B Computer Program Listings for Cement Hydration Model

Program disrealnew.c

```
/*************************************************************************/
/*                                                                       */
/*      Program disrealnew.c to hydrate three-dimensional cement         */
/*              particles in a 3-D box with periodic boundaries.         */
/*              Uses cellular automata techniques and preserves correct  */
/*              hydration volume stoichiometry.                          */
/*      Programmer: Dale P. Bentz                                        */
/*                  Building and Fire Research Laboratory                */
/*                  NIST                                                 */
/*                  100 Bureau Drive Mail Stop 8615                      */
/*                  Gaithersburg, MD  20899   USA                        */
/*                  (301) 975-5865      FAX: 301-990-6891                */
/*                  E-mail: dale.bentz@nist.gov                          */
/*                                                                       */
/*************************************************************************/
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

/* Heat of formation data added: 12/92 */
/* Three-dimensional version created 7/94 */
/* General C version created 8/94 */
/* Modified to have pseudo-continuous dissolution 11/94 */
/* In this program, dissolved silicates are located close */
/* to dissolution source within a 17*17*17 box centered at dissolution source */
/* Hydration under self-desiccating conditions included 9/95 */
/* Additions for adiabatic hydration conditions included 9/96 */
/* Adjustments for pozzolanic reaction included 11/96 */
/* Additions for calcium chloride to Freidel's salt added 4/97 */
/* Additions for stratlingite formation added 4/97 */
/* Additions for anhydrite to gypsum conversion added 5/97 */
/* Additions for calcium aluminodisilicate added 7/97 */
/* Temperature-variable C-S-H properties added 8/98 */
/* molar volumes and water consumption based on values */
/* given in Geiker thesis on chemical shrinkage */
/* Modelling of induction period based on an impermeable layer */
/* theory */
/* Ettringite unstable above 70 C added 9/98 */
```

```
/* Hemihydrate to gypsum conversion added 9/98 */
/* Decided to base alpha only on four major clinker phases 9/98 */
/* Updated dissolution to allow next nearest neighbors, etc. 9/98 */
/* Created stable iron-rich ettringite 3/99 */
/* Phases renumbered 1/00 */
/* Sulfate reactions modified by Claus-Jochen Haecker 6/00 */
/* Reactions between CaCO3 and Afm phase added 8/00 */
/* Slag incorporation 2/01 */
/* pH and pore solution concentration added 8/01 */
/* Influence of pH on hydration added 2/02 */
/* Induction and gypsum acceleration - influence of w/c modified 11/04 */
/* Possibility of resaturation added 12/04 */
/* C-S-H precipitating as plates added 01/05 */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#define MAXCYC 30000    /* Maximum number of cycles of hydration */
                        /* For hydration under sealed conditions: */
#define CUBEMAX 7       /* Maximum cube size for checking pore size */
#define CUBEMIN 3       /* Minimum cube size for checking pore size */
#define SYSIZE 100      /* System size in pixels per dimension */
#define SYSIZEM1 99     /* System size -1 */
#define DISBIAS 30.0    /* Dissolution bias- to change all dissolution rates */
#define DISMIN 0.001    /* Minimum dissolution for C3S dissolution */
#define DISMIN2 0.00025  /* Minimum dissolution for C2S dissolution */
#define DISMINSLAG 0.0001  /* Minimum dissolution for SLAG dissolution */
#define DISMINASG 0.0005  /* Minimum dissolution for ASG dissolution */
#define DISMINCAS2 0.0005  /* Minimum dissolution for CAS2 dissolution */
#define DISMIN_C3A_0 0.002  /* Minimum dissolution for C3A dissolution */
#define DISMIN_C4AF_0 0.0005  /* Minimum dissolution for C4AF dissolution */
#define DETTRMAX 1200   /* Maximum allowed # of ettringite diffusing species */
#define DGYPMAX 2000    /*  Maximum allowed # of gypsum diffusing species */
#define DCACO3MAX 1000  /*  Maximum allowed # of CaCO3 diffusing species */
#define DCACL2MAX 2000  /* Maximum allowed # of CaCl2 diffusing species */
#define DCAS2MAX 2000   /* Maximum allowed # of CAS2 diffusing species */
#define CHCRIT 50.0     /* Scale parameter to adjust CH dissolution probability */
#define C3AH6CRIT 10.0  /* Scale par. to adjust C3AH6 dissolution prob. */
#define C3AH6GROW 0.01  /* Probability for C3AH6 growth */
#define CHGROW 1.0      /* Probability for CH growth */
#define CHGROWAGG 1.0   /* Probability for CH growth on aggregate surface */
#define ETTRGROW 0.002  /* Probability for ettringite growth */
#define C3AETTR 0.001   /* Probability for reaction of diffusing C3A
                             with ettringite  */
#define C3AGYP 0.001    /*  Probability for diffusing C3A to react with diffusing
gypsum */
                        /* diffusing anhydrite, and diffusing hemihydrate */
#define SOLIDC3AGYP 0.5 /* Prob. of solid C3A to react with diffusing sulfate */
#define SOLIDC4AFGYP 0.1 /* Prob. of solid C4AF to react with diffusing sulfate */
#define PPOZZ 0.05      /* base probability for pozzolanic reaction */
#define PCSH2CSH 0.002  /* probability for CSH dissolution */
                        /* for conversion of C-S-H to pozz. C-S-H */
#define A0_CHSOL 1.325  /* Parameters for variation of CH solubility with */
#define A1_CHSOL 0.008162   /* temperature (data from Taylor- Cement Chemistry) */
/* changed CSHSCALE to 70000 6/15/01  to better model induction CS */
#define CSHSCALE 70000. /*scale factor for CSH controlling induction */
```

```
#define WCSCALE 0.4     /* scale factor for influence of w/c on induction */
#define WCSULFSCALE 0.5 /* scale factor for influence of w/c on sulfate
acceleration of silicates and aluminates */
#define C3AH6_SCALE 2000. /*scale factor for C3AH6 controlling induction of
aluminates */
#define NEIGHBORS 26    /* number of neighbors to consider (6, 18, or 26) */
/* define IDs for each phase used in model */
/* To add a solid phase, insert new phase before ABSGYP */
/*  and increment all subsequent IDs */
/* To add a diffusing species, insert just before DIFFCACL2 */
/*  and increment all subsequent IDs */
#define POROSITY 0
#define C3S 1
#define C2S 2
#define C3A 3
#define C4AF 4
#define GYPSUM 5
#define HEMIHYD 6
#define ANHYDRITE 7
#define POZZ 8
#define INERT 9
#define SLAG 10
#define ASG 11          /* aluminosilicate glass */
#define CAS2 12
#define CH 13
#define CSH 14
#define C3AH6 15
#define ETTR 16
#define ETTRC4AF 17     /* Iron-rich stable ettringite phase */
#define AFM 18
#define FH3 19
#define POZZCSH 20
#define SLAGCSH 21      /* Slag gel-hydration product */
#define CACL2 22
#define FREIDEL 23      /* Freidel's salt */
#define STRAT 24        /* stratlingite (C2ASH8) */
#define GYPSUMS 25      /* Gypsum formed from hemihydrate and anhydrite */
#define CACO3 26
#define AFMC 27
#define INERTAGG 28
#define ABSGYP 29
#define DIFFCSH 30
#define DIFFCH 31
#define DIFFGYP 32
#define DIFFC3A 33
#define DIFFC4A 34
#define DIFFFH3 35
#define DIFFETTR 36
#define DIFFCACO3 37
#define DIFFAS 38
#define DIFFANH 39
#define DIFFHEM 40
#define DIFFCAS2 41
#define DIFFCACL2 42
#define EMPTYP 45         /* Empty porosity due to self desiccation */
#define OFFSET 50         /* Offset for highlighted potentially soluble pixel */
/* define heat capacities for all components in J/g/C */
```

```
/* including free and bound water */
#define Cp_cement 0.75
#define Cp_pozz 0.75
#define Cp_agg 0.84
#define Cp_CH 0.75
#define Cp_h2o 4.18     /* Cp for free water */
#define Cp_bh2o 2.2     /* Cp for bound water */
#define WN 0.23        /* water bound per gram of cement during hydration */
#define WCHSH 0.06     /* water imbibed per gram of cement during chemical
                  shrinkage (estimate) */

/* data structure for diffusing species - to be dynamically allocated */
/* Use of a doubly linked list to allow for easy maintenance */
/* (i.e. insertion and deletion) */
/* Added 11/94 */
/* Note that if SYSIZE exceeds 256, need to change x, y, and z to */
/* int variables */
struct ants{
        unsigned char x,y,z,id;
        int cycbirth;
        struct ants *nextant;
        struct ants *prevant;
};

/* data structure for elements to remove to simulate self-desiccation */
/* once again a doubly linked list */
struct togo{
        int x,y,z,npore;
        struct togo *nexttogo;
        struct togo *prevtogo;
};

/* Global variables */
/* Microstructure stored in array mic of type char to minimize storage */
/* Initial particle IDs stored in array micpart (for assessing set point) */
static char mic [SYSIZE] [SYSIZE] [SYSIZE];
static char micorig [SYSIZE] [SYSIZE] [SYSIZE];
static short int micpart [SYSIZE] [SYSIZE] [SYSIZE];
static short int cshage [SYSIZE] [SYSIZE] [SYSIZE];
static short int faces [SYSIZE] [SYSIZE] [SYSIZE];
/* counts for dissolved and solid species */
long int discount[EMPTYP+1], count[EMPTYP+1];
long int ncshplategrow=0,ncshplateinit=0;
/* Counts for pozzolan reacted, initial pozzolan, gypsum, ettringite,
initial porosity, and aluminosilicate reacted */
long int npr,nfill,ncsbar,netbar,porinit,nasr,nslagr,slagemptyp=0;
/* Initial clinker phase counts */
long int c3sinit,c2sinit,c3ainit,c4afinit,anhinit,heminit,chold,chnew;
long int nmade,ngoing,gypready,poregone,poretodo,countpore=0;
long int countkeep,water_left,water_off,pore_off;
int ncyc,cyccnt,cubesize,sealed,outfreq;
int icyc,burnfreq,setfreq,setflag,sf1,sf2,sf3,porefl1,porefl2,porefl3;
int xoff[27]={1,0,0,-1,0,0,1,1,-1,-1,0,0,0,0,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,0};
int yoff[27]={0,1,0,0,-1,0,1,-1,1,-1,1,-1,1,-1,0,0,0,0,1,-1,1,-1,1,1,-1,-1,0};
int zoff[27]={0,0,1,0,0,-1,0,0,0,0,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,0};
/* Parameters for kinetic modelling ---- maturity approach */
float ind_time,temp_0,temp_cur,time_step=0.0,time_cur,E_act,beta,heat_cf;
```

```
float w_to_c=0.0,s_to_c,krate,totfract=1.0,tfractw04=0.438596,fractwithfill=1.0;
float tfractw05=0.384615,surffract=0.0,pfract,pfractw05=0.615385,sulf_conc;
long int scntcement=0,scnttotal=0;
float U_coeff=0.0,T_ambient=25.;
float alpha_cur,heat_old,heat_new,cemmass,mass_agg,mass_water,mass_fill,Cp_now;
float alpha,CH_mass,mass_CH,mass_fill_pozz,E_act_pozz,chs_new,cemmasswgyp;
float flyashmass,alpha_fa_cur;
float E_act_slag;
/* Arrays for variable CSH molar volume and water consumption */
float molarvcsh[MAXCYC],watercsh[MAXCYC],heatsum,molesh2o,saturation=1.0;
/* Arrays for dissolution probabilities for each phase */
float disprob[EMPTYP+1],disbase[EMPTYP+1],gypabsprob,ppozz;
/* Arrays for specific gravities, molar volumes, heats of formation, and */
/* molar water consumption for each phase */
float specgrav[EMPTYP+1],molarv[EMPTYP+1],heatf[EMPTYP+1],waterc[EMPTYP+1];
/* Solubility flags and diffusing species created for each phase */
/* Also flag for C1.7SH4.0 to C1.1SH3.9 conversion */
int soluble[EMPTYP+1],creates[EMPTYP+1],csh2flag,adiaflag,chflag,nummovsl;
float cs_acc=1.0;     /* increases disprob[C3S] and disprob[C2S] if gypsum is
present */
float ca_acc=1.0;    /* increases disprob[C3A] and disprob[C4AF] if gypsum is
present */
float dismin_c3a=DISMIN_C3A_0;
float dismin_c4af=DISMIN_C4AF_0;
float gsratio2=0.0,onepixelbias=1.0;
/* Slag probabilities */
float p1slag;   /*  probability SLAG is converted to SLAGCSH */
float p2slag;   /*  probability SLAG is converted to POROSITY or EMPTYP */
float p3slag;   /*  probability adjoining pixel is converted to SLAGCSH */
float p4slag;   /*  probability CH is consumed during SLAG reaction */
float p5slag;   /*  probability a C3A diffusing species is created */
float slagcasi,slaghydcasi;   /* Ca/Si ratios for SLAG and SLAGCSH */
float slagh2osi;     /* H/S ratio of SLAGCSH */
float slagc3a;       /* C3A/slag molar ratio */
float siperslag;     /* S ratio of SLAG (per mole) */
float slagreact;     /* Base dissolution reactivity factor for SLAG */
long int DIFFCHdeficit=0,slaginit=0;   /* Deficit in CH due to SLAG reaction */
long int slagcum=0,chgone=0;
long int nch_slag=0;    /* number of CH consumed by SLAG reaction */
long int sulf_cur=0;
long int sulf_solid;
int *seed;       /* Random number seed */
char heatname[80],adianame[80],phname[80],ppsname[80],ptsname[80],phrname[80];
char chshrname[80],moviename[80],parname[80],micname[80];
char cmdnew[120],pHname[80],fileroot[80];
struct ants *headant,*tailant;
FILE *heatfile,*chsfile,*ptmpfile,*movfile,*pHfile,*micfile;
/* Variables for alkali predictions */
float pH_cur,totsodium,totpotassium,rssodium,rspotassium;
/* Array for whether pH influences phase solubility  -- added 2/12/02 */
float pHeffect[EMPTYP+1],pHfactor=0.0;
int pHactive,resatcyc,cshgeom;
/* Make conccaplus global to speed up execution and moles_syn_precip */
/* global to accumulate properly */
double conccaplus=0.0,moles_syn_precip=0.0,concsulfate=0.0;
int primevalues[6]={2,3,5,7,11,13};
int cshboxsize;              /* Box size for location of extra diffusing C-S-H */
```

```
/* Supplementary programs */
#include "ran1.c"             /* random number generation */
#include "burn3d.c"           /* percolation of porosity assessment */
#include "burnset.c"          /* set point assessment */
#include "parthyd.c"          /* particle hydration assessment */
#include "hydrealnew.c"       /* hydration execution */
#include "pHpred.c"           /* pore solution pH prediction */

/* routine to initialize values for solubilities, molar volumes, etc. */
/* Called by main program */
/* Calls no other routines */
void init()
{
        int i;
        float slagin,CHperslag;
        FILE *slagfile,*alkalifile;

        for(i=0;i<=EMPTYP;i++){

                creates[i]=0;
                soluble[i]=0;
                disprob[i]=0.0;
                disbase[i]=0.0;
                pHeffect[i]=0.0;
        }

        /* soluble [x] - flag indicating if phase x is soluble */
        /* disprob [x] - probability of dissolution (relative diss. rate) */
        gypabsprob=0.01;      /* One sulfate absorbed per 100 CSH units */
        /* Source is from Taylor's Cement Chemistry  for K and Na absorption */
        /* Note that for the first cycle, of the clinker phases only the */
        /* aluminates and gypsum are soluble */
        soluble[C4AF]=1;
        disprob[C4AF]=disbase[C4AF]=0.067/DISBIAS;
        creates[C4AF]=POROSITY;
        pHeffect[C4AF]=1.0;
        soluble[C3S]=0;
        disprob[C3S]=disbase[C3S]=0.7/DISBIAS;
        creates[C3S]=DIFFCSH;
        pHeffect[C3S]=1.0;
        soluble[C2S]=0;
        disprob[C2S]=disbase[C2S]=0.1/DISBIAS;
        creates[C2S]=DIFFCSH;
        pHeffect[C2S]=1.0;
        soluble[C3A]=1;
        /* increased back to 0.4 from 0.25 7/8/99 */
        disprob[C3A]=disbase[C3A]=0.4/DISBIAS;
        creates[C3A]=POROSITY;
        pHeffect[C3A]=1.0;
        soluble[GYPSUM]=1;
        /* Changed from 0.05 to 0.015  9/29/98 */
        /* Changed to 0.040 10/15/98 */
        /* back to 0.05 from 0.10 7/8/99 */
        /* from 0.05 to 0.02 4/4/00 */
        /* from 0.02 to 0.025 8/13/01 */
        disprob[GYPSUM]=disbase[GYPSUM]=0.025;
```

```
        /* dissolved gypsum distributed at random throughout microstructure */
        creates[GYPSUM]=POROSITY;
        soluble[GYPSUMS]=1;
        pHeffect[GYPSUM]=0.0;
        /* Changed from 0.05 to 0.015  9/29/98 */
        /* Changed to 0.020 10/15/98 */
        /* and also changed all sulfate based dissolution rates */
        disprob[GYPSUMS]=disbase[GYPSUMS]=disprob[GYPSUM];
        creates[GYPSUMS]=POROSITY;
        pHeffect[GYPSUMS]=0.0;
        soluble[ANHYDRITE]=1;
        /* set anhydrite dissolution at 4/5ths of that of gypsum */
        /* Source: Uchikawa et al., Cement and Concrete Research, 1984 */
        disprob[ANHYDRITE]=disbase[ANHYDRITE]=0.8*disprob[GYPSUM];
        /* dissolved anhydrite distributed at random throughout microstructure */
        creates[ANHYDRITE]=POROSITY;
        pHeffect[ANHYDRITE]=0.0;
        soluble[HEMIHYD]=1;
        /* set hemihydrate dissolution at 3 times that of gypsum */
        /* Source: Uchikawa et al., Cement and Concrete Research, 1984 */
        /* Changed to 1.5 times that of gypsum 6/1/00 */
        disprob[HEMIHYD]=disbase[HEMIHYD]=1.5*disprob[GYPSUM];
        /* dissolved hemihydrate distributed at random throughout microstructure */
        creates[HEMIHYD]=POROSITY;
        pHeffect[HEMIHYD]=0.0;
        /* CH soluble to allow for Ostwald ripening of crystals */
        soluble[CH]=1;
        disprob[CH]=disbase[CH]=0.5/DISBIAS;
        creates[CH]=DIFFCH;
        pHeffect[CH]=0.0;
        /* CaCO3 is only mildly soluble */
        soluble[CACO3]=1;
        disprob[CACO3]=disbase[CACO3]=0.10/DISBIAS;
        creates[CACO3]=DIFFCACO3;
        pHeffect[CACO3]=0.0;
        /* Slag is not truly soluble, but use its dissolution probability for
reaction probability */
        soluble[SLAG]=0;
        disprob[SLAG]=disbase[SLAG]=0.005/DISBIAS;
        creates[SLAG]=0;
        pHeffect[SLAG]=1.0;
        soluble[C3AH6]=1;
        disprob[C3AH6]=disbase[C3AH6]=0.01/DISBIAS; /* changed from 0.5 to 0.01
06.09.00 */
        creates[C3AH6]=POROSITY;
        pHeffect[C3AH6]=0.0;
        /* Ettringite is initially insoluble */
        soluble[ETTR]=0;
        /* Changed to 0.008 from 0.020  3/11/99 */
        disprob[ETTR]=disbase[ETTR]=0.008/DISBIAS;
        creates[ETTR]=DIFFETTR;
        pHeffect[ETTR]=0.0;
        /* Iron-rich ettringite is always insoluble */
        soluble[ETTRC4AF]=0;
        disprob[ETTRC4AF]=disbase[ETTRC4AF]=0.0;
        creates[ETTRC4AF]=ETTRC4AF;
        pHeffect[ETTRC4AF]=0.0;
```

```
/* calcium chloride is soluble */
soluble[CACL2]=1;
disprob[CACL2]=disbase[CACL2]=0.1/DISBIAS;
creates[CACL2]=DIFFCACL2;
pHeffect[CACL2]=0.0;
/* aluminosilicate glass is soluble */
soluble[ASG]=1;
disprob[ASG]=disbase[ASG]=0.2/DISBIAS;
creates[ASG]=DIFFAS;
pHeffect[ASG]=1.0;
/* calcium aluminodisilicate is soluble */
soluble[CAS2]=1;
disprob[CAS2]=disbase[CAS2]=0.2/DISBIAS;
creates[CAS2]=DIFFCAS2;
pHeffect[CAS2]=1.0;

/* establish molar volumes and heats of formation */
/* molar volumes are in cm^3/mole */
/* heats of formation are in kJ/mole */
/* See paper by Fukuhara et al., Cem. & Conc. Res., 11, 407-14, 1981. */
/* values for Porosity are those of water */
molarv[POROSITY]=18.068;
heatf[POROSITY]=(-285.83);
waterc[POROSITY]=1.0;
specgrav[POROSITY]=0.99707;

molarv[C3S]=71.129;
heatf[C3S]=(-2927.82);
waterc[C3S]=0.0;
specgrav[C3S]=3.21;
/* For improvement in chemical shrinkage correspondence */
/* Changed molar volume of C(1.7)-S-H(4.0) to 108 5/24/95 */
molarv[CSH]=108.;
heatf[CSH]=(-3283.0);
waterc[CSH]=4.0;
specgrav[CSH]=2.11;

molarv[CH]=33.1;
heatf[CH]=(-986.1);
waterc[CH]=1.0;
specgrav[CH]=2.24;

/* Assume that calcium carbonate is in the calcite form */
molarv[CACO3]=36.93;
waterc[CACO3]=0.0;
specgrav[CACO3]=2.71;
heatf[CACO3]=(-1206.92);

molarv[AFMC]=261.91;
waterc[AFMC]=11.0;
specgrav[AFMC]=2.17;
/* Need to fill in heat of formation at a later date */
heatf[AFMC]=(0.0);

molarv[C2S]=52.513;
heatf[C2S]=(-2311.6);
waterc[C2S]=0.0;
```

```
        specgrav[C2S]=3.28;

        molarv[C3A]=88.94;
        heatf[C3A]=(-3587.8);
        waterc[C3A]=0.0;
        specgrav[C3A]=3.038;

        molarv[GYPSUM]=74.21;
        heatf[GYPSUM]=(-2022.6);
        waterc[GYPSUM]=0.0;
        specgrav[GYPSUM]=2.32;
        molarv[GYPSUMS]=74.21;
        heatf[GYPSUMS]=(-2022.6);
        waterc[GYPSUMS]=0.0;
        specgrav[GYPSUMS]=2.32;

        molarv[ANHYDRITE]=52.16;
        heatf[ANHYDRITE]=(-1424.6);
        waterc[ANHYDRITE]=0.0;
        specgrav[ANHYDRITE]=2.61;

        molarv[HEMIHYD]=52.973;
        heatf[HEMIHYD]=(-1574.65);
        waterc[HEMIHYD]=0.0;
        specgrav[HEMIHYD]=2.74;

        molarv[C4AF]=130.29;
        heatf[C4AF]=(-5090.3);
        waterc[C4AF]=0.0;
        specgrav[C4AF]=3.73;

        molarv[C3AH6]=150.12;
        heatf[C3AH6]=(-5548.);
        waterc[C3AH6]=6.0;
        specgrav[C3AH6]=2.52;

/* Changed molar volume of FH3 to 69.8 (specific gravity of 3.06) 5/23/95 */
        molarv[FH3]=69.803;
        heatf[FH3]=(-823.9);
        waterc[FH3]=3.0;
        specgrav[FH3]=3.062;

        molarv[ETTRC4AF]=735.01;
        heatf[ETTRC4AF]=(-17539.0);
        waterc[ETTRC4AF]=26.0;
        specgrav[ETTRC4AF]=1.7076;
/* Changed molar volue of ettringite to 735 (spec. gr.=1.7076)  5/24/95 */
        molarv[ETTR]=735.01;
        heatf[ETTR]=(-17539.0);
        waterc[ETTR]=26.0;
        specgrav[ETTR]=1.7076;

        molarv[AFM]=312.82;
        heatf[AFM]=(-8778.0);
        /* Each mole of AFM which forms requires 12 moles of water, */
        /* two of which are supplied by gypsum in forming ETTR  */
        /* leaving 10 moles to be incorporated from free water */
```

```
        waterc[AFM]=10.0;
        specgrav[AFM]=1.99;

        molarv[CACL2]=51.62;
        heatf[CACL2]=(-795.8);
        waterc[CACL2]=0;
        specgrav[CACL2]=2.15;

        molarv[FREIDEL]=296.662;
        /* No data available for heat of formation */
        heatf[FREIDEL]=(0.0);
        /* 10 moles of H2O per mole of Freidel's salt */
        waterc[FREIDEL]=10.0;
        specgrav[FREIDEL]=1.892;

        /* Basic reaction is 2CH + ASG + 6H --> C2ASH8 (Stratlingite) */
        molarv[ASG]=49.9;
        /* No data available for heat of formation */
        heatf[ASG]=0.0;
        waterc[ASG]=0.0;
        specgrav[ASG]=3.247;

        molarv[CAS2]=100.62;
        /* No data available for heat of formation */
        heatf[CAS2]=0.0;
        waterc[CAS2]=0.0;
        specgrav[CAS2]=2.77;

        molarv[STRAT]=215.63;
        /* No data available for heat of formation */
        heatf[STRAT]=0.0;
        /* 8 moles of water per mole of stratlingite */
        waterc[STRAT]=8.0;
        specgrav[STRAT]=1.94;

        molarv[POZZ]=27.0;
        /* Use heat of formation of SiO2 (quartz) for unreacted pozzolan */
        /* Source- Handbook of Chemistry and Physics */
        heatf[POZZ]=-907.5;
        waterc[POZZ]=0.0;
        specgrav[POZZ]=2.22;

        /* Data for Pozzolanic CSH based on work of Atlassi, DeLarrard, */
        /* and Jensen */
        /* gives a chemical shrinkage of 0.2 g H2O/g CSF */
        /* heat of formation estimated based on heat release of */
        /* 780 J/g Condensed Silica Fume */
        /* MW is 191.8 g/mole */
        /* Changed molar volume to 101.81  3/10/97 */
        molarv[POZZCSH]=101.81;
        waterc[POZZCSH]=3.9;
        specgrav[POZZCSH]=1.884;
        heatf[POZZCSH]=(-2299.1);

/* Assume inert filler has same specific gravity and molar volume as SiO2 */
        molarv[INERT]=27.0;
        heatf[INERT]=0.0;
```

```
        waterc[INERT]=0.0;
        specgrav[INERT]=2.2;

        molarv[ABSGYP]=74.21;
        heatf[ABSGYP]=(-2022.6);
        waterc[ABSGYP]=0.0;
        specgrav[ABSGYP]=2.32;

        molarv[EMPTYP]=18.068;
        heatf[EMPTYP]=(-285.83);
        waterc[EMPTYP]=0.0;
        specgrav[EMPTYP]=0.99707;

        /* Read in values for alkali characteristics and */
        /* convert them to fractions from percentages */
        alkalifile=fopen("alkalichar.dat","r");
        fscanf(alkalifile,"%f",&totsodium);
        fscanf(alkalifile,"%f",&totpotassium);
        fscanf(alkalifile,"%f",&rssodium);
        fscanf(alkalifile,"%f",&rspotassium);
        fclose(alkalifile);
        totsodium/=100.;
        totpotassium/=100.;
        rssodium/=100.;
        rspotassium/=100.;
        /* Read in values for slag characteristics */
        slagfile=fopen("slagchar.dat","r");
        fscanf(slagfile,"%f",&slagin);
        fscanf(slagfile,"%f",&slagin);
        fscanf(slagfile,"%f",&slagin);
        specgrav[SLAG]=slagin;
        fscanf(slagfile,"%f",&slagin);
        specgrav[SLAGCSH]=slagin;
        fscanf(slagfile,"%f",&slagin);
        molarv[SLAG]=slagin;
        fscanf(slagfile,"%f",&slagin);
        molarv[SLAGCSH]=slagin;
        fscanf(slagfile,"%f",&slagcasi);
        fscanf(slagfile,"%f",&slaghydcasi);
        fscanf(slagfile,"%f",&siperslag);
        fscanf(slagfile,"%f",&slagin);
        waterc[SLAGCSH]=slagin*siperslag;
        fscanf(slagfile,"%f",&slagc3a);
        fscanf(slagfile,"%f",&slagreact);
        waterc[SLAG]=0.0;
        heatf[SLAG]=0.0;
        heatf[SLAGCSH]=0.0;
        fclose(slagfile);
        /* Compute slag probabilities as defined above */
        CHperslag=siperslag*(slaghydcasi-slagcasi)+3.*slagc3a;
        if(CHperslag<0.0){CHperslag=0.0;}

p2slag=(molarv[SLAG]+molarv[CH]*CHperslag+molarv[POROSITY]*(waterc[SLAGCSH]-
CHperslag+waterc[C3AH6]*slagc3a)-molarv[SLAGCSH]-
molarv[C3AH6]*slagc3a)/molarv[SLAG];
        p1slag=1.0-p2slag;
        p3slag=(molarv[SLAGCSH]/molarv[SLAG])-p1slag;
```

```
        p4slag=CHperslag*molarv[CH]/molarv[SLAG];
        p5slag=slagc3a*molarv[C3A]/molarv[SLAG];
        if(p5slag>1.0){
           p5slag=1.0;
           printf("Error in range of C3A/slag value...  reset to 1.0 \n");
        }
}

/* routine to check if a pixel located at (xck,yck,zck) is on an edge */
/* (in contact with pore space) in 3-D system */
/* Called by passone */
/* Calls no other routines */
int chckedge(xck,yck,zck)
        int xck,yck,zck;
{
        int edgeback,x2,y2,z2;
        int ip;

        edgeback=0;

        /* Check all six neighboring pixels */
        /* with periodic boundary conditions */
        for(ip=0;((ip<NEIGHBORS)&&(edgeback==0));ip++){

                x2=xck+xoff[ip];
                y2=yck+yoff[ip];
                z2=zck+zoff[ip];
                if(x2>=SYSIZE){x2=0;}
                if(x2<0){x2=SYSIZEM1;}
                if(y2>=SYSIZE){y2=0;}
                if(y2<0){y2=SYSIZEM1;}
                if(z2>=SYSIZE){z2=0;}
                if(z2<0){z2=SYSIZEM1;}
                if(mic [x2] [y2] [z2]==POROSITY){
                        edgeback=1;
                }
        }
        return(edgeback);
}

/* routine for first pass through microstructure during dissolution */
/* low and high indicate phase ID range to check for surface sites */
/* Called by dissolve */
/* Calls chckedge */
void passone(low,high,cycid,cshexflag)
        int low,high,cycid,cshexflag;
{
        int i,xid,yid,zid,phid,iph,edgef,phread,cshcyc;

        /* gypready used to determine if any soluble gypsum remains */
        if((low<=GYPSUM)&&(GYPSUM<=high)){
                gypready=0;
        }
        /* Zero out count for the relevant phases */
        for(i=low;i<=high;i++){
            count[i]=0;
        }
```

```
        /* Scan the entire 3-D microstructure */
        for(xid=0;xid<SYSIZE;xid++){
        for(yid=0;yid<SYSIZE;yid++){
        for(zid=0;zid<SYSIZE;zid++){

             phread=mic[xid][yid][zid];
             /* Update heat data and water consumed for solid CSH */
             if((cshexflag==1)&&(phread==CSH)){
                   cshcyc=cshage[xid][yid][zid];
                   heatsum+=heatf[CSH]/molarvcsh[cshcyc];
                   molesh2o+=watercsh[cshcyc]/molarvcsh[cshcyc];
             }
        /* Identify phase and update count */
        phid=60;
        for(i=low;((i<=high)&&(phid==60));i++){

                if(mic [xid][yid][zid]==i){
                        phid=i;
                        /* Update count for this phase */
                        count[i]+=1;
                        if((i==GYPSUM)||(i==GYPSUMS)){
                                gypready+=1;
                        }
                        /* If first cycle, then accumulate initial counts */
                        if((cycid==1)||((cycid==0)&&(ncyc==0))){
                                if(i==POROSITY){porinit+=1;}
                        /* Ordered in terms of likely volume fractions */
                        /* (largest to smallest) to speed execution */
                                else if(i==C3S){c3sinit+=1;}
                                else if(i==C2S){c2sinit+=1;}
                                else if(i==C3A){c3ainit+=1;}
                                else if(i==C4AF){c4afinit+=1;}

                                else if(i==GYPSUM){ncsbar+=1;}
                                else if(i==GYPSUMS){ncsbar+=1;}
                                else if(i==ANHYDRITE){anhinit+=1;}
                                else if(i==HEMIHYD){heminit+=1;}
                                else if(i==POZZ){nfill+=1;}
                                else if(i==SLAG){slaginit+=1;}
                                else if(i==ETTR){netbar+=1;}
                                else if(i==ETTRC4AF){netbar+=1;}
                        }
                }
        }

        if(phid!=60){
                /* If phase is soluble, see if it is in contact with porosity */
                if((cycid!=0)&&(soluble[phid]==1)){
                        edgef=chckedge(xid,yid,zid);
                        if(edgef==1){
/* Surface eligible species has an ID OFFSET greater than its original value */
                                mic [xid][yid][zid]+=OFFSET;
                        }
                }
        }
        }  /* end of zid */
        }  /* end of yid */
```

```
        }  /* end of xid */
}

/* routine to locate a diffusing CSH species near dissolution source */
/* at (xcur,ycur,zcur) */
/* Called by dissolve */
/* Calls no other routines */
int loccsh(xcur,ycur,zcur,extent)
        int xcur,ycur,zcur,extent;
{
            int xpmax,ypmax,effort,tries,xmod,ymod,zmod;
            struct ants *antnew;

            effort=0;    /* effort indicates if appropriate location found */
            tries=0;
            /* 500 tries in immediate vicinity */
            while((effort==0)&&(tries<500)){
                tries+=1;
                xmod=(-extent)+(int)((2.*(float)extent+1.)*ran1(seed));
                ymod=(-extent)+(int)((2.*(float)extent+1.)*ran1(seed));
                zmod=(-extent)+(int)((2.*(float)extent+1.)*ran1(seed));
                if(xmod>extent){xmod=extent;}
                if(ymod>extent){ymod=extent;}
                if(zmod>extent){zmod=extent;}
                xmod+=xcur;
                ymod+=ycur;
                zmod+=zcur;
                /* Periodic boundaries */
                if(xmod>=SYSIZE){xmod-=SYSIZE;}
                else if(xmod<0){xmod+=SYSIZE;}
                if(zmod>=SYSIZE){zmod-=SYSIZE;}
                else if(zmod<0){zmod+=SYSIZE;}
                if(ymod<0){ymod+=SYSIZE;}
                else if(ymod>=SYSIZE){ymod-=SYSIZE;}

                if(mic[xmod][ymod][zmod]==POROSITY){
                        effort=1;
                        mic[xmod][ymod][zmod]=DIFFCSH;
                        nmade+=1;
                        ngoing+=1;
                        /* Add this diffusing species to the linked list */
                        antnew=(struct ants *)malloc(sizeof(struct ants));
                        antnew->x=xmod;
                        antnew->y=ymod;
                        antnew->z=zmod;
                        antnew->id=DIFFCSH;
                        antnew->cycbirth=cyccnt;
                     /* Now connect this ant structure to end of linked list */
                        antnew->prevant=tailant;
                        tailant->nextant=antnew;
                        antnew->nextant=NULL;
                        tailant=antnew;
                }
        }
            return(effort);
}
```

```
/* routine to count number of pore pixels in a cube of size boxsize */
/* centered at (qx,qy,qz) */
/* Called by makeinert */
/* Calls no other routines */
int countbox(boxsize,qx,qy,qz)
        int boxsize,qx,qy,qz;
{
        int nfound,ix,iy,iz,qxlo,qxhi,qylo,qyhi,qzlo,qzhi;
        int hx,hy,hz,boxhalf;

        boxhalf=boxsize/2;
        nfound=0;
        qxlo=qx-boxhalf;
        qxhi=qx+boxhalf;
        qylo=qy-boxhalf;
        qyhi=qy+boxhalf;
        qzlo=qz-boxhalf;
        qzhi=qz+boxhalf;
        /* Count the number of requisite pixels in the 3-D cube box */
        /* using periodic boundaries */
        for(ix=qxlo;ix<=qxhi;ix++){
                hx=ix;
                if(hx<0){hx+=SYSIZE;}
                else if(hx>=SYSIZE){hx-=SYSIZE;}
        for(iy=qylo;iy<=qyhi;iy++){
                hy=iy;
                if(hy<0){hy+=SYSIZE;}
                else if(hy>=SYSIZE){hy-=SYSIZE;}
        for(iz=qzlo;iz<=qzhi;iz++){
                hz=iz;
                if(hz<0){hz+=SYSIZE;}
                else if(hz>=SYSIZE){hz-=SYSIZE;}
                /* Count if porosity, diffusing species, or empty porosity */
                if((mic [hx] [hy] [hz]<C3S)||(mic[hx] [hy] [hz]>ABSGYP)){
                        nfound+=1;
                  }
        }
        }
        }

        return(nfound);
}

/* routine to count number of special pixels in a cube of size boxsize */
/* centered at (qx,qy,qz) */
/* special pixels are those not belonging to one of the cement clinker, */
/* calcium sulfate, or pozzolanic mineral admixture phases */
/* Called by addrand */
/* Calls no other routines */
int countboxc(boxsize,qx,qy,qz)
        int boxsize,qx,qy,qz;
{
        int nfound,ix,iy,iz,qxlo,qxhi,qylo,qyhi,qzlo,qzhi;
        int hx,hy,hz,boxhalf;

        boxhalf=boxsize/2;
        nfound=0;
```

```
        qxlo=qx-boxhalf;
        qxhi=qx+boxhalf;
        qylo=qy-boxhalf;
        qyhi=qy+boxhalf;
        qzlo=qz-boxhalf;
        qzhi=qz+boxhalf;
        /* Count the number of requisite pixels in the 3-D cube box */
        /* using periodic boundaries */
        for(ix=qxlo;ix<=qxhi;ix++){
                hx=ix;
                if(hx<0){hx+=SYSIZE;}
                else if(hx>=SYSIZE){hx-=SYSIZE;}
        for(iy=qylo;iy<=qyhi;iy++){
                hy=iy;
                if(hy<0){hy+=SYSIZE;}
                else if(hy>=SYSIZE){hy-=SYSIZE;}
        for(iz=qzlo;iz<=qzhi;iz++){
                hz=iz;
                if(hz<0){hz+=SYSIZE;}
                else if(hz>=SYSIZE){hz-=SYSIZE;}
                /* Count if not cement clinker */
                if((mic [hx] [hy] [hz]<C3S)||(mic[hx] [hy] [hz]>POZZ)){
                        nfound+=1;
                  }
        }
        }
        }
        return(nfound);
}

/* routine to create ndesire pixels of empty pore space to simulate */
/* self-desiccation */
/* Called by dissolve */
/* Calls countbox */
void makeinert(ndesire)
        long int ndesire;
{
        struct togo *headtogo,*tailtogo,*newtogo,*lasttogo,*onetogo;
        long int idesire;
        int px,py,pz,placed,cntpore,cntmax;

        /* First allocate the first element of the linked list */
        headtogo=(struct togo *)malloc(sizeof(struct togo));
        headtogo->x=headtogo->y=headtogo->z=(-1);
        headtogo->npore=0;
        headtogo->nexttogo=NULL;
        headtogo->prevtogo=NULL;
        tailtogo=headtogo;
        cntmax=0;

        printf("In makeinert with %ld needed elements \n",ndesire);
        fflush(stdout);
        /* Add needed number of elements to the end of the list */
        for(idesire=2;idesire<=ndesire;idesire++){
                newtogo=(struct togo *)malloc(sizeof(struct togo));
                newtogo->npore=0;
                newtogo->x=newtogo->y=newtogo->z=(-1);
```

```
        tailtogo->nexttogo=newtogo;
        newtogo->prevtogo=tailtogo;
        tailtogo=newtogo;
}

/* Now scan the microstructure and rank the sites */
for(px=0;px<SYSIZE;px++){
for(py=0;py<SYSIZE;py++){
for(pz=0;pz<SYSIZE;pz++){
        if(mic[px][py][pz]==POROSITY){
                cntpore=countbox(cubesize,px,py,pz);
                if(cntpore>cntmax){cntmax=cntpore;}
                /* Store this site value at appropriate place in */
                /* sorted linked list */
                if(cntpore>(tailtogo->npore)){
                        placed=0;
                        lasttogo=tailtogo;
                        while(placed==0){
                                newtogo=lasttogo->prevtogo;
                                if(newtogo==NULL){
                                        placed=2;
                                }
                                else{
                                        if(cntpore<=(newtogo->npore)){
                                                placed=1;
                                        }
                                }
                                if(placed==0){
                                        lasttogo=newtogo;
                                }
                        }
                        onetogo=(struct togo *)malloc(sizeof(struct togo));
                        onetogo->x=px;
                        onetogo->y=py;
                        onetogo->z=pz;
                        onetogo->npore=cntpore;
                        /* Insertion at the head of the list */
                        if(placed==2){
                                onetogo->prevtogo=NULL;
                                onetogo->nexttogo=headtogo;
                                headtogo->prevtogo=onetogo;
                                headtogo=onetogo;
                        }
                            if(placed==1){
                                onetogo->nexttogo=lasttogo;
                                onetogo->prevtogo=newtogo;
                                lasttogo->prevtogo=onetogo;
                                newtogo->nexttogo=onetogo;
                        }
                        /* Eliminate the last element */
                        lasttogo=tailtogo;
                        tailtogo=tailtogo->prevtogo;
                        tailtogo->nexttogo=NULL;
                        free(lasttogo);
                }
        }
}
```

```
        }
        }

        /* Now remove the sites */
        /* starting at the head of the list */
        /* and deallocate all of the used memory */
        for(idesire=1;idesire<=ndesire;idesire++){
                px=headtogo->x;
                py=headtogo->y;
                pz=headtogo->z;
                if(px!=(-1)){
                        mic [px] [py] [pz]=EMPTYP;
                        count[POROSITY]-=1;
                        count[EMPTYP]+=1;
                }
                lasttogo=headtogo;
                headtogo=headtogo->nexttogo;
                free(lasttogo);
         }
        /* If only small cubes of porosity were found, then adjust */
        /* cubesize to have a more efficient search in the future */
        if(cubesize>CUBEMIN){
                if((2*cntmax)<(cubesize*cubesize*cubesize)){
                       cubesize-=2;
                }
           }
}


/* routine to add extra SLAG CSH when SLAG reacts */
/* SLAG located at (xpres,ypres,zpres) */
/* Called by dissolve */
/* Calls moveone and edgecnt */
void extslagcsh(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int check,sump,xchr,ychr,zchr,fchr,i1,plok,action,numnear;
        long int tries;
        int mstest,mstest2;

/* first try 6 neighboring locations until  */
/*    a) successful                         */
/*    b) all 6 sites are tried or           */
/*    c) 100 tries are made                 */
/* try to grow slag C-S-H as plates         */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=100)&&(fchr==0)&&(sump!=30030));i1++){

                /* determine location of neighbor (using periodic boundaries) */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                action=0;
                sump*=moveone(&xchr,&ychr,&zchr,&action,sump);
                if(action==0){printf("Error in value of action in extpozz \n");}
                check=mic[xchr][ychr][zchr];
```

```
                /* Determine the direction of the neighbor selected and */
                /* the plates possible for growth */
                if(xchr!=xpres){
                  mstest=1;
                  mstest2=2;
                }
                if(ychr!=ypres){
                  mstest=2;
                  mstest2=3;
                }
                if(zchr!=zpres){
                  mstest=3;
                  mstest2=1;
                }

                /* if neighbor is porosity, locate the SLAG CSH there */
                if(check==POROSITY){

      if((faces[xpres][ypres][zpres]==0)||(mstest==faces[xpres][ypres][zpres])||(ms
test2==faces[xpres][ypres][zpres])){
                        mic[xchr][ychr][zchr]=SLAGCSH;
                        faces[xchr][ychr][zchr]=faces[xpres][ypres][zpres];
                        count[SLAGCSH]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                  }
                }
        }

        /* if no neighbor available, locate SLAG CSH at random location */
        /* in pore space */
        tries=0;
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the extra SLAG CSH there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,SLAG,CSH,SLAGCSH);
                        /* Be sure that one neighboring species is CSH or */
                        /* SLAG material */
                        if((tries>5000)||(numnear<26)){
                              mic[xchr][ychr][zchr]=SLAGCSH;
                              count[SLAGCSH]+=1;
                              count[POROSITY]-=1;
                              fchr=1;
                        }
                }
        }
}
```

```
/* routine to implement a cycle of dissolution */
/* Called by main program */
/* Calls passone, loccsh, and makeinert */
void dissolve(cycle)
        int cycle;
{
        int nc3aext,ncshext,nchext,nfh3ext,ngypext,nanhext,plok,edgef;
        int nsum5,nsum4,nsum3,nsum2,nhemext,nsum6,nc4aext;
        int xpmax,ypmax,phid,phnew,plnew,cread;
        int i,xloop,yloop,zloop,ngood,ix1,iy1,xc,yc,valid,xc1,yc1;
        int iz1,zc,zc1,cycnew;
        long int ctest;
        int placed,cshrand,ntrycsh,maxsulfate,msface;
        long int ncshgo,nsurf,suminit;
        long int xext,nhgd,npchext,nslagc3a=0;
        float pdis,plfh3,fchext,fc3aext,fanhext,mass_now,mass_fa_now,tot_mass;
        float dfact,dfact1,molesdh2o,h2oinit,heat4,fhemext,fc4aext,heatfill;
        float pconvert,pc3scsh,pc2scsh,calcx,calcy,calcz,tdisfact;
        float frafm,frettr,frhyg,frtot,mc3ar,mc4ar,p3init;
        FILE *phfile,*difffile;
        struct ants *antadd;

        /* Initialize variables */
        nmade=0;
        npchext=ncshgo=cshrand=0; /* counter for number of csh diffusing species */
                        /* to be located at random locations in microstructure */

        heat_old=heat_new; /* new and old values for heat released */

        /* Initialize dissolution and phase counters */
        nsurf=0;
        for(i=0;i<=EMPTYP;i++){
                discount[i]=0;
                count[i]=0;
        }

        /* Pass one- highlight all edge points which are soluble */
        soluble[C3AH6]=0;
        heatsum=molesh2o=0.0;
        passone(0,EMPTYP,cycle,1);
        printf("Returned from passone \n");
        fflush(stdout);

        sulf_solid=count[GYPSUM]+count[GYPSUMS]+count[HEMIHYD]+count[ANHYDRITE];
        /* If first cycle, then determine all mixture proportions based */
        /* on user input and original microstructure */
        if(cycle==1){
                /* Mass of cement in system */
                cemmass=(specgrav[C3S]*(float)count[C3S]+specgrav[C2S]*
                (float)count[C2S]+specgrav[C3A]*(float)count[C3A]+
                specgrav[C4AF]*(float)count[C4AF]);
      /*    +specgrav[GYPSUM]*
                (float)count[GYPSUM]+specgrav[ANHYDRITE]*(float)
            count[ANHYDRITE]+specgrav[HEMIHYD]*(float)count[HEMIHYD]); */
                cemmasswgyp=(specgrav[C3S]*(float)count[C3S]+specgrav[C2S]*
                (float)count[C2S]+specgrav[C3A]*(float)count[C3A]+
                specgrav[C4AF]*(float)count[C4AF]+specgrav[GYPSUM]*
```

```
            (float)count[GYPSUM]+specgrav[ANHYDRITE]*(float)
            count[ANHYDRITE]+specgrav[HEMIHYD]*(float)count[HEMIHYD]);
            flyashmass=(specgrav[ASG]*(float)count[ASG]+specgrav[CAS2]*
            (float)count[CAS2]+specgrav[POZZ]*(float)count[POZZ]);
            CH_mass=specgrav[CH]*(float)count[CH];
            /* Total mass in system neglecting single aggregate */
            tot_mass=cemmass+(float)count[POROSITY]+specgrav[INERT]*
            (float)count[INERT]+specgrav[CACL2]*(float)count[CACL2]+
            specgrav[ASG]*(float)count[ASG]+
            specgrav[SLAG]*(float)count[SLAG]+
            specgrav[HEMIHYD]*(float)count[HEMIHYD]+
            specgrav[ANHYDRITE]*(float)count[ANHYDRITE]+
            specgrav[CAS2]*(float)count[CAS2]+
            specgrav[CSH]*(float)count[CSH]+
            specgrav[GYPSUM]*(float)count[GYPSUM]+
            specgrav[ANHYDRITE]*(float)count[ANHYDRITE]+
            specgrav[HEMIHYD]*(float)count[HEMIHYD]+
            specgrav[GYPSUMS]*(float)count[GYPSUMS]+
            specgrav[POZZ]*(float)count[POZZ]+CH_mass;
            /* water-to-cement ratio */
            if(cemmass!=0.0){
                  w_to_c=(float)count[POROSITY]/(cemmass+
                  specgrav[GYPSUM]*(float)count[GYPSUM]+
                  specgrav[ANHYDRITE]*(float)count[ANHYDRITE]+
                  specgrav[HEMIHYD]*(float)count[HEMIHYD]);
            }
            else{
                  w_to_c=0.0;
            }
/* totfract is the total cement volume count including calcium sulfates */
/* fractwithfill is the total count of cement and solid fillers and */
/*    mineral admixtures   DPB- 10/04 */
       totfract=count[C3S]+count[C2S];
       totfract+=(count[C3A]+count[C4AF]);
       totfract+=(count[GYPSUM]+count[ANHYDRITE]+count[HEMIHYD]);
       fractwithfill=totfract+count[CACO3]+count[SLAG]+count[INERT];
       fractwithfill+=count[POZZ]+count[CAS2]+count[ASG]+count[CACL2];
       totfract/=(float)SYSIZE;
       totfract/=(float)SYSIZE;
       totfract/=(float)SYSIZE;
       fractwithfill/=(float)SYSIZE;
       fractwithfill/=(float)SYSIZE;
       fractwithfill/=(float)SYSIZE;
       /* Adjust masses for presence of aggregates in concrete */
       mass_water=(1.-mass_agg)*(float)count[POROSITY]/tot_mass;
       mass_CH=(1.-mass_agg)*CH_mass/tot_mass;
       /* pozzolan-to-cement ratio */
       if(cemmass!=0.0){
             s_to_c=(float)(count[INERT]*specgrav[INERT]+
             count[CACL2]*specgrav[CACL2]+count[ASG]*specgrav[ASG]+
             count[CAS2]*specgrav[CAS2]+
             count[SLAG]*specgrav[SLAG]+
             count[POZZ]*specgrav[POZZ])/cemmass;
       }
       else{
             s_to_c=0.0;
       }
```

```
                /* Conversion factor to kJ/kg for heat produced */
            if(cemmass!=0.0){

      heatfill=(float)(count[INERT]+count[SLAG]+count[POZZ]+count[CACL2]+count[ASG]
+count[CAS2])/cemmass;
            }
            else{
                  heatfill=0.0;
            }
            if(w_to_c>0.01){
                      heat_cf=0.001*(0.3125+w_to_c+heatfill);
            }
            else{
                  /* Need volume per 1 gram of silica fume */

      heat_cf=0.001*((1./specgrav[POZZ])+(float)(count[POROSITY]+count[CH]+count[IN
ERT])/(specgrav[POZZ]*(float)count[POZZ]));
            }
                mass_fill_pozz=(1.-
                mass_agg)*(float)(count[POZZ]*specgrav[POZZ])/tot_mass;
                mass_fill=(1.-mass_agg)*(float)(count[INERT]*specgrav[INERT]+
                count[ASG]*specgrav[ASG]+count[SLAG]*specgrav[SLAG]+
                count[CAS2]*specgrav[CAS2]+count[POZZ]*specgrav[POZZ]+
                count[CACL2]*specgrav[CACL2])/tot_mass;
                printf("Calculated w/c is %.4f\n",w_to_c);
                printf("Calculated s/c is %.4f \n",s_to_c);
                printf("Calculated heat conversion factor is %f \n",heat_cf);
  printf("Calculated mass fractions of water and filler are %.4f  and %.4f \n",
                mass_water,mass_fill);
        }

        molesdh2o=0.0;
        alpha=0.0;      /* degree of hydration */
        /* heat4 contains measured heat release for C4AF hydration from  */
        /* Fukuhara et al., Cem. and Conc. Res. article */
        heat4=0.0;
        mass_now=0.0;    /* total cement mass corrected for hydration */
        suminit=c3sinit+c2sinit+c3ainit+c4afinit;
        /* suminit=c3sinit+c2sinit+c3ainit+c4afinit+ncsbar+anhinit+heminit; */
        /* ctest is number of gypsum likely to form ettringite */
        /* 1 unit of C3A can react with 2.5 units of Gypsum */
        ctest=count[DIFFGYP];
        printf("ctest is %ld\n",ctest);
        fflush(stdout);
        if((float)ctest>(2.5*(float)(count[DIFFC3A]+count[DIFFC4A]))){
                 ctest=2.5*(float)(count[DIFFC3A]+count[DIFFC4A]);
        }
        for(i=0;i<=EMPTYP;i++){
                if((i!=0)&&(i<=ABSGYP)&&(i!=INERTAGG)&&(i!=CSH)){
                        heatsum+=(float)count[i]*heatf[i]/molarv[i];
                        /* Tabulate moles of H2O consumed by reactions so far */
                        molesh2o+=(float)count[i]*waterc[i]/molarv[i];
                }
                /* assume that all C3A which can, does form ettringite */
                if(i==DIFFC3A){
                      heatsum+=((float)count[DIFFC3A]-
                      (float)ctest/2.5)*heatf[C3A]/molarv[C3A];
```

```
    }
    /* assume that all C4A which can, does form ettringite */
    if(i==DIFFC4A){
          heatsum+=((float)count[DIFFC4A]-
          (float)ctest/2.5)*heatf[C4AF]/molarv[C4AF];
    }
    /* assume all gypsum which can, does form ettringite */
    /* rest will remain as gypsum */
    if(i==DIFFGYP){
           heatsum+=(float)(count[DIFFGYP]-
           ctest)*heatf[GYPSUM]/molarv[GYPSUM];
      /* 3.3 is the molar expansion from GYPSUM to ETTR */
           heatsum+=(float)ctest*3.30*heatf[ETTR]/molarv[ETTR];
           molesdh2o+=(float)ctest*3.30*waterc[ETTR]/molarv[ETTR];
    }
      else if(i==DIFFCH){
           heatsum+=(float)count[DIFFCH]*heatf[CH]/molarv[CH];
           molesdh2o+=(float)count[DIFFCH]*waterc[CH]/molarv[CH];
    }
    else if(i==DIFFFH3){
           heatsum+=(float)count[DIFFFH3]*heatf[FH3]/molarv[FH3];
           molesdh2o+=(float)count[DIFFFH3]*waterc[FH3]/molarv[FH3];
    }
    else if(i==DIFFCSH){
      /* use current CSH properties */
            heatsum+=(float)count[DIFFCSH]*heatf[CSH]/molarvcsh[cycle];
     molesdh2o+=(float)count[DIFFCSH]*watercsh[cycle]/molarvcsh[cycle];
    }
    else if(i==DIFFETTR){
            heatsum+=(float)count[DIFFETTR]*heatf[ETTR]/molarv[ETTR];
           molesdh2o+=(float)count[DIFFETTR]*waterc[ETTR]/molarv[ETTR];
    }
    else if(i==DIFFCACL2){
           heatsum+=(float)count[DIFFCACL2]*heatf[CACL2]/molarv[CACL2];
        molesdh2o+=(float)count[DIFFCACL2]*waterc[CACL2]/molarv[CACL2];
    }
    else if(i==DIFFAS){
            heatsum+=(float)count[DIFFAS]*heatf[ASG]/molarv[ASG];
            molesdh2o+=(float)count[DIFFAS]*waterc[ASG]/molarv[ASG];
    }
    else if(i==DIFFCAS2){
            heatsum+=(float)count[DIFFCAS2]*heatf[CAS2]/molarv[CAS2];
           molesdh2o+=(float)count[DIFFCAS2]*waterc[CAS2]/molarv[CAS2];
    }
/* assume that all diffusing anhydrite leads to gypsum formation */
    else if(i==DIFFANH){
         heatsum+=(float)count[DIFFANH]*heatf[GYPSUMS]/molarv[GYPSUMS];
         /* 2 moles of water per mole of gypsum formed */
         molesdh2o+=(float)count[DIFFANH]*2.0/molarv[GYPSUMS];
    }
/* assume that all diffusing hemihydrate leads to gypsum formation */
    else if(i==DIFFHEM){
         heatsum+=(float)count[DIFFHEM]*heatf[GYPSUMS]/molarv[GYPSUMS];
         /* 1.5 moles of water per mole of gypsum formed */
         molesdh2o+=(float)count[DIFFHEM]*1.5/molarv[GYPSUMS];
    }
    else if(i==C3S){
```

```
     alpha+=(float)(c3sinit-count[C3S]);
     mass_now+=specgrav[C3S]*(float)count[C3S];
     heat4+=.517*(float)(c3sinit-count[C3S])*specgrav[C3S];
}
else if(i==C2S){
     alpha+=(float)(c2sinit-count[C2S]);
     mass_now+=specgrav[C2S]*(float)count[C2S];
     heat4+=.262*(float)(c2sinit-count[C2S])*specgrav[C2S];
}
else if(i==C3A){
     alpha+=(float)(c3ainit-count[C3A]);
     mass_now+=specgrav[C3A]*(float)count[C3A];
     mc3ar=(c3ainit-(float)count[C3A])/molarv[C3A];
     mc4ar=(c4afinit-(float)count[C4AF])/molarv[C4AF];
     if((mc3ar+mc4ar)>0.0){
  frhyg=(mc3ar/(mc3ar+mc4ar))*(float)count[C3AH6]/molarv[C3AH6];
     }
     else{
              frhyg=0.0;
     }
     frettr=(float)count[ETTR]/molarv[ETTR];
     frafm=3*(float)count[AFM]/molarv[AFM];
     frtot=frafm+frettr+frhyg;
     if(frtot>0.0){
        frettr/=frtot;
        frafm/=frtot;
        frhyg/=frtot;
        heat4+=frafm*1.144*(float)(c3ainit-
              count[C3A])*specgrav[C3A];
        heat4+=frhyg*0.908*(float)(c3ainit-
              count[C3A])*specgrav[C3A];
        heat4+=frettr*1.672*(float)(c3ainit-
              count[C3A])*specgrav[C3A];
     }
}
else if(i==C4AF){
        alpha+=(float)(c4afinit-count[C4AF]);
        mass_now+=specgrav[C4AF]*(float)count[C4AF];
        mc3ar=(c3ainit-(float)count[C3A])/molarv[C3A];
        mc4ar=(c4afinit-(float)count[C4AF])/molarv[C4AF];
        if((mc3ar+mc4ar)>0.0){
  frhyg=(mc4ar/(mc3ar+mc4ar))*(float)count[C3AH6]/molarv[C3AH6];
        }
        else{
              frhyg=0.0;
        }
        frettr=(float)count[ETTRC4AF]/molarv[ETTRC4AF];
        frtot=frettr+frhyg;
        if(frtot>0.0){
        frettr/=frtot;
        frhyg/=frtot;
        heat4+=frhyg*.418*(float)(c4afinit-
              count[C4AF])*specgrav[C4AF];
        heat4+=frettr*.725*(float)(c4afinit-
              count[C4AF])*specgrav[C4AF];
        }
}
```

```
/*              else if(i==GYPSUM){
                        alpha+=(float)(ncsbar-count[GYPSUM]);
                        mass_now+=specgrav[GYPSUM]*(float)count[GYPSUM];
                }   */
                  /* 0.187 kJ/g anhydrite for anhydrite --> gypsum conversion */
                  else if(i==ANHYDRITE){
                       /*  alpha+=(float)(anhinit-count[ANHYDRITE]);
                       mass_now+=specgrav[ANHYDRITE]*(float)count[ANHYDRITE]; */
                       heat4+=.187*(float)(anhinit-
                        count[ANHYDRITE])*specgrav[ANHYDRITE];
                  /* 2 moles of water consumed per mole of anhydrite reacted */
                       molesh2o+=(float)(anhinit-
                        count[ANHYDRITE])*2.0/molarv[ANHYDRITE];
                }
            /* 0.132 kJ/g hemihydrate for hemihydrate-->gypsum conversion */
                else if(i==HEMIHYD){
/*                     alpha+=(float)(heminit-count[HEMIHYD]);
                       mass_now+=specgrav[HEMIHYD]*(float)count[HEMIHYD]; */
                       heat4+=.132*(float)(heminit-
                        count[HEMIHYD])*specgrav[HEMIHYD];
                  /* 1.5 moles of water consumed per mole of anhydrite reacted */
                       molesh2o+=(float)(heminit-
                        count[HEMIHYD])*1.5/molarv[HEMIHYD];
                }
          }
      mass_fa_now = specgrav[ASG]*(float)count[ASG];
      mass_fa_now += specgrav[CAS2]*(float)count[CAS2];
      mass_fa_now += specgrav[POZZ]*(float)count[POZZ];
      if(suminit!=0){
              alpha=alpha/(float)suminit;
      }
      else{
            alpha=0.0;
      }
        /* Current degree of hydration on a mass basis */
      if(cemmass!=0.0){
              alpha_cur=1.0-(mass_now/cemmass);
      }
      else{
            alpha_cur=0.0;
      }
      if(flyashmass!=0.0){
              alpha_fa_cur=1.0-(mass_fa_now/flyashmass);
      }
      else{
            alpha_fa_cur=0.0;
      }
      h2oinit=(float)porinit/molarv[POROSITY];

      /* Assume 780 J/g S for pozzolanic reaction */
      /* Each unit of silica fume consumes 1.35 units of CH, */
      /* so divide npr by 1.35 to get silca fume which has reacted */
      heat4+=0.78*((float)npr/1.35)*specgrav[POZZ];

      /* Assume 800 J/g S for slag reaction */
      /* Seems reasonable with measurements of Biernacki and Richardson */
      heat4+=0.8*((float)nslagr)*specgrav[SLAG];
```

```
      /* Assume 800 J/g AS for stratlingite formation (DeLarrard) */
      /* Each unit of AS consumes 1.3267 units of CH, */
      /* so divide nasr by 1.3267 to get ASG which has reacted */
      heat4+=0.80*((float)nasr/1.3267)*specgrav[ASG];

      /* Should be additional code here for heat release due to CAS2 to */
      /* stratlingite conversion, but data unavailable at this time */

      /* Adjust heat sum for water left in system */
      water_left=(long int)((h2oinit-molesh2o)*molarv[POROSITY]+0.5);
      countkeep=count[POROSITY];
      heatsum+=(h2oinit-molesh2o-molesdh2o)*heatf[POROSITY];
      if(cyccnt==0){
           heatfile=fopen(heatname,"w");
  fprintf(heatfile,"Cycle time(h) alpha_vol alpha_mass heat4(kJ/kg_solid) Gsratio2
G-s_ratio\n");
           fclose(heatfile);
      }
      heat_new=heat4;     /* use heat4 for all adiabatic calculations */
                            /* due to best agreement with calorimetry data */
      if(cyccnt==0){
        chsfile=fopen(chshrname,"w");
  fprintf(chsfile,"Cycle  time(h) alpha_mass  Chemical shrinkage (ml/g cement)\n");
        fclose(chsfile);
      }
      chs_new=((float)(count[EMPTYP]+count[POROSITY]-water_left)*heat_cf/1000.);
/*    if((molesh2o>h2oinit)&&(sealed==1)){  */
      if(((water_left+water_off)<0)&&(sealed==1)){
                printf("All water consumed at cycle %d \n",cyccnt);
                fflush(stdout);
                exit(1);
      }
      /* Attempt to create empty porosity to account for self-desiccation */
      if((sealed==1)&&((count[POROSITY]-water_left)>0)){
                poretodo=(count[POROSITY]-pore_off)-(water_left-water_off);
                poretodo-=slagemptyp;
                if(poretodo>0){
                      makeinert(poretodo);
                      poregone+=poretodo;
                }
      }
      /* Output phase counts */
      /* phfile for reactant and product phases */
      if(cyccnt==0){
            phfile=fopen(phname,"w");
         fprintf(phfile,"Cycle Porosity C3S C2S C3A C4AF GYPSUM HEMIHYD ANHYDRITE
POZZ INERT SLAG ASG CAS2 CH CSH C3AH6 ETTR ETTRC4AF AFM FH3 POZZCSH SLAGCSH CACL2
FREIDEL STRAT GYPSUMS CACO3 AFMC AGG ABSGYP EMPTYP water_left \n");
      }
      else{
            phfile=fopen(phname,"a");
      }
      fprintf(phfile,"%d ",cyccnt);
            for(i=0;i<=EMPTYP;i++){
            if((i<DIFFCSH)||(i>=EMPTYP)){
                      fprintf(phfile,"%ld ",count[i]);
```

```
      }
      printf("%ld ",count[i]);
}
printf("\n");
fprintf(phfile,"%ld\n",water_left);
fclose(phfile);

if(cycle==0){
          return;
}
cyccnt+=1;
printf("Cycle %d \n",cyccnt);
fflush(stdout);
/* Update current volume count for CH */
chold=chnew;
chnew=count[CH];

/* See if ettringite is soluble yet */
/* Gypsum 80% consumed, changed 06.09.00 from 90% to 80% */
/* Gypsum 75% consumed, changed 09.09.01 from 80% to 75% */
/* or system temperature exceeds 70 C */
if(((ncsbar+anhinit+heminit)!=0.0)||(temp_cur>=70.0)){
/* Account for all sulfate sources and forms */
if((soluble[ETTR]==0)&&((temp_cur>=70.0)||(count[AFM]!=0)||
((((float)count[GYPSUM]+1.42*(float)count[ANHYDRITE]+1.4*
(float)count[HEMIHYD]+(float)count[GYPSUMS])/((float)ncsbar+
1.42*(float)anhinit+1.4*(float)heminit+((float)netbar/3.30)))<0.25))){
          soluble[ETTR]=1;
          printf("Ettringite is soluble beginning at cycle %d \n",cycle);
          /* identify all new soluble ettringite */
          passone(ETTR,ETTR,2,0);
}
} /* end of soluble ettringite test */

/* Adjust ettringite solubility */
/* if too many ettringites already in solution */
if(count[DIFFETTR]>DETTRMAX){
          disprob[ETTR]=0.0;
}
else{
          disprob[ETTR]=disbase[ETTR];
}
/* Adjust CaCl2 solubility */
/* if too many CaCl2 already in solution */
if(count[DIFFCACL2]>DCACL2MAX){
          disprob[CACL2]=0.0;
}
else{
          disprob[CACL2]=disbase[CACL2];
}
/* Adjust CaCO3 solubility */
/* if too many CaCO3 already in solution */
if((count[DIFFCACO3]>DCACO3MAX)&&(soluble[ETTR]==0)){
          disprob[CACO3]=0.0;
}
else if(count[DIFFCACO3]>(4*DCACO3MAX)){
          disprob[CACO3]=0.0;
```

```
	}
	else{
		disprob[CACO3]=disbase[CACO3];
	}
	/* Adjust solubility of CH */
	/* based on amount of CH currently diffusing */
	/* Note that CH is always soluble to allow some */
	/* Ostwald ripening of the CH crystals */
	if((float)count[DIFFCH]>=CHCRIT){
		disprob[CH]=disbase[CH]*CHCRIT/(float)count[DIFFCH];
	}
	else{
		disprob[CH]=disbase[CH];
	}
	/* Adjust solubility of CH for temperature */
	/* Fit to data provided in Taylor, Cement Chemistry */
	/* Scale to a reference temperature of 25 C */
	/* and adjust based on availability of pozzolan */
	printf("CH dissolution probability goes from %f ",disprob[CH]);
	disprob[CH]*=((A0_CHSOL-A1_CHSOL*temp_cur)/(A0_CHSOL-A1_CHSOL*25.0));
	if((ppozz>0.0)&&(nfill>0)){
	      disprob[CH]*=ppozz/PPOZZ;

	}
	printf("to %f \n",disprob[CH]);

	/* Adjust solubility of ASG and CAS2 phases */
	/* based on pH rise during hydration */
	/* To be added at a later date */
	disprob[ASG]=disbase[ASG];
	disprob[CAS2]=disbase[CAS2];
	/* Address solubility of C3AH6 */
	/* If lots of gypsum or reactive ettringite, allow C3AH6 to dissolve */
	/* to generate diffusing C3A species */
	if(((count[GYPSUM]+count[GYPSUMS])>(int)(((float)ncsbar+
	1.42*(float)anhinit+1.4*(float)heminit)*0.05))
	||(count[ETTR]>500)){
		soluble[C3AH6]=1;
		passone(C3AH6,C3AH6,2,0);
		/* Base C3AH6 solubility on maximum sulfate in solution */
		/* from gypsum or ettringite available for dissolution */

		/* The more the sulfate, the higher this solubility should be */
		maxsulfate=count[DIFFGYP];
		if((maxsulfate<count[DIFFETTR])&&(soluble[ETTR]==1)){
			maxsulfate=count[DIFFETTR];
		}
/* Adjust C3AH6 solubility based on potential gypsum which will dissolve */
		if(maxsulfate<(int)((float)gypready*disprob[GYPSUM]*
		(float)count[POROSITY]/1000000.)){
			maxsulfate=(int)((float)gypready*disprob[GYPSUM]*
			(float)count[POROSITY]/1000000.);
		}
		if(maxsulfate>0){
		      disprob[C3AH6]=disbase[C3AH6]*(float)maxsulfate/C3AH6CRIT;
		      if(disprob[C3AH6]>0.5){disprob[C3AH6]=0.5;}
		}
```

```
                else{
                        disprob[C3AH6]=disbase[C3AH6];
                }
        }
        else{
                soluble[C3AH6]=0;
        }

        /* See if silicates are soluble yet */

if((soluble[C3S]==0)&&((cycle>1)||(count[ETTR]>0)||(count[AFM]>0)||(count[ETTRC4AF]
>0))){
                soluble[C2S]=1;
                soluble[C3S]=1;
                /* identify all new soluble silicates */
                passone(C3S,C2S,2,0);
      } /* end of soluble silicate test */
      /* Adjust solubility of C3S and C2S with CSH concentration */
      /* for simulation of induction period */

      tdisfact=A0_CHSOL-temp_cur*A1_CHSOL;
/*    printf("tdisfact is %f\n",tdisfact);
      fflush(stdout); */

   /* Calculation of cs_acc; acceleration of C3S and C2S reaction by CaSO4 */
   /* Calculation of ca_acc; acceleration of C3A and C4AF reaction by CaSO4 */
   /* November 2004 --- modified to be on a sulfate per unit initial cement */
   /*    per unit porosity basis */
   /*  Try using current porosity count to see if this helps in initial */
   /*   hydration rates of sealed vs. saturated  */
   pfract=(float)count[POROSITY]/(float)(SYSIZE*SYSIZE*SYSIZE);
   if((ncsbar+anhinit+heminit)==0.0){
                cs_acc=1.0;
                ca_acc=1.0;
                dismin_c3a=5.0*DISMIN_C3A_0;
                dismin_c4af=5.0*DISMIN_C4AF_0;
   }
   else{
        sulf_conc=sulf_cur*tfractw05*pfractw05/totfract/pfract;
        if(sulf_conc<10.0){
                cs_acc=1.0;
                ca_acc=1.0;
                dismin_c3a=DISMIN_C3A_0;
                dismin_c4af=DISMIN_C4AF_0;
        }
        else if(sulf_conc<20.0){
                cs_acc=1.0+((sulf_conc)-10.0)/10.0;
                ca_acc=1.0;
                dismin_c3a=DISMIN_C3A_0;
                dismin_c4af=DISMIN_C4AF_0;
      }
      else{
                cs_acc=1.0+(double)log10(sulf_conc-10.0);
                ca_acc=1.0;
                dismin_c3a=(6.0-(double)log10(sulf_conc))*DISMIN_C3A_0;
                dismin_c4af=(6.0-(double)log10(sulf_conc))*DISMIN_C4AF_0;
                if(dismin_c3a<DISMIN_C3A_0){dismin_c3a=DISMIN_C3A_0;}
```

```
                if(dismin_c4af<DISMIN_C4AF_0){dismin_c4af=DISMIN_C4AF_0;}
      }
   }

      /* Suggest change WCSCALE/w_to_c to (0.3125+WCSCALE)/(0.3125+w_to_c) */
      /* to have the induction period scaling depend on volume of CSH */
      /* produced per volume (not mass) of cement */
/*
dfact=tdisfact*((float)count[CSH]/((float)CSHSCALE*(0.3125+WCSCALE)/(w_to_c+0.3125)
))*((float)count[CSH]/((float)CSHSCALE*(0.3125+WCSCALE)/(w_to_c+0.3125)))*cs_acc;*/
      /* October 2004 --- changed to truly scale with volume of cement in */
      /* system for both plain portland cements and filled systems */

dfact=tdisfact*((float)count[CSH]/((float)CSHSCALE*surffract*totfract/tfractw04))*(
(float)count[CSH]/((float)CSHSCALE*surffract*totfract/tfractw04))*cs_acc;
       disprob[C3S]=DISMIN+dfact*disbase[C3S];
       disprob[C2S]=DISMIN2+dfact*disbase[C2S];
       if(disprob[C3S]>(1.*disbase[C3S])){disprob[C3S]=(1.*disbase[C3S]);}
       if(disprob[C2S]>(1.*disbase[C2S])){disprob[C2S]=(1.*disbase[C2S]);}

       /* Also adjust slag and fly ash dissolution rates here */
       /* Really slow down initial slag and fly ash dissolutions */
       /* Ultimately should be linked to pH of pore solution, most likely */
       disprob[SLAG]=slagreact*(DISMINSLAG+dfact*disbase[SLAG])/10.0;

if(disprob[SLAG]>(slagreact*disbase[SLAG])){disprob[SLAG]=(slagreact*disbase[SLAG])
;}
       if(disprob[C3S]==disbase[C3S]){disprob[SLAG]=slagreact*disbase[SLAG];}
       disprob[ASG]=DISMINASG+dfact*disbase[ASG]/5.0;
       if(disprob[ASG]>(1.*disbase[ASG])){disprob[ASG]=(1.*disbase[ASG]);}
       if(disprob[C3S]==disbase[C3S]){disprob[ASG]=disbase[ASG];}
       disprob[CAS2]=DISMINCAS2+dfact*disbase[CAS2]/5.0;
       if(disprob[CAS2]>(1.*disbase[CAS2])){disprob[CAS2]=(1.*disbase[CAS2]);}
       if(disprob[C3S]==disbase[C3S]){disprob[CAS2]=disbase[CAS2];}
       /* Adjust CAS2 solubility */
       /* if too many CAS2 already in solution */
       if(count[DIFFCAS2]>DCAS2MAX){
                disprob[CAS2]=0.0;
       }
    printf("Silicate probabilities: %f %f\n",disprob[C3S],disprob[C2S]);
    fflush(stdout);
       /* Assume that aluminate dissolution controlled by formation */
       /* of impermeable layer proportional to CSH concentration */
       /* if sulfates are present in the system */

/*
dfact1=tdisfact*((float)count[CSH]/CSHSCALE)*((float)count[CSH]/CSHSCALE)*ca_acc;
*/
    if((ncsbar+heminit+anhinit)>1000){
/*
dfact1=tdisfact*((float)count[CSH]/((float)CSHSCALE*(0.3125+WCSCALE)/(0.3125+w_to_c
)))*((float)count[CSH]/((float)CSHSCALE*(0.3125+WCSCALE)/(0.3125+w_to_c)))*ca_acc;
*/
      /* October 2004 --- changed to truly scale with volume of cement in */
      /* system for both plain portland cements and filled systems */
```

```
dfact1=tdisfact*((float)count[CSH]/((float)CSHSCALE*surffract*totfract/tfractw04))*
((float)count[CSH]/((float)CSHSCALE*surffract*totfract/tfractw04))*ca_acc;
    disprob[C3A]=dismin_c3a+dfact1*disbase[C3A];
    disprob[C4AF]=dismin_c4af+dfact1*disbase[C4AF];
    if(disprob[C3A]>(1.*disbase[C3A])){disprob[C3A]=(1.*disbase[C3A]);}
    if(disprob[C4AF]>(1.*disbase[C4AF])){disprob[C4AF]=(1.*disbase[C4AF]);}

    /* Location to add in dissolution reduction in calcium sulfate phases */
    /* if needed */
    disprob[GYPSUM]=(disbase[GYPSUM]/15.)+dfact1*disbase[GYPSUM];
    if(disprob[GYPSUM]>(disbase[GYPSUM])){disprob[GYPSUM]=(disbase[GYPSUM]);}
    disprob[GYPSUMS]=(disbase[GYPSUMS]/15.)+dfact1*disbase[GYPSUMS];
    if(disprob[GYPSUMS]>(disbase[GYPSUMS])){disprob[GYPSUMS]=(disbase[GYPSUMS]);}
    /* Adjust gypsum solubility */
    /* if too many diffusing gypsums already in solution */
    if(count[DIFFGYP]>DGYPMAX){
           disprob[GYPSUM]=disprob[GYPSUMS]=0.0;
    }
    disprob[HEMIHYD]=(disbase[HEMIHYD]/15.)+dfact1*disbase[HEMIHYD];
    if(disprob[HEMIHYD]>(disbase[HEMIHYD])){disprob[HEMIHYD]=(disbase[HEMIHYD]);}
    disprob[ANHYDRITE]=(disbase[ANHYDRITE]/15.)+dfact1*disbase[ANHYDRITE];

if(disprob[ANHYDRITE]>(disbase[ANHYDRITE])){disprob[ANHYDRITE]=(disbase[ANHYDRITE])
;}

    }
    else{
      /* Cause flash set by increasing dissolution rates of C3A and C4AF */
      /* each by a factor of four */
      disprob[C3A]=4.*disbase[C3A];
      disprob[C4AF]=4.*disbase[C4AF];
      disprob[GYPSUM]=disbase[GYPSUM];
      disprob[HEMIHYD]=disbase[HEMIHYD];
      disprob[ANHYDRITE]=disbase[ANHYDRITE];
    }
    /* Reduce dissolution probabilities based on saturation of system */
    if((count[EMPTYP]>0)&&((count[POROSITY]+count[EMPTYP])<220000)){
        if(countpore==0){countpore=count[EMPTYP];}
        saturation=(float)(count[POROSITY])/(float)(count[POROSITY]+(count[EMPTYP]-
countpore));
        /* Roughly according to results of Jensen in CCR, powers for RH
           sensitivity are:
            C3S-19
            C2S-29
            C3A, C4AF-6    */
        /* Adjust fly ash silicates (ASG and CAS2) and pozzolanic reactivity */
        /* by same factor as C3S (also CH) */
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
```

```
disprob[C3S]*=(saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
```

```
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation);
        disprob[C3A]*=(saturation*saturation);
        disprob[C3A]*=(saturation*saturation);
        disprob[C3A]*=(saturation*saturation);
        disprob[C4AF]*=(saturation*saturation);
        disprob[C4AF]*=(saturation*saturation);
        disprob[C4AF]*=(saturation*saturation);
    }
      printf("Silicate and aluminate probabilities: %f %f %f
%f\n",disprob[C3S],disprob[C2S],disprob[C3A],disprob[C4AF]);
      printf("cs_acc is %f  and ca_acc is %f sulf_cur is
%ld\n",cs_acc,ca_acc,sulf_cur);
      fflush(stdout);
      /* Pass two- perform the dissolution of species */
      /* Determine the pH factor to use */
      pHfactor=0.0;

if((pHactive==1)&&(count[CSH]>((CSHSCALE*surffract*surffract*totfract*totfract/tfra
ctw04/tfractw04)/8.0))){
         pHfactor=1.5;
         if(pH_cur>12.5){pHfactor=1.0;}
            /* 2/02*/
/*          pHfactor=0.30*(((14.1-pH_cur)/0.3)-1.0); */
          /* 3/02*/
          /*  pHfactor=0.60*(((14.1-pH_cur)/0.5)-1.0); */
          if(pH_cur>12.75){pHfactor=0.667;}
          if(pH_cur>13.00){pHfactor=0.333;}
          if(pH_cur>13.25){pHfactor=(0.0);}
          if(pH_cur>13.75){pHfactor=(-0.25);}
          pHfactor+=concsulfate;   /* influence of sulfate on reactivity */
        }
        nhgd=0;
      /* Update molar volume ratios for CSH formation */
      pc3scsh=molarvcsh[cyccnt]/molarv[C3S]-1.0;
      pc2scsh=molarvcsh[cyccnt]/molarv[C2S]-1.0;
      /* Once again, scan all pixels in microstructure */
      slagemptyp=0;
      for(xloop=0;xloop<SYSIZE;xloop++){
      for(yloop=0;yloop<SYSIZE;yloop++){
      for(zloop=0;zloop<SYSIZE;zloop++){
                if(mic[xloop][yloop][zloop]>OFFSET){
                        phid=mic[xloop][yloop][zloop]-OFFSET;
                        /* attempt a one-step random walk to dissolve */
                        plnew=(int)((float)NEIGHBORS*ran1(seed));
                        if((plnew<0)||(plnew>=NEIGHBORS)){ plnew=NEIGHBORS-1;}
                        xc=xloop+xoff[plnew];
                        yc=yloop+yoff[plnew];
                        zc=zloop+zoff[plnew];
                        if(xc<0){xc=(SYSIZEM1);}
```

```
                        if(yc<0){yc=(SYSIZEM1);}
                        if(xc>=SYSIZE){xc=0;}
                        if(yc>=SYSIZE){yc=0;}
                        if(zc<0){zc=(SYSIZEM1);}
                        if(zc>=SYSIZE){zc=0;}

                       /* Generate probability for dissolution */
                       pdis=ran1(seed);
                       /* Bias dissolution for one pixel particles as */
                       /* indicated by a pixel value of zero in the */
                       /* particle microstructure image */

if(((pdis<=(disprob[phid]/(1.+pHfactor*pHeffect[phid])))||((pdis<=(onepixelbias*dis
prob[phid]/(1.+pHfactor*pHeffect[phid])))&&(micpart[xloop][yloop][zloop]==0)))&&(mi
c[xc][yc][zc]==POROSITY)){
                                discount[phid]+=1;
                                cread=creates[phid];
                                count[phid]-=1;
                                mic[xloop][yloop][zloop]=POROSITY;
                                if(phid==C3AH6){nhgd+=1;}
                                /* Special dissolution for C4AF */
                                if(phid==C4AF){
                                        plfh3=ran1(seed);
                                        if((plfh3<0.0)||(plfh3>1.0)){
                                                plfh3=1.0;
                                        }
                                /* For every C4AF that dissolves, 0.5453 */
                                /* diffusing FH3 species should be created */
                                        if(plfh3<=0.5453){
                                                cread=DIFFFH3;
                                        }
                                }
                                if(cread==POROSITY){
                                count[POROSITY]+=1;
                         }
                         if(cread!=POROSITY){
                                        nmade+=1;
                                        ngoing+=1;
                                        phnew=cread;
                                        count[phnew]+=1;
                                        mic[xc][yc][zc]=phnew;
                            antadd=(struct ants *)malloc(sizeof(struct ants));
                                        antadd->x=xc;
                                        antadd->y=yc;
                                        antadd->z=zc;
                                        antadd->id=phnew;
                                        antadd->cycbirth=cyccnt;
                       /* Now connect this ant structure to end of linked list */
                                        antadd->prevant=tailant;
                                        tailant->nextant=antadd;
                                        antadd->nextant=NULL;
                                        tailant=antadd;
                                 }
                        /* Extra CSH diffusing species based on current temperature
*/
                                 if((phid==C3S)||(phid==C2S)){
                                        plfh3=ran1(seed);
```

```
if(((phid==C2S)&&(plfh3<=pc2scsh))||(plfh3<=pc3scsh)){
                                        cshboxsize=int(2.+6.*(40.-temp_cur)/20.);
                                        if(cshboxsize<1){cshboxsize=1;}
                                              placed=loccsh(xc,yc,zc,cshboxsize);
                                              if(placed!=0){
                                                      count[DIFFCSH]+=1;
                                                      count[POROSITY]-=1;
                                              }
                                              else{
                                                      cshrand+=1;
                                              }
                                        }
                                  }

                                  if((phid==C2S)&&(pc2scsh>1.0)){
                                        plfh3=ran1(seed);
                                        if(plfh3<=(pc2scsh-1.0)){
                                          cshboxsize=int(2.+6.*(40.-temp_cur)/20.);
                                          if(cshboxsize<1){cshboxsize=1;}
                                              placed=loccsh(xc,yc,zc,cshboxsize);
                                              if(placed!=0){
                                                      count[DIFFCSH]+=1;
                                                      count[POROSITY]-=1;
                                              }
                                              else{
                                                      cshrand+=1;
                                              }
                                        }
                                  }
                            }
                            else{
                                  mic[xloop][yloop][zloop]-=OFFSET;
                            }

      } /* end of if edge loop */
      /* Now check if CSH to pozzolanic CSH conversion is possible */
      /* Only if CH is less than 15% in volume */
      /* Only if CSH is in contact with at least one porosity */
      /* and user wishes to use this option */

if((count[POZZ]>=13000)&&(chnew<(0.15*SYSIZE*SYSIZE*SYSIZE))&&(csh2flag==1)){
            if(mic[xloop][yloop][zloop]==CSH){
            if((countbox(3,xloop,yloop,zloop))>=1){
                  pconvert=ran1(seed);
                  if(pconvert<PCSH2CSH){
                        count[CSH]-=1;
                        plfh3=ran1(seed);
                        /* molarvcsh units of C1.7SHx goes to */
                        /* 101.81 units of C1.1SH3.9 */
                        /* with 19.86 units of CH */
                        /* so p=calcy */
                        calcz=0.0;
                        cycnew=cshage[xloop][yloop][zloop];
                        calcy=molarv[POZZCSH]/molarvcsh[cycnew];
                        if(calcy>1.0){
                              calcz=calcy-1.0;
```

```
                                    calcy=1.0;
                                    printf("Problem of not creating enough
pozzolanic CSH during CSH conversion \n");
                                    printf("Current temperature is %f
C\n",temp_cur);
                              }

                              if(plfh3<=calcy){
                                    mic[xloop][yloop][zloop]=POZZCSH;
                                    count[POZZCSH]+=1;
                              }
                              else{
                                    mic[xloop][yloop][zloop]=DIFFCH;
                                    nmade+=1;
                                    ncshgo+=1;
                                    ngoing+=1;
                                    count[DIFFCH]+=1;
                                 antadd=(struct ants *)malloc(sizeof(struct ants));
                                    antadd->x=xloop;
                                    antadd->y=yloop;
                                    antadd->z=zloop;
                                    antadd->id=DIFFCH;
                                    antadd->cycbirth=cyccnt;
                        /* Now connect this ant structure to end of linked list */
                                    antadd->prevant=tailant;
                                    tailant->nextant=antadd;
                                    antadd->nextant=NULL;
                                    tailant=antadd;
                              }
                              /* Possibly need even more pozzolanic CSH */
                              /* Would need a diffusing pozzolanic
CSH species??? */
/*                            if(calcz>0.0){
                                    plfh3=ran1(seed);
                                    if(plfh3<=calcz){
                                          cshrand+=1;
                                    }
                              }    */


                              plfh3=ran1(seed);
                              calcx=(19.86/molarvcsh[cycnew])-(1.-calcy);
                              /* Ex. 0.12658=(19.86/108.)-(1.-0.94269) */
                              if(plfh3<calcx){
                                    npchext+=1;
                              }
                        }
                  }
                  }
            }
                /* See if slag can react --- in contact with at least one porosity
*/
            if(mic[xloop][yloop][zloop]==SLAG){
                  if((countbox(3,xloop,yloop,zloop))>=1){
                        pconvert=ran1(seed);
                        if(pconvert<(disprob[SLAG]/(1.+pHfactor*pHeffect[SLAG]))){
                                    nslagr+=1;
```

```
                                        count[SLAG]-=1;
                                        discount[SLAG]+=1;
                                        /* Check on extra C3A generation */
                                        plfh3=ran1(seed);
                                        if(plfh3<p5slag){
                                            nslagc3a+=1;
                                        }
                                        /* Convert slag to reaction products */
                                        plfh3=ran1(seed);
                                        if(plfh3<p1slag){
                                             mic[xloop][yloop][zloop]=SLAGCSH;
                                  /* Assign a plate axes identifier to this slag C-S-H
voxel */
                                             msface=(int)(3.*ran1(seed)+1.);
                                             if(msface>3){msface=1;}
                                               faces[xloop][yloop][zloop]=msface;
                                               count[SLAGCSH]+=1;
                                             }
                                             else{
                                             if(sealed==1){
                                           /* Create empty porosity at slag site */
                                                  slagemptyp+=1;
                                                  mic[xloop][yloop][zloop]=EMPTYP;
                                                  count[EMPTYP]+=1;
                                  }
                                  else{
                                        mic[xloop][yloop][zloop]=POROSITY;
                                        count[POROSITY]+=1;
                                  }
                                }
                                        /* Add in extra SLAGCSH as needed */
                                        p3init=p3slag;
                                        while(p3init>1.0){
                                             extslagcsh(xloop,yloop,zloop);
                                             p3init-=1.0;
                                        }
                                        plfh3=ran1(seed);
                                        if(plfh3<p3init){
                                             extslagcsh(xloop,yloop,zloop);
                                        }
                                    }
                        }
                    }
          } /* end of zloop */
          } /* end of yloop */
          } /* end of xloop */

      if(ncshgo!=0){printf("CSH dissolved is %ld \n",ncshgo);}

      if(npchext>0){printf("npchext is %ld at cycle %d \n",npchext,cycle);}
          /* Now add in the extra diffusing species for dissolution */
          /* Expansion factors from Young and Hansen and */
          /* Mindess and Young (Concrete) */
          ncshext=cshrand;
          if(cshrand!=0){
                     printf("cshrand is %d \n",cshrand);
```

```
        }
        /* CH, Gypsum, and diffusing C3A are added at totally random */
        /* locations as opposed to at the dissolution site */
        fchext=0.61*(float)discount[C3S]+0.191*(float)discount[C2S]+
        0.2584*(float)discount[C4AF];

        nchext=fchext;
        if(fchext>(float)nchext){
                pdis=ran1(seed);
                  if((fchext-(float)nchext)>pdis){
                        nchext+=1;
                }
        }
        nchext+=npchext;
        /* Adjust CH addition for slag consumption and maintain deficit as needed
*/
        slagcum+=discount[SLAG];
        chgone=(int)(p4slag*(float)slagcum);
        nchext-=chgone;
        slagcum-=(int)((float)chgone/p4slag);
        nchext-=DIFFCHdeficit;
        DIFFCHdeficit=0;
        if(nchext<0){
               DIFFCHdeficit-=nchext;
               nchext=0;
        }
        fc3aext=discount[C3A]+0.5917*(float)discount[C3AH6];
        nc3aext=fc3aext+nslagc3a;
        if(fc3aext>(float)nc3aext){
                pdis=ran1(seed);
                if((fc3aext-(float)nc3aext)>pdis){
                        nc3aext+=1;
                }
        }
        fc4aext=0.696*(float)discount[C4AF];
        nc4aext=fc4aext;
        if(fc4aext>(float)nc4aext){
            pdis=ran1(seed);
            if((fc4aext-(float)nc4aext)>pdis){
                  nc4aext+=1;
            }
        }
        /* both forms of GYPSUM form same DIFFGYP species */
        ngypext=discount[GYPSUM]+discount[GYPSUMS];
        /* Convert to diffusing anhydrite at volume necessary for final */
        /* gypsum formation  (1 anhydrite --> 1.423 gypsum) */
        /* Since hemihydrate can now react with C3A, etc., can't */
        /* do expansion here any longer  7/99 */
/*    fanhext=1.423*(float)discount[ANHYDRITE]; */
        fanhext=(float)discount[ANHYDRITE];
        nanhext=fanhext;
        if(fanhext>(float)nanhext){
                pdis=ran1(seed);
                if((fanhext-(float)nanhext)>pdis){
                        nanhext+=1;
                }
        }
```

```
        /* Convert to diffusing hemiydrate at volume necessary for final */
        /* gypsum formation  (1 hemihydrate --> 1.4 gypsum) */
        /* Since hemihydrate can now react with C3A, etc., can't */
        /* do expansion here any longer  7/99 */
        fhemext=(float)discount[HEMIHYD];
/*      fhemext=1.3955*(float)discount[HEMIHYD];  */

        nhemext=fhemext;
        if(fhemext>(float)nhemext){
                pdis=ran1(seed);
                  if((fhemext-(float)nhemext)>pdis){
                        nhemext+=1;
                }
        }
        count[DIFFGYP]+=ngypext;
        count[DIFFANH]+=nanhext;
        count[DIFFHEM]+=nhemext;
        count[DIFFCH]+=nchext;
        count[DIFFCSH]+=ncshext;
        count[DIFFC3A]+=nc3aext;
        count[DIFFC4A]+=nc4aext;

        nsum2=nchext+ncshext;
        nsum3=nsum2+nc3aext;
        nsum4=nsum3+nc4aext;
        nsum5=nsum4+ngypext;
        nsum6=nsum5+nhemext;
        fflush(stdout);
        for(xext=1;xext<=(nsum6+nanhext);xext++){
        plok=0;
        do{
                xc=(int)((float)SYSIZE*ran1(seed));
                yc=(int)((float)SYSIZE*ran1(seed));
                zc=(int)((float)SYSIZE*ran1(seed));
                if(xc>=SYSIZE){xc=0;}
                if(yc>=SYSIZE){yc=0;}
                if(zc>=SYSIZE){zc=0;}

                if(mic[xc][yc][zc]==POROSITY){
                        plok=1;
                        phid=DIFFCH;
                        count[POROSITY]-=1;
                        if(xext>nsum6){phid=DIFFANH;}
                        else if(xext>nsum5){phid=DIFFHEM;}
                        else if(xext>nsum4){phid=DIFFGYP;}
                        else if(xext>nsum3){phid=DIFFC4A;}
                        else if(xext>nsum2){phid=DIFFC3A;}
                        else if(xext>nchext){phid=DIFFCSH;}
                        mic[xc][yc][zc]=phid;
                        nmade+=1;
                        ngoing+=1;
                        antadd=(struct ants *)malloc(sizeof(struct ants));
                        antadd->x=xc;
                        antadd->y=yc;
                        antadd->z=zc;
                        antadd->id=phid;
                        antadd->cycbirth=cyccnt;
```

```
                    /* Now connect this ant structure to end of linked list */
                       antadd->prevant=tailant;
                       tailant->nextant=antadd;
                       antadd->nextant=NULL;
                       tailant=antadd;
                }
        } while (plok==0);

        } /* end of xext for extra species generation */

        printf("Dissolved- %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld
%ld\n",count[DIFFCSH],
      count[DIFFCH],count[DIFFGYP],count[DIFFC3A],count[DIFFFH3],
      count[DIFFETTR],count[DIFFAS],count[DIFFANH],count[DIFFHEM],
      count[DIFFCAS2],count[DIFFCACL2],count[DIFFCACO3]);
      sulf_cur=count[DIFFGYP]+count[DIFFANH]+count[DIFFHEM];
/*    difffile=fopen("diffuse.out","a");
      fprintf(difffile,"%d %ld %f %f %f %f %f %f %f\n",cycle, sulf_cur, cs_acc,
ca_acc, disprob[C3S], disprob[C3A], disprob[C4AF], dfact, dfact1);

      fclose(difffile); */

            /* if too many diffusing gypsums already in solution */
      if(sulf_cur>DGYPMAX){
                disprob[GYPSUM]=disprob[GYPSUMS]=0.0;
      }
      else{
                disprob[GYPSUM]=disbase[GYPSUM];
                disprob[ANHYDRITE]=disbase[ANHYDRITE];
                disprob[HEMIHYD]=disbase[HEMIHYD];
                disprob[GYPSUMS]=disbase[GYPSUMS];
      }


      printf("C3AH6 dissolved- %ld with prob. of %f \n",nhgd,disprob[C3AH6]);
      fflush(stdout);
}

/* routine to add nneed one pixel elements of phase randid at random */
/* locations in microstructure */
/* Special features for addition of 1-pixel CACO3 and INERT particles */
/* added 5/26/2004 */
/* Called by main program */
/* Calls no other routines */
void addrand(randid,nneed)
        int randid;
        long int nneed;
{
        int ix,iy,iz;
        long int ic;
        int success,cpores;

        /* Add number of requested phase pixels at random pore locations */
        for(ic=1;ic<=nneed;ic++){
                success=0;
                while(success==0){
                        ix=(int)((float)SYSIZE*ran1(seed));
```

```
                        iy=(int)((float)SYSIZE*ran1(seed));
                        iz=(int)((float)SYSIZE*ran1(seed));
                        if(ix==SYSIZE){ix=0;}
                        if(iy==SYSIZE){iy=0;}
                        if(iz==SYSIZE){iz=0;}
                        if(mic[ix][iy][iz]==POROSITY){
                        if((randid!=CACO3)&&(randid!=INERT)){
                                mic[ix][iy][iz]=randid;
                                micorig[ix][iy][iz]=randid;
                                success=1;
                      }
                      else{
                                cpores=countboxc(3,ix,iy,iz);
                                if(cpores>=26){
                                    mic[ix][iy][iz]=randid;
                                    micorig[ix][iy][iz]=randid;
                                    success=1;
                                }
                       }
                     }
                  }
       }
}

/* Routine measuresurf to measure initial surface counts for cement */
/* and for all phases (cement= C3S, C2S, C3A, C4AF, and calcium sulfates */
void measuresurf()
{
      int sx,sy,sz,jx,jy,jz,faceid;

      for(sx=0;sx<SYSIZE;sx++){
      for(sy=0;sy<SYSIZE;sy++){
      for(sz=0;sz<SYSIZE;sz++){
           if(mic[sx][sy][sz]==POROSITY){
           for(faceid=0;faceid<6;faceid++){
              if(faceid==1){
                   jx=sx-1;
                   if(jx<0){jx=SYSIZE-1;}
                   jy=sy;
                   jz=sz;
             }
             else if(faceid==0){
                   jx=sx+1;
                   if(jx>(SYSIZE-1)){jx=0;}
                   jy=sy;
                   jz=sz;
             }
             else if(faceid==2){
                   jy=sy+1;
                   if(jy>(SYSIZE-1)){jy=0;}
                   jx=sx;
                   jz=sz;
             }
             else if(faceid==3){
                   jy=sy-1;
                   if(jy<0){jy=SYSIZE-1;}
                   jx=sx;
```

```
                    jz=sz;
               }
               else if(faceid==4){
                    jz=sz+1;
                    if(jz>(SYSIZE-1)){jz=0;}
                    jx=sx;
                    jy=sy;
               }
               else if(faceid==5){
                    jz=sz-1;
                    if(jz<0){jz=SYSIZE-1;}
                    jx=sx;
                    jy=sy;
               }
              /* If the neighboring pixel is solid, update surface counts */

      if((mic[jx][jy][jz]==C3S)||(mic[jx][jy][jz]==C2S)||(mic[jx][jy][jz]==C3A)||(m
ic[jx][jy][jz]==C4AF)||(mic[jx][jy][jz]==INERT)||(mic[jx][jy][jz]==CACO3)){
            scnttotal+=1;

      if((mic[jx][jy][jz]==C3S)||(mic[jx][jy][jz]==C2S)||(mic[jx][jy][jz]==C3A)||(m
ic[jx][jy][jz]==C4AF)){
                    scntcement+=1;
             }
             }
             }
            }

      }
      }
      }
      printf("Cement surface count is %ld \n",scntcement);
      printf("Total surface count is %ld \n",scnttotal);
      surffract=(float)scntcement/(float)scnttotal;
      printf("Surface fraction is %f \n",surffract);
      fflush(stdout);
}

/* Routine resaturate to resaturate all empty porosity */
/* and continue with hydration under saturated conditions */
void resaturate()
{
      int sx,sy,sz;
      long int nresat=0;

      for(sx=0;sx<SYSIZE;sx++){
      for(sy=0;sy<SYSIZE;sy++){
      for(sz=0;sz<SYSIZE;sz++){
            if(mic[sx][sy][sz]==EMPTYP){
                  mic[sx][sy][sz]=POROSITY;
                  nresat++;
            }
      }
      }
      }
      if(nresat>0){
            porefl1=porefl2=porefl3=1;
```

```
      }
      printf("Number resaturated is %ld \n",nresat);
      fflush(stdout);
}

/* Calls init, dissolve and addrand */
main()
{
        int ntimes,valin,nmovstep,stopflag=0;
        int cycflag,ix,iy,iz,phtodo;
        int iseed,phydfreq,oflag;
        long int nadd;
        int xpl,xph,ypl,yph,fidc3s,fidc2s,fidc3a,fidc4af,fidgyp,fidagg,ffac3a;
        int fidhem,fidanh,fidcaco3,nlen,pixtmp;
        float pnucch,pscalech,pnuchg,pscalehg,pnucfh3,pscalefh3;
        float pnucgyp,pscalegyp;
        float thtimelo,thtimehi,thtemplo,thtemphi;
        float mass_cement,mass_cem_now,mass_cur,kpozz,kslag;
        FILE *infile,*outfile,*adiafile,*thfile;
        char filei[80],fileo[80],filetemp[80];

        ngoing=0;
        porefl1=porefl2=porefl3=1;
        pore_off=water_off=0;
        cycflag=0;
        heat_old=heat_new=0.0;
        chold=chnew=0;     /* Current and previous cycle CH counts */
        time_cur=0.0;      /* Elapsed time according to maturity principles */
        cubesize=CUBEMAX;
        ppozz=PPOZZ;
        poregone=poretodo=0;
        /* Get random number seed */
        printf("Enter random number seed \n");
        scanf("%d",&iseed);
        printf("%d\n",iseed);
        seed=(&iseed);

        printf("Dissolution bias is set at %f \n",DISBIAS);
        /* Open file and read in original cement particle microstructure */
        printf("Enter name of file to read initial microstructure from \n");
        scanf("%s",filei);
        printf("%s\n",filei);
        nlen=strcspn(filei,".");
        sprintf(fileroot,"");
        strncat(fileroot,filei,nlen);
        printf("nlen is %d and fileroot is now %s \n",nlen,fileroot);
        fflush(stdout);
        /* Get phase assignments for original microstructure */
        /* to transform to needed ID values */
     printf("Enter IDs in file for C3S, C2S, C3A, C4AF, Gypsum, Hemihydrate,
Anhydrite, Aggregate CaCO3\n");
    scanf("%d %d %d %d %d %d %d %d
%d",&fidc3s,&fidc2s,&fidc3a,&fidc4af,&fidgyp,&fidhem,&fidanh,&fidagg,&fidcaco3);
       printf("%d %d %d %d %d %d %d %d
%d\n",fidc3s,fidc2s,fidc3a,fidc4af,fidgyp,fidhem,fidanh,fidagg,fidcaco3);
        printf("Enter ID in file for C3A in fly ash  (default=35)\n");
        scanf("%d",&ffac3a);
```

```
printf("%d\n",ffac3a);
fflush(stdout);

infile=fopen(filei,"r");

for(ix=0;ix<SYSIZE;ix++){
for(iy=0;iy<SYSIZE;iy++){
for(iz=0;iz<SYSIZE;iz++){
    cshage[ix][iy][iz]=0;
    faces[ix][iy][iz]=0;

    fscanf(infile,"%d",&valin);
    mic[ix][iy][iz]=valin;
    if(valin==fidc3s){
                mic[ix][iy][iz]=C3S;
    }
    else if(valin==fidc2s){
                mic[ix][iy][iz]=C2S;
    }
    else if((valin==fidc3a)||(valin==ffac3a)){
                mic[ix][iy][iz]=C3A;
    }
    else if(valin==fidc4af){
                mic[ix][iy][iz]=C4AF;
    }
    else if(valin==fidgyp){
                mic[ix][iy][iz]=GYPSUM;
    }
    else if(valin==fidanh){
                mic[ix][iy][iz]=ANHYDRITE;
    }
    else if(valin==fidhem){
                mic[ix][iy][iz]=HEMIHYD;
    }
    else if(valin==fidcaco3){
                mic[ix][iy][iz]=CACO3;
    }
    else if(valin==fidagg){
                mic[ix][iy][iz]=INERTAGG;
    }
    micorig[ix][iy][iz]=mic[ix][iy][iz];
}
}
}
fclose(infile);
fflush(stdout);

/* Now read in particle IDs from file */
printf("Enter name of file to read particle IDs from \n");
scanf("%s",filei);
printf("%s\n",filei);
infile=fopen(filei,"r");

for(ix=0;ix<SYSIZE;ix++){
for(iy=0;iy<SYSIZE;iy++){
for(iz=0;iz<SYSIZE;iz++){
        fscanf(infile,"%d",&valin);
```

```
            micpart[ix][iy][iz]=valin;
    }
    }
    }

    fclose(infile);
    fflush(stdout);

    /* Initialize counters, etc. */
    npr=nasr=nslagr=0;
    nfill=0;
    ncsbar=0;
    netbar=0;
    porinit=0;
    cyccnt=0;
    setflag=0;
    c3sinit=c2sinit=c3ainit=c4afinit=anhinit=heminit=slaginit=0;

    /* Initialize structure for ants */
    headant=(struct ants *)malloc(sizeof(struct ants));
    headant->prevant=NULL;
    headant->nextant=NULL;
    headant->x=0;
    headant->y=0;
    headant->z=0;
    headant->id=100;       /* special ID indicating first ant in list */
    headant->cycbirth=0;
    tailant=headant;

  /* Allow user to iteratively add one pixel particles of various phases */
  /* Typical application would be for addition of silica fume */
    printf("Enter number of one pixel particles to add (0 to quit) \n");
    scanf("%ld",&nadd);
    printf("%ld\n",nadd);
    while(nadd>0){
            printf("Enter phase to add \n");
            printf(" C3S 1 \n");
            printf(" C2S 2 \n");
            printf(" C3A 3 \n");
            printf(" C4AF 4 \n");
            printf(" GYPSUM 5 \n");
            printf(" HEMIHYD 6 \n");
            printf(" ANHYDRITE 7 \n");
            printf(" POZZ 8 \n");
            printf(" INERT 9 \n");
            printf(" SLAG 10 \n");
            printf(" ASG 11 \n");
            printf(" CAS2 12 \n");
            printf(" CH 13 \n");
            printf(" CSH 14 \n");
            printf(" C3AH6 15 \n");
            printf(" Ettringite 16 \n");
            printf(" Stable Ettringite from C4AF 17 \n");
            printf(" AFM 18 \n");
            printf(" FH3 19 \n");
            printf(" POZZCSH 20 \n");
            printf(" SLAGCSH 21 \n");
```

```
            printf(" CACL2 22 \n");
            printf(" Friedels salt 23 \n");
            printf(" Stratlingite 24 \n");
            printf(" Calcium carbonate 26 \n");
            scanf("%d",&phtodo);
            printf("%d \n",phtodo);
            if((phtodo<0)||(phtodo>CACO3)){
                  printf("Error in phase input for one pixel particles \n");
                  exit(1);
            }
            addrand(phtodo,nadd);
           printf("Enter number of one pixel particles to add (0 to quit) \n");
            scanf("%ld",&nadd);
            printf("%ld\n",nadd);
      }
      fflush(stdout);

      init();
      printf("After init routine \n");
      printf("Enter number of cycles to execute \n");
      scanf("%d",&ncyc);
      printf("%d \n",ncyc);
printf("Do you wish hydration under 0) saturated or 1) sealed conditions \n");
      scanf("%d",&sealed);
      printf("%d \n",sealed);
      printf("Enter max. # of diffusion steps per cycle (500) \n");
      scanf("%d",&ntimes);
      printf("%d \n",ntimes);
      printf("Enter nuc. prob. and scale factor for CH nucleation \n");
      scanf("%f %f",&pnucch,&pscalech);
      printf("%f %f \n",pnucch,pscalech);
      printf("Enter nuc. prob. and scale factor for gypsum nucleation \n");
      scanf("%f %f",&pnucgyp,&pscalegyp);
      printf("%f %f \n",pnucgyp,pscalegyp);
      printf("Enter nuc. prob. and scale factor for C3AH6 nucleation \n");
      scanf("%f %f",&pnuchg,&pscalehg);
      printf("%f %f \n",pnuchg,pscalehg);
      printf("Enter nuc. prob. and scale factor for FH3 nucleation \n");
      scanf("%f %f",&pnucfh3,&pscalefh3);
      printf("%f %f \n",pnucfh3,pscalefh3);
      printf("Enter cycle frequency for checking pore space percolation \n");
      scanf("%d",&burnfreq);
      printf("%d\n",burnfreq);
printf("Enter cycle frequency for checking percolation of solids (set) \n");
      scanf("%d",&setfreq);
      printf("%d\n",setfreq);
printf("Enter cycle frequency for checking hydration of particles \n");
      scanf("%d",&phydfreq);
      printf("%d\n",phydfreq);
printf("Enter cycle frequency for outputting hydrating microstructure \n");
      scanf("%d",&outfreq);
      printf("%d\n",outfreq);
      /* Parameters for adiabatic temperature rise calculation */
      printf("Enter the induction time in hours \n");
      scanf("%f",&ind_time);
      printf("%f \n",ind_time);
      time_cur+=ind_time;
```

```
        printf("Enter the initial temperature in degrees Celsius \n");
        scanf("%f",&temp_0);
        printf("%f \n",temp_0);
        temp_cur=temp_0;
        printf("Enter the ambient temperature in degrees Celsius \n");
        scanf("%f",&T_ambient);
        printf("%f \n",T_ambient);
        printf("Enter the overall heat transfer coefficient in J/g/C/s \n");
        scanf("%f",&U_coeff);
        printf("%f \n",U_coeff);
        printf("Enter apparent activation energy for hydration in kJ/mole \n");
        scanf("%f",&E_act);
        printf("%f \n",E_act);
        printf("Enter apparent activation energy for pozzolanic reactions in
kJ/mole \n");
        scanf("%f",&E_act_pozz);
        printf("%f \n",E_act_pozz);
        printf("Enter apparent activation energy for slag reactions in kJ/mole
\n");
        scanf("%f",&E_act_slag);
        printf("%f \n",E_act_slag);
        printf("Enter kinetic factor to convert cycles to time for 25 C \n");
        scanf("%f",&beta);
        printf("%f \n",beta);
        printf("Enter mass fraction of aggregate in concrete \n");
        scanf("%f",&mass_agg);
        printf("%f \n",mass_agg);
        printf("Hydration under 0) isothermal, 1) adiabatic or 2) programmed
temperature history conditions \n");
        scanf("%d",&adiaflag);
        printf("%d \n",adiaflag);
        if(adiaflag==2){
            thfile=fopen("temphist.dat","r");
            fscanf(thfile,"%f %f %f %f",&thtimelo,&thtimehi,&thtemplo,&thtemphi);
            printf("%f %f %f %f\n",thtimelo,thtimehi,thtemplo,thtemphi);
        }
        printf("CSH to pozzolanic CSH 0) prohibited or 1) allowed \n");
        scanf("%d",&csh2flag);
        printf("%d \n",csh2flag);
      printf("CH precipitation on aggregate surfaces 0) prohibited or 1) allowed
\n");
        scanf("%d",&chflag);
        printf("%d \n",chflag);
        printf("Number of slices in hydration movie \n");
        scanf("%d",&nummovsl);
        printf("%d \n",nummovsl);
        nmovstep=1;
        if(nummovsl>0){
            nmovstep=ncyc/nummovsl;
                if(nmovstep<1){nmovstep=1;}
        }
        printf("Dissolution bias factor for one-pixel particles \n");
        scanf("%f",&onepixelbias);
        printf("%f\n",onepixelbias);
        printf("Enter number of cycles before executing total resaturation \n");
        scanf("%d\n",&resatcyc);
```

```
        printf("%d\n",resatcyc);

        printf("Enter choice for C-S-H geometry 0) random or 1) plates \n");
        scanf("%d\n",&cshgeom);
        printf("%d \n",cshgeom);
        printf("Does pH influence hydration kinetics 0) no or 1) yes \n");
        scanf("%d\n",&pHactive);
        printf("%d\n",pHactive);
        fflush(stdout);

sprintf(heatname,"%s.heat.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaf
lag,sealed);

sprintf(moviename,"%s.mov.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaf
lag,sealed);

sprintf(chshrname,"%s.chs.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaf
lag,sealed);

sprintf(adianame,"%s.adi.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafl
ag,sealed);

sprintf(parname,"%s.par.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafla
g,sealed);
      /* Store filename for pH file and initialize with column headings */

sprintf(pHname,"%s.phv.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaflag
,sealed);
/*      pHfile=fopen(pHname,"w");
        fprintf(pHfile,"Cycle time(h) alpha_mass pH sigma [Na+] [K+] [Ca++] [SO4--]
activityCa activityOH activitySO4 activityK molesSyngenite\n");
        fclose(pHfile); */

sprintf(fileo,"%s.img.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaflag,
sealed);

sprintf(phname,"%s.pha.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaflag
,sealed);

sprintf(ppsname,"%s.pps.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafla
g,sealed);
        /* Store parameters input in parameter file */
        sprintf(cmdnew,"cp disrealnew.out %s",parname);
        system(cmdnew);
        if(burnfreq<=ncyc){
           ptmpfile=fopen(ppsname,"w");
           fprintf(ptmpfile,"Cycle time(h) alpha_mass conn_por total_por
frac_conn\n");
           fclose(ptmpfile);
       }

sprintf(ptsname,"%s.pts.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafla
g,sealed);
        if(setfreq<=ncyc){
           ptmpfile=fopen(ptsname,"w");
           fprintf(ptmpfile,"Cycle time(h) alpha_mass conn_solid total_solid
frac_conn\n");
```

```
           fclose(ptmpfile);
       }

sprintf(phrname,"%s.phr.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafla
g,sealed);
        krate=exp(-(1000.*E_act/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        /* Determine pozzolanic and slag reaction rate constants */
        kpozz=exp(-(1000.*E_act_pozz/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        kslag=exp(-(1000.*E_act_slag/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        /* Update probability of pozzolanic reaction */
        /* based on ratio of pozzolanic reaction rate to hydration rate */
        ppozz=PPOZZ*kpozz/krate;
        /* Assume same holds for dissolution of fly ash phases */
        disprob[ASG]=disbase[ASG]*kpozz/krate;
        disprob[CAS2]=disbase[CAS2]*kpozz/krate;
        /* Update probability of slag dissolution */
        disprob[SLAG]=slagreact*disbase[SLAG]*kslag/krate;
        printf("%s\n",adianame);
        fflush(stdout);
        adiafile=fopen(adianame,"w");
fprintf(adiafile,"Time(h) Temperature  Alpha  Krate   Cp_now  Mass_cem kpozz/khyd
kslag/khyd\n");
        /* Set initial properties of CSH */
        molarvcsh[0]=molarv[CSH];
        watercsh[0]=waterc[CSH];
        /* Determine surface counts */

        measuresurf();
        for(icyc=1;icyc<=ncyc;icyc++){
            if((sealed==1)&&(icyc==(resatcyc+1))&&(resatcyc!=0)){
                  resaturate();
                  sealed=0;
            }
        if(temp_cur<=80.0){
            molarvcsh[icyc]=molarv[CSH]-8.0*((temp_cur-20.)/(80.-20.));
            watercsh[icyc]=waterc[CSH]-1.3*((temp_cur-20.)/(80.-20.));
        }
        else{
            molarvcsh[icyc]=molarv[CSH]-8.0;
            watercsh[icyc]=waterc[CSH]-1.3;
        }
                if(icyc==ncyc){cycflag=1;}
                printf("Calling dissolve \n");
                fflush(stdout);
                dissolve(icyc);
printf("Number dissolved this pass- %ld total diffusing- %ld \n",nmade,ngoing);
                fflush(stdout);
                if(icyc==1){
                     printf("ncsbar is %ld   netbar is %ld \n",ncsbar,netbar);
                }

hydrate(cycflag,ntimes,pnucch,pscalech,pnuchg,pscalehg,pnucfh3,pscalefh3,pnucgyp,ps
calegyp);
                printf("Returned from hydrate \n");
                fflush(stdout);
                temp_0=temp_cur;
                /* Handle adiabatic case first */
```

```
            /* Cement + aggregate +water + filler=1;  that's all there is */
            mass_cement=1.-mass_agg-mass_fill-mass_water-mass_CH;
            mass_cem_now=mass_cement;
            if(adiaflag==1){
                    /* determine heat capacity of current mixture, */
                    /* accounting for imbibed water if necessary */
                    if(sealed==1){
                            Cp_now=mass_agg*Cp_agg;
                            Cp_now+=Cp_pozz*mass_fill;
                            Cp_now+=Cp_cement*mass_cement;
                            Cp_now+=Cp_CH*mass_CH;
  Cp_now+=(Cp_h2o*mass_water-alpha_cur*WN*mass_cement*(Cp_h2o-Cp_bh2o));
                            mass_cem_now=mass_cement;
                      }
            /* Else need to account for extra capillary water drawn in */
    /* Basis is WCHSH(0.06) g H2O per gram cement for chemical shrinkage */
            /* Need to adjust mass basis to account for extra imbibed H2O */
              else{
                            mass_cur=1.+WCHSH*mass_cement*alpha_cur;
                            Cp_now=mass_agg*Cp_agg/mass_cur;
                            Cp_now+=Cp_pozz*mass_fill/mass_cur;
                            Cp_now+=Cp_cement*mass_cement/mass_cur;
                            Cp_now+=Cp_CH*mass_CH/mass_cur;
    Cp_now+=(Cp_h2o*mass_water-alpha_cur*WN*mass_cement*(Cp_h2o-Cp_bh2o));
                            Cp_now+=(WCHSH*Cp_h2o*alpha_cur*mass_cement);
                            mass_cem_now=mass_cement/mass_cur;
                      }
            /* Determine rate constant based on Arrhenius expression */
            /* Recall that temp_cur is in degrees Celsius */
        krate=exp(-(1000.*E_act/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        /* Determine pozzolanic and slag reaction rate constant */
     kpozz=exp(-(1000.*E_act_pozz/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
     kslag=exp(-(1000.*E_act_slag/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        /* Update probability of pozzolanic and slag reactions */
/* based on ratio of pozzolanic (slag) reaction rate to hydration rate */
        ppozz=PPOZZ*kpozz/krate;
        disprob[ASG]=disbase[ASG]*kpozz/krate;
        disprob[CAS2]=disbase[CAS2]*kpozz/krate;
        disprob[SLAG]=slagreact*disbase[SLAG]*kslag/krate;

            /* Update temperature based on heat generated and current Cp */
        if(mass_cem_now>0.01){
                    temp_cur=temp_0+mass_cem_now*heat_cf*(heat_new-
                          heat_old)/Cp_now;
        }
        else{
                    temp_cur=temp_0+mass_fill_pozz*heat_cf*(heat_new-
                          heat_old)/Cp_now;

        }
        /* Update system temperature due to heat loss/gain to/from */
        /* surroundings (semi-adiabatic case)  */
        temp_cur-=(temp_cur-T_ambient)*time_step*U_coeff/Cp_now;
        }
        else if(adiaflag==2){
              /* Update system temperature based on current time */
              /* and requested temperature history */
```

```
                  while((time_cur>thtimehi)&&(!feof(thfile))){
                        fscanf(thfile,"%f %f %f
%f",&thtimelo,&thtimehi,&thtemplo,&thtemphi);
                        printf("New temperature history values : \n");
                        printf("%f %f %f
%f\n",thtimelo,thtimehi,thtemplo,thtemphi);
                  }
            if((thtimehi-thtimelo)>0.0){
                  temp_cur=thtemplo+(thtemphi-thtemplo)*(time_cur-
thtimelo)/(thtimehi-thtimelo);
            }
            else{
                  temp_cur=thtemplo;
            }
         krate=exp(-(1000.*E_act/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
         kpozz=exp(-(1000.*E_act_pozz/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
         kslag=exp(-(1000.*E_act_slag/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
            ppozz=PPOZZ*kpozz/krate;
            disprob[ASG]=disbase[ASG]*kpozz/krate;
            disprob[CAS2]=disbase[CAS2]*kpozz/krate;
            disprob[SLAG]=slagreact*disbase[SLAG]*kslag/krate;
            }
                /* Update time based on simple numerical integration */
                /* simulating maturity approach */
                /* with parabolic kinetics (Knudsen model) */
                if(cyccnt>1){
                  time_cur+=(2.*(float)(cyccnt-1)-1.0)*beta/krate;
                  time_step=(2.*(float)(cyccnt-1)-1.0)*beta/krate;
            }
                fprintf(adiafile,"%f %f %f %f %f %f %f %f\n",time_cur,temp_cur,
                 alpha_cur,krate,Cp_now,mass_cem_now,kpozz/krate,kslag/krate);
                fflush(adiafile);
                gsratio2=0.0;
                gsratio2+=(float)(count[CH]+count[CSH]+count[C3AH6]+count[ETTR]);

gsratio2+=(float)(count[POZZCSH]+count[SLAGCSH]+count[FH3]+count[AFM]+count[ETTRC4A
F]);

gsratio2+=(float)(count[FREIDEL]+count[STRAT]+count[ABSGYP]+count[AFMC]);

gsratio2=(gsratio2)/(gsratio2+(float)(count[POROSITY]+count[EMPTYP]));
                heatfile=fopen(heatname,"a");
                if(w_to_c!=0.0){
                          fprintf(heatfile,"%d %f %f %f %f %f  %f \n",
                     cyccnt-
1,time_cur,alpha,alpha_cur,heat_new*heat_cf,gsratio2,((0.68*alpha_cur)/(0.32*alpha_
cur+w_to_c)));
                }
                else{
                        fprintf(heatfile,"%d %f %f %f %f %f  %f \n",
               cyccnt-1,time_cur,alpha,alpha_cur,heat_new*heat_cf,gsratio2,0.0);
                }
                fclose(heatfile);
                chsfile=fopen(chshrname,"a");
                fprintf(chsfile,"%d %f %f %f\n",
                           cyccnt-1,time_cur,alpha_cur,chs_new);
                fclose(chsfile);
```

```
            pHpred();
            printf("Returned from call to pH \n");
            fflush(stdout);
      /* Check percolation of pore space */
      /* Note that first variable passed corresponds to phase to check */
      /* Could easily add calls to check for percolation of CH, CSH, etc. */
      if(((icyc%burnfreq)==0)&&((porefl1+porefl2+porefl3)!=0)){
            porefl1=burn3d(0,1,0,0);
            porefl2=burn3d(0,0,1,0);
            porefl3=burn3d(0,0,0,1);
         /* Switch to self-desiccating conditions when porosity */
         /* disconnects */
         if(((porefl1+porefl2+porefl3)==0)&&(sealed==0)){
               water_off=water_left;
               pore_off=countkeep;
               sealed=1;
               printf("Switching to self-desiccating at cycle %d \n",cyccnt);
               fflush(stdout);
         }
      }
      /* Check percolation of solids (set point) */
      if(((icyc%setfreq)==0)&&(setflag==0)){
            sf1=burnset(1,0,0);
            sf2=burnset(0,1,0);
            sf3=burnset(0,0,1);
            setflag=sf1*sf2*sf3;
      }


      /* Check hydration of particles */
      if((icyc%phydfreq)==0){
            parthyd();
      }
      /* Output movie microstructure if desired */
            if((nummovsl>0)&&((icyc%nmovstep)==0)){
                     if(icyc==nmovstep){
                     movfile=fopen(moviename,"w");
                     }
                     else{
                     movfile=fopen(moviename,"a");
                     }

               for(ix=0;ix<SYSIZE;ix++){
               for(iy=0;iy<SYSIZE;iy++){
                     fprintf(movfile,"%d\n",(int)mic[50][ix][iy]);
               }
               }
               fclose(movfile);
        }
        /* Output complete microstructure every outfreq cycles */
            if((icyc>0)&&((icyc%outfreq)==0)){

sprintf(micname,"%s.ima.%d.%d.%1d%1d%1d",fileroot,icyc,(int)temp_0,csh2flag,adiafla
g,sealed);
               micfile=fopen(micname,"w");

               for(ix=0;ix<SYSIZE;ix++){
```

```
            for(iy=0;iy<SYSIZE;iy++){
            for(iz=0;iz<SYSIZE;iz++){
                        pixtmp=(int)mic[ix][iy][iz];
                        if(pixtmp==DIFFCSH){
                           pixtmp=CSH;
                        }
                        else if (pixtmp==DIFFANH){
                           pixtmp=ANHYDRITE;
                        }
                        else if (pixtmp==DIFFHEM){
                           pixtmp=HEMIHYD;
                        }
                        else if (pixtmp==DIFFGYP){
                           pixtmp=GYPSUM;
                        }
                        else if (pixtmp==DIFFCACL2){
                           pixtmp=CACL2;
                        }
                        else if (pixtmp==DIFFCACO3){
                           pixtmp=CACO3;
                        }
                        else if (pixtmp==DIFFCAS2){
                           pixtmp=CAS2;
                        }
                        else if (pixtmp==DIFFAS){
                           pixtmp=ASG;
                        }
                        else if (pixtmp==DIFFETTR){
                           pixtmp=ETTR;
                        }
                        else if (pixtmp==DIFFC3A){
                           pixtmp=C3A;
                        }
                        else if (pixtmp==DIFFC4A){
                           pixtmp=C3A;
                        }
                        else if (pixtmp==DIFFFH3){
                           pixtmp=FH3;
                        }
                        else if (pixtmp==DIFFCH){
                           pixtmp=CH;
                        }
                  fprintf(micfile,"%d\n",pixtmp);
            }
            }
            }
            fclose(micfile);
      }

  }
/* Last call to dissolve to terminate hydration */
dissolve(0);
/* Check percolation of pore space */
/* Note that first variable passed corresponds to phase to check */
/* Could easily add calls to check for percolation of CH, CSH, etc. */
if((burnfreq!=0)&&(burnfreq<=ncyc)&&((porefl1+porefl2+porefl3)!=0)){
         porefl1=burn3d(0,1,0,0);
```

```
            porefl2=burn3d(0,0,1,0);
            porefl3=burn3d(0,0,0,1);
      }
      /* Check percolation of solids (set point) */
      if((setfreq!=0)&&(setfreq<=ncyc)){
            setflag=burnset(1,0,0);
            setflag+=burnset(0,1,0);
            setflag+=burnset(0,0,1);
      }

      /* Output last lines of heat and chemical shrinkage files */
      if(cyccnt>1){
            time_cur+=(2.*(float)cyccnt-1.0)*beta/krate;
            time_step=(2.*(float)cyccnt-1.0)*beta/krate;
      }
      fprintf(adiafile,"%f %f %f %f %f %f %f %f\n",time_cur,temp_cur,
          alpha_cur,krate,Cp_now,mass_cem_now,kpozz/krate,kslag/krate);
      fflush(adiafile);
      fclose(adiafile);
      gsratio2=0.0;
      gsratio2+=(float)(count[CH]+count[CSH]+count[C3AH6]+count[ETTR]);

gsratio2+=(float)(count[POZZCSH]+count[SLAGCSH]+count[FH3]+count[AFM]+count[ETTRC4A
F]);
      gsratio2+=(float)(count[FREIDEL]+count[STRAT]+count[ABSGYP]+count[AFMC]);
      gsratio2=(gsratio2)/(gsratio2+(float)(count[POROSITY]+count[EMPTYP]));
      heatfile=fopen(heatname,"a");
      fprintf(heatfile,"%d %f %f %f %f %f %f\n",

cyccnt,time_cur,alpha,alpha_cur,heat_new*heat_cf,gsratio2,((0.68*alpha_cur)/(0.32*a
lpha_cur+w_to_c)));
      fclose(heatfile);
      chsfile=fopen(chshrname,"a");
      fprintf(chsfile,"%d %f %f %f\n",
      cyccnt,time_cur,alpha_cur,((float)(count[EMPTYP]+count[POROSITY]-
water_left)*heat_cf/1000.));
      fclose(chsfile);
      cyccnt+=1;
      pHpred();
      printf("Final count for ncshplategrow is %ld \n",ncshplategrow);
      printf("Final count for ncshplateinit is %ld \n",ncshplateinit);
      /* Output final microstructure if desired */
      outfile=fopen(fileo,"w");

      for(ix=0;ix<SYSIZE;ix++){
      for(iy=0;iy<SYSIZE;iy++){
      for(iz=0;iz<SYSIZE;iz++){
            fprintf(outfile,"%d\n",(int)mic[ix][iy][iz]);
      }
      }
      }
      fclose(outfile);
}
```