SOFTWARE SOURCE CODE

1. Modeling Diffusive Noble Gas uptake in Droplets

The following subsections include Pascal source code that was used to produce the model outputs for Figures 5, S1, S2, S3, and S4. All programs run within a command prompt (terminal in Linux) and were compiled using the Free Pascal compiler fpc. The main program is realmodel3.pas and it in turn "uses" unit fogmodel. That unit in turn uses unit surfacestuff. An example batch file that was used to create the model values for Figure S2 is included as run-10-65-1000.bat.

1.1. Main program realmodel3.pas.

The program uses 5 input line parameters:

- Temperature at the collection point (°C).
- Altitude in m a.s.l. of the starting point for the droplet.
- Altitude of the sample collection point in m a.s.l.
- Diameter of the droplet in mm.
- Lapse rate in °C per km.

```
uses fogmodel;

const
  nl=10000;

type
  layers=array[0..nl] of double;

var
  ng:ngtype;
  alt1,alt2,diam,c1,temp,ctemp,casw1,casw2,fc2,c2:extended;
  code,i:integer;
  watfact,icefact:nvect;
  vels,temps,alts,times:layers;
  lapse,dl:double;

function safeexp(x:double):extended;
begin
  if x<-100 then safeexp:=0
  else safeexp:=exp(x);
end;

function cavreal(diam,c1:double; ng:ngtype):double;
var
  phi,tau,diffs:layers;
  n,nmax:longint;
  isdone:boolean;
  i:integer;
  sum1,sum2,sum3,xn,y,y1,y2,a,mi:extended;
  dtau,dphi,x,sxn:extended;

begin
  for i:=0 to nl do diffs[i]:=ngd(ng,temps[i]);
  for i:=0 to nl do phi[i]:=ngasw(alts[i],temps[i],ng);
  a:=0.5*0.1*diam;
  nmax := 1000000;
  tau[0]:=0.0;
  for i:=1 to nl do
  tau[i]:=tau[i-1]+(times[i]-times[i-1])*0.5*(diffs[i]+diffs[i-1]);
  sum1:=0.0;
  sum2:=0.0;
  sum3:=0.0;
  n:=0;
  isdone:=false;
  repeat
    inc(n); xn:=n;
    y:=safeexp(-tau[nl]*sqr(n*pi/a));
    sum1:=sum1+y/sqr(xn);
    isdone:=(y<=1.0e-20)or(n>=nmax);
  until isdone;
  for i:=1 to nl do
    begin
      n:=0;
      isdone:=false;
      dtau:=tau[i]-tau[i-1];
      dphi:=phi[i]-phi[i-1];
      mi:=dphi/dtau;
      repeat
        inc(n); xn:=n;
        y1:=safeexp(sqr(n*pi/a)*(tau[i]-tau[nl]));
        y2:=safeexp(sqr(n*pi/a)*(tau[i-1]-tau[nl]));
        sxn:=sqr(xn);
        sum2:=sum2+y1*(phi[i]-sqr(a/xn/pi)*mi)/sxn;
        sum3:=sum3+y2*(phi[i-1]-sqr(a/xn/pi)*mi)/sxn;
        isdone:=(n>=nmax)or((y1<=1.0e-20)and(y2<=1.0e-20));
      until isdone;
    end;
    x:=(6.0/sqr(pi))*((sum1*c1)+sum2-sum3);
    cavreal:=x;
end;


begin
  if paramcount<>5 then
    begin
      writeln('usage realmodel3 T alt1 alt2 diam lapse');
      halt(-1);
    end;
  val(paramstr(1),ctemp,code); if code<>0 then halt(-1);
  val(paramstr(2),alt1,code); if code<>0 then halt(-1);
  val(paramstr(3),alt2,code); if code<>0 then halt(-1);
  val(paramstr(4),diam,code); if code<>0 then halt(-1);
  val(paramstr(5),lapse,code); if code<>0 then halt(-1);
  temp:=ctemp+zk;
  dl:=(alt1-alt2)/nl;
  for i:=0 to nl do alts[i]:=alt1-i*dl;
  for i:=nl downto 0 do temps[i]:=temp-lapse*dl*(nl-i)/1000.0;{lapse in
  deg/km}
  for i:=0 to nl do vels[i]:=dtermvel(diam,alts[i]);
  times[0]:=0.0;
  for i:=1 to nl do times[i]:=times[i-1]+dl/(0.5*(vels[i]+vels[i-1]));
  for ng:=he to xe do
    begin
      watfact[ng]:=1;
      icefact[ng]:=0;
    end;
  icefact[he]:=2.0; icefact[ne]:=0.80;
  for ng:=he to xe do
    begin
      casw1:=ngasw(alt1,temp,ng);
      c1:=watfact[ng]*casw1;
      casw2:=ngasw(alt2,temp,ng);
      c2:=cavreal(diam,c1,ng);
      fc2:=c2/casw2;
      write((ord(ng)+1):14,' ',fc2:14);
      c1:=icefact[ng]*casw1;
      c2:=cavreal(diam,c1,ng);
      fc2:=c2/casw2;
      writeln(' ',fc2:14);
    end;
end.
```

1.2. **Unit fogmodel.pas.** Program realmodel3.pas uses many of the functions and procedures in unit fogmodel.pas.

- function PFactor(h,t:extended):extended; {This calculates the pressure factor for noble gas partial pressures in the atmosphere at altitude h (m) and temperature t (K). The temperature is needed because the vapor pressure of water must be subtracted.}
- function vapour(t:extended):extended; {Vapor pressure of water at temperature t.}
- function ngd(i:ngtype; T:extended):extended; {Returns diffusion coefficient for noble gas "i" at temperature "T".}
- function dtermvel(d,h:double):extended; {Terminal velocity for droplet with diameter "d" at altitude "h".}
- function ngasw(h,t:double; ng:ngtype):extended; {ASW concentration for noble gas "ng" at altittude "h" and temperature "t". The default ASW value uses normal Henry's constants, but this can be altered to use modified constants from the Mercury et al. 2003,2004 model by first calling procedure filllnkp.}
- procedure filllnkp(t,p:double); {Alters the Henry's constants assuming a temperature "t" (K) and an internal pressure difference "p" in bars (Mercury et al., 2003, 2004).}

```
unit fogmodel;

INTERFACE

const
  h0 = 8350.0;
  zk = 273.15;


type
  ngtype=(he,ne,ar,kr,xe);
  nvect=array[ngtype] of double;

var
  lnkp:nvect;


function PFactor(h,t:extended):extended;
function vapour(t:extended):extended;
function ngd(i:ngtype; T:extended):extended;
function dtermvel(d,h:double):extended;
function ngasw(h,t:double; ng:ngtype):extended;
procedure filllnkp(t,p:double);


IMPLEMENTATION

type
  KCoeffArr=array[1..4,1..3] of extended;
  SalCoeffArr=array[1..4,1..3] of extended;
  parmarr=array[1..5] of double;

const

  Rjoule=8.31441;
  Eact:nvect =(11700,14840,20627,20200,21610);
  vs:nvect =(2.88,5.59,16.1,22.8,37.9);
  vair=20.1;

  D0:nvect=(8.18E-03,1.608e-2,0.106,6.393e-2,9.007e-2);

  PNoble:array[1..4] of extended =(1.667e-9, 3.183e-9, 6.582e-11,
  2.370e-12);
  PHe=5.24e-6;
  u1=3.4279e2;
  u2=-5.0866e-3;
  u3=9.4690e-7;
  u4=-2.0525;
  u5=3.1159e3;
  u6=-1.8289e2;
  u7=-8.0325e3;
  u8=4.21452e6;
  u9=2.1417;
  rgasbar=83.145;
  arparms:parmarr=(2.512,2874.324,12.761,-128156,-128574);
  heparms:parmarr=(1.4528,291.5,22.912,-117478,-88826);
  neparms:parmarr=(1.871,1311.8,18.902,-121696,-106064);
  krparms:parmarr=(2.6242,3151.9,11.67,-129302,-95437);
  xeparms:parmarr=(3.146,4426.4,6.661,-134570,-92634);


var


  KCoeff:KCoeffArr=(( 48.4575, -54.7275, -16.8278),
                    ( 64.9182, -84.9524, -23.6508),
                    ( 66.9928, -91.0166, -24.2207),
                    ( 74.7398, -105.210, -27.4664));

  SalCoeff:SalCoeffArr=(( -11.9556,  18.4062,  5.5464),
                        ( -10.6951,  16.7513,  4.9551),
                        ( -9.9787,   15.7619,  4.6181),
                        ( -14.5524,  22.5255,  6.7513));

  currparms:parmarr;
  integ,temperature:double;
  TrapzdIt: integer;
  tng:ngtype;

function spower(x,y:double):double;
var z,p:double;
begin
  z:=abs(x);
  if z=0 then z:=1e-35;
  z:=ln(z);
  p:=z*y;
  if p<-70 then p:=-70;
  if p>70 then p:=70;
  spower:=exp(p);
end;


function vapour(t:extended):extended;
const
  a=-5375.83585;
  b=21.2023734;
var x:extended;
begin
  x:=(a/t)+b;
  vapour:=exp(x);
end;


function PFactor(h,t:extended):extended;
const scaleh=8350.0;
var x:extended;
begin
  x:=760.0*exp(-h/scaleh)-vapour(t); {this is compatible with Stute}
  PFactor:=x/760.0;
end;
```

```
function ngd(i:ngtype; T:extended):extended;
begin
  ngd:=D0[i]*exp(-Eact[i]/(Rjoule*T));
end;

function dtermvel(d,h:double):extended;
const scaleh=8350.0;
var x,p,v0:extended;
begin
  p:=exp(0.4*h/scaleh);
  x:=ln(d/1.77);
  x:=x*1.147;
  x:=exp(x);
  v0:=9.43*(1-exp(-x));
  dtermvel:=v0*p;
end;

function epsilon(t,p:double):double;
var
  eps1000,c,b:double;
begin
  eps1000:=u1*exp(u2*t+u3*sqr(t));
  c:=u4+u5/(u6+t);
  b:=u7+u8/t+u9*t;
  epsilon:=eps1000+c*ln((b+p)/(b+1000));
end;

function born(t,p:double):double;
var
  tmp,eps:double;
  eps1000,c,b:double;
begin
  eps1000:=u1*exp(u2*t+u3*sqr(t));
  c:=u4+u5/(u6+t);
  b:=u7+u8/t+u9*t;
  eps:=eps1000+c*ln((b+p)/(b+1000));
  tmp:=c/((b+p)*sqr(eps));
  born:=tmp;
end;

function v0(t,p:double; var a:parmarr):double;
const
  psi=2600;
  theta=228;
begin
  v0:=10.0*(a[1]+a[2]/(psi+p)+a[3]/(t-theta)
       +a[4]/((psi+p)*(t-theta))-born(t,p)*a[5]);
end;

FUNCTION func(x: double): double;
begin
  func:=v0(temperature,x,currparms);
end;

PROCEDURE trapzd(a,b: double;
                 VAR s: double;
```

```
                      n: integer);
VAR
   j: integer;
   x,tnm,sum,del: double;
BEGIN
   IF n = 1 THEN BEGIN
      s := 0.5*(b-a)*(func(a)+func(b));
      TrapzdIt := 1
   END
   ELSE BEGIN
      tnm := TrapzdIt;
      del := (b-a)/tnm;
      x := a+0.5*del;
      sum := 0.0;
      FOR j := 1 TO TrapzdIt DO BEGIN
         sum := sum+func(x);
         x := x+del
      END;
      s := 0.5*(s+(b-a)*sum/tnm);
      TrapzdIt := 2*TrapzdIt
   END
END;

PROCEDURE qtrap(a,b: double;
                VAR s: double);
LABEL 99;
CONST
   eps = 1.0e-8;
   jmax = 30;
VAR
   j: integer;
   olds: double;
BEGIN
   olds := -1.0e30;
   FOR j := 1 TO jmax DO BEGIN
      trapzd(a,b,s,j);
      IF abs(s-olds) < eps*abs(olds) THEN GOTO 99;
      olds := s
   END;
   writeln ('pause in QTRAP - too many steps');
   readln;
99:
END;

procedure filllnkp(t,p:double);
var pressure:double;
begin
    temperature:=t;
    pressure:=p;
    currparms:=heparms;
    qtrap(1.01325,pressure,integ);
    integ:=integ/(rgasbar*temperature);
    lnkp[he]:=integ;
    currparms:=neparms;
    qtrap(1.01325,pressure,integ);
    integ:=integ/(rgasbar*temperature);
```

```
      lnkp[ne]:=integ;
      currparms:=arparms;
      qtrap(1.01325,pressure,integ);
      integ:=integ/(rgasbar*temperature);
      lnkp[ar]:=integ;
      currparms:=krparms;
      qtrap(1.01325,pressure,integ);
      integ:=integ/(rgasbar*temperature);
      lnkp[kr]:=integ;
      currparms:=xeparms;
      qtrap(1.01325,pressure,integ);
      integ:=integ/(rgasbar*temperature);
      lnkp[xe]:=integ;
  end;


  function lnK(inoble:integer; T:extended):extended;
  begin
    lnK:=KCoeff[inoble,1]-9.1971774+KCoeff[inoble,2]*100.0/T
         +KCoeff[inoble,3]*ln(0.01*T);
  end;

  function sal(inoble:integer; T:extended):extended;
  begin
    sal:=SalCoeff[inoble,1]+100.0*SalCoeff[inoble,2]/T
         +SalCoeff[inoble,3]*ln(0.01*T);
  end;

  function hehenry(t:double):double;
  const rgas=1.98717e-3;
  var x:double;
  begin
    x:=-167.2178+216.3442*(100.0/t)+139.2032*ln(t/100.0)
        -22.6202*t/100.0;
    hehenry:=exp(x)/1000.0;
  end;

  function ngasw(h,t:double; ng:ngtype):extended;
  const
    molvol=2.24138e4;
    watermolwt=18.015;
    scaleh=8350.0;
    isofracs:array[ne..xe] of double = (0.905,0.003364,0.57,0.2689);
  var vol,pf,ts,asw:extended;
      i:integer;
      na:double;
  begin
    na:=0;
    if ng=he then
      begin
        pf:=exp(-h/scaleh);
        ngasw:=pf*hehenry(t)*exp(-lnkp[ng]);
      end
    else
      begin
        i:=ord(ng);
        pf:=PFactor(h,t);
        ts:=sal(i,T);
        asw:=pf*PNoble[i]*exp(-Na*ts-lnK(i,T)-lnkp[ng]);
        vol:=asw*molvol/(isofracs[ng]*watermolwt);
        ngasw:=vol;
      end;
    end;


    begin
      for tng:=he to xe do lnkp[tng]:=0.0;
    end.
```

1.3. **Unit surfacestuff.pas.** This unit provides surface tension and Young-Laplace pressure functions.

- function surftension(tcel:double):double; {Returns the surface tension of water at temperature "tcel" (C).}
- function plaplace(d,tcel:double):double; {Returns the Young-Laplace pressure (bars) inside a droplet with diameter "d" (microns) and at temperature "tcel" (C).}

```pascal
unit surfacestuff;

interface

function surftension(tcel:double):double;
function plaplace(d,tcel:double):double;

implementation

const
  zk = 273.15;

function surftension(tcel:double):double;{in mN/m}
const
  tc=647.096;
var lnx,x,t,y:double;
begin
  t:=tcel+zk;
  x:=1.0-t/tc;
  lnx:=ln(x)*1.256;
  y:=1.0-0.625*x;
  surftension:=y*235.8*exp(lnx);
end;{surftension}

function plaplace(d,tcel:double):double;{d in microns, P in bar}
const bar=1.0e5;
var
  gam,r:double;
begin
  gam:=surftension(tcel)*0.001;
  r:=0.5*d*1.0e-6;
  plaplace:=gam*2.0/(r*bar);;
end;{plaplace}

begin
end.
```

1.4. **Program tsurf.pas.** This is a test program to check out some of the features of units susrfacestuff and fogmodel, but it was used to generate the Mercury et al. (2003, 2004) model plots in Fig. 6 of the main text. When the user inputs a droplet diameter value, the internal Young-Laplace pressure is calculated and used to modify the noble gas Henry's constants using a call to filllnpk. Note that if a pressure "p" value of zero is passed to the procedure, the normal solubility constants will be used by the function ngasw. The Pascal language allows for labels to be mapped into a range of ordinals, which explains why it is allowed to have a "for" loop iterating from he to xe. The type "ngtype" is defined in the unit fogmodel.

55
```
uses surfacestuff,fogmodel;

var
  tcel,diam,p:double;
  ng:ngtype;

begin
  write('Input T in Celcius ');
  readln(tcel);
  writeln('Surface tension in mN/m = ',surftension(tcel));
  write('Input drop diameter in microns ');
  readln(diam);
  p:=plaplace(diam,tcel);
  writeln('Laplace P in bars = ',plaplace(diam,tcel));
  filllnkp(tcel+zk,p);
  for ng:=he to xe do writeln(ord(ng),' ',lnkp[ng]);
  writeln('Sea Level ASW Values');
  for ng:=he to xe do writeln(ord(ng),' ',ngasw(0,tcel+zk,ng));
end.
```

2. ES Cluster Layer Modeling

These two programs were used to generate model values for figures S5 and S6 in the supplementary material. They both use the unit fdensity, which empoys the water density model of Vedamuthu et al. (1994).

2.1. **Program DelGibbs.pas.**

This program was used to generate the model data used in Fig. S5. It takes two commandline parameters, the temperature in °C and the number of water molcules in a cluster. The output is the value of ΔG as a function of the thickness of an ES-rich surface layer, relative to a water droplet with no internal Young-Laplace pressure.

```
uses fdensity,surfacestuff;

const
  wh2o=0.018;

var
  i,j:integer;
  a,{aprime,}w:double;
  tc:double;
  nh2o:integer;
  tmin,x:double;
  s:string;
  code:word;

function cube(x:double):double;
begin
  cube:=x*x*x;
end;

function rhoi(tc,q:double):double;
begin
  rhoi:=rhoe(tc)+(rhow(tc)-rhoe(tc))*cube(q);
end;

function meout(tc,a,thick:double):double;
var
  aprime,vshell,meorg,menew:double;
begin
  aprime:=a-thick;
  vshell:=4*pi*(cube(a)-cube(aprime))/3;
  menew:=vshell*rhoe(tc);
  meorg:=vshell*fe(tc)*rhow(tc);
  meout:=menew-meorg;
end;

function molshell(tc,a,thick,w:double):double;
var
  aprime,vshell,menew:double;
begin
  aprime:=a-thick;
  vshell:=4*pi*(cube(a)-cube(aprime))/3;
  menew:=vshell*rhoe(tc);
  molshell:=menew/w;
end;


function mcin(tc,a,thick:double):double;
var
  aprime,vshell:double;
begin
  aprime:=a-thick;
  vshell:=4*pi*(cube(a)-cube(aprime))/3;
  mcin:=fc(tc)*vshell*rhow(tc);
end;

function delueout(tc,a,thick:double):double;
```

```
var
  q,aprime:double;
begin
  aprime:=a-thick;
  q:=a/aprime;

  delueout:=meout(tc,a,thick)*1.0e5*0.5*plaplace(a*1e6,tc)*(1.0/rhoe(tc)-
  1.0/rhoi(tc,q));
end;

function delucin(tc,a,thick:double):double;
var
  q,aprime:double;
begin
  aprime:=a-thick;
  q:=a/aprime;

  delucin:=mcin(tc,a,thick)*1.0e5*0.5*plaplace(a*1e6,tc)*(1.0/rhoi(tc,q)-
  1.0/rhoc(tc));
end;

function delh(tc,a,thick:double):double;
begin
  delh:=-delueout(tc,a,thick)-delucin(tc,a,thick);
end;

function mctot(tc,a:double):double;
begin
  mctot:=4*pi*fc(tc)*rhow(tc)*cube(a)/3;
end;

function tdelscin(tc,a,thick,w:double):double;
const
  zk=273.15;
  rgas=8.312;
var
  molc:double;
  dels:double;
  tk:double;
  {aprime:double;}
begin
  tk:=tc+zk;
  molc:=mcin(tc,a,thick)/w;
  {aprime:=a-thick;}
  dels:=3*rgas*ln(1.0-thick/a);
  tdelscin:=molc*tk*dels;
end;

function delgibbs(x:double):double;
begin

  delgibbs:=(delh(tc,a,x*1.0e-9)-tdelscin(tc,a,x*1.0e-9,w))/molshell(tc,a
  ,x*1e-9,w);
end;
```

```
begin
  if paramcount<>2 then
    begin
      writeln('usage delgibbs tc nh2o');
      halt(1);
    end;
  s:=paramstr(1);
  val(s,tc,code);
  if code<>0 then
    begin
      writeln('incorrect input format for tc');
      halt(1);
    end;
  s:=paramstr(2);
  val(s,nh2o,code);
  if code<>0 then
    begin
      writeln('incorrect input format for nh2o');
      halt(1);
    end;
  w:=wh2o*nh2o;
  writeln;
  for i:=1 to 40 do
    begin
      tmin:=0.1*i;
      x:=tmin;
      write(tmin:8:2);
      for j:=1 to 5 do
        begin
          a:=j*1.0e-6;
          write(' ',delgibbs(x):13);
        end;
      {aprime:=a-thick;}
      writeln;
    end;
end.
```

2.2. **Program Gibbs5.pas.** This program takes one commandline parameter, the average number of water molecules in a cluster. The then calculates the expected drop in enthalpy caused by the formation of an ES-rich surface layer for a range of temperatures and drople sizes. For each calculation the expected shell thickness is found using shellmin by assuming that the is no change for the Gibbs free energy compared to an unpressurized droplet.

```
uses fdensity,surfacestuff;

const
  wh2o=0.018;

var
  i,j:integer;
  a,{aprime,}w:double;
  tc:double;
  nh2o:integer;
  tmin:double;
  s:string;
  code:word;

function cube(x:double):double;
begin
  cube:=x*x*x;
end;

function rhoi(tc,q:double):double;
begin
  rhoi:=rhoe(tc)+(rhow(tc)-rhoe(tc))*cube(q);
end;

function meout(tc,a,thick:double):double;
var
  aprime,vshell,meorg,menew:double;
begin
  aprime:=a-thick;
  vshell:=4*pi*(cube(a)-cube(aprime))/3;
  menew:=vshell*rhoe(tc);
  meorg:=vshell*fe(tc)*rhow(tc);
  meout:=menew-meorg;
end;

function molshell(tc,a,thick,w:double):double;
var
  aprime,vshell,menew:double;
begin
  aprime:=a-thick;
  vshell:=4*pi*(cube(a)-cube(aprime))/3;
  menew:=vshell*rhoe(tc);
  molshell:=menew/w;
end;


function mcin(tc,a,thick:double):double;
var
  aprime,vshell:double;
begin
  aprime:=a-thick;
  vshell:=4*pi*(cube(a)-cube(aprime))/3;
  mcin:=fc(tc)*vshell*rhow(tc);
end;

function delueout(tc,a,thick:double):double;
```

```
var
  q,aprime:double;
begin
  aprime:=a-thick;
  q:=a/aprime;

  delueout:=meout(tc,a,thick)*1.0e5*0.5*plaplace(a*1e6,tc)*(1.0/rhoe(tc)-
  1.0/rhoi(tc,q));
end;

function delucin(tc,a,thick:double):double;
var
  q,aprime:double;
begin
  aprime:=a-thick;
  q:=a/aprime;

  delucin:=mcin(tc,a,thick)*1.0e5*0.5*plaplace(a*1e6,tc)*(1.0/rhoi(tc,q)-
  1.0/rhoc(tc));
end;

function delh(tc,a,thick:double):double;
begin
  delh:=-delueout(tc,a,thick)-delucin(tc,a,thick);
end;

function mctot(tc,a:double):double;
begin
  mctot:=4*pi*fc(tc)*rhow(tc)*cube(a)/3;
end;

function tdelscin(tc,a,thick,w:double):double;
const
  zk=273.15;
  rgas=8.312;
var
  molc:double;
  dels:double;
  tk:double;
  {aprime:double;}
begin
  tk:=tc+zk;
  molc:=mcin(tc,a,thick)/w;
  {aprime:=a-thick;}
  dels:=3*rgas*ln(1.0-thick/a);
  tdelscin:=molc*tk*dels;
end;

procedure shellmin(tc,a,w:double; var tmin:double);
{finds thickness where delta-H balances T*delta-S}
var x1,x2,xacc:double;

    function fx(x:double):double;
    var tmp:double;
    begin
      tmp:=delh(tc,a,x*1.0e-9)-tdelscin(tc,a,x*1.0e-9,w);
      {writeln(tmp);}
      fx:=tmp;
    end;{fx}

    FUNCTION rtflsp(x1,x2,xacc: double): double;
    {Modified from rtflsp in Press et al., 1989, Numerical Recipes in
    Pascal.}
    LABEL 99;
    CONST
       maxit = 3000;
    VAR
       xl,xh,swap,fl: double;
       dx,del,f,fh,rtf: double;
       j: integer;
    BEGIN
       fl := fx(x1);
       fh := fx(x2);
       IF fl*fh > 0.0 THEN BEGIN
          writeln('pause in routine RTFLSP');
          writeln('Root must be bracketed for false position');
          readln
       END;
       IF fl < 0.0 THEN BEGIN
          xl := x1;
          xh := x2
       END
       ELSE BEGIN
          xl := x2;
          xh := x1;
          swap := fl;
          fl := fh;
          fh := swap
       END;
       dx := xh-xl;
       FOR j := 1 TO maxit DO BEGIN
          rtf := xl+dx*fl/(fl-fh);
          f := fx(rtf);
          IF f < 0.0 THEN BEGIN
             del := xl-rtf;
             xl := rtf;
             fl := f
          END
          ELSE BEGIN
             del := xh-rtf;
             xh := rtf;
             fh := f
          END;
          dx := xh-xl;
          IF (abs(del) < xacc) OR (f = 0.0) THEN
             GOTO 99
       END;
       writeln('pause in routine RTFLSP');
       writeln('maximum number of iterations exceeded');
       readln;
    99:
       rtflsp := rtf
```

```
      END;

  begin{shellmin}
    xacc:=1.0e-5;
    x1:=0.1;
    x2:=100;
    tmin:=rtflsp(x1,x2,xacc);
  end;{shellmin}

  begin
    if paramcount<>1 then
      begin
        writeln('usage gibbs5 nh2o');
        halt(1);
      end;
    s:=paramstr(1);
    val(s,nh2o,code);
    if code<>0 then
      begin
        writeln('incorrect input format for nh2o');
        halt(1);
      end;
    w:=wh2o*nh2o;
    for i:=1 to 50 do
      begin
        a:=i*0.5e-6;
        write(2*a*1e6:8:2);
        for j:=0 to 5 do
          begin
            tc:=5.0*j;
            shellmin(tc,a,w,tmin);
            write(' ',delh(tc,a,tmin)/molshell(tc,a,tmin,w):13);
          end;
        {aprime:=a-thick;}
        writeln;
      end;
  end.
```

2.3. **Unit fdensity.pas.** This unit implements the water density model of Vedamuthu et al. (1994) and exports the following 5 functions:

(1) function fc(tc:double):double; {returns the mass fraction of CS clusters at temperature tc (Celsius)}
(2) function fe(tc:double):double; {returns the mass fraction of ES clusters at temperature tc (Celsius)}
(3) function rhoe(tc:double):double; {ES cluster density at temperature tc in kg/m^3}
(4) function rhoc(tc:double):double; {CS cluster density at temperature tc in kg/m^3}
(5) function rhow(tc:double):double; {bulk water density in in kg/m^3}

```
unit fdensity;

interface
uses math;

function fc(tc:double):double;
function fe(tc:double):double;
function rhoe(tc:double):double;
function rhoc(tc:double):double;
function rhow(tc:double):double;


implementation


const

{fit ii}
A = 0.0454866;
B = 0.00036522;
C = 0.0869196;
T0 = 225.334;
V1T0 = 1.08739;
rho1T0 = 0.9196;
V11T0 = 0.84745;
rho11T0 = 1.18;
alpha1 = 0.000457558;
alpha11 = 0.00129374;
beta1 = 0;
beta11 = 0;

zk=273.15;

function v1(tc:double):double;
var t:double;
begin
  t:=tc+zk;
  v1:=V1T0*(1+alpha1*(t-t0)+beta1*sqr(t-t0));
end;

function v2(tc:double):double;
var t:double;
begin
  t:=tc+zk;
  v2:=V11T0*(1+alpha11*(t-t0)+beta11*sqr(t-t0));
end;

function v(tc:double):double;
begin
  v:=fc(tc)*v2(tc)+fe(tc)*v1(tc);
end;

function fc(tc:double):double;
var t:double;
begin
  t:=tc+zk;
  fc:=TANH((A*(t-t0)+B*sqr(t-t0))/(1+C*(t-t0)));
end;

function fe(tc:double):double;
begin
  fe:=1-fc(tc);
end;

function rhoe(tc:double):double;
begin
  rhoe:=1000.0/v1(tc);
end;

function rhoc(tc:double):double;
begin
  rhoc:=1000.0/v2(tc);
end;

function rhow(tc:double):double;
begin
  rhow:=1000.0/v(tc);
end;

begin
end.
```