:, 1, 4, 7) = (/-3.966160e-05, 5.889630e-06/)

k(:, 1, 5, 7) = (/ 0.000000e+00, 5.889630e-06/)

k(:, 2, 1, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 8) = (/ 1.677950e-06,-1.389780e-08/)

k(:, 1, 2, 8) = (/ 1.677950e-05,-1.389780e-07/)

k(:, 1, 3, 8) = (/ 1.238040e-06, 8.786340e-07/)

k(:, 1, 4, 8) = (/-3.946220e-06,-1.404240e-07/)

k(:, 1, 5, 8) = (/-3.946220e-06,-1.404240e-07/)

k(:, 2, 1, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 3, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 4, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 5, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1,10) = (/ 1.730000e-04, 6.490000e-04/)

k(:, 1, 2,10) = (/ 1.730000e-03, 6.490000e-03/)

k(:, 1, 3,10) = (/ 3.920000e-03,-7.108160e-04/)

k(:, 1, 4,10) = (/ 5.549190e-04, 2.400000e-03/)

k(:, 1, 5,10) = (/ 5.549190e-04, 2.400000e-03/)

k(:, 2, 1,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 3,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 4,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 5,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5,11) = (/ 0.000000e+00, 0.000000e+00/)

!--- END generated by "python gen\_uijparam.py u12-t\_param.txt 1"

case(13)

!--- BEGIN generated by "python gen\_uijparam.py u13-t\_param.txt 1"

k(:, 1, 1, 1) = (/ 1.108530e-05, 0.000000e+00/)

k(:, 1, 2, 1) = (/ 1.108530e-04, 5.132520e-06/)

k(:, 1, 3, 1) = (/ 1.043710e-04, 1.929240e-05/)

k(:, 1, 4, 1) = (/ 1.318670e-04,-2.090110e-06/)

k(:, 1, 5, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1, 1) = (/-6.587610e-07, 0.000000e+00/)

k(:, 2, 2, 1) = (/-6.587610e-06,-2.093880e-07/)

k(:, 2, 3, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 2) = (/ 6.782220e-05, 0.000000e+00/)

k(:, 1, 2, 2) = (/ 6.782220e-04, 1.740420e-06/)

k(:, 1, 3, 2) = (/ 3.819440e-04, 7.941420e-06/)

k(:, 1, 4, 2) = (/-2.734530e-04, 9.269190e-06/)

k(:, 1, 5, 2) = (/-2.734530e-04, 0.000000e+00/)

k(:, 2, 1, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 3) = (/-1.873890e-06, 0.000000e+00/)

k(:, 1, 2, 3) = (/-1.873890e-05, 1.607880e-07/)

k(:, 1, 3, 3) = (/-3.970050e-05, 9.590160e-07/)

k(:, 1, 4, 3) = (/-2.175030e-06, 1.002970e-06/)

k(:, 1, 5, 3) = (/-2.175030e-06, 0.000000e+00/)

k(:, 2, 1, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 4) = (/ 1.110960e-05, 0.000000e+00/)

k(:, 1, 2, 4) = (/ 1.110960e-04, 1.227640e-07/)

k(:, 1, 3, 4) = (/ 4.603570e-05, 1.433790e-06/)

k(:, 1, 4, 4) = (/-7.701760e-05, 2.118810e-06/)

k(:, 1, 5, 4) = (/-7.701760e-05, 2.118810e-06/)

k(:, 2, 1, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 5) = (/-2.006590e-05, 2.537640e-07/)

k(:, 1, 3, 5) = (/-3.765110e-05, 3.973740e-07/)

k(:, 1, 4, 5) = (/-1.512450e-05, 1.334030e-06/)

k(:, 1, 5, 5) = (/-1.512450e-05, 1.334030e-06/)

k(:, 2, 1, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 6) = (/ 6.109150e-06, 4.133380e-08/)

k(:, 1, 3, 6) = (/-2.168970e-07, 8.695480e-07/)

k(:, 1, 4, 6) = (/ 6.900410e-05, 2.749200e-06/)

k(:, 1, 5, 6) = (/ 6.900410e-05, 2.749200e-06/)

k(:, 2, 1, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 7) = (/-2.082470e-05, 7.975570e-07/)

k(:, 1, 3, 7) = (/-2.122660e-05, 2.897200e-06/)

k(:, 1, 4, 7) = (/-2.704370e-06, 7.528350e-06/)

k(:, 1, 5, 7) = (/-2.704370e-06, 0.000000e+00/)

k(:, 2, 1, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 8) = (/-4.555340e-05, 5.356330e-07/)

k(:, 1, 3, 8) = (/-4.013790e-05, 3.255770e-06/)

k(:, 1, 4, 8) = (/ 9.978290e-05, 2.754410e-06/)

k(:, 1, 5, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 3, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 4, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 5, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1, 9) = (/-7.491630e-06, 0.000000e+00/)

k(:, 2, 2, 9) = (/-7.491630e-05,-4.316920e-08/)

k(:, 2, 3, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2,10) = (/ 5.130000e-03, 1.485000e-02/)

k(:, 1, 3,10) = (/ 1.480000e-03, 4.802000e-02/)

k(:, 1, 4,10) = (/-2.045000e-02, 2.725000e-02/)

k(:, 1, 5,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 3,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 4,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 5,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1,11) = (/-5.007750e-06, 0.000000e+00/)

k(:, 2, 2,11) = (/-5.007750e-05,-1.521600e-07/)

k(:, 2, 3,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5,11) = (/ 0.000000e+00, 0.000000e+00/)

!--- END generated by "python gen\_uijparam.py u13-t\_param.txt 1"

case(23)

!--- BEGIN generated by "python gen\_uijparam.py u23-t\_param.txt 1"

k(:, 1, 1, 1) = (/ 3.080780e-05, 0.000000e+00/)

k(:, 1, 2, 1) = (/ 3.080780e-04, 3.101950e-06/)

k(:, 1, 3, 1) = (/-1.744460e-05, 3.392200e-06/)

k(:, 1, 4, 1) = (/-1.113940e-04,-1.402690e-06/)

k(:, 1, 5, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 1) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 2) = (/ 4.534340e-06, 0.000000e+00/)

k(:, 1, 2, 2) = (/ 4.534340e-05, 9.405500e-08/)

k(:, 1, 3, 2) = (/-4.708560e-05, 4.713280e-07/)

k(:, 1, 4, 2) = (/-7.667080e-05,-9.933570e-07/)

k(:, 1, 5, 2) = (/-7.667080e-05, 0.000000e+00/)

k(:, 2, 1, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 2) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 3) = (/-1.501560e-05, 0.000000e+00/)

k(:, 1, 2, 3) = (/-1.501560e-04, 1.661390e-07/)

k(:, 1, 3, 3) = (/-6.065620e-05, 8.299980e-07/)

k(:, 1, 4, 3) = (/ 1.482230e-05,-2.747200e-07/)

k(:, 1, 5, 3) = (/ 1.482230e-05, 0.000000e+00/)

k(:, 2, 1, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 3) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 4) = (/ 1.403630e-05, 0.000000e+00/)

k(:, 1, 2, 4) = (/ 1.403630e-04, 1.939340e-08/)

k(:, 1, 3, 4) = (/ 1.966730e-05, 1.755230e-08/)

k(:, 1, 4, 4) = (/-1.062540e-05,-4.497190e-07/)

k(:, 1, 5, 4) = (/-1.062540e-05,-4.497190e-07/)

k(:, 2, 1, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 4) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 5) = (/-2.618850e-05, 1.174180e-07/)

k(:, 1, 3, 5) = (/-4.449210e-05, 4.052410e-07/)

k(:, 1, 4, 5) = (/ 2.124070e-04,-4.271490e-06/)

k(:, 1, 5, 5) = (/ 2.124070e-04,-4.271490e-06/)

k(:, 2, 1, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 5) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 6) = (/-5.736600e-05,-2.828080e-07/)

k(:, 1, 3, 6) = (/-1.353800e-04,-2.794580e-06/)

k(:, 1, 4, 6) = (/ 2.268140e-04, 3.924680e-06/)

k(:, 1, 5, 6) = (/ 2.268140e-04, 3.924680e-06/)

k(:, 2, 1, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 6) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 7) = (/ 1.303790e-05,-4.460330e-07/)

k(:, 1, 3, 7) = (/ 8.400290e-05, 1.651570e-07/)

k(:, 1, 4, 7) = (/-1.930350e-04,-3.966430e-08/)

k(:, 1, 5, 7) = (/-1.930350e-04, 0.000000e+00/)

k(:, 2, 1, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 7) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 8) = (/ 2.200940e-05, 9.830790e-08/)

k(:, 1, 3, 8) = (/ 4.424690e-05, 1.128610e-06/)

k(:, 1, 4, 8) = (/ 1.146040e-05,-1.061750e-07/)

k(:, 1, 5, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 8) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 3, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 4, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 5, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5, 9) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2,10) = (/-1.080000e-03, 7.000000e-03/)

k(:, 1, 3,10) = (/-2.830000e-03, 9.010000e-03/)

k(:, 1, 4,10) = (/ 1.330000e-03, 4.556870e-04/)

k(:, 1, 5,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5,10) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 1,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 2,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 3,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 4,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 1, 5,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 1,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 2,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 3,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 4,11) = (/ 0.000000e+00, 0.000000e+00/)

k(:, 2, 5,11) = (/ 0.000000e+00, 0.000000e+00/)

!--- END generated by "python gen\_uijparam.py u23-t\_param.txt 1"

end select

end subroutine assignparamuij\_t

end subroutine pot

!=================================================================

! \*def preopt

! Dummy subroutine for interfacing with ANT.

!=================================================================

subroutine prepot

implicit none

return

end subroutine prepot