D Implementation new ship routine in GEOS-Chem

D.1 Ship routine

In the standard model, ship emissions are determined in the file 'emfossil.f'. Here the NO_x emissions are emitted as 10 molecules of O_3 and 1 molecule HNO_3. These emissions are stored in the array EMISRRN, which is later on stored in the emissions array REMIS (in the routine 'SETEMIS'), which is passed (as a reaction rate in RRATE) to the solver. As we need to know all the (current) values of the parameters of the LUT, it is not possible to build our new ship routine in the file 'emfossil.f', as the photolysis values are not know at the moment the ship emissions are determined. The photolysis values are determined in the routine 'CALCRATE' (which is called after 'SETEMIS'), in the routine 'CALCRATE' also the emissions are stored in the array RRATE. We moved the ship emissions calculation to 'calcrate.f', after the calculation of the photolysis values. The default diagram of the 'CALCRATE' routine and the new diagram are given in Figure 48.

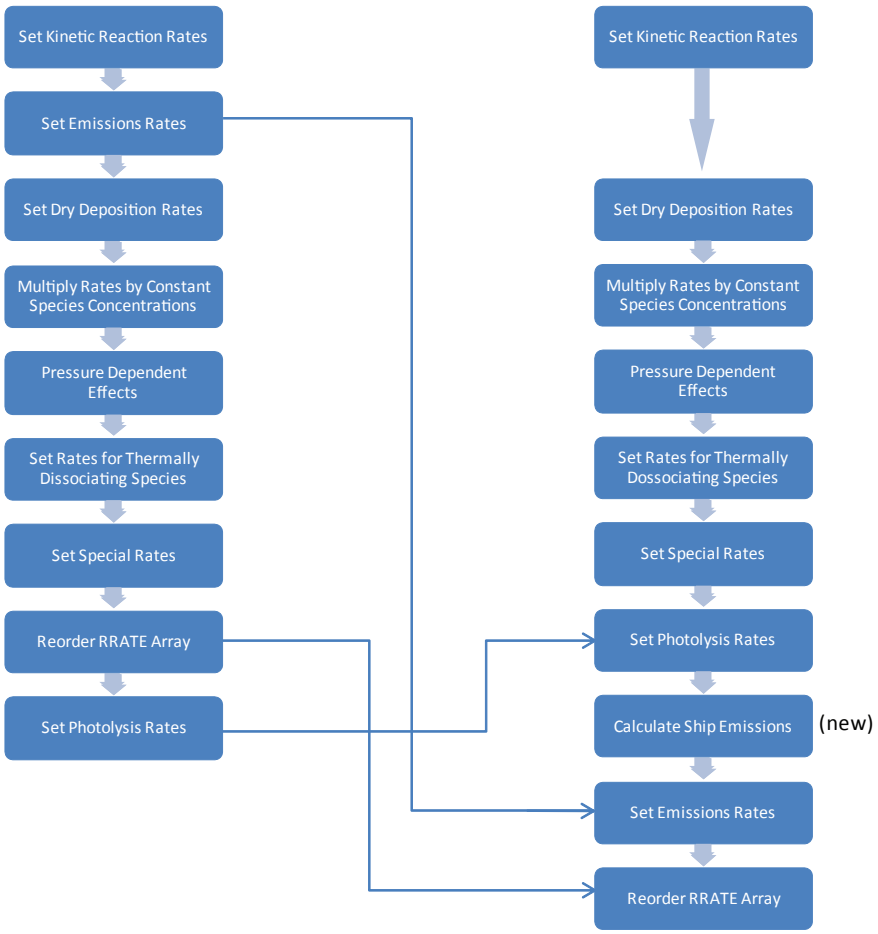

Figure 48: Diagram showing the standard approach as applied in the routine 'CALCRATE' and the new approach used in our SHIP model.

As the emissions are already stored in the array REMIS before 'calcrate' is called, we need to directly store the ship emissions in this REMIS array as well. The code we included in 'calcrate.f'

is based on the original code in 'emfossil.f' and 'setemis.f' and is given below.

```
C*****************************************************************
C             UPDATE SHIP EMISSIONS NOW THAT J-VALUES ARE KNOWN
C*****************************************************************

      DO KLOOP = 1, KTLOOP
c        JLOOP = LREORDER( KLOOP + JLOOPLO )
        JLOOP = KLOOP + JLOOPLO

        ! I-J-L indices
        IX = IXSAVE(JLOOP)
        IY = IYSAVE(JLOOP)
        IZ = IZSAVE(JLOOP)

        !-------------------------------------------------------------
        ! Add ship emissions (gvinken, 08/2010)
        !-------------------------------------------------------------
        ! DO it only once (1st level)
        IF ( IZ == 1 ) THEN

           ! Emission timestep [s]
            DTSRCE   = GET_TS_EMIS() * 60d0

           ! Surface area of grid box
           AREA_CM2 = GET_AREA_CM2( IY )

           NN = IDTNOX

           ! Reset
           SHIP = 0D0

           ! handle global inventory first
           IF ( LEDGARSHIP ) THEN

           ! Get SHIP EDGAR emissions for NOx [molec/cm2/s]
           SHIP = GET_EDGAR_NOx( IX, IY,
     &                       MOLEC_CM2_S=.TRUE., SHIP=.TRUE.)


           ! ICOADS ship emissions (cklee,7/09/09)
           ELSE IF ( LICOADSSHIP ) THEN

              ! Get ICOADS  emissions for NOx [molec/cm2/s]
              SHIP = GET_ICOADS_SHIP( IX, IY, NN, MOLEC_CM2_S=.TRUE. )

           ENDIF

           ! Overwrite Europe
           IF ( LEMEPSHIP ) THEN

             ! Prevent overwriting ICOADS data with a 0 value from EMEP
             IF ( (GET_EUROPE_MASK( IX, IY ) > 0d0) .and.
     &            (IX .gt. 61) .and. (IY .lt. 80)) THEN

             ! Get SHIP EMEP emissions for NOx [molec/cm2/s]
             SHIP = GET_EMEP_ANTHRO( IX, IY, NN, SHIP=.TRUE.)
             ENDIF
           ENDIF

           ! Convert molec/cm2/s to kg/box/timestep to get same
           ! units as EMISRN, ie default GEIA emiss (phs, 7/9/09)
           SHIP = SHIP * ( DTSRCE * AREA_CM2 ) / XNUMOL(NN)

           ! Add possible scaling of NOx emissions
           SHIP = SHIP * NOx_SCALING

           CALL INTERPOLATE_LUT(IX,IY,CBLK(KLOOP,IDO3),
     &                CBLK(KLOOP,IDCO),DENAIR(KLOOP),fraction_nox,
     &                int_ope)


           !=============================================
           ! STORE EMISSIONS DIRECTLY IN REMIS ARRAY, based on setemis.f
           !=============================================

           !=============================================================
           ! For GEOS-3: distribute surface emissions throughout the
           ! entire boundary layer.  Define some variables here.
           ! (bdf, 6/15/01)
```

```
      !=========================================================

      ! Top level of the boundary layer
      ! guard for b.l. being in first level.
      TOP = FLOOR( GET_PBL_TOP_L( IX, IY ) )
      IF ( TOP == 0 ) TOP = 1

      ! Check for possible emissions above PBL (phs, 27/10/09)
      ! Not necessary as ship emissions occur only in lowest level
      !TOPMIX = MIN( TOP, NOXEXTENT )

      ! Add option for non-local PBL (Lin, 03/31/09)
      IF (LNLPBL) TOP = 1

      ! Pressure thickness of entire boundary layer [hPa]
      TOTPRES = GET_PEDGE(IX,IY,1) - GET_PEDGE(IX,IY,TOP+1)

      !=========================================================
      ! SHIP NOx emissions [molec/box/s]
      ! Distribute emissions thru the entire boundary layer
      !=========================================================

      ! COEF1 = molecules of emission species / molecules of tracer

      COEF1 = 1.0 + CTRMB(NN, IDEMIS(NN))


      ! Loop over the boundary layer
      DO L = 1, TOP
        JLOOP   = JLOP(IX,IY,L)
        EMIS_BL = 0d0

        IF ( JLOOP /= 0 ) THEN

           ! Thickness of level L [mb]
           DELTPRES = GET_PEDGE(IX,IY,L) -
     &                GET_PEDGE(IX,IY,L+1)

           ! Add option for non-local PBL (Lin, 03/31/09)
           IF (LNLPBL) DELTPRES = TOTPRES

           ! Of the total anthro NOx, the fraction DELTPRES/TOTPRES
           ! goes into level L, since that is the fraction of the
           ! boundary layer occupied by level L.  Also divide NOx
           ! by COEF1 to convert from [molec NOx/box/s] to
           ! [molec NO/box/s], which is really what gets emitted.
           !
           ! Convert SHIP (kg/box/timestep) to the REMIS format
           ! (molec NO/box/s) and multiply by the fraction of NOx
           ! remaining

           EMIS_BL = ( (SHIP*fraction_nox* XNUMOL(NN) / DTSRCE )
     &               / COEF1   ) * ( DELTPRES / TOTPRES )

           ! Convert anthro NOx emissions from [molec NO/box/s]
           ! to [molec NO/cm3/s] and add to the REMIS array
           REMIS(JLOOP,IDENOX) = REMIS(JLOOP,IDENOX) +
     &                           EMIS_BL / VOLUME(JLOOP)
        ENDIF
      ENDDO

      !=========================================================
      ! SHIP HNO3 emissions [molec/box/s]
      ! Distribute emissions thru the entire boundary layer
      !=========================================================

      NN = IDTHNO3

      ! COEF1 = molecules of emission species / molecules of tracer
      COEF1 = 1.0 + CTRMB(NN, IDEMIS(NN))

      ! Loop over the boundary layer
      DO L = 1, TOP
        JLOOP   = JLOP(IX,IY,L)
        EMIS_BL = 0d0

        IF ( JLOOP /= 0 ) THEN

           ! Thickness of level L [mb]
           DELTPRES = GET_PEDGE(IX,IY,L) -
```

```
&                             GET_PEDGE(IX,IY,L+1)

           ! Add option for non-local PBL (Lin, 03/31/09)
           IF (LNLPBL) DELTPRES = TOTPRES

           ! Of the total ship HNO3, the fraction DELTPRES/TOTPRES
           ! goes into level L, since that is the fraction of the
           ! boundary layer occupied by level L.  Also divide HNO3
           ! by COEF1 to convert from [molec NOx/box/s] to
           ! [molec NO/box/s], which is really what gets emitted.
           !
           ! Convert SHIP (kg/box/timestep) to the REMIS format
           ! (molec NO/box/s) and multiply by the 1 - fraction of
           ! NOx remaining

           EMIS_BL = ( (SHIP*(1D0 - fraction_nox)* XNUMOL(NN)
&                       / DTSRCE )
&                           / COEF1   ) * ( DELTPRES / TOTPRES )

           ! Convert ship HNO3 emissions from [molec NO/box/s]
           ! to [molec NO/cm3/s] and store in the REMIS array
           REMIS(JLOOP,IDEHNO3) = REMIS(JLOOP,IDEHNO3) +
&                                  EMIS_BL / VOLUME(JLOOP)
         ENDIF
       ENDDO

       !=================================================
       ! SHIP O3 emissions [molec/box/s]
       ! Distribute emissions thru the entire boundary layer
       !=================================================

       NN = IDTOX

       ! COEF1 = molecules of emission species / molecules of tracer
       COEF1 = 1.0 + CTRMB(NN, IDEMIS(NN))

       ! Loop over the boundary layer
       DO L = 1, TOP
         JLOOP   = JLOP(IX,IY,L)
         EMIS_BL = 0d0

         IF ( JLOOP /= 0 ) THEN

           ! Thickness of level L [mb]
           DELTPRES = GET_PEDGE(IX,IY,L) -
&                             GET_PEDGE(IX,IY,L+1)

           ! Add option for non-local PBL (Lin, 03/31/09)
           IF (LNLPBL) DELTPRES = TOTPRES

           ! Of the total ship O3, the fraction DELTPRES/TOTPRES
           ! goes into level L, since that is the fraction of the
           ! boundary layer occupied by level L.  Also divide O3
           ! by COEF1 to convert from [molec NOx/box/s] to
           ! [molec NO/box/s], which is really what gets emitted.
           !
           ! Convert SHIP (kg/box/timestep) to the REMIS format
           ! (molec NO/box/s)

           EMIS_BL = ( (int_ope * SHIP * XNUMOL(NN)
&                       / DTSRCE )
&                           / COEF1   ) * ( DELTPRES / TOTPRES )

           ! Convert ship O3 emissions from [molec NO/box/s]
           ! to [molec NO/cm3/s] and store in the REMIS array
           REMIS(JLOOP,IDEOX) = REMIS(JLOOP,IDEOX) +
&                               EMIS_BL / VOLUME(JLOOP)
         ENDIF
       ENDDO

       !=================================================
       ! Update diagnostics
       !=================================================
       !ND36 = Anthro source diagnostic...store as [molec/cm2]
       !and convert to [molec/cm2/s] in DIAG3.F
       IF ( ND36 > 0 ) THEN

          AD36(IX,IY,IDEHNO3) = AD36(IX,IY,IDEHNO3) + SHIP *
&                               (1D0 - fraction_nox) *
&                               XNUMOL(IDTHNO3) / AREA_CM2
```

```
            AD36(IX,IY,IDEOX) = AD36(IX,IY,IDEOX) + SHIP *
     &                          int_ope *
     &                          XNUMOL(IDTOX) / AREA_CM2

            AD36(IX,IY,IDENOX) = AD36(IX,IY,IDENOX) + SHIP *
     &                           fraction_nox *
     &                           XNUMOL(IDTNOX) / AREA_CM2
          ENDIF

          ! ND32 = save anthro NOx for levels L=1,NOXEXTENT [molec/cm2/s]
          IF ( ND32 > 0 ) THEN
             AD32_an(IX,IY,IZ) = AD32_an(IX,IY,IZ) +
     &            ( SHIP * fraction_nox * XNUMOL(NN) /
     &                              ( DTSRCE * AREA_CM2 ) )
          ENDIF

       ENDIF


      ENDDO
```

The routine 'INTERPOLATE_LUT' interpolates the values of the fraction of NO_x remaining and integrated OPE in the LUT to the current values of the parameters. This routine is given below.

```
      subroutine INTERPOLATE_LUT(I,J,o3,co,dens,fraction_nox,int_ope)
     !=================================================================
     !
     ! INTERPOLATE_LUT:    Return FracNOx or IntOPE from the LUT
     !
     ! temp   : model temperature
     ! jno2   : J(NO2) value
     ! cao3   : concentration O3 in ambient air
     ! alfa0 : solar elevation angle 5 hours ago
     ! alfa5 : solar elevation angle at this time
     ! jo1d   : ratio J(O1D)/J(NO2)
     ! caco   : concentration CO in ambient air
     !
     ! o3     : incoming o3 concentration
     ! co     : incoming co concentration
     ! dens   : incoming air density
     !
     ! Based on routine GetAmf, by Henk Eskes, KNMI, nov 1999
     !
     !                   G.C.M. Vinken, KNMI, june 2010
     !=================================================================
      USE DAO_MOD, ONLY : TS, SUNCOS, SUNCOS5
      USE TIME_MOD, ONLY   : GET_LOCALTIME

#     include "cmn_fj.h"               ! Photolysis parameters
#     include "CMN_O3"                ! fracnox, intope, jvalues

      INTEGER,    INTENT(IN)        :: I, J
      REAL*8,       INTENT(IN)        :: o3,co,dens
      REAL,       INTENT(OUT)       ::  fraction_nox,int_ope

      ! Local Variables
      INTEGER                       :: IJLOOP
      integer,parameter             :: ntemp  = 4
      integer,parameter             :: njno2  = 4
      integer,parameter             :: ncao3  = 4
      integer,parameter             :: nalfa0 = 12
      integer,parameter             :: nalfa5 = 12
      integer,parameter             :: njo1d  = 4
      integer,parameter             :: ncaco  = 4

      real,dimension(ntemp)         :: templev
      real,dimension(njno2)         :: jno2lev
      real,dimension(ncao3)         :: cao3lev
      real,dimension(nalfa0)        :: alfa0lev
      real,dimension(nalfa5)        :: alfa5lev
      real,dimension(njo1d)         :: jo1dlev
      real,dimension(ncaco)         :: cacolev

      real              :: temp_tmp,jno2_tmp,cao3_tmp
      ! Temporary variable storage
      REAL              :: alfa0_tmp,alfa5_tmp,jo1d_tmp,caco_tmp
```

```
 real,dimension(2) :: xtemp,xjno2,xcao3,xalfa0
 ! Interpolation parameters
 real,dimension(2) :: xalfa5,xjo1d,xcaco
 ! Interpolation parameters

 ! For loops
 integer           :: itemp, ijno2, icao3, ialfa0
 integer           :: ialfa5, ijo1d, icaco
 integer           :: i0,i1,i2,i3,i4,i5,i6,i7

 REAL,DIMENSION(7) :: var_array
 ! array contain temp, jno2, cao3, alfa_0, alfa_5, jo1d, caco

 ! Set the levels that were chosen in the look up table
 templev = (/ 275. , 280. , 285. , 300. /)
 jno2lev = (/ 5.e-4, 0.0025, 0.0050, 0.012 /)
 cao3lev = (/ 5., 20., 35., 75. /)
 alfa0lev = (/ -90., -60., -45., -30., -15., 0., 15., 30.,
$              45., 60., 75., 90. /)
 alfa5lev = (/ -90., -60., -45., -30., -15., 0., 15., 30.,
$              45., 60., 75., 90. /)
 jo1dlev = (/ 5.e-4, 0.0015, 0.0025, 0.0055 /)
 cacolev = (/ 50., 100., 150., 1200. /)

 ! Set the variables
 IJLOOP = ( (J-1) * IIPAR ) + I
 var_array(1) = TS(I,J)                          ! Temperature
 var_array(2) = jvalues(I,J,1)                   ! J(NO2)
 var_array(3) = o3 / dens * 1.E9                 ! [O3] in ppbv
 var_array(4) = ASIND(SUNCOS5(IJLOOP))           ! alfa0
 var_array(5) = ASIND(SUNCOS(IJLOOP))            ! alfa5
 var_array(6) = jvalues(I,J,2) / jvalues(I,J,1)  ! J(O1D)/J(NO2)
 var_array(7) = co / dens * 1.E9                 ! [CO] in ppbv

 if (jvalues(I,J,1) .eq. 0.) var_array(6) = 0.   ! prevent NaN when jvalues are 0.

 !
 ! Determine reference index (itemp,ijno2,icao3,ialfa0,ialfa5,ialfajo1d,icaco)
 !
 !=============================================================================
 ! Find smallest temperature reference level (i) for which actual temperature
 ! is smaller, then do
 !
 ! x(1) = (  temperature_level(i+1) - actual temperature  )
 !        ------------------------------------------------
 !        ( temperature_level(i+1) - temperature_level(i) )
 !
 ! then x(2) = 1.0 - x(1)
 !
 !=============================================================================

 ! Temperature:
 temp_tmp = var_array(1)
 if ( var_array(1) > templev(ntemp) ) temp_tmp = templev(ntemp)
 ! If temperature larger than largest in LUT, assign largest temp
 if ( var_array(1) < templev(1) ) temp_tmp = templev(1)
 ! If temp smaller, assign smallest temp level
 do i0=1,ntemp-1
    itemp = i0
    if( templev(itemp+1) > temp_tmp ) exit
 end do
 xtemp(1)=(templev(itemp+1)-temp_tmp)/
$                        (templev(itemp+1)-templev(itemp))
 xtemp(2)=1.0-xtemp(1)

 ! J(NO2):
 jno2_tmp = var_array(2)
 if ( var_array(2) > jno2lev(njno2) ) jno2_tmp = jno2lev(njno2)
 ! If larger than largest in LUT, assign largest level values
 if ( var_array(2) < jno2lev(1) ) jno2_tmp = jno2lev(1)
 ! If smaller, assign smallest level value
 do i0=1,njno2-1
    ijno2 = i0
    if( jno2lev(ijno2+1) > jno2_tmp ) exit
 end do
 xjno2(1)=(jno2lev(ijno2+1)-jno2_tmp)/
$                        (jno2lev(ijno2+1)-jno2lev(ijno2))
 xjno2(2)=1.0-xjno2(1)

 ! [O3]:
```

```
 cao3_tmp = var_array(3)
 if ( var_array(3) > cao3lev(ncao3) ) cao3_tmp = cao3lev(ncao3)
 ! If larger than largest in LUT, assign largest level values
 if ( var_array(3) < cao3lev(1) ) cao3_tmp = cao3lev(1)
 ! If smaller, assign smallest level value
 do i0=1,ncao3-1
    icao3 = i0
    if( cao3lev(icao3+1) > cao3_tmp ) exit
 end do
 xcao3(1)=(cao3lev(icao3+1)-cao3_tmp)/
$                        (cao3lev(icao3+1)-cao3lev(icao3))
 xcao3(2)=1.0-xcao3(1)

 ! alfa0:
 alfa0_tmp = var_array(4)
 if ( var_array(4) > alfa0lev(nalfa0) ) alfa0_tmp =
$                                       alfa0lev(nalfa0)
! If larger than largest in LUT, assign largest level values
 if ( var_array(4) < alfa0lev(1) ) alfa0_tmp = alfa0lev(1)
 ! If smaller, assign smallest level value
 do i0=1,nalfa0-1
    ialfa0 = i0
    if( alfa0lev(ialfa0+1) > alfa0_tmp ) exit
 end do
 xalfa0(1)=(alfa0lev(ialfa0+1)-alfa0_tmp)/
$                        (alfa0lev(ialfa0+1)-alfa0lev(ialfa0))
 xalfa0(2)=1.0-xalfa0(1)

 ! alfa5:
 alfa5_tmp = var_array(5)
 if ( var_array(5) > alfa5lev(nalfa5) ) alfa5_tmp =
$                                       alfa5lev(nalfa5)
! If larger than largest in LUT, assign largest level values
 if ( var_array(5) < alfa5lev(1) ) alfa5_tmp = alfa5lev(1)
 ! If smaller, assign smallest level value
 do i0=1,nalfa5-1
    ialfa5 = i0
    if( alfa5lev(ialfa5+1) > alfa5_tmp ) exit
 end do
 xalfa5(1)=(alfa5lev(ialfa5+1)-alfa5_tmp)/
$                        (alfa5lev(ialfa5+1)-alfa5lev(ialfa5))
 xalfa5(2)=1.0-xalfa5(1)

 ! jo1d:
 jo1d_tmp = var_array(6)
 if ( var_array(6) > jo1dlev(njo1d) ) jo1d_tmp = jo1dlev(njo1d)
 ! If larger than largest in LUT, assign largest level values
 if ( var_array(6) < jo1dlev(1) ) jo1d_tmp = jo1dlev(1)
 ! If smaller, assign smallest level value
 do i0=1,njo1d-1
    ijo1d = i0
    if( jo1dlev(ijo1d+1) > jo1d_tmp ) exit
 end do
 xjo1d(1)=(jo1dlev(ijo1d+1)-jo1d_tmp)/
$                        (jo1dlev(ijo1d+1)-jo1dlev(ijo1d))
 xjo1d(2)=1.0-xjo1d(1)

 ! [CO]:
 caco_tmp = var_array(7)
 if ( var_array(7) > cacolev(ncaco) ) caco_tmp = cacolev(ncaco)
 ! If larger than largest in LUT, assign largest level values
 if ( var_array(7) < cacolev(1) ) caco_tmp = cacolev(1)
 ! If smaller, assign smallest level value
 do i0=1,ncaco-1
    icaco = i0
    if( cacolev(icaco+1) > caco_tmp ) exit
 end do
 xcaco(1)=(cacolev(icaco+1)-caco_tmp)/
$                        (cacolev(icaco+1)-cacolev(icaco))
 xcaco(2)=1.0-xcaco(1)

 !
 ! Linear interpolation
 !
 fraction_nox=0.0
 int_ope = 0.0
 do i1=1,2
  do i2=1,2
   do i3=1,2
    do i4=1,2
```

```
     do i5=1,2
      do i6=1,2
       do i7=1,2
         !IF ENCOUNTER -999 IN THE LUT PRINT ERROR!!
         IF (fracnox(itemp+i1-1,ijno2+i2-1,icao3+i3-1,ialfa0+i4-1,
$      ialfa5+i5-1,ijo1d+i6-1,icaco+i7-1) .LT. 0.) print*,"##E#"
         ! fracnox is the array with the actual lut data
         fraction_nox=fraction_nox+xtemp(i1)*xjno2(i2)
$             *xcao3(i3)*xalfa0(i4)*xalfa5(i5)*xjo1d(i6)*
$             xcaco(i7)*fracnox(itemp+i1-1,ijno2+i2-1,icao3+i3-1,
$             ialfa0+i4-1,ialfa5+i5-1,ijo1d+i6-1,icaco+i7-1)
         ! intope is the array with the actual lut data
         int_ope=int_ope+xtemp(i1)*xjno2(i2)*xcao3(i3)*
$                xalfa0(i4)*xalfa5(i5)*xjo1d(i6)*xcaco(i7)*
$                intope(itemp+i1-1,ijno2+i2-1,icao3+i3-1,
$                ialfa0+i4-1,ialfa5+i5-1,ijo1d+i6-1,icaco+i7-1)
       end do
      end do
     end do
    end do
   end do
  end do
 end do
 !

 end subroutine INTERPOLATE_LUT
```

D.2 Reading in the LUT

We read the LUT in the routine 'DO_EMISSIONS'. For this we edited the header 'CMN_O3.h' to contain the LUT arrays, this way it can be accessed in different routines. We added to 'CMN_O3.h' the following code:

```
 REAL*4 fracnox
 REAL*4 intope
 REAL jvalues
 COMMON /FRO3/
&   fracnox(4,4,4,12,12,4,4),
&   intope(4,4,4,12,12,4,4),
&   jvalues(IIPAR,JJPAR,2)
```

To the file 'emissions_mod.f' (which contains 'DO_EMISSIONS') we added at line 255:

```
! Read ICOADS ship emissions once per month (cklee, 7/09/09)
    IF ( LICOADSSHIP .and. ITS_A_NEW_MONTH() ) THEN
         CALL EMISS_ICOADS_SHIP
         open (unit=97,
$          file="/usr/people/vinken/FracNOx_binary.dat",
$          form="binary")
         read (97) fracnox
         close(97)
         open (unit=98,
$          file="/usr/people/vinken/IntOPE_binary.dat",
$          form="binary")
         read (98) intope
         close(98)
    ENDIF
```