DOE/PC/89774--T5

AUG 20 1992

Effects of Catalytic Mineral Matter on CO/CO_2 Temperature and Burning Time for Char Combustion

DOE/PC/89774--T5

DE92 041297

DE-FG22-89PC89774

Prof. John P. Longwell

Prof. Adel F. Sarofim

Chun-Hyuk Lee

Quarterly Progress Report No. 11

April - June 1992

Prepared for

Norey B. Laug
U.S. Department of Energy
Pittsburgh Energy Technology Center
P.O. Box 10940, MS 921-165
Pittsburgh, Pennsylvania 15236-0940

by

Massachusetts Institute of Technology
Department of Chemical Engineering
Cambridge, Ma 02139

MASTER

Introduction

The high temperature oxidation of char is of interest in a number of applications in which coal must be burned in confined spaces. These include: the conversion of oil-fired boilers to coal using coal-water slurries, the development of a new generation of pulverized-coal-fired cyclone burners, the injection of coal into the tuyeres of blast furnaces, the use of coal as a fuel in direct-fired gas turbines in large-bore low-speed diesels, and entrained flow gasifiers. In addition, there is a need to better understand the temperature history of char particles in conventional pulverized-coal-fired boilers in order to better understand the processes governing the formation of pollutants and the transformation of mineral matter.

The temperature of a char particle burning in an oxygen containing atmosphere is the product of a strongly coupled balance between particle size and physical properties, heat transfer from the particle, surface reactivity, CO/CO_2 ratio and gas phase diffusion in the surrounding boundary layer and within the particle. In addition to its effects on burning rate, particle temperature has major effects on ash properties and mineral matter vaporization. Measurements of the temperature of individual burning char particles have become available in recent years and have clearly demonstrated large particle to particle temperature variations which depend strongly on particle size and on particle composition. These studies, done with pulverized coal, do not allow direct determination of the CO/CO_2 ratio produced at the char surface or the catalytic effects of mineral matter in the individual char particles and it has generally been assumed that CO is the only product of the carbon-oxygen reaction and that CO_2 is formed by subsequent gas phase reaction More recent work, however, has pointed out the need to take CO_2 production into consideration in order to account for observed particle temperatures.

The importance of the CO/CO_2 ratio of carbon oxidation products is illustrated by examination of the heats of reaction for formation of these two products

$C + 1/2O_2 = CO$ $\Delta H_f = 27.2$ kcal/mol

$CO + 1/2O_2 = CO_2$ $\Delta Hf = 66.8$ kcal/mol

$C + O_2 = CO_2$ $\Delta H_f = 94$ kcal/mol

The heat released by formation of CO_2 is a factor of 3.5 higher than for CO so the temperature of a particle will depend strongly on the CO/CO_2 ratio produced. If gas diffusion through the boundary layer is fast, increased direct production of CO_2 produces a higher temperature and a higher burning rate. If the supply of oxygen to the surface is limited by diffusion through the boundary layer, production of CO_2 consumes half as much carbon as production of CO so carbon consumption rate is reduced even though temperature may be somewhat higher. Models of these complex interactions have been developed; however, the CO/CO_2 ratio produced by the carbon-oxygen reaction must, at present be assumed or inferred from measurement of particle temperature.

CO_2/CO ratios can be strongly influenced by catalytic material in the carbon and by the char temperature. In this program we are measuring the CO_2/CO ratio for both catalyzed and uncatalyzed chars over a wide range of temperature. These results will then be used to develop predictive models for char temperature and burning rates. The electrodynamic balance has been successfully used to make such measurements for a single 200 μm spherocarb particle. A few theoretical approaches to model a single particle oxidation have been made, but most of them assumed an infinitely thin reaction zone at the particle surface. These approaches do not take into account pore diffusion limitation, structural change, or reaction at low temperatures inside the particle. There is a need to develop a model which combines both solid and gas phase reactions, heat and mass transfer. In this report, progress on modeling, programming, and some results are reported.

Results

(1) Stiffness matrices

Based on the governing equations and the Galerkin finite element method we can construct tridiagonal matrices. Matrices for species conservation equation are

$$\underline{\underline{A}}\,\underline{Y_i} = \underline{B}$$

where $\underline{A} = \rho v_r \int_{R_o}^{R_-} \Phi^l \frac{d\Phi^k}{dr} r^2 dr - \rho V_i \int_{R_o}^{R_-} \frac{d\Phi^l}{dr} \Phi^k r^2 dr$, and $\underline{B} = R_i \int_{R_o}^{R_-} \Phi^l r^2 dr$

Boundary conditions for O_2, CO, and CO_2 should be considered at the solid-gas interface. The term of $\rho v_r \big]_{r=R_o} R_o^2$ is added to the diagonal element of $\underline{A}$ and $R_o^2 * flux$ is added to the corresponding element of residual vector. The values for the *flux* of O_2, CO, CO_2 are $r_c \frac{4}{3}(f_{co} - 2)$, $r_c \frac{7}{3} f_{co}$, and $r_c \frac{11}{3}(1 - f_{co})$.

A matrix for an energy conservation equation is

$$\underline{\underline{A}}\,\underline{T} = \underline{B}$$

where $\underline{A} = \rho(v_r C_{p,g} + \sum Y_i V_i C_{p,i}) \int_{R_o}^{R_-} \Phi^l \frac{d\Phi^k}{dr} r^2 dr + \lambda_g \int_{R_o}^{R_-} \frac{d\Phi^k}{dr} \frac{d\Phi^l}{dr} r^2 dr$

, and $\underline{B} = -(R_i H_i) \int_{R_o}^{R_-} \Phi^l r^2 dr$

Essential boundary conditions can be incorporated by setting a diagonal element as one, and changing the corresponding value of residual vector to the essential condition value.

(2) Diffusion velocity

The diffusion velocity V_i is calculated by following.

$$V_i = -\frac{1}{X_i} D_{im} \frac{dX_i}{dr} - \frac{D^T}{\rho Y_i T} \frac{dT}{dr}$$

There are two ways of calculating multicomponent diffusion coefficient D_{im} . One is mixture-average formulation based on the binary diffusion coefficient, and the other is using ordinary multicomponent diffusion coefficients.

$$D_{im} = \frac{1 - Y_i}{\sum_{j \neq i}^{K} X_j / D_{ji}}$$

$$D_{im} = -\frac{\sum_{j \neq i}^{K} M_j D_{ij} \nabla X_j}{\overline{M} \nabla X_i}$$

A problem with mixture averaged formulation is that it is not well defined if the mixture goes to a pure species. To satisfy the constraint of $\sum_{i=1}^{K} V_i Y_i = 0$, a corrective term V_C defined by $V_c = -\sum_{i=1}^{K} V_i Y_i$ is used. And updated value of V_i satisfying the restriction condition is obtained by $V_i\ (new) = V_i\ (old) + V_C$. An alternative approach can be used if one species is present in excess. An excess component mass fraction is computed simply by subtracting the sum of the remaining mass fractions from one.

Because it has a differential term, the formulation of diffusion velocity reduces the order of differential equation from second to first. When we solve the equations, we first evaluate the diffusion velocities from initial guess mass fractions and temperature distribution. Then the diffusion velocities are updated from the computed values. If we can neglect the thermal diffusion related term, the structure of the stiffness matrices reduces to the form which is similar to the one from energy conservation equation.

Because thermal diffusion effect is not negligible if the temperature gradient is substantial, it is necessary to include this term for this case.

(3) Mass transfer coefficient

The mass transfer coefficient k_g can be calculated from mass fraction distribution. We calculated the mass transfer coefficient for the solid-only model in Progress Report #4. But we need to correlate the gas phase mass fraction distribution with the mass transfer coefficient.

$$k_g = \rho Y_i (V_i + v_r) \frac{M_C}{M_{O_2}} (1 + f_{CO}) \Big/ (P_b - P_s)$$

And P_S can be obtained from ideal gas law.

$$P_s = \frac{\rho RT}{M_{O_2}} Y_{O_2} \Big]_{r=R_o}$$

The Stefan flow velocity at r=R_O is

$$v_r = \frac{r_c f_{CO} R T_p}{2 M_C P}$$

, and continuity equation is used to calculate v_r at each radial position. $r^2 \rho v_r = const$

(4) CO_2/CO ratio using CONP

Before getting into the main problem, preliminary modeling efforts to predict CO_2/CO ratio has been tried by using a CHEMKIN driver called CONP, which is a code for adiabatic and constant pressure conditions. The heterogeneous reaction CO_2/CO ratio data from Tognotti reported in Progress Report #1 were used for initial condition of the gas phase reaction. The reactions considered are in Table.1 and the results of 100% oxygen are in Fig.1. The results show a close resemblance to the experimental values except that the ratio is about the half of the Tognotti's results and the threshold temperature is slightly higher than the reported value. A more quantitively accurate value can be obtained through FEM modeling. One calculation has been performed for 20% oxygen condition, and the result was consistent with the former results. But adding water vapor at 1250K didn't increase the CO_2/CO ratio, which were increased 3 ~ 4 times at experiments. This will be

further studied. Also the effects of hydrogen on CO_2/CO ratio have been tested, because there exists a small fraction of hydrogen in a spherocarb. But the effects were negligible based on the calculations using CONP code.

(5) Modeling results from FEM

The calculation of SCOM (Single Carbon Oxidation Model) was performed on Cray-Y/MP of University of Nevada, Las Vegas Supercomputer center. The average CPU times consumed were around 10-15 minutes. The solutions were obtained until the particle surface temperature reached up to 1500K. The reactions considered were $CO+O = CO_2$, $CO+O_2 = CO_2+O$, and $2O = O_2$. The total summation of the errors (73 nodes and 4 species) was $10^{-9} \sim 10^{-10}$ at low temperatures, but reached 10^{-2} at 1600 K which is over limitation. The summation of temperature errors was $10^{-6} \sim 10^{-8}$. The temperature profile and mass fraction distribution were presented in Fig.2. The distribution profile changed faster above ignition temperature which was about 1350K.

The results show oxygen concentration drops rapidly after ignition and falls down to zero about 1500-1600 K. The exact cause of this problem is not clear now. But the followings points needs to be considered. If this situation is a real phenomena, it is hard to resolve because we need the mass fraction gradient at gas-solid interface to calculate mass transfer rate. We may use the negative sum of other component gradients but this gives considerable error.

Another possibility is the numerical method itself. The method currently used is 'successive substitution' ,which updates the parameters continuously from the calculated values. But this method converges relatively slowly and error accumulation may be a potential cause of error at high temperatures. A more efficient method is called Newton method. Newton method has good and bad points. Good point is it is a quadratic converging scheme so it converges fast near solutions. But bad points are it needs a good

initial guess and large computation time to evaluate the Jacobian matrix. And in case of nonlinear problem, we can calculate Jacobian only by numerical finite difference.

The oxygen partial pressure of 0.2 and 0.9999 atm were tried to find out whether existence of excess component might help the problem solvability. The latter was almost same as 1.0 atm case, because the error became the same order of magnitude of nitrogen concentration. That may imply we need Newton method at high temperatures to satisfy the error limitation. In the former case the particle did not ignite ,which will be further studied.

Experimental Work

Poor CO_2 laser performance was traced to less of gas pressure. The laser was sent to manufacturer for refill.

Future Program

Next quarter a simpler model will be tested to find out the major cause of the problem. The entire temperature range behavior will be traced and the effects of important parameters will be tested. The experimental program will be resumed using the repaired laser.

Nomenclature

[subscript]

'g' means gas mixture in the boundary layer

'i' means the i-th species in the gas mixture

[symbol]

$C_{p,i}$	specific heat of 'i' th species
$C_{p,g}$	gas mixture specific heat at constant pressure
D_{im}	multicomponent diffusion coefficient of 'i' th species
Dij	binary diffusion coefficient
f_i	i-th reaction fraction
H_i	specific enthalpy
M_i	molecular weight of 'i' th species
P	pressure
R	gas constsnt
R_i	mass production rate
R_O	particle radius
r	radial coordinate
r_C	carbon consumption rate [gC/cm^{2}s]
T	temperature of the gas phase
T_m	mean temperature of boundary layer
T_O	temperature at t=0
T_p	particle temperature
t	time
v_r	fluid velocity in 'r' direction
V_i	diffusion velocity of 'i' th species
W_i	molecular weight of i
X_i	mole fraction
Y_i	mass fraction
γ_i	g of i generated/g C consumed
λ_g	thermal conductivity
Φ	basis function for Galerkin finite element method
$\Delta\nu_i$	net stoichiometry of i-th reaction
ρ_g	density of the gas mixture in the boundary layer

[Table 1] Reactions for CO_2/CO ratio using CONP

```
ELEMENTS C   O   H   N  END
SPECIES  O2  CO  CO2  H2 H O OH HO2 H2O2 H2O N N2 NO END
REACTIONS
   H2+O2=2OH                          0.170E+14  0.00  47780
   OH+H2=H2O+H                        0.117E+10  1.30   3626 ! D-L&W
   O+OH=O2+H                          0.400E+15 -0.50      0 ! JAM 1986
   O+H2=OH+H                          0.506E+05  2.67   6290 ! KLEMM,ET AL
   H+O2+M=HO2+M                       0.361E+18 -0.72      0 ! DIXON-LEWIS
       H2O/18.6/  H2/2.86/  N2/1.26/  CO2/4.2/  CO/2.1/
   OH+HO2=H2O+O2                      0.750E+13  0.00      0 ! D-L
   H+HO2=2OH                          0.140E+15  0.00   1073 ! D-L
   O+HO2=O2+OH                        0.140E+14  0.00   1073 ! D-L
   2OH=O+H2O                          0.600E+09  1.30      0 ! COHEN-WEST.
   H+H+M=H2+M                         0.100E+19 -1.00      0 ! D-L
     H2O/0.0/  H2/0.0/  CO2/0.0/
   H+H+H2=H2+H2                       0.920E+17 -0.60      0
   H+H+H2O=H2+H2O                     0.600E+20 -1.25      0
   H+OH+M=H2O+M                       0.160E+23 -2.00      0 ! D-L
       H2O/5/
   H+O+M=OH+M                         0.620E+17 -0.60      0 ! D-L
       H2O/5/
   O+O+M=O2+M                         0.189E+14  0.00  -1788 ! NBS
   H+HO2=H2+O2                        0.125E+14  0.00      0 ! D-L
   HO2+HO2=H2O2+O2                    0.200E+13  0.00      0
   H2O2+M=OH+OH+M                     0.130E+18  0.00  45500
   H2O2+H=HO2+H2                      0.160E+13  0.00   3800
   H2O2+OH=H2O+HO2                    0.100E+14  0.00   1800
   O+N2=NO+N                          0.140E+15  0.00  75800
   N+O2=NO+O                          0.640E+10  1.00   6280
   OH+N=NO+H                          0.400E+14  0.00      0
   CO+O+M=CO2+M                       0.617E+15  0.00   3000 ! Miller
   CO+OH=CO2+H                        0.151E+08  1.30   -758 ! et. al
   CO+O2=CO2+O                        0.253E+13  0.00  47688 ! from
   HO2+CO=CO2+OH                      0.580E+14  0.00  22934 ! Mitchell
END
```

CO2/CO ratio = f (T, H2)

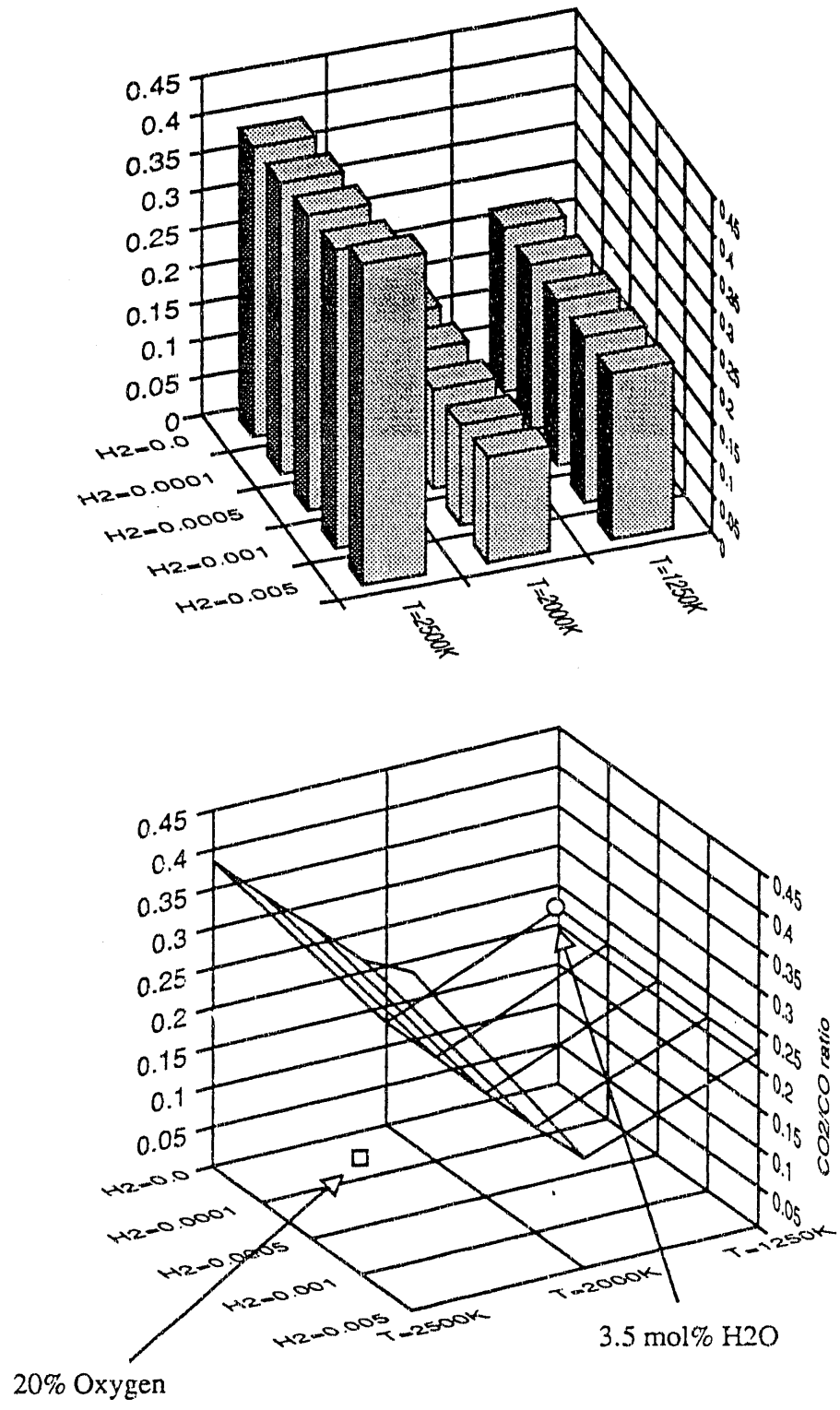

Figure 1. CO_2/CO ratio

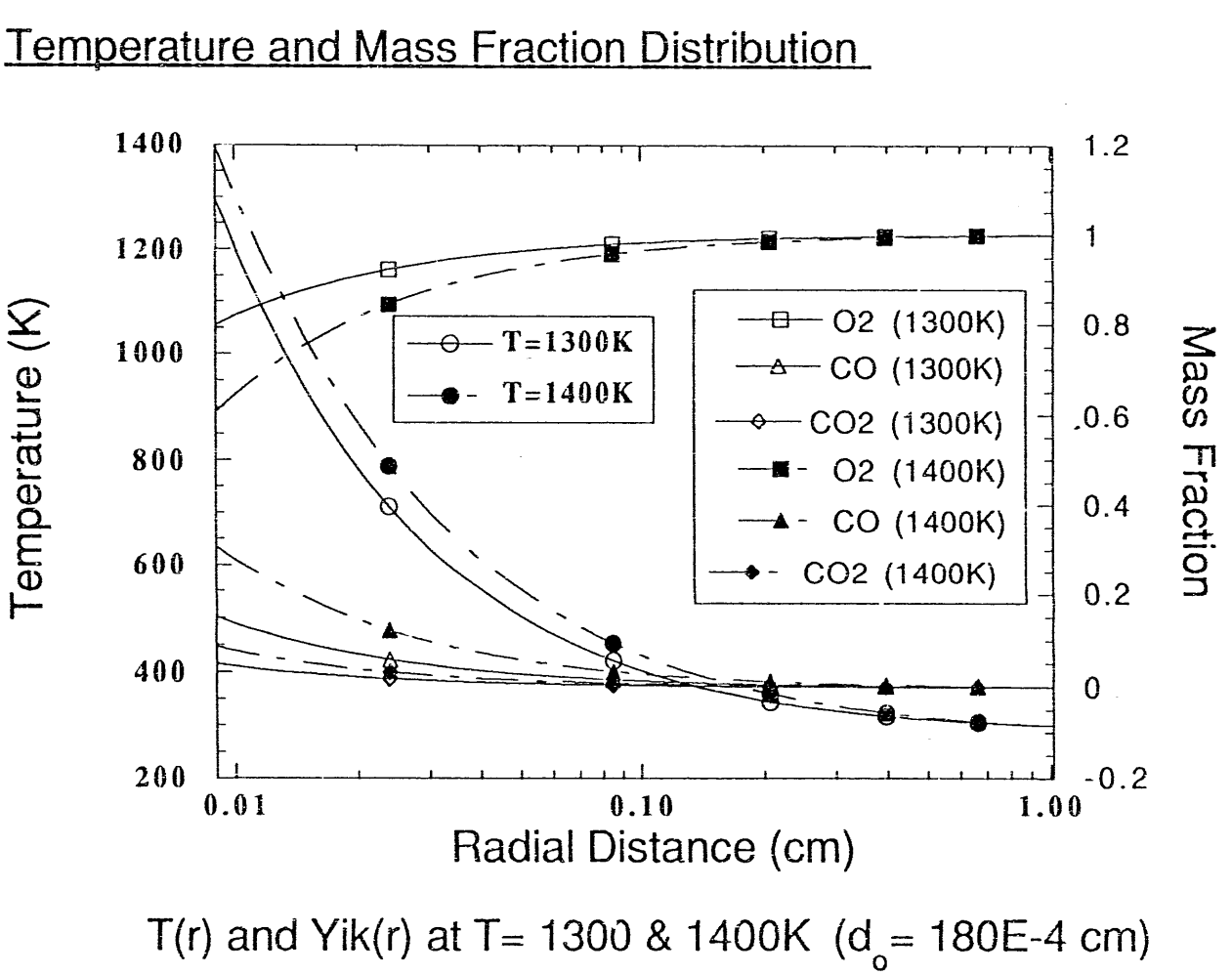

Figure 2. Temperature and mass fraction distribution in the boundary layer

APPENDICES

A1. SCOM fortran source code

```
c=======================================================================
c     This program calculates the time-dependent temperature profile
c     and concentration & temperature  using heat balance equation.
c        1) solid phase using Thiele modulus approach
c        2) gas phase reaction by FEM
c     This program requires at least CHEMKIN II 2.6 & TRANFIT 1.9 for
c     calculating thermodynamic and multicomponent transport data.
c-----------------------------------------------------------------------
c     made by Chun-hyuk Lee ,MIT, Oct-Nov,90
c     modified by Chun-hyuk Lee, MIT, Mar-May,92
c-----------------------------------------------------------------------
c     SCOM (Single Carbon Oxidation Model)
c     VERSION 1.1  (April 30, 92)
c     VERSION 1.2  (May 12, 92)
c     VERSION 1.3  (May 22, 92)
c     VER 1.3.1 : generate matrix output for gas T(r)
c     * This version includes second order differential term of
c     Yi to solve the instability and divergence resulted from
c     evaluating Vi from known mass fraction distribution.
c     * Vectorize the Do-loop for the initial guess for T
c     ---> diverges at T=1050K
c     * Calculate yo2 = 1 - ysum(ic=2,ncomp)
c     ---> laser flux needs to be increased to get to T > 1050K
c     ---> fails at 1711K due to Yo2 is negative
c     * Mass transfer coeff. needs to be correlated to the mass
c     fraction distribution in the gas phase.
c     * Sum of Yik*Vi should be 0.
c     ---> use Vc = -sum(Yik*Vi) : Vi(new)= Vi+Vc
c     ---> fails due to at high T,  Vc is getting bigger than Vi
c     * Calculate Dm by using ordinary multi-component diff. coeff.
c     ---> dXk=-sum(dXj) when calculate Dkm
c     ---> set minimum for dXk
c     ---> check the condition of Vi > vr,Vc
c     ---> set maximum for Vi
c     * Back to averaged diff.coeff. give more iteration
c     ---> Vi(O2)= -sum(Yik*Vi)/Yik(O2) when calculate rkg
c=======================================================================

      implicit real (a-h,o-z)
c
c     parameters for chemkin
c
      PARAMETER (LINKCK=25,LINKTP=35,LTRAN=31,LOUT=6,KDIM=15, NO=4,
     1           LENICK=2000, LENRCK=2000, LENCCK=100,
     2           LENIMC=100, LENRMC=6000)
c-----------------------------------------------------------
c     LENIMC=4*KK+NLITE
c     LENRMC=(19+2*NO+NO*NLITE)*KK+(15+NO)*KK**2
c     LENICK,LENRCK,LENCCK: from interpreter output
c     NO: order of polynomial fit, default=4
c     NLITE: # of species of molecular weight < 5
c     KDIM: >= KK number of species
c-----------------------------------------------------------
        common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
        common/mat2/to
        common/mat3/iband
        common/mat4/axis,xfact
        common/mat5/nnx,nxel,nntol,nodtol,nrhs,ncomp
        common/dat1/conv,eff,rc,xco,d,qcon
        common/dat2/dtold
        common/dat3/rkg,po2
        common/dat4/rho(200),vr(200),vi(15,200),ri(15,200),cpg(200),
```

```
     #          cpi(15,200),rhamda(200),hi(15,200),xi(15),yi(15)
           common/dat5/viy(15,200),vit(15,200),vc(200)
c-----------------------------------------------------------------------
           common/par1/ICKWRK(2000),RCKWRK(2000)
           common/par2/KK,P,RU,RUC,WT(15),WDOT(15), HMS(15),
     #          CPBMS,CPMS(15),RHOG,xik(15,200)
           common/tdat1/IMCWRK(100), RMCWRK(6000)
           common/tdat2/EPS(15),SIG(15),DIP(15),POL(15),ZROT(15),
     1          NLIN(15),DTH(15),COND,DM(15),Dkm(15,15)
C
      LOGICAL IERR
      CHARACTER CCKWRK(100)*16, KSYM(15)*16
C-----------------------------------------------------------------------
C
C     *****open CHEMKIN & TRANSPORT link files
      OPEN (LINKCK,STATUS='OLD', FORM='UNFORMATTED', FILE='cklink')
      OPEN (LINKTP,STATUS='OLD', FORM='UNFORMATTED', FILE='tplink')
c
c     ***** open output & solution files ****
c
      open(2,file='gast.out')
      open(3,file='gasy.out')
      open(4,file='solid.out')

C
C          INITIALIZE CHEMKIN
C
      CALL CKINIT(LENICK, LENRCK, LENCCK, LINKCK, LOUT, ICKWRK,
     1            RCKWRK, CCKWRK)
      CALL CKINDX(ICKWRK, RCKWRK, MM, KK, II, NFIT)
      IF (KK .GT. 15) THEN
         WRITE (LOUT, 1000) 15
         STOP
 1000    format(/,1x,'15 needs to be increased! at least',/)
      ENDIF
C
      CALL CKSYMS(CCKWRK, LOUT, KSYM, IERR)
      CALL CKWT(ICKWRK, RCKWRK, WT)
      CALL CKRP(ICKWRK, RCKWRK, RU, RUC, PATM)
      P = PATM
c
c     Initialize Transport
c
      call MCINIT(LINKTP,LOUT,LENIMC,LENRMC,IMCWRK,RMCWRK)
      call MCPRAM(IMCWRK,RMCWRK,EPS,SIG,DIP,POL,ZROT,NLIN)
c
c     initialization of data for solid & gas subroutines
c-----gas phase ncomp(max)=15 nxel(max)=199
      nxel=72
      nrhs=1
      ncomp=KK
      axis=1.0
      xfact=2.35
      to=298.15
      po2=1.0
c-----solid phase
      d0=180.0e-4
      temp=to
      d=d0
      wc=3.141592*180.e-4**3*0.56/6.
      time=0.000
      dt=0.0004
      convm=0.0
      efffm=1.0
c      write(6,1099) time,temp,convm,efffm
      write(6,1099) time,temp,convm
```

```
c      write(4,1099) time,temp,convm,efffm
       write(4,1099) time,temp,convm
c      ***** initialize temperature and mass fractions for the gas phase *****
       call init
c-------------------------------------------------------------
c
c      Integration of the ODEs using R-G method
c
c-------------------------------------------------------------

       do 100 it=1,3000
          time=time+dt
          tempi=temp
          wci=wc

c
c      call gas subroutine for temperature and mass fraction distribution
c
       call gas(it,temp)
c
c      calculate particle temperature
c
       call ode(temp,wc,f1,f2)
          g10=dt*f1
          g20=dt*f2
          temp=tempi+0.5*g10
          wc=wci+0.5*g20
       call ode(temp,wc,f1,f2)
          g11=dt*f1
          g21=dt*f2
          temp=tempi+0.5*g11
          wc=wci+0.5*g21
       call ode(temp,wc,f1,f2)
          g12=dt*f1
          g22=dt*f2
          temp=tempi+g12
          wc=wci+g22
       call ode(temp,wc,f1,f2)
          g13=dt*f1
          g23=dt*f2
c
c      Adjusting the dt to catch the steep temperature change
c
           dtold=dt
          dtemp=(g10+2.*g11+2.*g12+g13)/6.
c          if (abs(dtemp).gt.25.0) then
c             dt=dt/10.
c          elseif (abs(dtemp).gt.10.0.and.abs(dtemp).lt.25.0) then
          if (abs(dtemp).gt.10.0) then
             dt=dt/4.
          elseif (abs(dtemp).gt.5.0.and.abs(dtemp).lt.10.0) then
             dt=dt/2.
          elseif (temp.gt.1400.0.and.abs(dtemp).gt.1.0) then
             dt=dt/2.
          elseif (temp.gt.1600.0.and.abs(dtemp).gt.0.6) then
             dt=dt/2.
          elseif (temp.gt.1700.0.and.abs(dtemp).gt.0.4) then
             dt=dt/2.
          elseif (abs(dtemp).lt.0.1.and.abs(dtemp).gt.0.02) then
             dt=dt*2.
          elseif (abs(dtemp).lt.0.02)  then
             dt=dt*4.
          endif
          temp=tempi+dtemp
          wc=wci+(g20+2.*g21+2.*g22+g23)/6.
c
```

```
c     call ode again to calculate values for common variables
c
         call ode(temp,wc,f1,f2)
         convm=conv
         efffm=eff

c
c     print solutions
c
      write(6,1001)
c      write(6,1005) (t(i),i=1,nodtol)
      write(2,1005) (t(i),i=1,nodtol)
c
c-----------------------------------------------
      do 96 ic=1,ncomp
c         write(6,1005) (yik(ic,i),i=1,nodtol)
         write(3,1005) (yik(ic,i),i=1,nodtol)
 96   continue
c-----------------------------------------------
c     solid phase solution
c-----------------------------------------------
c      write(6,1099) time,temp,convm,efffm
      write(6,1099) time,temp,convm
c-----
      write(6,*) 'dt=  ',dt
c      write(4,1099) time,temp,convm,efffm
      write(4,1099) time,temp,convm
1099  format(e20.7,f20.4,f15.5)
c1099 format(e20.7,f20.4,2f15.5)
c
100   continue

 1001 format(/,1x,'### RESULT ###',/)
 1005 format(/,5(1x,f12.6,2x))
c 1007 format(25f7.2)
      stop
      end

c
c     Subroutine calculating temperature and weight change
c     of the particle
c
      subroutine ode(temp,wc,f1,f2)
      implicit real (a-h,o-z)
      real lo,lo0
      common/mat2/to
      common/dat1/conv,eff,rc,xco,d,qcon
      common/dat3/rkg,po2
c-------------------------------------------------------------------------
c     initial diameter
      d0=180.e-4
c     initial density
      lo0=0.56
      pei=3.141592
      wc0=pei*d0**3*lo0/6.
c     power of the mass ratio
      a=0.25
      b=(1.-a)/3.
c     gas coefficinet
      rg=1.987
      rgc=82.05
cc----initializing data-----------------------------------
      tg=to
c     x:conversion tor:tortuisity wo2:molecular weight of O2
c     po2:partial pressure of O2 (atm)
      tor=3.
```

```
      wo2=32.
c     po2=1.0
      emm=0.85
      sigma=5.676e-12
      abs=0.85
c     flux=300.
      flux=450.
c-----Flux needs to be increased to achieve higher
c     temperature if we consider gas phase reaction
c     which means more convective heat loss
      if ((temp.gt.1350.).or.(f1.lt.0.)) flux=0.
cc-------------------------------------------------------
c     calculate conversion
      x=1.-wc/wc0
      conv=x
c     calculate CO rxn fraction
      xco=1./(1.+0.02*po2**0.21*exp(6000./(rg*temp)))
c     stoichiometric const
c      xx=12.011*(1.+xco)
c     intrinsic rxn rate coefficient
      rks=0.184*exp(-33000./(rg*temp))
c      rks=0.3*exp(-36000./(rg*temp))
c     Hurt's intrinsic reaction rate
c      rks=5.0*0.3*exp(-36000./(rg*temp))
c     external mass transfer coefficient
      d=d0*(1.-x)**b
      lo=lo0*(1.-x)**a
      tm=(temp+tg)/2.
c     calculate convective heat transfer coefficient
c      ck=10.4e-5+5.56e-7*tm
c      h=2.*ck/d
c     calculate correction for mass transfer due to stefan flow
c     factm=(2.-xco)*log(2./((2.-xco)*(1.+xo2s*xco/(2-xco))))
c      factm=(2.-xco)*log(2./(2.-xco))/xco
c      db=1.0255e-5*tm**1.75
c      rkg=2.*db/(d*rgc*tm)*xx*factm
      por=(1.-lo/2.15)
c     sg: internal surface area
c      sg=(4.85*x**3-5.5566*x**2+0.8215*x+0.8688)*1.e7
      sg=(963.7928+585.2440*x-3609.9223*x**2+4179.2473*x**3
     #    -1619.3156*x**4)*1.0e4
c     calculate thiele modulus
c      dk=19400.*por/(sg*lo)*(temp/wo2)**0.5
c      dbp=1.0255e-5*temp**1.75
c      de=1./(1./dbp+1./dk)
c      deff=por*de/tor
c      pi=(d/6.)*(lo*sg*rks*rgc*temp/(deff*xx))**0.5
c     calculate effectiveness factor
c      if (pi.lt.1.e-3) then
        eff=1.0
c      elseif (pi.gt.1.e3) then
c         eff=1./pi
c      else
c         eff=(1./tanh(3.*pi)-1./(3.*pi))/pi
c      endif
c     calculate carbon consumption rate
      rc=po2/(1./rkg+1./(eff*rks*sg*lo*d/6.))

c     calculate heat of rxn
      heat1=-(-26416+3.21*(temp-tg)+0.24e-3*(temp**2-tg**2)
     &+0.09e5*(1./tg-1./temp))*4.184/12.011
      heat2=-(-94051+3.41*(temp-tg)+0.55e-3*(temp**2-tg**2)
     &-1.66e5*(1./tg-1./temp))*4.184/12.011
      heat=heat1*xco+heat2*(1.-xco)
c     calculate Cp of Superocarb
      cp=0.92+4.7e-4**temp
```

```
c     calculate correction factor due to Stefan flow for heat transfer
c      cpo2=(7.16+1.e-3*tm-0.4e5/tm**2)*4.184
c      bc=-1.*0.5*xco*rc*d*cpo2/(2.*12.011*ck)
c      if (dabs(bc).lt.1.e-2) then
c         fact=1.
c      else
c         fact=bc/(exp(bc)-1.)
c      endif
c-----ODEs
c      f1=(abs*flux/4.+rc*heat-h*(temp-tm)*fact-sigma*emm*(temp**4-tg**4))
c     &/(lo*d*cp/6.)
       f1=(abs*flux/4.+rc*heat-qcon-sigma*emm*(temp**4-tg**4))
      &/(lo*d*cp/6.)
       f2=-1.*rc*pei*d**2
       return
       end

c
c-----Initialize T(i) & Yik(ic,i)
c
       subroutine init
       implicit real (a-h,o-z)
         common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
         common/mat2/to
         common/mat5/nnx,nxel,nntol,nodtol,nrhs,ncomp
       nnx=nxel+1
c      ***** initial condition for species ***
       yo2=1.0
c      yn2=0.0
       yio=1.0e-12
c      ****** end of data
       do 705 ic=1,ncomp
          do 705 kk=1,nnx
             yik(ic,kk)=yio
             if (ic.eq.1) yik(ic,kk)=yo2
 705   continue
       do 707 kk=1,nnx
          t(kk)=to
 707   continue
       return
       end



c---------------------------------------------------------------------
c      This sub-program calculates the temperature and concentration
c      distribution in the boundary layer.
c      - Galerkin FEM is used in this modeling.
c        * linear basis function is used.
c      - The convective and conductive heat transfer terms at the solid-
c        gas interface are matched iteratively to adjust the temperature
c        calculated from the lumped solid phase model.
c      - For the thermodynamic and transport data, the package from
c        Sandia National Lab has been used
c      MADE BY Chun-hyuk Lee , Apr-May  1992
c---------------------------------------------------------------------
       SUBROUTINE gas(it,tp)
         implicit real (a-h,o-z)
         common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
         common/mat2/to
         common/mat3/iband
         common/mat4/axis,xfact
         common/mat5/nnx,nxel,nntol,nodtol,nrhs,ncomp
         common/dat1/conv,eff,rc,fco,d,qcon
         common/dat2/dtold
         common/dat3/rkg,po2
```

```
      common/dat4/rho(200),vr(200),vi(15,200),ri(15,200),cpg(200),
     #      cpi(15,200),rhamda(200),hi(15,200),xi(15),yi(15)
      common/dat5/viy(15,200),vit(15,200),vc(200)
c-----------------------------------------------------------------------
      common/par1/ICKWRK(2000),RCKWRK(2000)
      common/par2/KK,P,RU,RUC,WT(15),WDOT(15), HMS(15),
     #      CPBMS,CPMS(15),RHOG,xik(15,200)
      common/tdat1/IMCWRK(100), RMCWRK(6000)
      common/tdat2/EPS(15),SIG(15),DIP(15),POL(15),ZROT(15),
     1      NLIN(15),DTH(15), COND,DM(15),Dkm(15,15)
c
c-------set dimension for local variables
c       *old*(A,B) A: # of components  B: bigger than nnx
c       ncomp(max)=15, nxel(max)=199
c        dimension told(200), yold(15,200),told2(200),yold2(15,200)
c     #       ,ttmp(200),wtm(200)
      dimension ttmp(200),wtm(200)
c
c-------set iteration conditions
      error=1.0e-9
      niter=10
      evalue=1.0e-12
c
c     ***** adjusting maximum iteration *****
      if (tp.gt.1000.) niter=15
      if (tp.gt.1300.) niter=20


c
c       ***** calculate various parameters *****
c
      nnx=nxel+1
      nuc=nnx+1
      nlc=nnx+1
c       iband=2*nlc+1
      iband=3
      nodtol=nnx
      nntol=nxel
c
c       ***** calculate the mesh points  *****
c
      call mesh
c
c       ***** special considerations for it=1 and 2 *****
      if (it.eq.1) then
         qcon=0.
         rkg=1.0e-10
         return
      endif
      if (it.eq.2) then
c          do 150 i=1,nnx
c          told(i)=t(i)
c          do 150 ic=1,ncomp
c          yold(ic,i)=yik(ic,i)
c 150      continue
         do 155 i=1,nnx
            t(i)=(tp-to)*(1./x(i)-1./(d/2.+axis))
     #           /(2./d-1./(d/2.+axis))+to
c-------------------
c             yik(1,i)=-1.0e-8*(1./x(i)-1./(d/2.+axis))
c     #            /(2./d-1./(d/2.+axis))+1.0
            yik(2,i)=1.0e-10*(1./x(i)-1./(d/2.+axis))
     #           /(2./d-1./(d/2.+axis))
            yik(3,i)=1.0e-8*(1./x(i)-1./(d/2.+axis))
     #           /(2./d-1./(d/2.+axis))
c            yik(3,1)=1.-yik(1,i)-yik(2,i)-evalue
```

```
              yik(1,i)=1.0-yik(2,i)-yik(3,i)-evalue
c----------------------
 155        continue
c            delto=dtold
           endif
c
c          ***** newton iteration loop *****
c
           iter=0
           ipass=0
           jpass=0
           summax=0.0

 300       continue
           iter=iter+1
           write(6,2006)iter
c
c          ***** form the matrix a and the vector b *****
c          ***** iterate over components *****
c
           sum1=0.0
           sum2=0.0
c
c       ----- call CHEMKIN & TRANSPORT subroutines -----
c
           do 77 ir=1,nnx
              do 771 ic=1,ncomp
                 yi(ic)=yik(ic,ir)
 771          continue
           call CKYTX(yi,ICKWRK,RCKWRK,xi)
              do 772 ic=1,ncomp
                 xik(ic,ir)=xi(ic)
 772          continue

 77        continue
           do 88 ir=1,nnx
c-----------------------------------------------------
              do 881 ic=1,ncomp
                 yi(ic)=yik(ic,ir)
 881          continue
           call CKYTX(yi,ICKWRK,RCKWRK,xi)
c-----------------------------------------------------
           call CKMMWY(yi,ICKWRK,RCKWRK,wtmm)
                wtm(ir)=wtmm
c--------
           call CKRHOY(P,T(ir),yi,ICKWRK,RCKWRK,RHOG)
                rho(ir)=RHOG
c       vr(ir)=7.*x(1)**2*rc*fco/(6.*x(ir)**2*rho(ir))
c       vr(ir)=x(1)**2*rc/(x(ir)**2*rho(ir))
        vr(ir)=rc*fco*RU*T(ir)/(24.0*P*RHOG*x(ir)**2)*rho(1)*x(1)**2
c                if (it.eq.3.and.ir.eq.1) write(6,*) 'vr= ',vr(1)
c          vr(ir)=0.0
           call CKCPBS(T(ir),yi,ICKWRK,RCKWRK,CPBMS)
                cpg(ir)=CPBMS
           call CKWYP(P,T(ir),yi,ICKWRK,RCKWRK,WDOT)
           call CKCPMS(T(ir),ICKWRK,RCKWRK,CPMS)
           call CKHMS(T(ir),ICKWRK,RCKWRK,HMS)
             do 883 ic=1,ncomp
                ri(ic,ir)=WT(ic)* WDOT(ic)
                cpi(ic,ir)=CPMS(ic)
                hi(ic,ir)=HMS(ic)
 883          continue
           call MCMCDT(P,T(ir),xi,IMCWRK,RMCWRK,ICKWRK,RCKWRK,DTH,COND)
           rhamda(ir)=COND
c----- Use different formula for Dm
           call MCADIF(P,T(ir),xi,RMCWRK,DM)
```

```
        sumvi=0.0
        do 884 ic=1,ncomp
             if (xi(ic).lt.evalue) then
                xi(ic)=evalue
             endif
             if (yi(ic).lt.evalue) then
                yi(ic)=evalue
             endif
             if (ir.eq.nnx) then
                vi(ic,ir)=0.9*vi(ic,ir-1)
                vit(ic,ir)=0.9*vit(ic,ir-1)
                viy(ic,ir)=rho(ir)*DM(ic)
                go to 884
             endif
c            delxik=(xik(ic,ir+1)-xik(ic,ir))
            delxik=(yik(ic,ir+1)-yik(ic,ir))
c----- Set minimum value of delxik
        if (delxik.le.0.0.and.ic.eq.1) then
              delxik=evalue
        elseif (delxik.ge.0.0.and.ic.ne.1) then
              delxik=-evalue
           endif
c-----End of delxik
          vi(ic,ir)=-DM(ic)*delxik/((x(ir+1)-x(ir))*yi(ic))
     #              -DTH(ic)*(t(ir+1)-t(ir))/((x(ir+1)-x(ir))
     #              *rho(ir)*yi(ic)*t(ir))
          vit(ic,ir)=DTH(ic)*(t(ir+1)-t(ir))/((x(ir+1)-x(ir))
     #              *t(ir))
          viy(ic,ir)=rho(ir)*DM(ic)
cc-----Set maximum for Vi
c          if (abs(vi(ic,ir)).gt.1.0e3.and.ic.ne.1) then
c             vi(ic,ir)=1.0e3
c          endif
c-----Calculation of Vc
          sumvi=sumvi-yi(ic)*vi(ic,ir)
 884    continue
c----- Correcting Vi
       vc(ir)=sumvi
c       vc(ir)=0.0
       do 885 ic=1,ncomp
       vi(ic,ir)=vi(ic,ir)+vc(ir)
 885   continue
c-----End of correcting
        if (vi(1,1).gt.1.0e-5) then
           write(6,*) ' Vi O2 is positive ! Vi= ',vi(1,1)
           stop
           endif
88        continue
c--------------
c          write(6,*) 'rho= ',rho(1),' vr= ',vr(1)
c--------------
c
c       ***** solve Energy equation *****
c
c---if temperatue converged already iterate Yik only
c
          if (jpass.eq.1) go to 411
c-------------------------------------------------
         call domit
         call boundt(tp)
c
c-------- Newton Iteration for Temperature
c
c         if(iter.gt.6.and.mod(iter,3).eq.1) then
c            call tnewton
c         endif
```

```
c-------
        call band
        do 401 i=1,nodtol
        err=t(i)-b(i)
        ttmp(i)=b(i)
        sum2=sum2+sqrt(err**2)
 401  continue
 411  continue
c
c       ***** solve Species Conservation equations *****
c
c         do 99 ic=1,ncomp
        do 99 ic=2,ncomp
c
        call domi(ic)
c
c       ***** insert boundary conditions *****
c
        call bound(ic)
c
c       ***** solve the system of linear equations ****
c
        call band
        do 400 i=1,nodtol
        err=yik(ic,i)-b(i)
        yik(ic,i)=b(i)
        if (yik(ic,i).lt.evalue) yik(ic,i)=evalue
c--------
c         write(6,*) 'yik(',ic,':',i,')= ',yik(ic,i)
c--------
        sum1=sum1+sqrt(err**2)
 400  continue
 99   continue
c-------Calculate yo2
       do 408 ir =1,nnx
         ysum=0.0
         do 409 ic=2,ncomp
            ysum=ysum+yik(ic,ir)
 409     continue
         yik(1,ir)=1.0-ysum
         if (yik(1,ir).lt.evalue) then
            yik(1,ir)=evalue
         do 419 ic=2,ncomp
            yik(ic,ir)=yik(ic,ir)/ysum-evalue
 419     continue
         endif
 408   continue
c-------End of yo2
      do 402 i=1,nodtol
         t(i)=ttmp(i)
c--------
c          write(6,*) 't(',i,')= ', t(i)
c--------
 402  continue
c
c       ***** check convergence *****
c
        if(ipass.eq.1) go to 600
        esum=sum1+sum2
        write(6,2007)sum1,sum2
        if(esum.le.error) go to 500
        summax=max(esum,summax)
        if(sum2.lt.1.0e-8) jpass=1
        if(iter.lt.niter)go to 300
c
c       ----- set condition if the iteration diverges -----
```

```
c       ----- get back to initial guess & solve again -----
        if(summax.eq.esum) then
c           do 553 i=1,nnx
c           t(i)=told2(i)
c           do 553 ic=1,ncomp
c              yik(ic,i)=yold2(ic,i)
c 553       continue
c           ipass=1
           write(6,2010)
           go to 600
c           stop
c           go to 300
        endif
        write(6,2008)
        go to 600
500     continue
        write(6,2009)
 600    continue
c----------
        qcon=1.0e-7*(rhamda(1)+rhamda(2))*(t(1)-t(2))/(x(2)-x(1))/2.
        pso2=rho(1)*82.05*tp*yik(1,1)/WT(1)
        delp=po2-pso2
        if (delp.lt.evalue) delp=evalue
        vi11=-1.*(yik(2,1)*vi(2,1)+yik(3,1)*vi(3,1))/yik(1,1)
        if (abs(vr(1)).gt.abs(vi11)) then
           vio2=evalue
           write(6,*) ' vr is bigger than Vi O2  '
           write(6,*) 'vr= ',vr(1),' Vi O2= ',vi(1,1)
        endif
        vio2=-1.*(vi11+vr(1))
c        vio2=abs(vi(1,1))
        if (vio2.lt.evalue) vio2=evalue
c        if (vio2.gt.1.0e+10) vio2=1.0e+10
        rkg=rho(1)*yik(1,1)*vio2*12.011*(1.+fco)/(32.*delp)
c        if (rkg.lt.1.0e-10) rkg=1.0e-10
c-----------
        write(6,*) 'kg= ',rkg
c-----------
c
c       ***** format *****
c
2006    format(/,1x,'iteration number (iter) =',i5,/)
2007    format(/,1x,'Yik error =',e15.8,2x,'Temp error =',e15.8,/)
2008    format(/,1x,'!!-- iteration did not converge ----',/,
     #          1x,'  -- but decreasing error!!',/)
2009    format(/,1x,'!--- iteration converged -----------',/)
2010     format(/,1x,'!!!- iteration diverges ---------!!!',/)
        return
        end
c
c
c
        subroutine mesh
        implicit real (a-h,o-z)
        common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
        common/mat4/axis,xfact
        common/mat5/nnx,nxel,nntol,nodtol,nrhs,ncomp
        common/dat1/conv,eff,rc,fco,d,qcon
c
c       ***** evaluate the mesh points at each ij node *****
c
        do 320 i=1,nnx
        x(i)=axis*(float(i-1)/float(nxel))**xfact+d/2.
320     continue
c
c       ***** write the mesh *****
```

```
c
c       write(2,3001)
c       write(2,3002)
c       write(2,3003) (x(i),i=1,nodtol)
c       write(5,3003) (x(i),i=1,nodtol)
        return
3001    format(/,1x,'-----------------------------------------------',//,
     11x,'          mesh     points           ',//,1x,
     2              '-----------------------------------------------',/)
3002    format(/,1x,'r coordinate',/)
3003    format(/,5(1x,f9.6,2x))
        end
c
c
c
c
c

        subroutine domi(ic)
        implicit real (a-h,o-z)
        common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
        common/mat3/iband
        common/mat5/nnx,nxel,nntol,nodtol,nrhs,ncomp
        common/dat4/rho(200),vr(200),vi(15,200),ri(15,200),cpg(200),
     #       cpi(15,200),rhamda(200),hi(15,200),xi(15),yi(15)
        common/dat5/viy(15,200),vit(15,200),vc(200)

        dimension temp(2,2),templ(2)
c
c       ***** initialize the stiffness  matrix and load vector *****
c
        do 420 i=1,nodtol
        b(i)=0.
        do 410 j=1,iband
        a(i,j)=0.
410     continue
420     continue
c
c       ***** iterate over elements in domain *****
c
        do 4100 nelem=1,nntol
        nm(1)=nelem
        nm(2)=nm(1)+1

c
c       ***** initialize working matrices for element integration *****
c
        do 440 iw=1,2
        templ(iw)=0.0
        do 440 jw=1,2
        temp(iw,jw)=0.0
440     continue
c
c       ***** calculate integral of derivatives *****
c
        do 460 iw =1,2
           templ(iw)=templ(iw)+ ri(ic,nm(iw))*pl(nm(1),nm(2),iw)
     #          -vit(ic,nm(iw))*dpl(nm(1),nm(2),iw)
c----
     #     +rho(nm(iw))*vc(nm(iw))*yik(ic,nm(iw))*dpl(nm(1),nm(2),iw)
c----
        do 460 jw =1,2
           temp(iw,jw)=temp(iw,jw)+rho(nm(iw))*vr(nm(iw))
     #       *dpkpl(nm(1),nm(2),iw,jw)
     #       +viy(ic,nm(iw))*dpkdpl(nm(1),nm(2),iw,jw)
c-----Add corrective diffusion velocity term
```

```
c     #       -rho(nm(iw))*vc(nm(iw))*pkdpl(nm(1),nm(2),iw,jw)
 460  continue

c
c       ***** stor the element integration matrix and vector in
c       ***** the global matrix a and vecotr b
c
        do 490 i=1,2
        irow=nm(i)
        b(irow)=b(irow)+temp1(i)
        do 490 j=1,2
        icol=nm(j)
        iloc=icol-irow+2
        a(irow,iloc)=a(irow,iloc)+temp(i,j)
490     continue
4100    continue

        return
        end

c
c
c
c
        subroutine bound(ic)
        implicit real (a-h,o-z)
        common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
        common/mat3/iband
        common/mat5/nnx,nxel,nntol,nodtol,nrhs,ncomp
        common/dat1/conv,eff,rc,fco,d,qcon
        common/dat4/rho(200),vr(200),vi(15,200),ri(15,200),cpg(200),
     #       cpi(15,200),rhamda(200),hi(15,200),xi(15),yi(15)
c-----------------------------------------------------------------
c             ic=1:O2  2:CO  3:CO2  4:O
c-----------------------------------------------------------------
c-----------------------------------------------------------------
c             insert boundary conditions at the boundary r=Ro
c-----------------------------------------------------------------
c
c       ***** store boundary conditions in global matrix *****
c
        irow=1
        flux=0.0
        if (ic.eq.1) then
           flux=rc*(fco-2.)*4./3.
        elseif (ic.eq.2) then
           flux=rc*fco*7./3.
        elseif (ic.eq.3) then
           flux=rc*(1.-fco)*11./3.
        endif
c------
        if (ic.gt.3) flux=0.
c-------
        b(irow)=b(irow)+x(1)**2*flux
c        if (ic.gt.3) b(irow)=0.0
        iloc=2
        a(irow,iloc)=a(irow,iloc)+rho(1)*vr(1)*x(1)**2
c        if (ic.gt.3) then
c           do 450 icol=1,iband
c              a(irow,icol)=0.0
c 450          continue
c              a(irow,iloc)=1.0
c        endif
c-----------------------------------------------------------------
c             insert essential b.c. at the boundary r=Roo
c-----------------------------------------------------------------
```

```
      irow=nnx
      b(irow)=1.0e-15
      if (ic.eq.1) b(irow)=1.0-1.0e-15
      do 120 icol=1,iband
      a(irow,icol)=0.0
120   continue
      a(irow,2)=1.0
      return
      end
c
c
c


      subroutine domit

      implicit real (a-h,o-z)
      common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
      common/mat3/iband
      common/mat5/nnx,nxel,nntol,nodtol,nrhs,ncomp
      common/dat4/rho(200),vr(200),vi(15,200),ri(15,200),cpg(200),
     #     cpi(15,200),rhamda(200),hi(15,200),xi(15),yi(15)

      dimension temp(2,2),templ(2)
c
c     ***** initialize the stiffness  matrix and load vector *****
c
      do 20 i=1,nodtol
      b(i)=0.
      do 10 j=1,iband
      a(i,j)=0.
10    continue
20    continue
c
c     ***** iterate over elements in domain *****
c
      do 100 nelem=1,nntol
      nm(1)=nelem
      nm(2)=nm(1)+1

c
c     ***** initialize working matrices for element integration *****
c
      do 40 iw=1,2
      templ(iw)=0.0
      do 40 jw=1,2
      temp(iw,jw)=0.0
40    continue
c
c     ***** calculate integral of derivatives *****
c
      do 60 iw =1,2
         suml=0.0
         do 62 icc=1,ncomp
            suml=suml+ri(icc,nm(iw))*hi(icc,nm(iw))
62          continue
         templ(iw)=templ(iw)-suml*pl(nm(1),nm(2),iw)
      do 60 jw =1,2
         sum=0.0
         do 61 icc=1,ncomp
            sum=sum+yik(icc,nm(iw))*vi(icc,nm(iw))*cpi(icc,nm(iw))
61          continue
         temp(iw,jw)=temp(iw,jw)+rho(nm(iw))*(vr(nm(iw))*cpg(nm(iw))
     #                  +sum)*dpkpl(nm(1),nm(2),iw,jw)
     #                  +rhamda(nm(iw))*dpkdpl(nm(1),nm(2),iw,jw)

60    continue
```

```
c
c       ***** stor the element integration matrix and vector in
c       ***** the global matrix a and vecotr b
c
        do 90 i=1,2
        irow=nm(i)
        b(irow)=b(irow)+temp1(i)
        do 90 j=1,2
        icol=nm(j)
        iloc=icol-irow+2
        a(irow,iloc)=a(irow,iloc)+temp(i,j)
90      continue
100     continue

        return
        end

c
c
c
c
        subroutine boundt(tp)
        implicit real (a-h,o-z)
        common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
        common/mat2/to
        common/mat3/iband
        common/mat5/nnx,nxel,nntol,nodtol,nrhs,ncomp
c-------------------------------------------------------------------
c             insert boundary conditions at the boundary r=Ro       |
c-------------------------------------------------------------------
c
c       ***** store boundary conditions in global matrix *****
c
        irow=1
        b(irow)=tp
        do 119 icol=1,iband
        a(irow,icol)=0.0
 119    continue
        a(irow,2)=1.0
c-------------------------------------------------------------------
c             insert essential b.c. at the boundary r=Roo           |
c-------------------------------------------------------------------
        irow=nnx
        b(irow)=to
        do 120 icol=1,iband
        a(irow,icol)=0.0
120     continue
        a(irow,2)=1.0
        return
        end

c
c
c
        subroutine band
c
c-------------------------------------------------------------------------
c
        implicit real (a-h,o-z)
        common/mat1/c(200,3),v(200,1),t1(200),t2(200),t3(15,200),det,n4(2)
        common/mat3/iband
        common/mat5/nnx,nxel,nntol,nod,nvarg,ncomp
c
c-------------------------------------------------------------------------
c
```

```
      mc=nod
      nc=iband
      nv=nvarg
c
c     ***** initialize det  *****
c
      eps=1.0e-15
      icount=0
      det=1.0
c
c     *** prepare the matrix c for processing by shifting undefined   ***
c     *** elements out of the upper-left-hand corner and inserting    ***
c     *** zeros in the lower right-hand-corner                        ***
c
      nc1=nc+1
      lr=nc1/2
      mr=lr-1
      ii=mc+1
      do 60 ir=1,mr
      ii=ii-1
      nr=lr-ir
      jj=nc1
      do 40 jr=1,nr
      do 20 jc=2,nc
      p=c(ir,jc)
      k=jc-1
      c(ir,k)=p
20    continue
      c(ir,nc)=0.0
      jj=jj-1
      c(ii,jj)=0.0
40    continue
60    continue
c
c     *** use row operations to eleminate the lower triangular part of a ***
c     *** apply these same operations to the right-hand-sides             ***
c
      do 400 ic=1,mc
      ipiv=ic
      piv=c(ic,1)
      pivmax=abs(piv)
      kr=ic+1
      if(kr.gt.lr)go to 140
c
c     ***** find the largest possible pivot in the current column.
c
      do 120  ir=kr,lr
      pivmag=abs(c(ir,1))
      if(pivmax.ge.pivmag)go to 120
      ipiv=ir
      pivmax=pivmag
120   continue
      piv=c(ipiv,1)
c
c     ***** check the pivot magnitude *******
c
140   continue
      if(pivmax.ge.eps)go to 150
c
c     **** a nonzero pivot smaller than eps has been found ****
c
      return
c
c     **** if necessary , swap row ipiv with row ic ****
c
150   continue
```

```
      if(ipiv.eq.ic)go to 200
      det=-det
      do 160 jc=1,nc
      t=c(ic,jc)
      c(ic,jc)=c(ipiv,jc)
      c(ipiv,jc)=t
160   continue
      do 180 jv=1,nv
      t=v(ic,jv)
      v(ic,jv)=v(ipiv,jv)
      v(ipiv,jv)=t
180   continue
c
c     ***** update the determinant value *****
c
200   continue
      if(abs(det).ge.1.0e+25)icount=icount+1
      if(abs(det).ge.1.0e+25)det=det/abs(det)
      det=det*piv
c
c     ***** normalize the pivot row *****
c
      theta=1./piv
      do 220 jc=2,nc
      c(ic,jc)=c(ic,jc)*theta
220   continue
      do 240 jv=1,nv
      v(ic,jv)=v(ic,jv)*theta
240   continue
c
c     *** eliminate the lower triangular elements in the current column ***
c
      if(kr.gt.lr)go to 400
      do 380 ir=kr,lr
      t=c(ir,1)
      do 230 jc=2,nc
      k=jc-1
      c(ir,k)=c(ir,jc)-t*c(ic,jc)
230   continue
      c(ir,nc)=0.0
      do 360 jv=1,nv
      v(ir,jv)=v(ir,jv)-t*v(ic,jv)
360   continue
380   continue
      if(lr.eq.mc)go to 400
      lr=lr+1
400   continue
c
c     ***** triangularization is complete *****
c
c
c     ***** back-substitute to compute the solution vector(s) ******
c
      kr=2
      lc=mc-1
      do 480 ic=1,lc
      iv=mc-ic
      do 460 jv=1,nv
      ii=iv
      do 440 jc=2,kr
      ii=ii+1
      v(iv,jv)=v(iv,jv)-c(iv,jc)*v(ii,jv)
440   continue
460   continue
      if(kr.eq.nc) go to 480
      kr=kr+1
```

```
480     continue
c
c       ****** the matrix equation is solved ******
c
        return
        end

c
c       ***** Functions *****
c       ***** calculate the intergral of basis function *****
c
        FUNCTION dpkpl(nm1,nm2,kk,ll)
        common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
        if (kk.eq.1.and.ll.eq.1) then
           dpkpl=x(nm2)*x(nm1)**2/(x(nm2)-x(nm1))-(x(nm2)*x(nm1)
     #     /(x(nm2)-x(nm1)))**2*log(x(nm2)/x(nm1))
        endif
        if (kk.eq.1.and.ll.eq.2) then
           dpkpl=-x(nm2)**2*x(nm1)/(x(nm2)-x(nm1))+(x(nm2)*x(nm1)
     #     /(x(nm2)-x(nm1)))**2*log(x(nm2)/x(nm1))
        endif
        if (kk.eq.2.and.ll.eq.1) then
           dpkpl=-x(nm2)*x(nm1)**2/(x(nm2)-x(nm1))+(x(nm2)*x(nm1)
     #     /(x(nm2)-x(nm1)))**2*log(x(nm2)/x(nm1))
        endif
        if (kk.eq.2.and.ll.eq.2) then
           dpkpl=x(nm2)**2*x(nm1)/(x(nm2)-x(nm1))-(x(nm2)*x(nm1)
     #     /(x(nm2)-x(nm1)))**2*log(x(nm2)/x(nm1))
        endif
        return
        end

        FUNCTION pkdpl(nm1,nm2,kk,ll)
        common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
        if (kk.eq.1.and.ll.eq.1) then
           pkdpl=x(nm2)*x(nm1)**2/(x(nm2)-x(nm1))-(x(nm2)*x(nm1)
     #     /(x(nm2)-x(nm1)))**2*log(x(nm2)/x(nm1))
        endif
        if (kk.eq.1.and.ll.eq.2) then
           pkdpl=-x(nm2)*x(nm1)**2/(x(nm2)-x(nm1))+(x(nm2)*x(nm1)
     #     /(x(nm2)-x(nm1)))**2*log(x(nm2)/x(nm1))
        endif
        if (kk.eq.2.and.ll.eq.1) then
           pkdpl=-x(nm2)**2*x(nm1)/(x(nm2)-x(nm1))+(x(nm2)*x(nm1)
     #     /(x(nm2)-x(nm1)))**2*log(x(nm2)/x(nm1))
        endif
        if (kk.eq.2.and.ll.eq.2) then
           pkdpl=x(nm2)**2*x(nm1)/(x(nm2)-x(nm1))-(x(nm2)*x(nm1)
     #     /(x(nm2)-x(nm1)))**2*log(x(nm2)/x(nm1))
        endif
        return
        end

        FUNCTION pl(nm1,nm2,ll)
        common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
        if (ll.eq.1) then
           pl=x(nm2)*x(nm1)*0.5*(x(nm1)+x(nm2))
     #     -(x(nm1)**2+x(nm1)*x(nm2)+x(nm2)**2)
        endif
        if (ll.eq.2) then
           pl=-x(nm2)*x(nm1)*0.5*(x(nm1)+x(nm2))
     #     +(x(nm1)**2+x(nm1)*x(nm2)+x(nm2)**2)
        endif
        return
        end
```

```
      FUNCTION dpl(nm1,nm2,ll)
      common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
      if (ll.eq.1) then
         dpl=-1.*x(nm2)*x(nm1)
      endif
      if (ll.eq.2) then
         dpl=x(nm2)*x(nm1)
      endif
      return
      end

      FUNCTION dpkdpl(nm1,nm2,kk,ll)
      common/mat1/a(200,3),b(200),t(200),x(200),yik(15,200),det,nm(2)
      if (kk.eq.1.and.ll.eq.1) then
         dpkdpl=x(nm2)*x(nm1)/(x(nm2)-x(nm1))
      endif
      if (kk.eq.1.and.ll.eq.2) then
         dpkdpl=-x(nm2)*x(nm1)/(x(nm2)-x(nm1))
      endif
      if (kk.eq.2.and.ll.eq.1) then
         dpkdpl=-x(nm2)*x(nm1)/(x(nm2)-x(nm1))
      endif
      if (kk.eq.2.and.ll.eq.2) then
         dpkdpl=x(nm2)*x(nm1)/(x(nm2)-x(nm1))
      endif
      return
      end
c
c     ***** END of FUNCTIONS *****
c
```

END

DATE
FILMED
12/21/92