```fortran



   module tbbop_emb

   use mod_precision
   use mod_const
   use mod_conf
   use mod_clock
   use topologia

#ifdef MPI
   use mpi
#endif

   implicit none

   integer, parameter :: nr = 2  ! number of regions (only TB and BOP so far)


   type mbtb_t
      logical :: ovl        ! overlap switch (not implemented yet)
      integer :: rlst(nr+1) ! atom offsets for regions
      integer :: n(nr+1)    ! number of atoms according to region
      integer :: nd         ! total number of atoms
      integer :: thsz, bhsz, ahsz ! sizes for TB, BOP and global Hamiltonians respectively
!                          posham,    aptr_tb    bptr_tb    aptr_dh   bptr_dh
      integer, allocatable :: pth(:), pbh(:), pah(:) ! atom offsets for Hamiltonians
      integer, allocatable :: ad(:,:)   ! coordinates of dynamic atoms
      integer, allocatable :: apt(:), bpt(:) ! neighbour lists for TB -> TB
      integer, allocatable :: apd(:), bpd(:) ! neighbour lists for TB -> BOP
      integer, allocatable :: apb(:), bpb(:) ! neighbour lists for BOP -> BOP

      real(dp), allocatable :: h(:,:), s(:,:) ! TB hamiltonian and overlap matrices in dense form
      real(dp), allocatable :: dh(:,:,:)      ! TB->BOP matrix elements in sparse form
      real(dp), allocatable :: q(:), rho(:,:,:,:) ! global atomic charges and density matrix in sparse
      form
      !, v(:), ! rho(mxnstat,mxnstat,jb,ia)

      complex(dp), allocatable :: a(:,:), ac(:,:), gt(:,:) !In short:
!       a = G_BOP dh'
!       gt = (z - (h + dh a))^{-1}
!       ac = a gt
! see the notes in 'formuli.pdf' pages: 4-6 for details.

      complex(dp), allocatable :: ggd(:) ! global Green matrix diagonal elements
      !, w(:,:), occ(:), gg(:,:)
   end type


   contains

   subroutine emb_sc()
!    self consistency loop.
!    This is the entry point for the embedding program.

      use mod_all_scalar, only : mag, quiet, dqmax, forces, mgmax, nd, qerr
      use mod_atom_ar, only : dq, mg

      include "../Include/Atom.array"


      type(mbtb_t) :: mbc ! embedding conf

      integer :: it, mxit, ia, mgit, mit
      real(dp) :: etott
      real(dp) :: avdde, sumz, merr
      real(qp), allocatable :: de_prev(:), dq_prev(:), dec(:),  td(:), dql(:)
      real(dp), allocatable :: mg_o(:)

      character(len=25) :: ffmt

      integer,parameter :: mixing_type = 2

      integer(8) :: c1,c2

      integer :: i
      real(dp) :: ef,nf

      integer :: ib, ierr
      real(dp) :: dqmax_old, aa, aamin, aamax

      ef = 0.0_dp
      !       mxit = 500
      mxit = rtc % bop_conf % mxit
      if (mag) then
        mgit = rtc % bop_conf % mgit
        merr = rtc % bop_conf % merr
        mit = 1
        if (merr < 0.0_dp) mgit = 1
      end if

      allocate(de_prev(nd), dq_prev(nd), td(nd), dec(nd), dql(nd))
      dec = real(de(:nd),qp)
      sumz = sum( (/ (real(zc(z(ia)), qp), ia = 1,nd) /) )

!
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++
! This is temporary while writing the code only! should certainly be changed to load it from the cell
file.

!       mbc % rlst(1) = 1
!       mbc % rlst(2) = 21
!       mbc % rlst(3) = 41
!
       mbc % rlst(1) = 1
       mbc % rlst(2) = 220
       mbc % rlst(3) = 271

      mbc % ovl = .false.
! The cell file will specify the numbers nt, nb, and nd then list all atoms ordered as follows:
! first the bop atoms then the tb atoms and afterwards the pure bop atoms. Pure bop atoms are not
consdered yet
! but should be kept in mind
!
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++

      call mbldh(mbc)

      if ((qerr < 0.0_dp).or.(mixing_type == 0)) mxit = 1
      if (.not. quiet) then
         write(9,"('Imposing LCN ...')")
         write(ffmt,"(a,i0,a)") '(a,":",', mbc%nd, '(x,f22.14))'
      end if

      if (mag) allocate(mg_o(mbc%nd))

      do
         if (mag) then
            if (.not. quiet) write(6,'(/,"mit:",i3)') mit
            mg_o = mg

            do ia = 1, mbc%nd
               dem(ia) = 0.5_dp*istn(z(ia))*mg(ia)
            end do

         end if

!           if linmix
         dqmax_old = 0.0_dp
         aamax=1.0_dp
         aamin=1.0e-4_dp
         aa=10.0_dp
!           endif

         it = 1
         do
      !           print "('sumzde,de:',2(x,f22.14))", sum( (/ (de(ia)*zc(z(ia)), ia = 1,nd) /) ), de(9)
            de(:mbc%nd) = real(dec - sum( (/ (dec(ia)*zc(z(ia)), ia = 1,mbc%nd) /) ) / sumz, dp)
      !           print "('bde:',x,f22.14)", de(9)
            if (.not. quiet) then
               write(6,'(/,"it:",i3)') it
               write(6, ffmt) 'de ', de(:mbc%nd)
               write(6, ffmt) 'dem', dem(:mbc%nd)
            end if

            call mbeq(mbc)

      !                      dec(:nd) = de(:nd)
            dql = real(dq,qp)


            print *, redb, 'dqmax:', dqmax, endc
            if (abs(dqmax) <= qerr .or. it == mxit) exit

!             if (it > 1) then
!                do ia = 1, nd
!       !     if (.not. quiet) print *, 'dif:',   (dec(ia) - de_prev(ia))/ (dq(ia) - dq_prev(ia) ), abs(dq(ia) - dq_prev(ia))
!                   if ( abs(dec(ia) - de_prev(ia)) > 6.0_qp*abs(dql(ia) - dq_prev(ia))) then
!                      td(ia) = -dql(ia)
!       !    if (.not. quiet) print *,red,'potential daner', (dec(ia) - de_prev(ia)), (dq(ia) - dq_prev(ia)),endc
!                   else
!                      td(ia) = dql(ia) * (dec(ia) - de_prev(ia)) / ( dql(ia) - dq_prev(ia) )
!                   endif
!                end do
!                de_prev = dec
!                dq_prev = dql
!                dec = dec - td
!       !               write(6,"('td,nde:',2(x,f22.14))") td(9),de(9)
!             else
!                de_prev = dec
!                dq_prev = dql
!                dec = dec + dql
!             end if

            if (     (dqmax_old*dqmax > 0.0_dp &
            &  .and. abs(dqmax_old) > abs(dqmax)) &
            &  .or. (dqmax_old*dqmax < 0.0_dp)) then
               aa=aa*abs(dqmax_old)/abs(dqmax_old-dqmax)
            end if

            aa=max(aamin,aa)
            aa=min(aamax,aa)

            if (abs(dqmax) <= qerr) aa=aamin


            print *, redb, 'aa:', aa, endc
            dec(:mbc%nd) = de(:mbc%nd) + aa*dq(:mbc%nd)


            dqmax_old = dqmax

            it = it + 1
         end do

         if (.not. mag) exit
         mgmax = maxval(abs(mg - mg_o))

         if (.not. quiet) then
            write(6,ffmt) 'mgo', mg_o
            write(6,ffmt) 'mg ', mg
            write(6,*) 'mgmax:', mgmax
         end if
         if (mgmax <= merr .or. mit == mgit) exit
         mit = mit + 1
      end do
```

```fortran

      if (mag) deallocate(mg_o)
      deallocate(de_prev, dq_prev, td, dec, dql)


      if (forces) call mbf(mbc)

   end subroutine emb_sc



   subroutine mbf(mbc)
      use mod_all_scalar, only : mag, quiet, lef, locc, dqmax, nbase
      type(mbtb_t), intent(inout) :: mbc

      call mbrho(mbc, lef)



   end subroutine mbf




   subroutine gg_diag(mbc, ez, isp)

      type(mbtb_t), intent(inout) :: mbc
      complex(dp), intent(in) :: ez
      integer, intent(in) :: isp

      integer :: i, j


      call a_and_ac(mbc, ez, isp)


!        write( 389, *) 'ez', ez

!        call print_c(mbc % ac, 389, 'ac'//achar(10))

!        call print_c(mbc % h, 389, 'h')


!The BOP G is already added, here we just add the mixed term ACA*

      do i = 1, mbc % bhsz
         do j = 1, mbc % thsz
            mbc % ggd(i) = mbc % ggd(i) + mbc % ac(i,j)* conjg(mbc % a(i,j))
         end do
      end do

      do i = 1, mbc % thsz
         mbc % ggd(mbc % bhsz + i) = mbc % gt(i,i)
      end do



!        calc the diagonal here; do over atoms and orbs  gb(ia,ia) +-  sum_ib(ac(ia,ib) * a(ia,ib))

!        stop
   end subroutine gg_diag



   subroutine gg_rho(mbc, ez, zp, isp)
      use ab_io
      use mod_ham

      type(mbtb_t), intent(inout) :: mbc
      complex(dp), intent(in) :: ez, zp
      integer, intent(in) :: isp

      integer :: ia, gia, ib, gib, it, git, ja, jb, jt, i, j, jt0

      complex(dp) :: gab(mxnstat, mxnstat)


      call a_and_ac(mbc, ez, isp)

      do ia = 1, mbc % n(1)

         call assoc_ab(ia,isp)
         call assoc_ham(ia) ! for cluster, pcluster and decipher

         gia = mbc % rlst(1) -1 + ia

!           BOP onsites
         gab = 0.0_dp
         call get_gab(ez, ia, ia, 1, gab)

         do i = 1, mbc % pbh(ia+1) - mbc % pbh(ia)
            do j = 1, mbc % pbh(ia+1) - mbc % pbh(ia)
               gab(i,j) = gab(i,j) + sum(mbc % ac(mbc % pbh(ia) + i, :) * conjg(mbc % a(mbc % pbh(ia)
               + j, :)))
            end do
         end do

         mbc % rho(:,:,1,gia) = mbc % rho(:,:,1,gia) + real(zp*gab)

!           BOP intersites

         ja = mbc % apb(ia)
         ib = mbc % bpb(ja)

         do while (ib /= eol)
            jb = decipher(ib)

            if (jb /= 0) then !jb == 0 => ib == ia
               gib = mbc%rlst(1)-1 + ib !

               gab = 0.0_dp
               call get_gab(ez, ia, ib, jb, gab)

               do i = 1, mbc % pbh(ia+1) - mbc % pbh(ia)
                  do j = 1, mbc % pbh(ib+1) - mbc % pbh(ib)
                     gab(i,j) = gab(i,j) + sum(mbc % ac(mbc % pbh(ia) + i, :) * conjg(mbc % a(mbc %
                     pbh(ib) + j, :)))
                  end do
               end do

               mbc % rho(:,:,jb,gia) = mbc % rho(:,:,jb,gia) + real(zp*gab)

            end if
            ja = ja + 1
            ib = mbc % bpb(ja)
         end do

      end do



      do it = 1, mbc % n(2)

         git = mbc % rlst(2) -1 + it

!           TB all

         jt = mbc % apt(it)
         jt0 = jt - 1
         ib = mbc % bpt(jt)

         do while (ib /= eol)

            gab = 0.0_dp
            gab(1 : mbc % pth (it + 1) - mbc % pth(it), 1 : mbc % pth (ib + 1) - mbc % pth(ib)) =  &
               & mbc % gt(mbc % pth(it) + 1 : mbc % pth(it + 1), mbc % pbh(ib) + 1 : mbc % pbh(ib +
               1))

            mbc % rho(:,:,jt-jt0,it) = mbc % rho(:,:,jt-jt0,it) + real(zp*gab)

            jt = jt + 1
            ib = mbc % bpt(jt)
         end do


!          TB->BOP intersites
         jt = mbc % apd(it)
         jt0 = jt - 1
         ib = mbc % bpd(ja)

         do while (ib /= eol)
            gib = mbc%rlst(2)-1 + ib

            gab = 0.0_dp
! this is incomplete:
!             gab(1 : mbc % pth (it + 1) - mbc % pth(it), 1 : mbc % pbh (ib + 1) - mbc % pbh(ib)) =  &
!              & - conjg(transpose(mbc % pbh(ib) + 1 : mbc % pbh(ib + 1)), mbc % ac(mbc % pth(it) + 1 : mbc % pth(it + 1) ))

            mbc % rho(:,:,jt-jt0,it) = mbc % rho(:,:,jt-jt0,it) + real(zp*gab)

            jt = jt + 1
            ib = mbc % bpt(jt)
         end do

!TODO:  complete the transpose BOP->TB elements .... or may be put it in the previous loop over ia

      end do



   end subroutine gg_rho




!    subroutine gtb(mbc, ez, isp)
!       use mod_all_scalar, only : mag
!
!
!       type(mbtb_t), intent(inout) :: mbc
!       complex(dp), intent(in) :: ez
!       integer, intent(in) :: isp
!
!       integer :: nstt1, nstt2, nsttb, l1, l2, lb, jt, it1, it2, ib, pt1, pt2, pb, it, git, lt
!       integer :: i, off
!
!       real(dp) :: dea
!       complex, parameter :: z_one = 1.0_dp, z_zero = 0.0_dp
!
!
!       integer :: info, lwork
!       real(dp), allocatable :: rwork(:)
!       complex(dp), allocatable :: work(:)
!
!       include '../Include/Atom.array'
!
!       ! (z*I*S - (H + dH'*Gb*dH))^-1 = W (z*I - V)^-1 W'
!
!       mbc % w = mbc % h
! !       mbc % w = 0.0_dp
!
!       off = mbc % rlst(2)-1
!
!       do it = 1, mbc % n(2)
!          git = off + it
!
!          dea = de(git)
!          if (mag) dea = dea + oppm(isp)*dem(git)
!
!          do lt = mbc % pth(it)+1, mbc % pth(it+1)
!             mbc % w(lt,lt) = mbc % w(lt,lt) + dea
!          end do
!       end do
!
!
!       do it1 = 1, mbc % n(2)
!          pt1 = mbc % pth(it1)
!          nstt1 = mbc % pth(it1+1) - pt1
!
!          do it2 = 1, mbc % n(2)
!             pt2 = mbc % pth(it2)
!             nstt2 = mbc % pth(it2+1) - pt2
!
!             jt = mbc % apd(it1)
!             ib = mbc % bpd(jt)
!             do while ( ib /= eol)
!                pb = mbc % pbh(ib)
!                nsttb = mbc % pbh(ib+1) - pb
!
!                do l1 = 1, nstt1
!                   do l2 = 1, nstt2
!                      do lb = 1, nsttb
!                         mbc % w(pt1+l1,pt2+l2) =  mbc % w(pt1+l1,pt2+l2) &
!                            & + mbc % dh(l1, lb, jt) * mbc % a (pb+lb,pt2+l2)
!                      end do
!                   end do
!                end do
!
!                jt = jt + 1
!                ib = mbc % bpd(jt)
!             end do
!
!          end do
!       end do
!
!       call print_c(mbc%w,500,'sgm'//achar(10))
!
! !       stop
!       allocate(work(1), rwork(3*mbc % thsz-2))
!       lwork = -1
!
!       if (.not. mbc%ovl) then
!          call zheev('v', 'l', mbc % thsz, mbc % w, mbc % thsz, mbc % v, work, lwork, rwork, info)
!       else
!          call zhegv(1, 'v', 'l', mbc % thsz, mbc % w, mbc % thsz, mbc % s, mbc % thsz, &
!                                                 & mbc % v, work, lwork, rwork, info)
!       end if
!
!       lwork = work(1)
!       deallocate(work)
!       allocate(work(lwork))
!
!       if (.not. mbc%ovl) then
!          call zheev('v', 'l', mbc % thsz, mbc % w, mbc % thsz, mbc % v, work, lwork, rwork, info)
!       else
!          call zhegv(1, 'v', 'l', mbc % thsz, mbc % w, mbc % thsz, mbc % s, mbc % thsz, &
!                                                 & mbc % v, work, lwork, rwork, info)
!       end if
!
!       deallocate(work,rwork)
!
! !       print *, info
!
!       call print_c(mbc%v,400,'v'//achar(10))
!
!       do i = 1, mbc % thsz
!          mbc % w (:,i) = mbc % w(:,i)/sqrt(ez - mbc % v(i))
!       end do
!
!       call print_c(mbc%w,400,'w'//achar(10))
!
!
!
!       call zgemm ('n', 'c', mbc % thsz, mbc % thsz, mbc % thsz, z_one, &
!                             & mbc % w, mbc % thsz, mbc % w, mbc % thsz, &
!                             & z_zero, mbc % gt, mbc % thsz)
!
!       stop 'remove me when you find out whats wrong with zgemm'
!
!       call print_c(mbc%gt,400,'gt'//achar(10))
!
!
! !       AC, C: gt. may be try zhemm?
!       call zgemm ('n', 'n', mbc % bhsz, mbc % thsz, mbc % thsz, z_one, &
!                             & mbc % a, mbc % bhsz, mbc % gt, mbc % thsz, &
!                             & z_zero, mbc % ac, mbc % bhsz)
!
!    end subroutine gtb
!


   subroutine gtbi(mbc, ez, isp)
      use mod_all_scalar, only : mag


      type(mbtb_t), intent(inout) :: mbc
      complex(dp), intent(in) :: ez
      integer, intent(in) :: isp

      integer :: nstt1, nstt2, nsttb, l1, l2, lb, jt, it1, it2, ib, pt1, pt2, pb, it, git, lt
      integer :: i, off, j

      real(dp) :: dea
      complex, parameter :: z_one = 1.0_dp, z_zero = 0.0_dp


      integer :: info, lwork
      real(dp), allocatable :: rwork(:)
      complex(dp), allocatable :: work(:)
      integer, allocatable :: ipiv(:)

      include '../Include/Atom.array' ! for dem only

      ! (z*I*S - (H + dH'*Gb*dH))^-1 = W (z*I - V)^-1 W'

      mbc % gt = 0.0_dp
      mbc % gt = mbc % h
!       mbc % gt = 0.0_dp

      off = mbc % rlst(2)-1

      do it = 1, mbc % n(2)
         git = off + it

         dea = de(git)
         if (mag) dea = dea + oppm(isp)*dem(git)

         do lt = mbc % pth(it)+1, mbc % pth(it+1)
            mbc % gt(lt,lt) = mbc % gt(lt,lt) + dea
         end do
      end do


      do it1 = 1, mbc % n(2)
         pt1 = mbc % pth(it1)
         nstt1 = mbc % pth(it1+1) - pt1

         do it2 = 1, mbc % n(2)
            pt2 = mbc % pth(it2)
            nstt2 = mbc % pth(it2+1) - pt2

            jt = mbc % apd(it1)
            ib = mbc % bpd(jt)
            do while ( ib /= eol)
               pb = mbc % pbh(ib)
               nsttb = mbc % pbh(ib+1) - pb

               do l1 = 1, nstt1
                  do l2 = 1, nstt2
```

```
                     do lb = 1, nsttb
                        mbc % gt(pt1+l1,pt2+l2) =  mbc % gt(pt1+l1,pt2+l2) &
                           & + mbc % dh(l1, lb, jt) * mbc % a (pb+lb,pt2+l2)
                     end do
                  end do
               end do

               jt = jt + 1
               ib = mbc % bpd(jt)
            end do

         end do
      end do

      mbc % gt = - mbc % gt

      if (.not. mbc % ovl) then
         do i = 1, mbc % thsz
            mbc % gt(i,i) = mbc % gt(i,i) + ez
         end do
      else
         mbc % gt = mbc % gt + mbc % s * ez
      end if


      allocate(ipiv(mbc % thsz))

      call zgetrf (mbc % thsz, mbc % thsz, mbc % gt, mbc % thsz, ipiv, info)
      if (info /= 0) stop 'zgetrf info /= 0'

      allocate(work(1))
      lwork = -1

      call zgetri(mbc % thsz, mbc % gt, mbc % thsz, ipiv, work, lwork, info)
      if (info /= 0) stop 'zgetri i info /= 0'

      lwork = work(1)
      deallocate(work)
      allocate(work(lwork))

      call zgetri(mbc % thsz, mbc % gt, mbc % thsz, ipiv, work, lwork, info)
      if (info /= 0) stop 'zgetri r info /= 0'

      deallocate(work,ipiv)



!       There is something going on with this zgemm
!       call zgemm ('n', 'n', mbc % bhsz, mbc % thsz, mbc % thsz, z_one, &
!                      & mbc % a, mbc % bhsz, mbc % gt, mbc % thsz, &
!                      & z_zero, mbc % ac, mbc % bhsz)

! NOTE: quite inefficient
      do j = 1, mbc % thsz
         do i = 1, mbc % bhsz
            mbc % ac(i,j) = sum(mbc % a(i,:) * mbc % gt(:,j))
         end do
      end do


   end subroutine gtbi


   subroutine gb_x_dh(mbc,ez,isp)
      use ab_io
      use mod_ham

      type(mbtb_t), intent(inout) :: mbc
      complex(dp), intent(in) :: ez
      integer, intent(in) :: isp

      integer :: ib, ia, it, jb, jt, la, lb, lt, gia, gib, git

      complex(dp) :: gab(mxnstat, mxnstat)
```

```fortran

      mbc%a = 0.0_dp

      do ia = 1, mbc % n(1)

         call assoc_ab(ia,isp)
         call assoc_ham(ia) ! for cluster, pcluster and decipher

         gia = mbc%rlst(1)-1 + ia ! gia == ia, not sure why am i doing the extra addition.
consistency?

         call get_gab(ez, ia, ia, 1, gab)

         do la = 1, mbc%pah(gia+1)-mbc%pah(gia)      ! it is a little dirty to sed ggd and gg here
            mbc % ggd(mbc%pah(gia) + la) = gab(la,la)  ! but in this way gab is reused and another
call get_gab is avoided
         end do


         print *, 'ia:', ia
         do it = 1, mbc % n(2)
            jt = mbc % apd(it)
            ib = mbc % bpd(jt)
            git = mbc % rlst(2) -1 + it
             print *, 'it:    ', it
            do while ( ib /= eol)
               jb = decipher(ib) ! ib is local for region 1 as is decipher
                print *, 'jb:        ', jb
               if (jb /= 0) then
                   print *, 'ib:             ', ib
                  gib = mbc%rlst(1)-1 + ib !

                  call get_gab(ez, ia, ib, jb, gab)

                  do lt = mbc % pth(it) + 1, mbc % pth(it + 1)
                     do la = mbc % pbh(ia)+1, mbc % pbh(ia + 1)
                        mbc%a(la, lt) = 0.0_dp
                        do lb = 1, mbc % pbh(ib + 1) - mbc % pbh(ib)
                           mbc % a(la, lt) = mbc % a(la, lt) &
                              & + gab(la - mbc % pbh(ia), lb) * mbc%dh(lt - mbc % pth(it), lb, jt)
                        end do
                     end do
                  end do

               end if
               jt = jt + 1
               ib = mbc % bpd(jt)
            end do
         end do
      end do

   end subroutine gb_x_dh



   subroutine get_gab(ez, ia, ib, jb, gab)
      use ab_io
      use mod_g0n
      use mod_all_scalar, only : momflg, term

      complex(dp), intent(in) :: ez
      integer, intent(in) :: ia, ib, jb
      complex(dp), intent(out) :: gab(mxnstat,mxnstat)

      complex(dp) :: g0n(0:mrec+2), hia(0:mrec+1), hib(1:mrec+1)

      integer :: la, lb, nla, nstta, nsttb, lla, nma, ma, nmax, nrec, za, zb

      include "../Include/Atom.array"

      za = z(ia) ! offset = 0 region 1  ! These are inside because the number of cross-neighbours is
expected to be low.
      call states(za, nla, nstta, llista) !

      if (momflg == 1) then
         lla = 0
         do la = 1,nla
            nma = 2*llista(la) + 1
            mlista(lla+1:lla+nma) = la
            lla = lla + nma
         enddo
      else
         do la = 1,nstta
            mlista(la) = la
         enddo
      endif

      gab = 0.0_dp

      nsttb = nstt(z(ib))

      do la = 1, nstta

         ma = mlista(la)
         nrec = lchain(ma)
         nmax = nrec + 1

         call getg0n(ez, arec(0:nrec,ma), brec(0:nrec,ma), g0n, nmax, nrec, lainf(ma), lbinf(ma))

         hia(0:nmax) = g0n(0:nmax  ) * g0n(0:nmax)
         hib(1:nmax) = g0n(0:nmax-1) * g0n(1:nmax)

         do lb = 1, nsttb
            if (la /= lb .or. jb /= 1) then
               gab(la,lb) =   sum(hia(0:nrec) * darec(0:nrec,la,lb,jb)) &
                        & + 2*sum(hib(1:nrec) * dbrec(1:nrec,la,lb,jb))
               if (term == 1) gab(la,lb) = gab(la,lb) + 2 * hib(nmax) * dbrec(nmax,la,lb,jb)
            else
               gab(la,la) = g00(ez, arec(0:nrec,ma), brec(0:nrec,ma), nrec, lainf(ma), lbinf(ma))
            end if
         end do

      end do

!       write(390,*) 'ez, ia, ib, jb',ez, ia, ib, jb
!       call print_c(gab, 390, 'gab'//achar(10))

   end subroutine get_gab


   subroutine gbop(mbc, ez, isp)
      use ab_io
      use mod_ham
      use mod_all_scalar, only : nbase

      type(mbtb_t), intent(inout) :: mbc
      complex(dp), intent(in) :: ez
      integer, intent(in) :: isp

      complex(dp) :: gab(mxnstat,mxnstat)
      complex(dp), allocatable :: g(:,:)

      integer :: ia, ib, la, lb, jb, pa, pb

      allocate(g(mbc % bhsz, mbc % bhsz))

      g = 0.0_dp

      do ia = 1, mbc % n(1)
         call assoc_ab(ia,isp)
         call assoc_ham(ia)

         pa = mbc % pbh(ia)

         do jb = 1, pcluster(nbase+1)-1
            ib = cluster(jb)
            pb = mbc % pbh(ib)

            call get_gab(ez, ia, ib, jb, gab)

            do la = 1, mbc % pbh(ia+1) - pa
               do lb = 1, mbc % pbh(ib+1) - pb
                  g(pa+la,pb+lb) = gab(la,lb)
               end do
            end do

!              write(700,*),'ia,ib,jb,ez:', ia,ib,jb, ez
!              call print_c(gab,700,'')
!
         end do
      end do

!       write(600,*) 'ez', ez
!       call print_c(g,600,'g'//achar(10),f='f8.3')

      deallocate(g)

   end subroutine gbop

   subroutine a_and_ac(mbc, ez, isp)
      type(mbtb_t), intent(inout) :: mbc
      complex(dp), intent(in) :: ez
      integer, intent(in) :: isp

!       call gbop(mbc,ez,isp)


      call gb_x_dh(mbc,ez,isp)
      call gtbi(mbc, ez, isp)
   end subroutine a_and_ac


   subroutine mbeq(mbc)
      use mod_all_scalar, only : mag, quiet, lef, locc, dqmax, nbase
      use mod_atom_ar
      use ab_io
      use mod_ham

      type(mbtb_t), intent(inout) :: mbc

      real(dp) :: ef
      character(len=25) :: ffmt
      integer :: ia, jb

      include '../Include/Atom.array' ! for dq, zc, z

      call recurse(.false.)

      do ia = 1, mbc % n(1)
         call assoc_ham(ia)
         do jb = 1, pcluster(nbase+1)-1
            decipher(cluster(jb)) = jb
         end do
      end do

      ef = embeff(mbc,locc,lef)
      lef = ef

      dqmax = 0.0_dp
      do ia = 1, mbc % nd
         dq(ia) = dq(ia) - zc(z(ia))
         if (abs(dq(ia)) > abs(dqmax)) dqmax = dq(ia)
      end do


      if (.not. quiet) then
         write(ffmt,"(a,i0,a)") '(a,":",', mbc % nd, '(x,f22.14))'
         write(6,'(a)',advance='no') grnb
         write(6, ffmt,advance='no') 'dq', dq(: mbc % nd)
         write(6,'(a)') endc
         if (mag) write(6, ffmt) 'dm', dem(:mbc % nd)
   !                if (mag) write(6, ffmt) 'mg', mg(:mbc%nd)
```

```fortran
      end if

!       call mbq

   end subroutine mbeq

   function embeff(mbc,locc,ef0) result(ef)
      use mod_all_scalar, only : totnia

      type(mbtb_t), intent(inout) :: mbc
      real(dp), intent(in) :: locc, ef0
      real(dp) :: ef
      real(dp) :: eflo,efhi,nello,nelhi
      real(dp) :: nel


      real(dp), parameter :: def = 1.0_dp, maxdn = 1.0e-3_dp



!       do it = 1, 100
!           if (it > 1) nel = mbq(mbc, ef)
!
!           if (.not. quiet) then
!             write(6,'(/,"it:",i3)') it
!             write(6,'("ef,nf:",4(x,f22.14))') ef_prev, ef, dnf_prev, dnf
!           end if
!
!           if (abs(dnf) <= 1.0e-10_dp) exit
!           def = dnf * (ef - ef_prev) / ( dnf - dnf_prev )
!
!           ef_prev = ef
!           dnf_prev = dnf
!           ef = ef - def
!
!       end do


      ef = ef0
      nel = mbq(mbc, ef)
      print *, 'ef,nel:',ef,nel
      if (nel < locc) then
         eflo = ef
         nello = nel
         efhi = eflo + def
   1       nelhi = mbq(mbc, efhi)
         print *, 'efhi,nelhi:', efhi,nelhi
         if (nelhi < locc) then
            eflo = efhi
            nello = nelhi
            efhi = efhi+def
            goto 1
         endif
      else
         efhi = ef
         nelhi = nel
         eflo = efhi - def
   4       nello = mbq(mbc, eflo)
         print *, 'eflo,nello:',eflo,nello
         if (nello > locc) then
            efhi = eflo
            nelhi = nello
            eflo = eflo-def
            goto 4
         endif
      endif


!    Use  binary sections to find the Fermi energy.

 2    ef = (eflo + efhi)*0.5_dp
      nel = mbq(mbc, ef)
      print *, 'ef,nel:',ef,nel
      if (nel > locc) then
```

```
         nelhi = nel
         efhi = ef
      else
         nello = nel
         eflo = ef
      endif
      if (abs(nel-locc) > maxdn) goto 2

      totnia = nel

   end function embeff


   function mbq(mbc, ef) result(cnel)
      use mod_all_scalar, only : kt, mfac, nsp
      use mod_atom_ar

      type(mbtb_t), intent(inout) :: mbc
      real(dp), intent(in) :: ef
      integer :: isp, la, ia

      real(dp) :: cnel

      complex(dp) :: zp,zfac,zep
      real(dp) :: phase,w0
      integer :: m, p


!      M     = NINT(MFAC*0.25D0*(EF-LAINF(LA)+2.0D0*LBINF(LA))/KT)
!        m     = nint(mfac*0.25_dp*(ef-lainf(la)+2*lbinf(la))/kt)
!       print *,ef, lainf(la), lbinf(la), (ef-lainf(la)+2.0_dp*lbinf(la))
!       m     = nint(mfac*0.25_dp*1.8_dp/kt)
      m = mfac

      print *, 'm:',m
!       write(100+iproc,*) ef, lainf(la), lbinf(la)
!       write(100+iproc,*) '      ',ef, arec(0:lchain(la),la), brec(0:lchain(la)+1,la)

      dq = 0.0_dp
      do isp = 1, nsp
         mbc % q = 0.0_dp
         mbc % ggd = 0.0_dp

         w0    = kt*(2*m)
         phase = pi/real(2*m, dp)
         zp    = cmplx(cos(phase),sin(phase), kind=dp)
         zfac  = zp*zp

         do p = 0, m-1
!             print *, 'p', p
            zep  = ef+w0*(zp-1)

            call gg_diag(mbc, zep, isp)

            do ia = 1, mbc % nd
!                print *, 'ia:', ia
               do la = mbc%pah(ia)+1, mbc%pah(ia+1)
!                   print *, '    la,g:', la-mbc%pah(ia), mbc % ggd(la)
                  mbc % q(ia) = mbc % q(ia) + real(zp * mbc % ggd(la))
               end do
            end do

            zp   = zfac*zp
         enddo

         mbc % q = 4 * kt * mbc % q

         dq(:mbc%nd) = dq(:mbc%nd) + mbc % q

         if (isp /= 1) then ! mag == true and isp == 2
            dq(:mbc%nd) = 0.5_dp * dq(:mbc%nd)
            mg(:mbc%nd) = dq(:mbc%nd) - mbc%q(:mbc%nd)
         end if

```

```fortran
      end do

      cnel = sum(dq(:mbc%nd))

!       stop
   end function mbq



   subroutine mbrho(mbc, ef)
      use mod_all_scalar, only : kt, mfac, nsp
      use mod_atom_ar

      type(mbtb_t), intent(inout) :: mbc
      real(dp), intent(in) :: ef
      integer :: isp, la, ia

      real(dp) :: cnel

      complex(dp) :: zp,zfac,zep
      real(dp) :: phase,w0
      integer :: m, p


!      M     = NINT(MFAC*0.25D0*(EF-LAINF(LA)+2.0D0*LBINF(LA))/KT)
!        m     = nint(mfac*0.25_dp*(ef-lainf(la)+2*lbinf(la))/kt)
!       print *,ef, lainf(la), lbinf(la), (ef-lainf(la)+2.0_dp*lbinf(la))
!       m     = nint(mfac*0.25_dp*1.8_dp/kt)
      m = mfac

      print *, 'm:',m
!       write(100+iproc,*) ef, lainf(la), lbinf(la)
!       write(100+iproc,*) '      ',ef, arec(0:lchain(la),la), brec(0:lchain(la)+1,la)

      mbc % rho = 0.0_dp

      do isp = 1, nsp

         w0    = kt*(2*m)
         phase = pi/real(2*m, dp)
         zp    = cmplx(cos(phase),sin(phase), kind=dp)
         zfac  = zp*zp

         do p = 0, m-1
!             print *, 'p', p
            zep  = ef+w0*(zp-1)
            call gg_rho(mbc, zep, zp, isp)
            zp   = zfac*zp
         enddo

!          mbc % q = 4 * kt * mbc % q
!
!          dq(:mbc%nd) = dq(:mbc%nd) + mbc % q
!
!          if (isp /= 1) then ! mag == true and isp == 2
!             dq(:mbc%nd) = 0.5_dp * dq(:mbc%nd)
!             mg(:mbc%nd) = dq(:mbc%nd) - mbc%q(:mbc%nd)
!          end if

      end do

   end subroutine mbrho








   subroutine areduce(n, pos, full, cnds)
      integer, intent(in) :: n, pos(1:)
      real(dp), intent(in) :: full(1:)
      real(dp), intent(out) :: cnds(1:)

      integer :: i

      do i = 1, n
         cnds(i) = cnds(i) + sum(full(pos(i)+1 : pos(i+1)))
      end do

   end subroutine areduce


   subroutine mbldh(mbc)
      use ab_io  , only : init_ab , free_ab , assoc_ab
      use mod_ham, only : init_ham, free_ham, assoc_ham
      use mod_chi, only : init_chi, free_chi
      use mod_all_scalar, only : nsp, nd, quiet
      use topologia, only : mpmap, iproc


      type(mbtb_t), intent(inout) :: mbc

      integer :: ia, ja, jb, p, mp, ib, ir
      integer :: loc_clusiz

      include "../Include/PosVel.array"
      include "../Include/NebList.array"

      mbc % nd = mbc % rlst(nr+1)-1
      do ir = 1, nr
         mbc % n(ir) = mbc%rlst(ir+1) - mbc%rlst(ir)
      end do

      if (.not. allocated(mbc % ad)) allocate(mbc % ad(3, mbc % nd))

      if (.not. allocated(mbc % apb)) allocate(mbc % apb(mbc % n(1)+1))
      if (.not. allocated(mbc % bpb)) allocate(mbc % bpb(mbc % n(1)*(mxnnb+1)))

      if (.not. allocated(mbc % apt)) allocate(mbc % apt(mbc % n(2)+1))
      if (.not. allocated(mbc % bpt)) allocate(mbc % bpt(mbc % n(2)*(mxnnb+1)))

      if (.not. allocated(mbc % apd)) allocate(mbc % apd(mbc % n(2)+1))
      if (.not. allocated(mbc % bpd)) allocate(mbc % bpd(mbc % n(2)*(mxnnb))) ! there will be at least
one neighbour in its own region


      mbc % ad = ad(:, :mbc % nd)


      call extract_neblist(aptr, bptr, 1, 1, mbc % rlst, mbc % apb, mbc % bpb)
      call extract_neblist(aptr, bptr, 2, 2, mbc % rlst, mbc % apt, mbc % bpt)
      call extract_neblist(aptr, bptr, 2, 1, mbc % rlst, mbc % apd, mbc % bpd) ! list of atoms from r2
which have neighbours in r1

      print *, 'in'
      call print_neblist(1, mbc % nd, aptr, bptr)
      print *, 'out, 2->1'
      call print_neblist(1, mbc % n(2), mbc % apd, mbc % bpd )
      print *, 'out, 2'
      call print_neblist(1, mbc % n(2), mbc % apt, mbc % bpt )

!        stop
!        BOP r1
      nd = mbc % n(1)
      mpmap(iproc) = 0
      mpmap(iproc+1) = mbc % n(1)

      aptr(:mbc % n(1)+1)           = mbc % apb
      bptr(:mbc % apb(mbc%n(1)+1)) = mbc % bpb(:mbc % apb(mbc%n(1)+1))


      call init_ab (1, mbc % n(1), nsp)
!        print *, 'init_ab done'
      call init_ham(1, mbc % n(1))
!        print *, 'init_ham done'

      do ia = 1, mbc % n(1) ! bop is r1
```

```fortran
            call assoc_ab(ia,1)
            call assoc_ham(ia)
            call getnch(ia)   !Find number of linear chains per site.
            call bldclus(ia)
!             loc_clusiz = loc_clusiz + pcluster(nbase+1)-1
            call bldh()
         end do
!
! End BOP r1


!          TB r2

         if (.not. allocated(mbc % pbh)) allocate(mbc % pbh( mbc % n(1)+1))
         call mpos(mbc%rlst(1), mbc%rlst(2)-1, mbc%pbh)
         mbc % bhsz = mbc % pbh(mbc % n(1)+1)
!          print *, 'pbh', mbc % pbh

         if (.not. allocated(mbc % pth)) allocate(mbc % pth( mbc % n(2)+1))
         call mpos(mbc%rlst(2), mbc%rlst(3)-1, mbc%pth)
         mbc % thsz = mbc % pth(mbc % n(2)+1)
          print *, 'pth', mbc % pth

         mbc % ahsz = mbc % bhsz + mbc % thsz
         if (.not. allocated(mbc % pah)) allocate(mbc % pah( mbc % nd+1))
         mbc % pah(:mbc % rlst(2)) = mbc % pbh
         mbc % pah(mbc % rlst(2) : mbc % rlst(3)) = mbc%bhsz + mbc % pth
!          print *, 'pah', mbc % pah
!          print *, 'thsz:', mbc % thsz

!          stop

         print *, 'thsz:', mbc%thsz
         if (.not. allocated(mbc%h)) allocate(mbc%h(mbc%thsz,mbc%thsz))
!          print *, 'h'
         if (.not. allocated(mbc%dh)) allocate(mbc%dh(mxnstat,mxnstat,mbc%apd(mbc%n(2)+1)))
!          print *, 'dh'
         if (mbc%ovl .and. (.not. allocated(mbc%s)))  allocate(mbc%s(mbc%thsz,mbc%thsz))

!          print *, 's'

         call bldmtbmat(mbc % rlst(2), mbc % rlst(3)-1, mbc % pth, mbc % apt, mbc % bpt, mbc % h)
         if (mbc%ovl) call bldmtbmat(mbc % rlst(2), mbc % rlst(3)-1, mbc % pth, mbc % apt, mbc % bpt, mbc
         % s)
         call bldsparsedh(mbc % apd, mbc % bpd, 2, 1, mbc % rlst,  mbc%dh )

!           a(:,:), ac(:,:), aca(:,:), v(:), w(:,:), occ(:)


         if (.not. allocated(mbc%a)) allocate(mbc % a(mbc % bhsz, mbc % thsz))
         if (.not. allocated(mbc%ac)) allocate(mbc% ac(mbc % bhsz, mbc % thsz))
!          if (.not. allocated(mbc%v)) allocate(mbc % v(mbc % thsz))
!          if (.not. allocated(mbc%occ)) allocate(mbc % occ(mbc % thsz))
!          if (.not. allocated(mbc%w)) allocate(mbc % w(mbc % thsz, mbc % thsz))
         if (.not. allocated(mbc%gt)) allocate(mbc % gt(mbc % thsz, mbc % thsz))
         if (.not. allocated(mbc%ggd)) allocate(mbc%ggd(mbc % ahsz))
         if (.not. allocated(mbc%q)) allocate(mbc%q(mbc % nd))


      end subroutine mbldh


      subroutine mpos(bgn, fin, p)
         integer, intent(in) :: bgn, fin
         integer, intent(out) :: p(:)
         integer :: ia
!        calculate offsets in matrices using the common nstt.
!        To embed atoms of the same type but with different number of orbitals it may be better if the
conf_t structures are used

         include "../Include/Atom.array"

         p(1) = 0
         do ia = 2, fin - bgn + 2 ! go one extra
            p(ia) = p(ia-1) + nstt(z( bgn  - 2 + ia)) ! gia = off + ia; gia: global ia
         enddo

      end subroutine mpos


      subroutine bldsparsedh(ap, bp, r1, r2, lst, h)
!    build sparse dH using the neighbour table

         integer, intent(in) :: ap(:), bp(:), r1, r2, lst(:)
         real(dp), intent(out) :: h(:,:,:)

         integer :: i,j,off1,off2, iap, ibp
         integer :: za,zb
         integer :: ja,ja0,ia,ib,la,lb,gia,gib
         integer :: nstta,nsttb

         real(dp) :: dr(3), dea
         real(dp) :: subh(mxnstat,mxnstat)

         real(dp) :: scfcut(14)

         include "../Include/Atom.array" ! for z mainly
         include "../Include/PosVel.array"


         dea = 0.0_dp
         scfcut = 1.0_dp


         off1 = lst(r1)-1
         off2 = lst(r2)-1

         do gia = lst(r1), lst(r1+1)-1
            za = z(gia)
            nstta = nstt(za)

            ja = ap(gia-off1) ! ia = gia-off1
            ja0 = ja - 1
            do while (bp(ja) /= eol)
               ib = bp(ja)
               gib = off2 + ib
               zb = z(gib)
               nsttb = nstt(zb)

               dr = ad(:,gia)-ad(:,gib)
!              print *, 'sparse: gia, gib, ia, ib, ja:', gia, gib, ia, ib, ja
               call matel(za,zb,dr,h(:,:,ja),dea,scfcut)

               ja = ja+1
            end do
         end do
      end subroutine bldsparsedh


      subroutine bldmtbmat(bgn, fin, ph, ap, bp, h)

!       Build the full size embedded molecular TB Hamiltonian matrix

         integer, intent(in) :: bgn, fin, ph(:), ap(:), bp(:)
         real(dp), intent(out) :: h(:,:)

         integer :: i,j,off, iap, ibp
         integer :: za,zb
         integer :: ja,ja0,ia,ib,la,lb,gia,gib
         integer :: nstta,nsttb

         real(dp) :: dr(3), dea
         real(dp) :: subh(mxnstat,mxnstat)


         real(dp) :: scfcut(14)

         include "../Include/Atom.array" ! for z mainly
         include "../Include/PosVel.array"


         dea = 0.0_dp
         scfcut = 1.0_dp

         h(:,:) = 0.0_dp

         off = bgn - 1

         do ia = 1, fin - off
            gia = off + ia
            za = z(gia)
            nstta = nstt(za)

            ja = ap(ia)
            ja0 = ja - 1
            do while (bp(ja) /= eol)
               ib = bp(ja)
               gib = off + ib
               zb = z(gib)
               nsttb = nstt(zb)

               dr = ad(:,gia)-ad(:,gib)

               call matel(za,zb,dr,subh,dea,scfcut)

               iap = ph(ia)
               ibp = ph(ib)

               h(iap+1:iap+nstta, ibp+1:ibp+nsttb) = subh(:nstta,:nsttb)

               ja = ja+1
            end do
         end do

      end subroutine bldmtbmat


      subroutine swap(n,a,b)
   !    utility routine to test different swap algorithms

         integer, intent(in) :: n
         integer, intent(inout) :: a(n), b(n)

         integer :: i, t

         do i = 1, n
            t = a(i)
            a(i) = b(i)
            b(i) = t
         end do

   !       do i = 1, n
   !          a(i) = ieor(a(i),b(i))
   !          b(i) = ieor(a(i),b(i))
   !          a(i) = ieor(a(i),b(i))
   !       end do


      end subroutine swap




      subroutine extract_neblist(api, bpi, r1, r2, lst, apo, bpo)
   !    extracts the output position tables apo, bpo from the input ones api, bpi, for the atoms listed in lst,
   !       such that the first atom of a bond is bound to be in region r1 and the second one in r2

         integer, intent(in) :: api(:), bpi(:), r1, r2, lst(:)
         integer, intent(out) :: apo(:), bpo(:)

         integer :: ia, ib, ja, jb, p, mp, prevmp
```

```fortran

      mp = 1
      do ia = lst(r1), lst(r1+1) - 1
         ja = ia - lst(r1) + 1
         apo(ja) = mp

         jb = api(ia)
         p = 0
         ib = bpi(jb+p)
         do while (ib /= eol)
            if (rmap(ib, lst) == r2) then
               bpo(mp) = ib - lst(r2) + 1
               mp = mp + 1
            end if
            p = p + 1
            ib = bpi(jb+p)
         end do
         bpo(mp) = eol

         if (bpo(max(mp-1,1)) /= eol) then
            mp = mp + 1
         else
            apo(ja) = apo(ja)-1
         end if
      end do

      apo(lst(r1+1)- lst(r1) + 1) = mp-2

      print *, 'apo:', apo
      print *, 'bpo:', bpo


   end subroutine extract_neblist



   subroutine print_neblist(s,e,a,b)
!    print the position tables ap, bp from starting atom s to endind atom e

      integer, intent(in) :: s,e, a(:), b(:)

      integer :: ia, ja, p, ib

      do ia = s, e
         print *, 'ia:', ia
         ja = a(ia)
         p = 0
         ib = b(ja+p)
         do while(ib/=eol)
            print *, '   p, ib', p, ib
            p = p + 1
            ib = b(ja+p)
         end do
      end do

   end subroutine print_neblist


   function rmap(ia, rlst)
!       range map, return the range in which atom ia resides according to the range's list rlst

      integer, intent(in) :: ia, rlst(:)
      integer :: rmap
!       To be generalised in the future. For now there are only 2 blocks

      if (ia < rlst(2)) then
         rmap = 1
      else if (ia >= rlst(2) .and. ia < rlst(3)) then
         rmap = 2
      else
         rmap = 3
      end if

   end function rmap

   end module tbbop_emb

```