IV. PROGRAM FLOWSHEETS AND LISTINGS

IV.I. First Model

Listing of First Model Program

```
%begin
%external %routine %spec prompt %alias "S#PROMPT"(%string (255) s)
%routine define(%string (255) s)
%external %routine %spec emas3(%string %name command,params, %integer
%name flag)
%integer flag
emas3("DEFINE",s,flag)
%end;  ! Of %routine define.
!
%integer kmax,n,s,i,k,j,flag,u,count,b,p
%long %real ph,pl,th,tl,phi,pli,thi,tli,tfhi,tfli
%long %real ktheta,ksec,dsec,dtheta
%long %real z,y,theta,vh,dvh,vl,dvl,sec,old,gamma,htc
%real dndt,dvdt,aa,bb,cc,a1,a2,a3,a4,a5,a6
%real vho,vlo,dvho,dvlo
%real a,psec,dmass,mass,dpower,power
%real tclose,salt,area,height
%long %real %array x,oldx,delta,e(1:100),dens(0:20,0:20)
aa = 7.9186968; bb = 1636.909; cc = 224.92
gamma = 1.135
!First the routines needed in the program are defined:
!ADIABAT contains the equations
!for adiabatic compression.
%routine ADIABAT(%real dvh,dvl,vh,vl,phi,pli,thi,tli,
%long %real %name ph,pl,th,tl)
%real aph,apl,gamma
!
gamma = 1.135
!
aph = -1*gamma*phi*dvh/vh
apl = -1*gamma*pli*dvl/vl
!
th = (gamma-1)*thi*aph/(gamma*phi)+thi
tl = (gamma-1)*tli*apl/(gamma*pli)+tli
!
ph = aph+phi
pl = apl+pli
!
%end
!
!PHYSPROP contains a few simple approximations
! for the variation of the properties
!of brine or water with temperature
!and/or salinity.
%routine PHYSPROP(%real %name s, %long %real %name t,r,cp,bpe, %long
%real %array %name a)
```

```
!
!First, the densities.....
!
%real dx,dy,x,y,p,q
%integer i,j,aa,bb,cc,dd
!
t = t-273.15 {this calculation works in degC, not K}
!
dx = a(2,0)-a(1,0)
dy = a(0,2)-a(0,1)
x = (s-a(1,0))/dx+1
y = (t-a(0,1))/dy+1
!
aa = intpt(x)
bb = intpt(x+1)
cc = intpt(y)
dd = intpt(y+1)
!
%if aa>8 %then aa = 8
%if aa<0 %then aa = 1
%if bb>8 %then bb = 8
%if bb<0 %then bb = 1
%if cc>17 %then cc = 17
%if cc<0 %then cc = 1
%if dd>17 %then dd = 17
%if dd<0 %then dd = 1
!
p = (a(aa,dd)-a(aa,cc))*(y-cc)+a(aa,cc)
q = (a(bb,dd)-a(bb,cc))*(y-cc)+a(bb,cc)
r = (q-p)*(x-aa)+p
!
!Next, the heat capacities....
!
t = t+273.15 {this bit works in K}
!
%real a0,a1,a2,a3,b0,b1,b2,b3,c0,c1,c2,c3,d0,d1,d2,d3
!
a1 = 5.328; a2 = -9.76@-2; a3 = 4.04@-4
b1 = -6.913@-3; b2 = 7.351@-4; b3 = -3.15@-6
c1 = 9.6@-6; c2 = -1.927@-6; c3 = 8.23@-9
d1 = 2.5@-9; d2 = 1.666@-9; d3 = -7.125@-12
!
a0 = a1+a2*s+a3*s**2
b0 = b1+b2*s+b3*s**2
c0 = c1+c2*s+c3*s**2
d0 = d1+d2*s+d3*s**2
!
cp = a0+b0*t+c0*t**2+d0*t**3
cp = cp*1000
!
!And, finally, the boiling point elevation....
!
%real conc,bpe1,bpe2,bpe3,bpe4,bpe5,bpe6
!
```

```
conc = s/1000
!
bpe1 = conc*t**2/13832
bpe2 = 0.001373*t
bpe3 = -0.00272*t*conc**0.5
bpe4 = 17.86*conc
bpe5 = -0.0152*conc*t*((t-225.9)/(t-236))
bpe6 = -(2583*conc*(1-conc))/t
!
bpe = bpe1*(1+bpe2+bpe3+bpe4+bpe5+bpe6)
!
!And thats all folks.....
!
%end
!
!EVALUATE contains the equations which describe
!compression with heat and mass transfer .
%routine evaluate(%long %real %array %name f,x, %real
phi,pli,thi,tli,tfhi,tfli,tclose,salt,a)
%long %real aa,bb,cc,a1,a2,a3,ca,ka,kb
%long %real cpw,cpb,gamma,rhow,rhob
%long %real lamw,lamb,hc,he,w,xw,xb,r,k
%long %real mr,mf,tmix,bpe,tfeed,conc
aa = 7.9186968; bb = 1636.909; cc = 224.92
!
xw = 0.3@-3; xb = 0.27@-3
lamw = 2258@3; lamb = 2258@3
r = 461.89; gamma = 1.135; k = 104
w = 0.85@-3; conc = salt/34.5
!
ca = (gamma-1)/gamma
!
cpw = 4216; rhow = 962
physprop(salt,x(4),rhob,cpb,bpe,dens)
mf = 12*salt/(salt-34.5)
mr = 0.085*a-mf
tfeed = ((conc-1)*(x(3)-tclose)+(x(4)-tclose))/conc
tmix = (x(4)*mr+tfeed*mf)/(mr+mf)
!
f(1) = (ca*thi*(x(5)-phi)/phi)+thi-x(1)
f(2) = (ca*tli*(x(6)-pli)/pli)+tli-x(2)
f(3) = a*cpw*rhow*x(10)*(tfhi-x(3))+
       x(9)*a*(x(7)-x(3))*dsec+k*a*dsec*(x(4)-x(3))/w
f(4) = cpb*(mf+mr)*dsec*(tfli-tmix)+
       cpb*a*xb*rhob*(x(4)-tfli)+6911*a* (x(4)-x(8))*dsec+k*a*dsec*(x(4)-x(3))/w
f(5) = (-1*gamma*phi*dvh/vh)-(x(9)*a*(x(7)-x(3))
       *dsec*r*x(1)/(vh*lamw))+phi-x(5)
f(6) = (-1*gamma*pli*dvl/vl)+(6911*a*(x(4)-x(8))
       *dsec*r*x(2)/(vl*lamb))+pli-x(6)
f(7) = (bb/(aa-0.434*log(x(5)*760/1.013@5)))
       -cc-x(7)+273.15
f(8) = (bb/(aa-0.434*log(x(6)*760/1.013@5)))
       -cc-x(8)+273.15+bpe
%if x(1)<x(7) %then x(1) = x(7)
```

```
%if x(2)<x(8) %then x(2) = x(8)
!
f(9) = (x(9)**4)*(x(7)-x(3))-1.26@16
f(10) = 1.2262@-16*x(9)*(x(7)-x(3))-(x(10)**3)
%return
%end
!
!SOLVER contains the Newton equation solving method
%routine SOLVER(%long %real %array %name x,e, %integer n, %integer
%name kmax,flag, %real phi,pli,thi,tli,tfhi,tfli)
%long %real det
%integer i,j,iter,m,h
%long %real %array b,dx,f(1:n),a(1:n,1:n)
%external %routine %spec solve ln eq(%long %real %array %name a,f,
%integer n, %long %real %name det)
%for i = 1,1,n %cycle
dx(i) = 10*e(i)
%repeat
!
%for iter = 1,1,kmax %cycle
flag = 1
!
evaluate(f,x,phi,pli,thi,tli,tfhi,tfli,tclose,salt,area)
!
%for m = 1,1,n %cycle
b(m) = -f(m)
%repeat
!
%for j = 1,1,n %cycle
x(j) = x(j)+dx(j)
evaluate(f,x,phi,pli,thi,tli,tfhi,tfli,tclose,salt,area)
%for i = 1,1,n %cycle
a(i,j) = (f(i)+b(i))/dx(j)
%repeat
!
x(j) = x(j)-dx(j)
%repeat
!
solve ln eq(a,b,n,det)
!
!
%if mod(det)<10@-8 %then flag = 2 %and %return
!
%for h = 1,1,n %cycle
x(h) = x(h)+b(h)
%if mod(b(h))>e(h) %then flag = 0
%repeat
%if flag=1 %then %return
!
%repeat
%return
%end
!
!
```

```
!Input of initial data from the terminal-
prompt("Initial pressure on h.p. side "); read(phi)
prompt("Initial pressure on l.p. side "); read(pli)
prompt("Initial temperature on h.p. side "); read(thi)
prompt("Initial temperature on l.p. side "); read(tli)
prompt("Initial water film temperature "); read(tfhi)
prompt("Initial brine film temperature "); read(tfli)
prompt("Temp. approach in H.Recov.  "); read(tclose)
prompt("Area for heat exchange "); read(area)
prompt("Brine concentration factor "); read(salt)
salt = salt*34.5
prompt("Swept angle "); read(ktheta)
ktheta = ktheta*pi/180
prompt("Period of oscillation "); read(ksec)
prompt("Number of timesteps "); read(s)
!Output files are defined-
define("1,theta")
define("2,temp")
define("3,delp")
define("4,time")
define("5,mass")
define("6,data")
define("10,emct12:general.densities")
!
selectinput(10)
%for i = 0,1,17 %cycle
%for j = 0,1,8 %cycle
read(dens(j,i))
%repeat
%repeat
closestream(10)
!
!The timestep size is calculated
dsec = ksec/(s)
n = 10
!Initial guess for x() variables
! x(1)=th;x(2)=tl;x(3)=tfh;x(4)=tfl;x(5)=ph
! x(6)=pl;x(7)=tsat;x(8)=tsat;x(9)=hc;x(10)=xw
x(1) = tli; x(2) = thi; x(3) = tfhi; x(4) = tfli
x(5) = pli; x(6) = phi
x(9) = 10000; x(10) = 0.2@-3
!Iteration tolerances are defined
e(1) = 0.05; e(2) = 0.05; e(3) = 0.05; e(4) = 0.05
e(5) = 1000; e(6) = 1000; e(7) = 0.05; e(8) = 0.05;
e(9) = 100; e(10) = 0.01@-3
!
count = 0
old = -ktheta/2
kmax = 500
!
!
u = 1
b = 1
dmass = 0
```

```
mass = 0
dpower = 0
power = 0
p = 0
k = 0
psec = 0
a = ktheta/ksec
y = 0
vho = 20*(4.5*(pi/2-old)+0.125*old-1.5)
vlo = 20*(4.5*(pi/2+old)-1.5-0.125*tan(old))
dvho = 0
dvlo = 0
! Time cycle starts here
%for i = 0,1,s+1 %cycle
! The program will go through a number of wave cycles.
sec = dsec*i
%if mod(sec-ksec/2)<0.01*dsec %and b=1 %then %start
selectinput(0)
prompt("New swept angle ")
read(ktheta)
ktheta = ktheta*pi/180
ksec = (ktheta/a)
dsec = ksec/s
%finish
%if i=1+s %then %start
%if k>4 %then %exit
k = k+1
thi = tl; tli = th; pli = ph; phi = pl
count = 0
old = -ktheta/2
b = -b
u = 1
vho = 20*(4.5*(pi/2-old)+0.125*old-1.5)
vlo = 20*(4.5*(pi/2+old)-1.5-0.125*tan(old))
i = 0
%finish
sec = dsec*i
!
! Theta describes the angle of rotation of the duck.
! In this case theta is a sine function .
theta = -(ktheta/2)*cos(3.142*sec/ksec)
dtheta = theta-old
old = theta
!
!
x(7) = (bb/(aa-0.434*log(x(5)*7.502@-3)))-cc+273
x(8) = (bb/(aa-0.434*log(x(6)*7.502@-3)))-cc+273
%for j = 1,1,10 %cycle
oldx(j) = x(j)
%repeat
!If u=1 then the valves are closed, if u=0 then they are open
%if u=1 %then %start
!
z = 1
```

```
!
dvh = 0; dvl = 0
vh = 34; vl = 34
a1 = x(5); a2 = x(6); a3 = x(1); a4 = x(2); a5 = x(3)
a6 = x(4)
!
kmax = 500
! SOLVER solves for Ps,Ts et.c when valves are closed
solver(x,e,n,kmax,flag,a1,a2,a3,a4,a5,a6)
!should the solution fail then the following messages are printed-
%if flag=2 %then %start
newline
printstring("System is ill-conditioned.")
%stop
%finish
!
%if flag=0 %then %start
newline
printstring("System has failed to converge.")
%exit
%finish
!
%if theta<0 %then %start
vho = 20*(4.5*(pi/2-theta)+0.125*theta-1.5)
vlo = 20*(4.5*(pi/2+theta)-0.125*tan(theta)-1.5)
dvho = -87.5*dtheta
dvlo = 20*(4.5-(0.125/(cos(theta)**2)))*dtheta
%finish
!
%if theta>=0 %then %start
vho = 20*(4.5*(pi/2-theta)+0.125*tan(theta)-1.5)
vlo = 20*(4.5*(pi/2+theta)-0.125*theta-1.5)
dvho = 20*(0.125/(cos(theta)**2)-4.5)*dtheta
dvlo = 87.5*dtheta
%finish
!
! ADIABAT solves for Ps,Ts in adiabatic compression .
adiabat(dvho,dvlo,vho,vlo,phi,pli,thi,tli,ph,pl,th,tl)
! If 5 timesteps are passed then the program prints data.
%finish
!
%if count=5 %then %start
count = 0
!
selectoutput(1)
print(psec,2,2); space; print(theta,1,2); space; print(th,3,2)
space; print(tl,3,2)
space; print(x(3),3,2); space; print(x(4),3,2)
space; print(ph/1@5,1,3); space; print(pl/1@5,1,3)
space; print(x(5)/1@5,1,3); space; print(x(6)/1@5,1,3)
newline
!
selectoutput(2)
newline
```

```
print(x(9),4,2); space; print(x(10)*1000,1,3)
!
selectoutput(3)
newline
print(psec,1,2); space; print(b*(ph-pl)/1@5,1,3)
!
selectoutput(4)
newline
%finish
! Test to see if the valves are open
! (with a 0.01 bar pressure drop).
%if x(5)+0.01@5<ph %and x(6)>pl+0.01@5 %then u = 0
! Update the x values according to previous changes .
%for j = 1,1,10 %cycle
delta(j) = x(j)-oldx(j)
x(j) = x(j)+0.75*delta(j)
%repeat
! Reset initial values .
phi = ph; pli = pl; thi = th; tli = tl
tfhi = x(3); tfli = x(4)
! Check if the valves are open .
%if u=0 %then %start
!
!
%if theta<0 %then %start
dvh = -87.5*dtheta
dvl = 20*(4.5-(0.125/(cos(theta)**2)))*dtheta
%finish
!
%if theta>=0 %then %start
dvh = 20*(0.125/(cos(theta)**2)-4.5)*dtheta
dvl = 87.5*dtheta
%finish
!
kmax = 200
! If the valves are open for the first time , then a
! different type of guess is needed .
%if z=1 %then %start
vh = vho+34
vl = vlo+34
kmax = 300
thi = 0.5*thi+0.5*x(1)
tli = 0.5*tli+0.5*x(2)
phi = 0.5*phi+0.5*x(5)
pli = 0.5*pli+0.5*x(6)
x(1) = thi
x(2) = tli
x(5) = phi
x(6) = pli
x(7) = (bb/(aa-0.434*log(phi*7.502@-3)))-cc+273
x(8) = (bb/(aa-0.434*log(pli*7.502@-3)))-cc+273
x(9) = 9000
x(10) = 0.2@-3
%finish
```

```
!
z = 0
! Solve for HE behaviour with open valves .
solver(x,e,n,kmax,flag,phi,pli,thi,tli,tfhi,tfli)
! If equation fails the following messages will be printed
!
%if flag=2 %then %start
newline
printstring("System is ill conditioned.")
%stop
%finish
%if flag=0 %then %start
newline
printstring("System has failed to converge.")
%exit
%finish
! Update initial values
ph = x(5); pl = x(6); th = x(1); tl = x(2)
tfhi = x(3); tfli = x(4)
%finish
%if theta<0 %then %start
vh = 20*(4.5*(pi/2-theta)+0.125*theta-1.5)+34
vl = 20*(4.5*(pi/2+theta)-0.125*tan(theta)-1.5)+34
%finish
!
%if theta>=0 %then %start
vh = 20*(4.5*(pi/2-theta)+0.125*tan(theta)-1.5)+34
vl = 20*(4.5*(pi/2+theta)-0.125*theta-1.5)+34
%finish
!
count = count+1
psec = psec+dsec
p = p+1
dmass = x(9)*area*(x(7)-x(3))/2258@3
dmass = dmass**2
mass = mass+dmass
!
dpower = b*(ph-pl)*90*dtheta/dsec
dpower = dpower**2
power = power+dpower
!
%repeat
! Take next timestep
!
mass = mass/p
mass = mass**0.5
!
power = power/p
power = power**0.5
!
selectoutput(6)
printstring("Area for heat exchange ="); print(area,4,1)
newline
printstring("Temp. approach ="); print(tclose,3,2)
```

```
newline
printstring("RMS Fresh Water Flowrate ="); print(mass,3,3)
printstring("kg/s"); newline
printstring("RMS Power Consumption ="); print(power/1000,3,3)
printstring("kW"); newline
printstring("RMS Specific Energy Usage ="); print(power/(mass*1000),3,3)
printstring("kJ/kg"); newline
printstring("Mean Wave Climate ="); print(power/20@3,2,2)
printstring("kW/m"); newline
height = (power/(22@3*ksec))**0.5
printstring("RMS Wave Height ="); print(height,2,2)
printstring("m"); newline
closestream(6)
!
%end %of %program
```

IV.II. Second Model

Listing of the Second Model Program

```
%begin
!
!
%external %routine %spec IMPDRUNGE(%routine f(%long %real x, %long %real
%array %name y,z), %long %real h,x0, %long %real %array %name y,res,
%integer m,n)
%external %routine %spec IMPDAVDEN(%routine f(%long %real %array %name
x, y), %integer n, %integer %name iflag, %long %real eps, %long %real %array
%name x,y)
%external %routine %spec DEFINE(%integer %name chan, %string %name
char)
%external %routine %spec EMAS3PROMPT(%string %name text)
%external %routine %spec VALVECONS(%long %real %name a,b)
%external %routine %spec SETUP(%long %real %name amp,per,sal,tf,mf, area,
cd,total)
%external %long %real %array %spec dens(0:20,0:20)
%long %real ti
!
!This program is the second attempt to model the
!desalination duck.
!The model includes the motion of the fluid
!piston, or an approximation to it, but
!the duck still moves as a sinusoid.
!
!Details of the model are held elsewhere.
!The ODE solution method used
!is the 4th order Runge-Kutta method
!held in the package IMPDRUNGE.
!
%routine PHYSPROP(%long %real %array %name var)
%external %long %real %fn %spec DENSITY(%long %real s,t)
%external %long %real %fn %spec GAMMA(%long %real t)
```

```
%external %long %real %fn %spec HEATCAP(%long %real s,t)
%external %long %real %fn %spec LATHEAT(%long %real s,t)
%external %long %real %fn %spec TCOND(%long %real s,t)
%external %long %real %fn %spec VOLUCV(%long %real t)
%external %long %real %fn %spec VISC(%long %real s,t)
%external %long %real %fn %spec SATV(%long %real s,vg)
!
%long %real vg1,vg2,tsat1,tsat2
!
vg1 = var(39)/var(8)
vg2 = var(40)/var(9)
tsat1 = SATV(0,vg1)
tsat2 = SATV(var(20),vg2)
!VAR() is a global array that holds all
!the process variables that need
!to be passed from one routine to another.
!PHYSPROP assigns those that are to be
!physical properties.
var(25) = LATHEAT(0,tsat1)
var(26) = LATHEAT(var(20),tsat2)
var(27) = VOLUCV(tsat1)
var(28) = VOLUCV(tsat2)
var(29) = HEATCAP(0,tsat1)
var(30) = HEATCAP(var(20),tsat2)
var(31) = DENSITY(0,tsat1)
var(32) = DENSITY(var(20),tsat2)
var(33) = GAMMA(var(3))
var(34) = GAMMA(var(4))
var(35) = GAMMA(var(5))
var(36) = GAMMA(var(6))
var(37) = VISC(0,tsat1)
var(38) = VISC(var(20),tsat2)
var(43) = TCOND(0,tsat1)
var(44) = TCOND(var(20),tsat2)
!
%return
%end
!
!
%routine RECOVER(%long %real %array %name var)
%external %routine %spec NDNEWTON(%routine f(%long %real %array %name
x,y), %long %real %array %name x,e, %integer n, %integer %name kmax, flag)
%long %real %array x,err(1:17)
%long %real s
%integer kmax,flag,i
!
%routine BAL(%long %real %array %name x,y)
%long %real %array del(1:5)
%long %real cp,u1,u2,a1,a2,lm1,lm2,lm3,s
!
s = var(20)/35
del(1) = x(1)-x(6); del(2) = x(2)-x(5)
del(3) = x(3)-x(4)
del(4) = x(7)-x(5); del(5) = x(8)-x(4)
```

```
!
%if del(1)*del(2)<=0 %then %start
lm1 = (del(1)+del(2))/2
%finish %else %start
lm1 = (del(1)-del(2))/log(mod(del(1)/del(2)))
%finish
!
%if del(2)*del(3)<=0 %then %start
lm2 = (del(2)+del(3))/2
%finish %else %start
lm2 = (del(2)-del(3))/log(mod(del(2)/del(3)))
%finish
!
%if del(4)*del(5)<=0 %then %start
lm3 = (del(4)+del(5))/2
%finish %else %start
lm3 = (del(4)-del(5))/log(mod(del(4)/del(5)))
%finish
!
cp = 4.186@3; u1 = 3000; u2 = 3000; a1 = 0; a2 = 1000
!
y(1) = cp*x(9)*(x(1)-x(2))-x(15)
y(2) = cp*x(10)*(x(6)-x(5))-x(15)
y(3) = u1*a1*lm1-x(15)
y(4) = cp*x(9)*(x(2)-x(3))-x(16)
y(5) = 0.5*u2*a2*lm2-x(16)
y(6) = cp*x(11)*(x(7)-x(8))-x(17)
y(7) = 0.5*u2*a2*lm3-x(17)
y(8) = cp*x(10)*(x(5)-x(4))-x(17)-x(16)
y(9) = x(12)-lm1
y(10) = x(13)-lm2
y(11) = x(14)-lm3
y(12) = var(45)-x(9)
y(13) = s*var(45)/(s-1)-x(10)
y(14) = x(11)+var(45)-x(10)
y(15) = var(11)-x(1)
y(16) = 20-x(4)
y(17) = var(12)-x(7)
!
%return
%end
s = var(20)/35
!
x(1) = var(11); x(2) = 100; x(3) = 24; x(4) = 20
x(5) = 95; x(6) = var(21); x(7) = var(12)
x(8) = var(47)
x(9) = var(45); x(10) = s*x(9)/(s-1)
x(11) = x(10)-x(9)
x(12) = 5; x(13) = 5; x(14) = 5
x(15) = 150@3; x(16) = 6@6; x(17) = 6@6
!
%for i = 1,1,14 %cycle
err(i) = 0.01
%repeat
```

```
err(3) = 0.001
!
err(15) = 15; err(16) = 6@2; err(17) = 6@2
!
kmax = 200
flag = 0
!
NDNEWTON(BAL,x,err,17,kmax,flag)
!
%if flag=0 %then %start
selectoutput(0)
printstring("NDNEWTON fails...unconverged after ")
print(kmax,3,1)
printstring(" iterations."); newline
%stop
%finish
!
%if flag=2 %then %start
selectoutput(0)
printstring("NDNEWTON fails...indeterminate problem.")
newline
%stop
%finish
!
var(21) = x(6)
var(46) = x(3)
var(47) = x(8)
var(24) = x(10)
!
%return
%end
!
!
!
!
!
!
%routine ALGEBRAIC(%long %real %array %name var)
%external %long %real %fn %spec FILMTHICK(%integer i, %long %real m,
tsat,tw,s,d)
!According to the mathematical model the liquid
!and vapour inside the duck must attain
!phase equilibrium instantaneously at each
!time step.
!The equations describing this process are
!algebraic and are solved in this routine
!using a DAVIDENKO package.
!A few other algebraics are also included.
!
%routine EQUIL(%long %real %array %name x,y)
%external %long %real %fn %spec SATV(%long %real s,v)
%long %real vg1,vg2,ts1,ts2,m1,m2
!
vg1 = var(39)/var(8); vg2 = var(40)/var(9)
```

```
ts1 = SATV(0,vg1); ts2 = SATV(var(20),vg2)
m1 = var(8)+var(13); m2 = var(9)+var(14)
!
!The equations below describe equilibration at constant volume
!mass and enthalpy
y(1) = var(27)*var(8)*(var(4)-ts1)+
       var(29)*var(13)*(var(11)-ts1)
y(1) = ((y(1)+var(29)*m1*(ts1-x(2)))/var(25))
       +var(8)-x(1)
!
y(2) = SATV(0,var(39)/x(1))-x(2)
!
y(3) = var(28)*var(9)*(var(5)-ts2)+
       var(30)*var(14)*(var(12)-ts2)
y(3) = ((y(3)+var(30)*m2*(ts2-x(4)))/var(26))
       +var(9)-x(3)
!
y(4) = SATV(var(20),var(40)/x(3))-x(4)
!
!
%end
!
!
!
%long %real %array x(1:12),res(1:12)
%long %real tw,alpha
%integer i,j
!
!
x(1) = var(8); x(2) = var(4)
x(3) = var(9); x(4) = var(5)
!The above equations are solved in IMPDAVDEN.
IMPDAVDEN(EQUIL,4,i,0.05,x,res)
!Error trapping....
%if i=1 %then %start
selectoutput(0)
printstring("DAVDEN fails"); newline
%for j = 1,1,12 %cycle
print(res(j),1,3)
%repeat
newline
closestream(0)
%finish
!Reset....
var(4) = x(2); var(11) = x(2); var(8) = x(1)
var(5) = x(4); var(12) = x(4); var(9) = x(3)
!
tw = 0.5*(var(11)+var(12))
!Liquid film thicknesses are
!calculated from correlations.
!This may be inaccurate due to the
!steady-state assumptions made
!in the derivation of such correlations.
var(13) = FILMTHICK(1,0,var(4),tw,0,2.5)*var(22)*var(31)
```

```
var(14) = FILMTHICK(0,0.5,var(5),tw,var(20),2.5)
        *var(22)*var(32)
!
%return
%end
!
!The routine which follows, MATHMOD,
!contains the differential
!equations at the core of the model.
!These are described in detail
!elsewhere.
!Note that the many deriv's set to 0 are time independent variables,
!eg. physprops that must however be included due to the structure of
!the Runge-Kutta package.
!
%routine MATHMOD(%long %real ti, %long %real %array %name var,deriv)
%external %routine %spec VOLUMES(%long %real ti,a,t,theta,dthetadt, %long
%real %name va,vd,dvadt,dvddt)
%external %long %real %fn %spec ORIFICE(%long %real ph,pl,t)
%external %long %real %fn %spec VALVECHAR(%long %real ph,pl,a,b)
%external %long %real %fn %spec SATV(%long %real s,vg)
%external %long %real %fn %spec UVAL(%long %real hc,he,k,x,ff1,ff2)
%external %long %real %fn %spec HTCOND(%integer i, %long %real tsat,tw,
l,d,n,h)
%external %long %real %fn %spec HTEVAP(%integer i, %long %real s,tsat,
tw,l,d,m,n,h)
!
%long %real %array vf(1:4)
%long %real r,k
%long %real va,vb,vc,vd,dvadt,dvddt,vgb,vgc
%long %real pa,pb,pc,pd,cdao
%long %real tsatb,tsatc,twb,twc,xb,xc,xw
%long %real dnadt,dnbdt,dncdt,dnddt,phi,eps,hb,hc,u
%long %real spring,inertia,kf,g
%long %real ta,tb,tc,td
%integer i
!
!Physical constants for the system...
r = 461.889; k = 0.392; xw = 0.5@-3; g = 9.812
kf = 60; spring = 3.352@6; inertia = 1.2906@6
!
ta = var(3)+273.15; tb = var(4)+273.15; tc = var(5)+273.15
td = var(6)+273.15
!
vb = var(39); vc = var(40)
!Calculation of instantaneous 'cylinder' volumes...
VOLUMES(ti,var(18),var(19),var(1),var(2),
        va,vd,dvadt,dvddt)
!Pressure is not a state variable but it
!must be calculated for use in other equations.
!The ideal gas law is assumed throughout.
pa = var(7)*ta*r/va
pb = var(8)*tb*r/vb
pc = var(9)*tc*r/vc
```

```
pd = var(10)*td*r/vd
!
cdao = var(23)
!Calculation of flow through the valves....
vf(1) = cdao*ORIFICE(pa,pb,var(3))*
       VALVECHAR(pa/1@5,pb/1@5,var(41), var(42))
vf(2) = cdao*ORIFICE(pd,pb,var(6))*
       VALVECHAR(pd/1@5,pb/1@5,var(41),var(42))
vf(3) = cdao*ORIFICE(pc,pa,var(5))*
       VALVECHAR(pc/1@5,pa/1@5,var(41),var(42))
vf(4) = cdao*ORIFICE(pc,pd,var(5))*
       VALVECHAR(pc/1@5,pd/1@5,var(41),var(42))
!
vgb = var(39)/var(8); vgc = var(40)/var(9)
tsatb = SATV(0,vgb); tsatc = SATV(var(20),vgc)
!
twb = 0.25*(3*var(11)+var(12))
twc = 0.25*(var(12)+3*var(11))
!Heat transfer coefficients, again only correlations...
hb = HTCOND(1,tsatb,twb,3,2.5,0,0)
hc = HTEVAP(1,var(20),tsatc,twc,3,2.5,0.5,0,0)
hb = hb/1000; hc = hc/1000
u = UVAL(hb,hc,k,xw,0,0)
!A more accurate calculation of wall temperature....
twb = var(11)-u*(var(11)-var(12))/hb
twc = var(12)+u*(var(11)-var(12))/hc
!Phi & eps are the mass fluxes
!condensing and evaporating .
phi = hb*var(22)*(var(4)-twb)/var(25)
eps = hc*var(22)*(twc-var(5))/var(26)
!Thus the overall mass fluxes to/from
!each chamber can be found....
dnadt = vf(3)-vf(1)
dnddt = vf(4)-vf(2)
dnbdt = vf(1)+vf(2)-phi
dncdt = eps-vf(3)-vf(4)
!Now for the actual ODE's....
!*****************************************************
!1 & 2 describe the motion of the fluid piston.
deriv(1) = var(2)
deriv(2) = (kf*(pa-pd)-spring*sin(var(1)))/inertia
!3-6 find dT/dt for each of the chambers
deriv(3) = (var(33)-1)*ta*((-dvadt/va)+(dnadt/var(7)))
deriv(3) = deriv(3)+vf(3)*(var(5)-var(3))/var(7)
!
deriv(4) = (var(34)-1)*tb*(dnbdt/var(8))
deriv(4) = deriv(4)+vf(1)*(var(3)-var(4))/var(8)+
        vf(2)*(var(6)-var(4))/var(8)
!
deriv(5) = (var(35)-1)*tc*(dncdt/var(9))
!
deriv(6) = (var(36)-1)*td*((-dvddt/vd)+(dnddt/var(10)))
deriv(6) = deriv(6)+vf(4)*(var(5)-var(6))/var(10)
!7-10 are self-explanatory
```

```
deriv(7) = dnadt
deriv(8) = dnbdt
deriv(9) = dncdt
deriv(10) = dnddt
!11 & 12 find dT/dt for the liquid films
deriv(11) = (var(25)*phi-k*var(22)*(twb-twc)/xw)
       /(var(29)*var(13))
deriv(12) = (-var(26)*eps-var(30)*var(24)*
       (var(12)-var(21)))
deriv(12) = (deriv(12)+k*var(22)*(twb-twc)/xw)/
       (var(30)*var(14))
!
!13-44 are all time independent variables
!
%for i = 13,1,44 %cycle
deriv(i) = 0
%repeat
!
%return
%end
!
!Now that all the routines have been set up, the real
!program begins.
!
%external %long %real %fn %spec HTCOND(%integer i, %long %real
tsat,tw,l,n,h)
%long %real %array y(1:50),result(1:44,0:1)
%long %real h,t,total,tout,mass1,mass2,tw,hc
%integer m,n,i,j,count,step1,step2,max
tout = 0
!
!
DEFINE(5,"emct12:general.densities")
selectinput(5)
%for i = 0,1,17 %cycle
%for j = 0,1,8 %cycle
read(dens(j,i))
%repeat
%repeat
selectinput(0)
closestream(5)
!SETUP is the 'front-end' data input routine.
SETUP(y(18),y(19),y(20),y(21),y(24),y(22),y(23),total)
!The user supplied variables (areas etc)
!are sent to a file for future use
!by the data processing program PROCESS.
define(9,"cons")
selectoutput(9)
%for i = 18,1,24 %cycle
print(y(i),0,3); space
%repeat
selectoutput(0)
closestream(9)
!VALVECONS gets data from the user
```

```
!and calculates valve characteristics.
VALVECONS(y(41),y(42))
!
h = 0.02
m = 44
n = 1
!Initialisation....
y(1) = 0.00; y(2) = 0.00; y(3) = 102; y(4) = 102
y(5) = 102; y(6) = 102; y(7) = 56.1; y(8) = 26.14
y(9) = 26.14; y(10) = 56.1; y(11) = 102; y(12) = 102
y(13) = 35; y(14) = 700; y(39) = (90-y(22)*0.5@-3)/2; y(40) = y(39)
!
y(15) = 0; y(16) = 0; y(17) = 0
y(46) = 25; y(47) = 23
!
max = int(total/h)
!
ti = 0.00; step1 = 0; step2 = 0; mass1 = 0
!
DEFINE(10,"result")
selectoutput(10)
!Enter the main, time-stepping cycle.
%for count = 1,1,max %cycle
!
PHYSPROP(y)
!
ALGEBRAIC(y)
!
tw = 0.25*(3*y(11)+y(12))
hc = HTCOND(1,y(4),tw,3,2.5,0,0)/1000
mass2 = hc*y(22)*(y(4)-tw)/y(25)
mass1 = mass1+mass2
y(45) = mass1/count
!
%if step2=10 %then %start
RECOVER(y)
step2 = 0
%finish
!
tout = 4.186*(y(11)+0.97*y(12)-1.97*y(21))+tout
!
%if ti<=total-2*y(19) %then step1 = 0
!
!Printout at equilibrium...but only every 5th step.
%if step1=5 %then %start
print(ti,3,3); space
%for i = 1,1,14 %cycle
print(y(i),3,3); space
%repeat
newline
step1 = 0
%finish
!
IMPDRUNGE(MATHMOD,h,ti,y,result,m,1)
```

```
!
%for j = 1,1,44 %cycle
y(j) = result(j,1)
%repeat
!Update....
step1 = step1+1
step2 = step2+1
ti = ti+h
!
%repeat
!
print(101010.10,6,2)
newline
selectoutput(0)
closestream(10)
newlines(3)
printstring("Finished")
!That's all Folks!
%end %of %program
```

Listing of External Function File

```
!All these routines/functions may called as
!externals in any program.
%external %routine %spec emas3(%string %name command,params, integer
%name flag)
%external %routine %spec emas3itos(%integer %name i, %string %name s)
%externallongrealarray dens(0:20,0:20)
!
%external %routine define(%integer %name chan, %string %name file)
%string (8) number
%integer flag
emas3itos(chan,number)
emas3("define","sto".number."","".file,flag)
%return
!
%end
!
%external %long %real %function density(%long %real s,t)
!Liquid density calculation for brines
!and pure water.
%externalroutinespec emas3(%stringname comm,parms %integer flag)
%integer aa,b,c,d,i,j,flag
%long %real x,y,p,q,dx,dy,r
!
!
%if t>180 %then %result=1000
%if t<20 %then %result=1000
%if s<=0 %then %start
r = 1.00386@3-(2.23115@-1*t)-(2.40169@-3*t**2)
%result=r
%finish %else %start
!
!
dx = dens(2,0)-dens(1,0)
dy = dens(0,2)-dens(0,1)
x = (s-dens(1,0))/dx+1
y = (t-dens(0,1))/dy+1
!
aa = intpt(x)
b = intpt(x+1)
c = intpt(y)
d = intpt(y+1)
!
p = (dens(aa,d)-dens(aa,c))*(y-c)+dens(aa,c)
q = (dens(b,d)-dens(b,c))*(y-c)+dens(b,c)
r = (q-p)*(x-aa)+p
%finish
%result = r
%end
!
%external %long %real %function VISC(%long %real s,t)
!Viscosity correlation for brines and
```

```
!pure water.
%long %real aa,a1,a2,bb,b1,b2,c1,c2,n20,n
!
!
aa = 1.37220; a1 = -0.001015; a2 = 0.000005
bb = 0.000813; b1 = 0.006102; b2 = -0.000040
c1 = 0.001550; c2 = 0.0000093
!
n20 = 1.002+c1*s+c2*(s**2)
!
n = ((t-20)/(t+109))*(aa*(1+a1*s+a2*(s**2))+
    bb*(t-20)*(1+b1*s+b2*(s**2)))
n = n20/10**(n)
n = n/1000
!
!
%result = n
%end
!
%external %long %real %function HEATCAP(%long %real s,t)
!Liquid heat capacity for brines and water.
%long %real aa,a1,a2,a3,bb,b1,b2,b3
%long %real cc,c1,c2,c3,dd,d1,d2,d3,ta,cp
!
a1 = 5.328; a2 = -9.76@-2; a3 = 4.04@-4
b1 = -6.913@-3; b2 = 7.351@-4; b3 = -3.15@-6
c1 = 9.6@-6; c2 = -1.927@-6; c3 = 8.23@-9
d1 = 2.5@-9; d2 = 1.666@-9; d3 = -7.125@-12
!
ta = t+273.15
!
aa = a1+a2*s+a3*(s**2)
bb = b1+b2*s+b3*(s**2)
cc = c1+c2*s+c3*(s**2)
dd = d1+d2*s+d3*(s**2)
!
cp = aa+bb*ta+cc*(ta**2)+dd*(ta**3)
!
%result = cp
%end
!
%external %long %real %function TCOND(%long %real s,t)
!Liquid thermal condictivity for brines and water.
%long %real lamc,tc,x,y,g,p,q,ta,k
!
ta = t+273.15
!
tc = 647
lamc = 240
x = s*0.0002
y = s*0.03
g = 343.5+(s*0.37)
!
p = log(lamc+x)
```

```
q = (2.3-(g/ta))*((1-(ta/(tc+Y)))**(1/3))
!
k = exp(p+q)
k = k/1000
%result = k
!
%end
!
%external %long %real %function BPE(%long %real s,t)
!Boiling point elevation of brines.
%long %real a,bb,c,d,e,f,x,ta,b
!
x = s/1000
!
ta = t+273.15
!
a = x*(ta**2)/13832
bb = 0.001373*ta
c = -0.00272*ta*(x**0.5)
d = 17.86*x
e = -0.0152*x*ta*((ta-225.9)/(ta-236))
f = -(2583*x*(1-x))/ta
!
b = a*(1+bb+c+d+e+f)
!
%result = b
%end
!
%external %long %real %function RECOV(%long %real rec,t)
!Calculates Gibb's free energy for recovery.
%long %real %array a(0:20,0:20)
%integer aa,b,c,d,i,j
%long %real x,y,p,q,dx,dy,delg
!
define(7,"general.delg")
selectinput(7)
!
%for i = 0,1,8 %cycle
%for j = 0,1,4 %cycle
read(a(i,j))
%repeat
%repeat
!
closestream(7)
!
dx = a(2,0)-a(1,0)
dy = a(0,2)-a(0,1)
x = (t-a(1,0))/dx+1
y = (rec-a(0,1))/dy+1
!
aa = intpt(x)
b = intpt(x+1)
c = intpt(y)
d = intpt(y+1)
```

```
!
p = (a(aa,d)-a(aa,c))*(y-c)+a(aa,c)
q = (a(b,d)-a(b,c))*(y-c)+a(b,c)
delg = (q-p)*(x-aa)+p
!
%result = delg
%end
!
%external %long %real %function BARCP(%long %real t)
!Heat capacity of vapour at constant pressure.
%long %real a,b,c,d
a = -1.28115; b = 9.76656@-2; c = -9.90726@-4; d = 3.44861@-6
%long %real cp
!
cp = a+b*t+c*t**2+d*t**3
!
%result = cp
%end
!
%external %long %real %function VOLUCV(%long %real t)
!Heat capacity of vapour at constant volume.
%external %long %real %fn %spec barcp(%long %real t)
%long %real r
r = 0.462
%long %real cv,cp
!
cp = barcp(t)
cv = cp-r
!
%result = cv
%end
!
%external %long %real %function GAMMA(%long %real t)
!Adiabatic compression index.
%long %real cp,cv,gamma
!
cp = barcp(t)
cv = volucv(t)
gamma = cp/cv
!
%result = gamma
%end
!
%external %routine SATEQ(%long %real %array %name x,y)
%external %long %real %fn %spec BPE(%long %real s,t)
!
%long %real a,b,c
a = 7.9186968; b = 1636.909; c = 224.92
!
y(1) = (b/(a-0.434294*log(x(3)*750.06)))-c+x(2)-x(1)
y(2) = BPE(x(4),x(1))-x(2)
!
%return
%end
```

```
!
!
!
%external %long %real %function SATTEMP(%long %real s,p)
!Saturation temperature at a given
!pressure and salinity.
%external %long %real %fn %spec BPE(%long %real s,t)
%external %routine %spec IMPDAVDEN(%routine f(%long %real %array %name
x,y), %integer n, %integer %name ifault, %long %real eps, %long %real %array
%name x,y)
%external %routine %spec SATEQ(%long %real %array %name x,y)
!
%long %real %array x(1:4),y(1:4)
%long %real a,b,c,tol
%integer flag
a = 7.9186968; b = 1636.909; c = 224.92; tol = 0.000001
%if p<=0 %then %result=60
!
x(1) = (b/(a-0.434294*log(750.06*p)))-c
%if s<=0 %then %start
%result = x(1)
%finish
x(2) = BPE(s,x(1))
x(3) = p
x(4) = s
!
IMPDAVDEN(SATEQ,2,flag,tol,x,y)
!
%if flag=1 %then printstring("Something wrong in DAVDEN")
!
%result = x(1)
!
%end
!
%external %long %real %function VAPPRESS(%long %real s,t)
!Calculates vapour pressure at a given
!temperature and salinity.
%external %long %real %fn %spec bpe(%long %real s,t)
%long %real a,b,c,p
!
a = 7.9186968; b = 1636.909; c = 224.92
!
%unless s<=0 %then c = c-bpe(s,t)
!
p = 10**(a-b/(c+t))
p = p*1.01325/760.15
!
%result = p
%end
!
%external %long %real %function LATHEAT(%long %real s,t)
!Latent heat of vaporisation of brine/water
!at given temperature and salinity.
%external %long %real %fn %spec vappress(%long %real s,t)
```

```
%external %long %real %fn %spec bpe(%long %real s,t)
%external %long %real %fn %spec density(%long %real s,t)
%long %real a,b,c,r,p,vg,vl,l,ta
!
a = 18.233473; b = 3769.122; c = 224.902; r = 8.314
ta = t+273.15
!
%unless s<=0 %then c = c-bpe(s,t)
!
p = vappress(s,t)
vg = r*ta/(p*1.8@3)
vl = 1/density(s,t)
l = -b*ta*(vl-vg)*p*1@2/((t+c)**2)
!
%result = l
%end
!
%external %long %real %function HTCOND(%integer i, %long %real
tsat,tw,l,d,nf,hf)
!Heat transfer film coefficient for condensation.
%external %long %real %fn %spec latheat(%long %real s,t)
%external %long %real %fn %spec visc(%long %real s,t)
%external %long %real %fn %spec density(%long %real s,t)
%external %long %real %fn %spec tcond(%long %real s,t)
!
%long %real h,k,rho,g,lam,tref,delt,u,re
!
g = 9.812; delt = tsat-tw; tref = tsat-3*delt/4
!
k = tcond(0,tref)
rho = density(0,tref)
lam = latheat(0,tref)*1000
u = visc(0,tref)
%if delt<=0 %then %result=1
!
%if i=1 %then %start
{Nusselt Euation Routine for Vertical Tubes}
!
h = (k**3)*(rho**2)*g*lam/(delt*l*u)
h = h**0.25
h = h*0.943
!
re = 4*delt*h*l/(lam*u)
%if re>2100 %then %start
!
h = re**0.4
h = h*0.0076
h = h*(((k**3)*(rho**2)*g/(u**2))**0.333)
!
%finish
!
%result = h
!
%finish
```

```
%end
! %external %long %real %function HTEVAP(%integer i, %long %real
s,tsat,tw,l,d,m,nf,hf)
!Heat transfer film coefficient for evaporation.
%external %long %real %fn %spec heatcap(%long %real s,t)
%external %long %real %fn %spec tcond(%long %real s,t)
%external %long %real %fn %spec visc(%long %real s,t)
%external %long %real %fn %spec density(%long %real s,t)
!
%long %real Nu,Pr,Re,h,k,u,g,rho,gamma,tref,delt,cp
!
%if i=1 %then gamma = m/d {plates, d=width}
%if i=2 %then gamma = m/3.142*d {tubes, d=diameter}
!
delt = tsat-tw; tref = tsat-3*delt/4; g = 9.812
!
u = visc(s,tref)
k = tcond(s,tref)
rho = density(s,tref)
cp = heatcap(s,tref)
!
Re = 4*gamma/u; Pr = u*cp/k
!
%unless i>2 %then %start
!
Nu = 0.565/(Pr+5.47)
Nu = 0.110-Nu
Nu = Nu*(Re**0.231)
!
h = Nu*k*((g*(rho**2)/(u**2))**0.333)
!
%result = h
%finish
!
%end
!
%external %long %real %function ORIFICE(%long %real ph,pl,t)
%long %real r,ta,f
!
!This function calculates (mass flow)/CdAo for incompressible
!vapour flow through an orifice.
!
r = 461.89
ta = t+273.15
!
%if ph>=pl %then %start
f=(mod(ph**2-pl**2)/(r*ta))**0.5
%result = f
%finish
!
%if ph<pl %then %start
f=-(mod(pl**2-ph**2)/(r*ta))**0.5
%result=f
%finish
```

```
%end
!
%external %long %real %function UVAL(%long %real hc,he,k,x,ff1,ff2)
!Calculates overall u-value from film
!coefficients, wall resistance and fouling factors.
%long %real u
!
u = (1/hc)+(1/he)+(x/k)+ff1+ff2
u = 1/u
!
%result = u
%end
!
%external %routine VOLUMES(%long %real t,a,ti,theta,dthetadt, long %real
%name va,vd,dvadt,dvddt)
!Calculates instantaneous duck volumes and derivatives.
%longreal arg,omega,epsi,depsdt,dwdt,hold1,hold2
%integer aa,b
!
arg=2*pi/ti
!
omega=a*sin(arg*t); dwdt=a*arg*cos(arg*t)
!
epsi=omega-theta; depsdt=dwdt-dthetadt
!
%if epsi>=0 %then aa=1 %and b=0
%if epsi<0 %then epsi=mod(epsi) %and aa=0 %and b=1
!
hold1=20*(4.5*((pi/2)-epsi)+0.1406*(tan(epsi)-epsi)-2.25)
hold2=20*(4.5*((pi/2)+epsi)-0.1406*(tan(epsi)+epsi)-2.25)
!
va=aa*hold1+b*hold2
vd=b*hold1+aa*hold2
!
hold1=20*depsdt*(0.1406*((1/(cos(epsi)**2))-1)-4.5)
hold2=20*depsdt*(4.5-0.1406*(1+(1/(cos(epsi)**2))))
!
dvadt=-depsdt*20*4.5
dvddt=-dvadt
!
%return
%end
!
%external %long %real %function FILMTHICK(%integer flag, %long %real
m,t1,t2,s,d)
!Calculates liquid film thicknesses.
%external %long %real %fn %spec visc(%long %real s,t)
%external %long %real %fn %spec density(%long %real s,t)
%external %long %real %fn %spec htcond(%integer i, %long %real th,tl,l,d,
nf,hf)
%external %long %real %fn %spec latheat(%long %real s,t)
!
%long %real u,rho,g,delta,gamma,Re,h,lat
!
```

```
g = 9.812
!
%if flag=0 %then %start
u = visc(s,t2); rho = density(s,t2)
gamma = m/d
Re = 4*gamma/u
%if Re<=2000 %then %start
delta = ((3*gamma*u)/(g*(rho**2)))**(1/3)
%result = delta
%finish %else %start
delta = 0.304*((gamma**1.75)*(u**0.25)/(g*(rho**2)))**(1/3)
%result = delta
%finish
%finish
!
lat = latheat(0,t2); rho = density(0,t2); u = visc(0,t2)
%if t2>=t1 %then %result=0.0005@-3
h = htcond(1,t1,t2,2,d,0,0)/1000
gamma = h*2*mod((t1-t2))/lat
delta = ((3*gamma*u)/(g*(rho**2)))**(1/3)
%result = delta
%end
!
%externallongrealfunction VALVECHAR(%longreal ph,pl,a,b)
!Calculates %opening of n-r valves for given Δp.
%longreal open,delp
!
delp=ph-pl
!
open=0.5*(1+(2/pi)*(arctan(1,(a*delp+b))))
!
%result=open
%end
!
%externalroutine VALVECONS(%longrealname a,b)
!Sets parameters for valvechar.
%externalroutinespec emas3prompt(%stringname text)
%longreal xf,yf,hold1,hold2
newline;printstring("Parameters determining the N/R valve behaviour-")
newline
!
emas3prompt("First give the pressure drop across the valve.....")
read(xf)
emas3prompt("And now the % valve opening at that dP.....")
read(yf)
!
hold1=(pi/2)*(0.02*yf-1)
hold2=(pi/2)*(2.222@-4*yf-1)
!
a=0.5*(tan(hold1)-tan(hold2))
a=a/xf
!
b=0.5*(tan(hold1)+tan(hold2))
!
```

```
%return
%end
!
%external %long %real %function SATV(%long %real s,vg)
%externallongrealfnspec BPE(%longreal s,t)
%longreal a,b,tsat
!
a=log(vg)
tsat=114.74-26.991*a
!
%if s=0 %then %result=tsat
!
b=bpe(s,tsat)
tsat=tsat+b
!
%result=tsat
!
%end
!
%external %routine NDNEWTON(%routine evaluate(%longrealarrayname
x,y),%long %real %array %name x,e, %integer n, %integer %name kmax,flag)
!Multi-Dimensional Newton-Raphson algorithm for
!the solution of algebraic equations.
%long %real det
%integer i,j,k,m,h
%long %real %array b,dx,f(1:n),a(1:n,1:n)
%external %routine %spec solve ln eq(%long %real %array %name a,f,
%integer n, %long %real %name det)
%for i = 1,1,n %cycle
dx(i) = 5*e(i)
%repeat
%for k = 1,1,kmax %cycle
flag = 1
evaluate(x,f)
!
%for m = 1,1,n %cycle
b(m) = -f(m)
%repeat
!
%for j = 1,1,n %cycle
x(j) = x(j)+dx(j)
evaluate(x,f)
%for i = 1,1,n %cycle
a(i,j) = (f(i)+b(i))/dx(j)
%repeat
!
x(j) = x(j)-dx(j)
%repeat
!
solve ln eq(a,b,n,det)
!
%if mod(det)<10@-8 %then flag = 2 %and %return
!
%for h = 1,1,n %cycle
```

```
x(h) = x(h)+b(h)
%if mod(b(h))>e(h) %then flag = 0
%repeat
!
%if flag=1 %then kmax = k %and %return
%repeat
%return
%end
%end %of %file
```

Listing of Typical Data Processing File

```
%begin
!This is one of a number of data
!processing programs used to
!manipulate the data produced by
!the modelling program into a
!more intelligible format
%external %routine %spec EMAS3(%string %name comms,parms, %integer
%name flag)
%external %routine %spec EMAS3ITOS(%integer %name k, %string %name s)
%external %routine %spec VOLUMES(%long %real ti,a,t,theta,dthetadt, %long
%real %name va,vd,dvadt,dvddt)
%external %routine %spec DEFINE(%integer %name chan, %string %name text)
%external %long %real %fn %spec sattemp(%long %real s,p)
%external %long %real %fn %spec LATHEAT(%long %real s,t)
%external %long %real %fn %spec HTCOND(%integer i, %long %real
th,tl,l,d,nf,hf)
%external %routine %spec READLINE(%string %name text)
%external %routine %spec EMAS3PROMPT(%string %name text)
%external %long %real %array %spec dens(0:20,0:20)
!
%string (255) file,no
%integer i,j,flag,k
%long %real r,va,vb,vc,vd,dvadt,dvddt,pa,pb,pc,pd
%long %real twb,twc,tsatb,hco,latb,mb,mass,dth,th,avep,power
%long %real vmin,vmax,ratio,ideal,still
%long %real %array cons(1:8),y(1:15)
!
cons(1) = 0.5; cons(2) = 8; cons(5) = 5000
file = "t#out"
!Define input and output streams:
define(5,"emct12:general.densities")
define(2,"emct12:mk3.result")
define(3,file)
define(60,"prod")
!
selectinput(5)
%for i = 0,1,17 %cycle
%for j = 0,1,8 %cycle
read(dens(j,i))
%repeat
%repeat
!
vmin = 0; vmax = 0
r = 461.889; flag = 0
!Call to VOLUMES to calculate
!the swept volume if the fluid piston
!were to remain still:
VOLUMES(0,cons(1),cons(2),0,0,still,still,dvadt,dvddt)
!
selectoutput(3)
selectinput(2)
```

```
closestream(5)
avep = 0; mass = 0
!
!The following cycles read in the data from result:
%for i = 1,1,6000 %cycle
!
%for j = 1,1,15 %cycle
read(y(j))
!The data file always ends with 101010.10;
%if y(j)=101010.10 %then flag = 1 %and %exit
%repeat
!
%if flag=1 %then %exit
!
!Instantaneous "cylinder" volumes:
!
VOLUMES(y(1),cons(1),cons(2),0,0,ideal,vd,dvadt,dvddt)
VOLUMES(y(1),cons(1),cons(2),y(2),y(3),va,vd,dvadt,dvddt)
th = cons(1)*sin(2*pi*y(1)/cons(2))
dth = (cons(1)*2*pi/cons(2))*cos(2*pi*y(1)/cons(2))
vb = 45; vc = 45
!
!Pressures calculates from the ideal gas law:
!
pa = y(8)*r*(y(4)+273)/va
pa = pa/1@5
pb = y(9)*r*(y(5)+273)/vb
pb = pb/1@5
pc = y(10)*r*(y(6)+273)/vc
pc = pc/1@5
pd = y(11)*r*(y(7)+273)/vd
pd = pd/1@5
!
!Calculation of the volume ratio:
vmax = (ideal-still)**2+vmax
vmin = (va-still)**2+vmin
!
ratio = sqrt(vmin/vmax)
!
twb = y(12)-0.25*(y(12)-y(13))
twc = y(13)+0.25*(y(12)-y(13))
hco = HTCOND(1,y(5),twb,3,2.5,0,0)/1000
tsatb = SATTEMP(0,pb)
latb = LATHEAT(0,tsatb)
!
!
!Calculation of condensate flowrate and power:
mb = hco*cons(5)*(y(5)-twb)/latb
power = -(pa-pd)*dvadt*100
!
avep = avep+power
mass = mass+mb
!
!Output to T#OUT:
```

```
!
print(y(1),2,2); print(pa,3,3); print(pb,3,3); print(pc,3,3)
print(pa-pd,3,3); print(va,3,3)
newline
!
!
%repeat
mass = mass/i
avep = avep/i
selectinput(0)
selectoutput(0)
closestream(3)
closestream(2)
selectoutput(60)
!Output to PROD:
printstring("The average mass flowrate of condensate is ")
print(mass,2,2)
printstring(" kg/s.")
!
newline
printstring("Mean Power input ="); print(avep,3,3); printstring("kW")
newline
printstring("Mean SEC ="); print(avep/mass,3,3); printstring %c
("kJ/kg")
newline
printstring("Ratio of swept volume to ideal volume = "); print(ratio,3,3)
selectoutput(0)
closestream(60)
%end %of %program
```