ภาคผนวก ง

โปรแกรมที่ใช้ในการคำนวณ

โปรแกรมที่ใช้ในการคำนวณ (ปรับปรุงจากโปรแกรมของ Simple ของ Urieli) ในส่วนนี้แสดง Source Code ของโปรแกรม MATLAB ที่ใช้ในการคำนวณ

ง1 โปรแกรมคำนวณกำลังบ่งชี้เครื่องยนต์ GPU-3

โปรแกรมเพื่อคำนวณหากำลังและความร้อนที่ให้กับเครื่องยนต์ ปรับปรุงจากโปรแกรม simple ของ Urieli ประกอบด้วยไฟล์

spec	ข้อมูลจำเพาะของเครื่องยนต์
cycle.m	ไฟล์ที่กำหนด อุณหภูมิและความเร็วรอบ เรียกการทำงานของโปรแกรมหลัก
sea.m	โปรแกรมหลัก กำหนดตัวแปร และเรียกโปรแกรมการคำนวณ
define.m	อ่านข้อมูลจากไฟล์ข้อมูลจำเพาะ กำหนดค่าจากข้อมูลจำเพาะให้กับตัวแปร คำนวณหาอุณหภูมิ
engine.m	กำหนดข้อมูลรูปแบบของเครื่องยนต์
`heatex.m`	กำหนดตัวแปรขนาดของ คูลเลอร์
`regen`	ตัวแปรขนาดของ ฮีทเตอร์ คูลเลอร์ และรีเจนเนอเรเตอร์
`gas.m`	กำหนดสารทำงานที่ใช้ และคำนวณหาค่าสมบัติของก๊าซ เช่น ความจุความร้อนจำเพาะ ความหนืดจลน์
operat.m	คำนวณมวลสารทำงานและทำการวิเคราะห์แบบ Schmidt
simple.m	วิเคราะห์ด้วยแบบจำลอง simple ซึ่งรวมผลของการถ่ายเทความร้อนและความดันลดในฮีทเตอร์ รีเจนเนอเรเตอร์ และคูลเลอร์
adiab.m	วิเคราะห์ด้วยแบบจำลองอะเดียบาติกและแสดงผลการคำนวณ
dadiab.m	กำหนดสมการอนุพันธ์ที่จะหาคำตอบด้วยวิธีรุงเงคุททาอันดับสี่
volume.m	คำนวณหาการเปลี่ยนแปลงปริมาตรในแต่ละมุมหมุนของเพลา
rk4.m	คำนวณด้วยวิธีรุงเงคุททาอันดับสี่
regsim.m	คำนวณผลและประสิทธิภาพของรีเจนเนอเรเตอร์
`worksim.m`	คำนวณผลของความดันลดและงานที่สูญเสีย

reynum.m	คำนวณเรย์โนลด์นัมเบอร์ในฮีทเตอร์และคูลเลอร์
reynum_t	คำนวณเรย์โนลด์นัมเบอร์ในรีเจนเนอเรเตอร์

spec

```
s
8.724e-006
7.757e-005
1.349e-005
8.117e-005
120.0
p
1.016e-003
0.066
152
t
2.082e-003
2.032e-003
2.032e-003
80
m
0.690
4.000e-005
p
3.020e-003
0.2453
96
he
10200000.0
349.0
933.0
50.0
```

---

cycle.m

```
clc;
clear all;
global res
global temp_h temp_k
global freq_h

temp=[965 968 969];
tempk=[337 347 353];
rpm=[41.66 50 58.33];

for(i = 1:1:3)

    temp_h=temp(i);
    temp_k=tempk(i);
    freq_h=rpm(i);
    sea(temp_h,temp_k,freq_h);
end
```

---

sea.m

```
% sea (stirling engine analysis) - main program
%Israel Urieli 7/20/02

 function sea(temp_h,temp_q,freq_h)
% Row indices of the var, dvar arrays:
 TC = 1;  % Compression space temperature (K)
 TE = 2;  % Expansion space temperature (K)
 QK = 3;  % Heat transferred to the cooler (J)
 QR = 4;  % Heat transferred to the regenerator (J)
 QH = 5;  % Heat transferred to the heater (J)
 WC = 6;  % Work done by the compression space (J)
 WE = 7;  % Work done by the expansion space (J)
 W  = 8;  % Total work done (WC + WE) (J)
 P  = 9;  % Pressure (Pa)
 VC = 10; % Compression space volume (m^3)
 VE = 11; % Expansion space volume (m^3)
 MC = 12; % Mass of gas in the compression space (kg)
 MK = 13; % Mass of gas in the cooler (kg)
 MR = 14; % Mass of gas in the regenerator (kg)
 MH = 15; % Mass of gas in the heater (kg)
 ME = 16; % Mass of gas in the expansion space (kg)
 TCK = 17; %Conditional temperature compression space / cooler (K)
 THE = 18; %Conditional temeprature heater / expansion space (K)
 GACK = 19;%Conditional mass flow compression space/cooler (kg/rad)
 GAKR = 20; % Conditional mass flow cooler / regenerator (kg/rad)
 GARH = 21; % Conditional mass flow regenerator / heater (kg/rad)
 GAHE = 22; % Conditional mass flow heater / expansion space (kg/rad)
 ROWV = 22; % number of rows in the var matrix
 ROWD = 16; % number of rows in the dvar matrix
 COL = 37; % number of columns in the matrices (every 10 degrees)
%=====================================================================
global tk tr th % cooler, regenerator, heater temperatures [K]
global vk % cooler void volume [m^3]
global vr % regen void volume [m^3]
global vh % heater void volume [m^3]
global res
global temp_h temp_k
global freq_h
define(temp_h,temp_k,freq_h);

res = fopen(filedata,'a');
[var,dvar] = simple(res);
fprintf('quitting simulation...\n');
status = fclose(res);
```

---

engine.m

```
function engine
% Define engine configuration and drive geometric parameters.
% Israel Urieli 4/14/02

global engine_type % s)inusoidal, y)oke (both alpha engines)
global new fid % new data file

engine_type = 'u';
```

```
while(strncmp(engine_type,'u',1))
      if(strncmp(new,'y',1))
            fprintf('Available engine types are:\n');
            fprintf('   s)inusoidal drive\n');
            fprintf('   y)oke drive (Ross)\n');
            engine_type = input('enter engine type ','s');
            fprintf(fid, '%c\n', engine_type(1));
      else
            engine_type = fscanf(fid, '%c',1);
      end
      if(strncmp(engine_type,'s',1))
            sindrive;
      else
            fprintf('engine type is undefined\n')
            engine_type = 'u';
   end
end
%===============================================================
function sindrive
% Sinusoidal drive engine configuration
% Israel Urieli 4/14/02

global vclc vcle % compression,expansion clearence vols [m^3]
global vswc vswe % compression, expansion swept volumes [m^3]
global alpha % phase angle advance of expansion space [radians]
global new fid % new data file

fprintf('sinusoidal drive engine configuration\n')
if(strncmp(new,'y',1))
     vclc = input('enter compression space clearence volume[m^3]: ');
     vswc = input('enter compression space swept volume [m^3]: ');
     vcle = input('enter expansion space clearence volume [m^3]: ');
     vswe = input('enter expansion space swept volume [m^3]: ');
     phase =input('enter expansion phase angle advance [degrees]: ');
      fprintf(fid, '%.3e\n', vclc);
      fprintf(fid, '%.3e\n', vswc);
      fprintf(fid, '%.3e\n', vcle);
      fprintf(fid, '%.3e\n', vswe);
      fprintf(fid, '%.1f\n', phase);
else
      vclc = fscanf(fid,'%e',1);
      vswc = fscanf(fid,'%e',1);
      vcle = fscanf(fid,'%e',1);
      vswe = fscanf(fid,'%e',1);
      phase = fscanf(fid, '%f',1);
end
fprintf('\nsinusoidal drive engine data summary:\n');
fprintf('comp clearence,swept vols%.1f,%.1f[cm^3]\n',
vclc*1e6,vswc*1e6);
fprintf(' exp clearence,swept vols %.1f,%.1f[cm^3]\n',
vcle*1e6,vswe*1e6);
fprintf(' expansion phase angle advance %.1f[degrees]\n', phase);
alpha = phase * pi/180;
%===============================================================
```

---

heatex.m

```
function heatex
% Specify heat exchanger geometric parameters
% Jeff Guess 1/26/03
cooler;
regen;
heater;

%==========================================================
function cooler
% Specify cooler geometric parameters
% Jeff Guess 1/26/03
global vk % cooler void volume [m^3]
global ak % cooler internal free flow area [m^2]
global awgk % cooler internal wetted area [m^2]
global dk % cooler hydraulic diameter [m]
global lk % cooler effective length [m]
global new fid % new data file

cooler_type = 'u';
while(strncmp(cooler_type,'u',1))
      if(strncmp(new,'y',1))
            fprintf('Available cooler types are:\n')
            fprintf('   p, for smooth pipes\n')
            fprintf('   a, for smooth annulus\n')
            fprintf('   s, for slots\n')
            cooler_type = input('enter cooler type ','s');
            fprintf(fid, '%c\n', cooler_type(1));
      else
            fscanf(fid, %c',1);
            cooler_type = fscanf(fid, '%c',1);
      end
      if(strncmp(cooler_type,'p',1))
            [vk,ak,awgk,dk,lk] = pipes;
      elseif(strncmp(cooler_type,'a',1))
            [vk,ak,awgk,dk,lk] = annulus;
      elseif(strncmp(cooler_type,'s',1))
            [vk,ak,awgk,dk,lk] = slots;
      else
            fprintf('cooler type is undefined\n')
            cooler_type = 'u';
      end
end

fprintf('cooler data summary:\n');
fprintf(' void volume(cc) %.4f\n', vk*1e6)
fprintf(' free flow area (cm^2) %.4f\n', ak*1e2)
fprintf(' wetted area (cm^2) %.4f\n', awgk*1e2)
fprintf(' hydraulic diameter(mm) %.4f\n', dk*1e3)
fprintf(' cooler length (cm) %.4f\n', lk*1e2)

%==========================================================
```

---

heater.m

```
function heater
% Specify heater geometric parameters
% Israel Urieli 4/15/02
```

```
global vh % heater void volume [m^3]
global ah % heater internal free flow area [m^2]
global awgh % heater internal wetted area [m^2]
global dh % heater hydraulic diameter [m]
global lh % heater effective length [m]
global new fid % new data file

heater_type = 'u';
while(strncmp(heater_type,'u',1))
      if(strncmp(new,'y',1))
            fprintf('Available heater types are:\n')
            fprintf('   p, for smooth pipes\n')
            fprintf('   a, for smooth annulus\n')
            fprintf('   s, for slots\n')
            heater_type = input('enter heater type ','s');
            fprintf(fid, '%c\n', heater_type(1));
      else
            fscanf(fid, '%c',1); % bypass the previous newline
character
            heater_type = fscanf(fid, '%c',1);
      end
    if(strncmp(heater_type,'p',1))
            [vh,ah,awgh,dh,lh] = pipes;
          vh=80.84e-6; % for GPU3
      elseif(strncmp(heater_type,'a',1))
            [vh,ah,awgh,dh,lh] = annulus;
      elseif(strncmp(heater_type,'s',1))
            [vh,ah,awgh,dh,lh] = slots;
      else
            fprintf('heater type is undefined\n')
      heater_type = 'u';
      end
end
fprintf('heater data summary:\n');
fprintf(' void volume(cc) %.2f\n', vh*1e6)
fprintf(' free flow area (cm^2) %.2f\n', ah*1e2)
fprintf(' wetted area (cm^2) %.2f\n', awgh*1e2)
fprintf(' hydraulic diameter(mm) %.2f\n', dh*1e3)
fprintf(' heater length (cm) %.2f\n', lh*1e2)

%=========================================================
function [v,a,awg,d,len] = pipes
% homogeneous smooth pipes heat exchanger
% Israel Urieli 4/15/02
global new fid % new data file

fprintf('homogeneous bundle of smooth pipes\n')
if(strncmp(new,'y',1))
      d = input('enter pipe inside diameter [m] : ');
      len = input('enter heat exchanger length [m] : ');
      num = input('enter number of pipes in bundle : ');
      fprintf(fid, '%.3e\n', d);
      fprintf(fid, '%.3f\n', len);
      fprintf(fid, '%d\n', num);
else
      d = fscanf(fid,'%e',1);
      len = fscanf(fid,'%f',1);
      num = fscanf(fid,'%d',1);
end
a = num*pi*d*d/4; %pi/4*d^2
v = a*len;
awg = num*pi*d*len; %wall/gas area
%==========================================================
function [v,a,awg,d,len] = annulus
% annular gap heat exchanger
% Israel Urieli 4/15/02
global new fid % new data file

fprintf(' annular gap heat exchanger\n')
if(strncmp(new,'y',1))
      dout = input('enter annular gap outer diameter [m] : ');
      din = input('enter annular gap inner diameter [m] : ');
      len = input('enter heat exchanger length [m] : ');
      fprintf(fid, '%.3f\n', dout);
      fprintf(fid, '%.3f\n', din);
      fprintf(fid, '%.3f\n', len);
else
      dout = fscanf(fid,'%f',1);
      din = fscanf(fid,'%f',1);
      len = fscanf(fid,'%f',1);
end

a = pi*(dout*dout - din*din)/4;
v = a*len;
awg = pi*(din + dout)*len;
d = dout - din;
%==========================================================
function [v,a,awg,d,len] = slots
% slots heat exchanger
% Israel Urieli 3/31/02
global new fid % new data file

fprintf(' slots heat exchanger\n')
if(strncmp(new,'y',1))
      w = input('enter width of slot [m] : ');
      h = input('enter height of slot [m] : ');
      len = input('enter heat exchanger length [m] : ');
      num = input('enter number of slots : ');
      fprintf(fid, '%.3e\n', w);
      fprintf(fid, '%.3e\n', h);
      fprintf(fid, '%.3f\n', len);
      fprintf(fid, '%d\n', num);
else
      w = fscanf(fid,'%f',1);
      h = fscanf(fid,'%f',1);
      len = fscanf(fid,'%f',1);
      num = fscanf(fid,'%d',1);
end

a = num*w*h;
v = a*len;
awg = num*2*(w + h)*len;
d = 4*v/awg;
%==========================================================
```

regen.m

```
function regen
% Specifies regenerator geometric and thermal properties
% Israel Urieli 04/20/02

global lr % regenerator effective length [m]
global cqwr % regenerator housing thermal conductance [W/K]
global new fid % new data file

regen_type = 'u';
while(strncmp(regen_type,'u',1))
      if(strncmp(new,'y',1))
            fprintf('Available regenerator configurations are:\n')
            fprintf('   t, for tubular regenerator set\n')
            fprintf('   a, for annular regenerator\n')
            regen_type = input('enter regenerator configuration','s');
            fprintf(fid, '%c\n', regen_type(1));
      else
          fscanf(fid, '%c',1); % bypass the previous newline character
            regen_type = fscanf(fid, '%c',1);
      end
      if(strncmp(regen_type,'t',1))
            fprintf('tubular regenerator housing\n')
            if(strncmp(new,'y',1))
      dout = input('enter tube housing external diameter [m] : ');
       din = input('enter tube housing internal diameter [m] : ');
                  lr = input('enter regenerator length [m] : ');
            num = input('enter number of tubes : ');
                  fprintf(fid, '%.3e\n', dout);
                  fprintf(fid, '%.3e\n', din);
                  fprintf(fid, '%.3e\n', lr);
                  fprintf(fid, '%d\n', num);
            else
                  dout = fscanf(fid, '%f',1);
                  din = fscanf(fid, '%f',1);
                  lr = fscanf(fid, '%f',1);
                  num = fscanf(fid, '%d',1);
            end
            dimat = 0;

      elseif(strncmp(regen_type,'a',1))
            fprintf('annular regenerator housing\n')
            if(strncmp(new,'y',1))
            dout = input('enter housing external diameter [m] : ');
            din = input('enter housing internal diameter [m] : ');
            dimat = input('enter matrix internal diameter [m] : ');
                  lr = input('enter regenerator length [m] : ');
                  fprintf(fid, '%.3e\n', dout);
                  fprintf(fid, '%.3e\n', din);
                  fprintf(fid, '%.3e\n', dimat);
                  fprintf(fid, '%.3e\n', lr);
            else
                  dout = fscanf(fid, '%f',1);
                  din = fscanf(fid, '%f',1);
                  dimat = fscanf(fid, '%f',1);
                  lr = fscanf(fid, '%f',1);
            end
            num = 1;
   else
      fprintf('regenerator configuration is undefined\n')
      regen_type = 'u';
   end
end

amat = num*pi*(din*din - dimat*dimat)/4; % regen matrix area
awr = num*pi*(dout*dout - din*din)/4; % regen housing wall area
kwr =19; % thermal conductivity [W/m/K]
cqwr = kwr*awr/lr; % regen wall thermal conductance [W/K]
matrix(amat);

%=================================================================
function matrix(amat)
% Specifies regenerator matrix geometric and thermal properties
% Israel Urieli 03/31/02

global matrix_type % m)esh or f)oil
global new fid % new data file

matrix_type = 'u';
while(strncmp(matrix_type,'u',1))
      if(strncmp(new,'y',1))
            fprintf('Available matrix types are:\n')
            fprintf('   m, for mesh matrix\n')
            fprintf('   f, for foil matrix\n')
            matrix_type = input('enter matrix type ','s');
            fprintf(fid, '%c\n', matrix_type(1));
      else
            fscanf(fid, '%c',1);
            matrix_type = fscanf(fid, '%c',1);
      end
      if(strncmp(matrix_type,'m',1))
            mesh(amat);
      elseif(strncmp(matrix_type,'f',1))
            foil(amat);
      else
            fprintf('matrix configuration is undefined\n')
      matrix_type = 'u';
      end
end
%=================================================================
function mesh(amat)
% Specifies mesh matrix geometric and thermal properties
% Israel Urieli 03/31/02

global vr % regen void volume [m^3]
global ar % regen internal free flow area [m^2]
global awgr % regen internal wetted area [m^2]
global lr % regenerator effective length [m]
global dr % regen hydraulic diameter [m]
global new fid % new data file

fprintf(' stacked wire mesh matrix\n')
if(strncmp(new,'y',1))
      porosity = input('enter matrix porosity : ');
      dwire = input('enter matrix wire diameter [m] : ');
      fprintf(fid, '%.3f\n', porosity);
      fprintf(fid, '%.3e\n', dwire);
else
      porosity = fscanf(fid,'%f',1);
      dwire = fscanf(fid,'%e',1);
end

ar = amat*porosity;
vr = ar*lr;
dr = dwire*porosity/(1 - porosity);
awgr = 4*vr/dr;

fprintf(' matrix porosity: %.3f\n', porosity)
fprintf(' matrix wire diam %.2f(mm)\n', dwire*1e3)
fprintf(' hydraulic diam %.3f(mm)\n', dr*1e3)
fprintf(' total wetted area %.3e(sq.m)\n', awgr)
fprintf(' regenerator length %.1f(mm)\n', lr*1e3)
fprintf(' void volume %.2f(cc)\n', vr*1e6)
%================================================================
function foil(amat)
% Specifies foil matrix geometric and thermal properties
% Israel Urieli 03/31/02

global vr % regen void volume [m^3]
global ar % regen internal free flow area [m^2]
global awgr % regen internal wetted area [m^2]
global lr % regenerator effective length [m]
global dr % regen hydraulic diameter [m]
global new fid % new data file

fprintf(' wrapped foil matrix\n')
if(strncmp(new,'y',1))
      fl = input('enter unrolled length of foil [m] : ');
      th = input('enter foil thickness [m] : ');
      fprintf(fid, '%.3f\n', fl);
      fprintf(fid, '%.3e\n', th);
else
      fl = fscanf(fid,'%f',1);
      th = fscanf(fid,'%e',1);
end

am = th*fl;
ar = amat - am;
vr = ar*lr;
awgr = 2*lr*fl;
dr = 4*vr/awgr;
porosity = ar/amat;

fprintf(' unrolled foil length: %.3f(m)\n', fl)
fprintf(' foil thickness %.3f(mm)\n',th*1e3)
fprintf(' hydraulic diam %.3f(mm)\n', dr*1e3)
fprintf(' total wetted area %f(sq.m)\n', awgr)
fprintf(' void volume %.2f(cc)\n', vr*1e6)
fprintf(' porosity %.3f\n', porosity)

%================================================================
```

gas.m

```
function gas
% specifies the working gas properties (he, h2, air)
% Israel Urieli 4/20/02

global rgas % gas constant [J/kg.K]
global cp % specific heat capacity at constant pressure [J/kg.K]
global cv % specific heat capacity at constant volume [J/kg.K]
global gama % ratio: cp/cv
global mu0 % dynamic viscosity at reference temp t0 [kg.m/s]
global t0 t_suth % reference temperature [K], Sutherland constant [K]
global prandtl % Prandtl number
global new fid % new data file

gas_type = 'un';
while(strncmp(gas_type,'un',2))
      if(strncmp(new,'y',1))
            fprintf('Available gas types are:\n');
            fprintf('   hy)drogen)\n');
            fprintf('   he)lium\n');
            fprintf('   ai)r\n');
            gas_type = input('enter gas type: ','s');
            gas_type = [gas_type(1), gas_type(2)];
            fprintf(fid, '%s\n', gas_type);
      else
            fscanf(fid, '%c',1); % bypass the previous newline
character
            gas_type = fscanf(fid, '%c',2);
      end
    if(strncmp(gas_type,'hy',2))
       fprintf('gas type is hydrogen\n')
       gama = 1.4;
       rgas = 4157.2;
       mu0 = 8.35e-6;
       t_suth = 84.4;
      elseif(strncmp(gas_type,'he',2))
            fprintf('gas type is helium\n')
            gama = 1.67;
            rgas = 2078.6
            mu0 =  18.85e-6 ;
            t_suth = 80.0;
      elseif(strncmp(gas_type,'ai',2))
            fprintf('gas type is air\n')
            gama = 1.4;
            rgas = 287.0;
            mu0 = 17.08e-6;  ; % < Organ 17.08e-6
            t_suth = 112.0;
      else
            fprintf('gas type is undefined\n')
            gas_type = 'un';
      end
end
cv = rgas/(gama - 1);
cp = gama*cv;
t0 = 273;
prandtl = 0.71;
```

operat.m

```
function operat(temp_h,temp_k,freq_h)
% Determine operating parameters and do Schmidt anlysis
% Israel Urieli 4/20/02

global pmean % mean (charge) pressure [Pa]
global tk tr th % cooler, regenerator, heater temperatures [K]
global freq omega % cyce frequency [herz], [rads/s]
global new fid % new data file
global temp_h temp_k freq_h
if(strncmp(new,'y',1))
      pmean = input('enter mean pressure (Pa) : ');
      tk = input('enter cold sink temperature (K) : ');
      th = input('enter hot source temperature (K) : ');
      freq = input('enter operating frequency (herz) : ');
      fprintf(fid, '%.1f\n', pmean);
      fprintf(fid, '%.1f\n', tk);
      fprintf(fid, '%.1f\n', th);
      fprintf(fid, '%.1f\n', freq);
else
      pmean = fscanf(fid,'%f',1);
      tk = fscanf(fid,'%f',1);
      th = fscanf(fid,'%f',1);
      freq = fscanf(fid,'%f',1);
end
 freq = freq_h;
fprintf(' Temperature ***ggggg****************** (K):
%.1f\n',temp_h);
th=temp_h;
tk=temp_k;
tkk= temp_k+20;
tk=419;
tr = (922-tk)/log(922/tk);
omega = 2*pi*freq;
fprintf('operating parameters:\n');
fprintf(' mean pressure (kPa): %.3f\n',pmean*1e-3);
fprintf(' cold sink temperature (K): %.1f\n',tk);
fprintf(' hot source temperature (K): %.1f\n',th);
fprintf(' effective regenerator temperature (K): %.1f\n',tr);
fprintf(' operating frequency (herz): %.1f\n',freq);

Schmidt; % Do Schmidt analysis
%===============================================================
function Schmidt
% Schmidt anlysis
% Israel Urieli 3/31/02

global mgas % total mass of gas in engine [kg]
global pmean % mean (charge) pressure [Pa]
global tk tr th % cooler, regen, heater temperatures [K]
global freq omega % cycle frequency [herz], [rads/s]
global vclc vcle % compression,expansion clearence vols [m^3]
global vswc vswe % compression, expansion swept volumes [m^3]
global alpha % phase angle advance of expansion space [radians]
global vk vr vh % cooler, regenerator, heater volumes [m^3]
global rgas % gas constant [J/kg.K]
trise=10;

% Schmidt analysis
c = (((vswe/th)^2 + (vswc/tk)^2 +
2*(vswe/th)*(vswc/tk)*cos(alpha))^0.5)/2;
s = (vswc/2 + vclc + vk)/tk+ vr/tr + (vswe/2 + vcle + vh)/th;
b = c/s;
sqrtb = (1 - b^2)^0.5;
bf = (1 - 1/sqrtb);
beta = atan(vswe*sin(alpha)/th/(vswe*cos(alpha)/th + vswc/tk));
fprintf(' pressure phase angle beta  %.1f(degrees)\n',beta*180/pi)
% total mass of working gas in engine
 mgas=pmean*s*sqrtb/rgas

format long e
fprintf(' total mass of gas: %.8f(kg)\n',mgas)
fprintf(' total mass of gas:  %.8f(gm)\n',mgas*1e3)

% work output
wc = (pi*vswc*mgas*rgas*sin(beta)*bf/c);
we = (pi*vswe*mgas*rgas*sin(beta - alpha)*bf/c);
w = (wc + we);
power = w*freq;
eff = w/we; % qe = we
% Printout Schmidt analysis results
fprintf('====================  Schmidt analysis  ===============\n')
fprintf(' Work(joules) %.3e,  Power(watts) %.3e\n', w,power);
fprintf(' Qexp(joules) %.3e,  Qcom(joules) %.3e\n', we,wc);
fprintf(' indicated efficiency %.3f\n', eff);
fprintf('========================================================\n')
 Plot Schmidt analysis pv and p-theta diagrams
fprintf('Do you want Schmidt analysis plots\n');
choice = input('y)es or n)o: ','s');
if(strncmp(choice,'y',1))
  plotpv
end
 Plot Alan Organ's particle mass distribution in Natural Coordinates
fprintf('Do you want particle mass distribution plot\n');
choice = input('y)es or n)o: ','s');
if(strncmp(choice,'y',1))
  plotmass
end
```

---

simple.m

```
function [var,dvar] = simple(temp_h)
% simple analysis - including heat transfer and pressure drop effects
% Israel Urieli, 7/22/2002 (modified 12/3/2003 for temp plots)
% Returned values:
%   var(22,37) array of variable values every 10 degrees (0 - 360)
% dvar(16,37) array of derivatives every 10 degrees (0 - 360)
% Row indices of the var, dvar arrays:
 TC = 1;  % Compression space temperature [K]
 TE = 2;  % Expansion space temperature [K]
 QK = 3;  % Heat transferred to the cooler [J]
 QR = 4;  % Heat transferred to the regenerator [J]
 QH = 5;  % Heat transferred to the heater [J]
 WC = 6;  % Work done by the compression space [J]
 WE = 7;  % Work done by the expansion space [J]
 W  = 8;  % Total work done (WC + WE) [J]
 P  = 9;  % Pressure [Pa]
 VC = 10; % Compression space volume [m^3]
 VE = 11; % Expansion space volume [m^3]
 MC = 12; % Mass of gas in the compression space [kg]
 MK = 13; % Mass of gas in the cooler [kg]
 MR = 14; % Mass of gas in the regenerator [kg]
 MH = 15; % Mass of gas in the heater [kg]
 ME = 16; % Mass of gas in the expansion space [kg]
 TCK = 17; % Conditional temperature compression space / cooler [K]
 THE = 18; % Conditional temeprature heater / expansion space [K]
 GACK = 19;% Conditional mass flow compression space / cooler [kg/rad]
 GAKR = 20; % Conditional mass flow cooler / regenerator [kg/rad]
 GARH = 21; % Conditional mass flow regenerator / heater [kg/rad]
 GAHE = 22; % Conditional mass flow heater / expansion space [kg/rad]
 ROWV = 22; % number of rows in the var matrix
 ROWD = 16; % number of rows in the dvar matrix
 COL = 37; % number of columns in the matrices (every 10 degrees)
%======================================================================

global freq % cycle frequency [herz]
global tk tr th % cooler, regenerator, heater temperatures [K]
global cqwr % regenerator housing thermal conductance [W/K]
global new fish res% new data file
global temp_h
global pmean
reavg =0;
%th= temp_h;
twk = tk; % Cooler wall temp - equal to initial cooler gas temp
twh = th; % Heater wall temp - equal to initial heater gas temp
epsilon = 1;% allowable temperature error bound for cyclic convergence
terror = 10*epsilon; % Initial temperature error (to enter loop)

while (terror>epsilon)
   [var,dvar] = adiab;
   tgh = hotsim(var,twh); % new heater gas temperature
   tgk = kolsim(var,twk); % new cooler gas temperature
   terror = abs(th - tgh) + abs(tk - tgk);
   th = tgh;
   tk = tgk;
   tr = (th-tk)/log(th/tk);
end

fprintf(' converged heater and cooler mean temperatures =====\n');
fprintf('heater wall/gas temperatures: Twh = %.1f[K], Th =
%.1f[K]\n',twh,th);
fprintf('cooler wall/gas temperatures: Twk = %.1f[K], Tk =
%.1f[K]\n',twk,tk);
% Print out ideal adiabatic analysis results
eff = var(W,COL)/var(QH,COL);% engine thermal efficency
Qkpower = var(QK,COL)*freq;  % Heat transferred to the cooler (W)
Qrpower = var(QR,COL)*freq;  % Heat transferred to the regenerator (W)
Qhpower = var(QH,COL)*freq;  % Heat transferred to the heater (W)
Wpower = var(W,COL)*freq;    % Total power output (W)
fprintf(' Work per cycle: %d\n',var(W,COL));
fprintf('========== ideal adiabatic analysis results ==========\n');
fprintf(' Heat transferred to the cooler: %.2f[W]\n', Qkpower);
fprintf(' Net heat transferred to the regenerator: %.2f[W]\n',
Qrpower);
fprintf(' Heat transferred to the heater: %.2f[W]\n', Qhpower);
fprintf(' Total power output: %.2f[W]\n', Wpower);
fprintf(' Thermal efficiency: %.1f[%%]\n', eff*100);
fprintf('======================================================\n');

fprintf('============= Regenerator simple analysis ============\n');
[reavg,u,mu,qrloss] = regsim(var);
fprintf(' Regenerator net enthalpy loss: %.1f[W]\n', qrloss*freq);
qwrl = cqwr*(twh - twk)/freq;
fprintf(' Regenerator wall heat leakage: %.1f[W]\n', qwrl*freq);

fprintf('========= pressure drop simple analysis =============\n');
res = fopen('p_drop.csv','a');
[rho,reynold,dwork]=worksim(var,dvar,res);
fprintf(' Pressure drop available work loss: %.1f[W]\n', dwork*freq)
actWpower = Wpower - dwork*freq;
actQhpower = Qhpower + qrloss*freq + qwrl*freq;
acteff = actWpower/actQhpower;
fprintf(' Actual power from simple analysis: %.1f[W]\n', actWpower);
fprintf(' Actual heat power in from simple analysis: %.1f[W]\n',
actQhpower);
fprintf(' Actual efficiency from simple analysis: %.1f[%%]\n',
acteff*100);

format short g;

fprintf(res, '%0.5d,%0.5d,%0.5d,%0.2d,%0.2d,%0.2d,%0.2d,%0.2d,%0.2d
\n', pmean,Wpower,actWpower,actQhpower,th,twh,tk,twk,(dwork*freq));
```

adiab.m

```
function [var,dvar] = adiab
% ideal adiabatic model simulation
% Israel Urieli, 7/6/2002
% Returned values:
%   var(22,37) array of variable values every 10 degrees (0 - 360)
%   dvar(16,37) array of derivatives every 10 degrees (0 - 360)

global tk th % cooler, heater temperatures [K]

%Row indices of the var, dvar matrices, and the y,dy variable vectors:
 TC = 1;  % Compression space temperature (K)
 TE = 2;  % Expansion space temperature (K)
 QK = 3;  % Heat transferred to the cooler (J)
 QR = 4;  % Heat transferred to the regenerator (J)
 QH = 5;  % Heat transferred to the heater (J)
 WC = 6;  % Work done by the compression space (J)
 WE = 7;  % Work done by the expansion space (J)
 W  = 8;  % Total work done (WC + WE) (J)
 P  = 9;  % Pressure (Pa)
 VC = 10; % Compression space volume (m^3)
 VE = 11; % Expansion space volume (m^3)
 MC = 12; % Mass of gas in the compression space (kg)
 MK = 13; % Mass of gas in the cooler (kg)
 MR = 14; % Mass of gas in the regenerator (kg)
 MH = 15; % Mass of gas in the heater (kg)
 ME = 16; % Mass of gas in the expansion space (kg)
 TCK = 17; % Conditional temperature compression space / cooler (K)
 THE = 18; % Conditional temeprature heater / expansion space (K)
 GACK =19; % Conditional mass flow compression space / cooler (kg/rad)
 GAKR = 20; % Conditional mass flow cooler / regenerator (kg/rad)
 GARH = 21; % Conditional mass flow regenerator / heater (kg/rad)
 GAHE = 22; % Conditional mass flow heater / expansion space (kg/rad)
 ROWV = 22; % number of rows in the var matrix
 ROWD = 16; % number of rows in the dvar matrix
 COL = 37; % number of columns in the matrices (every 10 degrees)

fprintf('============Ideal Adiabatic Analysis====================\n')
fprintf('Cooler Tk = %.1f[K], Heater Th = %.1f[K]\n', tk, th);
 epsilon = 3;  % Allowable error in temerature (K)
 max_iteration = 20;  % Maximum number of iterations to convergence
 ninc = 360; % number if integration increments (every degree)
 step = ninc/36; % for saving values in var, dvar matrices
 dtheta = 2.0*pi/ninc; % integration increment (radians)
% Initial conditions:
 y(THE) = th;
 y(TCK) = tk;
 y(TE) = th;
 y(TC) = tk;
 iter = 0;
 terror = 10*epsilon; % Initial error to enter the loop
% Iteration loop to cyclic convergence
 while ((terror >= epsilon)&(iter < max_iteration))
% cyclic initial conditions
    tc0 = y(TC);
    te0 = y(TE);
    theta = 0;
    y(QK) = 0;
    y(QR) = 0;
    y(QH) = 0;
    y(WC) = 0;
    y(WE) = 0;
    y(W) = 0;
    fprintf('iteration %d: Tc=%.1f[K],Te= %.1f[K]\n',iter,y(TC),y(TE))
    for(i = 1:1:ninc)
       [theta,y,dy] = rk4('dadiab',7,theta,dtheta,y);
    end
    terror = abs(tc0 - y(TC)) + abs(te0 - y(TE));
    iter = iter + 1;
  end

 if (iter >= max_iteration)
     fprintf('No convergence within %d iteration\n',max_iteration)
 end

 % Initial var and dvar matrix
 var = zeros(22,37);
 dvar = zeros(16,37);

 % a final cycle, to fill the var, dvar matrices
 theta=0;
 y(QK)=0;
 y(QR)=0;
 y(QH)=0;
 y(WC)=0;
 y(WE)=0;
 y(W)=0;
 [var,dvar] = filmatrix(1,y,dy,var,dvar);
 for(i = 2:1:COL)
     for(j = 1:1:step)
         [theta,y,dy] = rk4('dadiab',7,theta,dtheta,y);
     end
     [var,dvar] = filmatrix(i,y,dy,var,dvar);
 end

var_tran=var';
dvar_tran=dvar';
```

---

dadiab.m

```
function [y,dy] = dadiab(theta,y)
% Evaluate ideal adiabatic model derivatives
% Israel Urieli, 7/6/2002
% Arguments:  theta - current cycle angle [radians]
%             y(22) - vector of current variable values
% Returned values:
%             y(22) - updated vector of current variables
%             dy(16) vector of current derivatives
% Function invoked : volume.m

% global variables used from "define" functions
global vk % cooler void volume [m^3]
global vr % regen void volume [m^3]
global vh % heater void volume [m^3]
global rgas % gas constant [J/kg.K]
global cp % specific heat capacity at constant pressure [J/kg.K]
global cv % specific heat capacity at constant volume [J/kg.K]
global gama % ratio: cp/cv
global mgas % total mass of gas in engine [kg]
global tk tr th % cooler, regen, heater temperatures [K]

% Indices of the y, dy vectors:
 TC = 1;  % Compression space temperature (K)
 TE = 2;  % Expansion space temperature (K)
 QK = 3;  % Heat transferred to the cooler (J)
 QR = 4;  % Heat transferred to the regenerator (J)
 QH = 5;  % Heat transferred to the heater (J)
 WC = 6;  % Work done by the compression space (J)
 WE = 7;  % Work done by the expansion space (J)
 W  = 8;  % Total work done (WC + WE) (J)
 P  = 9;  % Pressure (Pa)
 VC = 10; % Compression space volume (m^3)
 VE = 11; % Expansion space volume (m^3)
 MC = 12; % Mass of gas in the compression space (kg)
 MK = 13; % Mass of gas in the cooler (kg)
 MR = 14; % Mass of gas in the regenerator (kg)
 MH = 15; % Mass of gas in the heater (kg)
 ME = 16; % Mass of gas in the expansion space (kg)
 TCK = 17; % Conditional temperature compression space / cooler (K)
 THE = 18; % Conditional temeprature heater / expansion space (K)
 GACK = 19;% Conditional mass flow compression space / cooler (kg/rad)
 GAKR = 20; % Conditional mass flow cooler / regenerator (kg/rad)
 GARH = 21; % Conditional mass flow regenerator / heater (kg/rad)
 GAHE = 22; % Conditional mass flow heater / expansion space (kg/rad)
%==================================================================
% Volume and volume derivatives:
 [y(VC),y(VE),dy(VC),dy(VE)] = volume(theta);

% Pressure and pressure derivatives:
 vot = vk/tk + vr/tr + vh/th;
 y(P) = (mgas*rgas/(y(VC)/y(TC) + vot + y(VE)/y(TE)));
 top = -y(P)*(dy(VC)/y(TCK) + dy(VE)/y(THE));
 bottom = (y(VC)/(y(TCK)*gama) + vot + y(VE)/(y(THE)*gama));
 dy(P) = top/bottom;

% Mass accumulations and derivatives:
 y(MC) = y(P)*y(VC)/(rgas*y(TC));
 y(MK) = y(P)*vk/(rgas*tk);
 y(MR) = y(P)*vr/(rgas*tr);
 y(MH) = y(P)*vh/(rgas*th);
 y(ME) = y(P)*y(VE)/(rgas*y(TE));
 dy(MC) = (y(P)*dy(VC) + y(VC)*dy(P)/gama)/(rgas*y(TCK));
 dy(ME) = (y(P)*dy(VE) + y(VE)*dy(P)/gama)/(rgas*y(THE));
 dpop = dy(P)/y(P);
 dy(MK) = y(MK)*dpop;
 dy(MR) = y(MR)*dpop;
 dy(MH) = y(MH)*dpop;

% Mass flow between cells:
 y(GACK) = -dy(MC);
 y(GAKR) = y(GACK) - dy(MK);
 y(GAHE) = dy(ME);
 y(GARH) = y(GAHE) + dy(MH);

% Conditional temperatures between cells:
 y(TCK) = tk;
 if(y(GACK)>0)
    y(TCK) = y(TC);
 end
 y(THE) = y(TE);
 if(y(GAHE)>0)
    y(THE) = th;
 end
% 7 derivatives to be integrated by rk4:
% Working space temperatures:
 dy(TC) = y(TC)*(dpop + dy(VC)/y(VC) - dy(MC)/y(MC));
 dy(TE) = y(TE)*(dpop + dy(VE)/y(VE) - dy(ME)/y(ME));

% Energy:
 dy(QK) = vk*dy(P)*cv/rgas - cp*(y(TCK)*y(GACK) - tk*y(GAKR));
 dy(QR) = vr*dy(P)*cv/rgas - cp*(tk*y(GAKR) - th*y(GARH));
 dy(QH) = vh*dy(P)*cv/rgas - cp*(th*y(GARH) - y(THE)*y(GAHE));
 dy(WC) = y(P)*dy(VC);
 dy(WE) = y(P)*dy(VE);
```

```
% Net work done:
 dy(W) = dy(WC) + dy(WE);
 y(W)=y(WC)+y(WE);
```

---

volume.m

```
function [vc,ve,dvc,dve] = volume(theta)
% determine working space volume variations and derivatives
% Israel Urieli, 7/6/2002
% Argument:  theta - current cycle angle [radians]
% Returned values:
%   vc, ve - compression, expansion space volumes [m^3]
%   dvc, dve - compression, expansion space volume derivatives

global engine_type % s)inusoidal, y)oke (both alpha engines)

if(strncmp(engine_type,'s',1))
   [vc,ve,dvc,dve] = e_rhombic(theta); %GPU3

end

%===============================================================
  function [vc,ve,dvc,dve] = e_rhombic(theta)

% Argument:  theta - current cycle angle [radians]
% Returned values:
%   vc, ve - compression, expansion space volumes [m^3]
%   dvc, dve - compression, expansion space volume derivatives
global vcd vhd % compression,expansion clearence vols [m^3]
global vswc vswe % compression, expansion swept volumes [m^3]
global alpha % phase angle advance of expansion space [radians]
global D L e r

D=0.0699;
r=0.01397;
e=0.02065;
L=0.04602;
d_rod=0.00953;

vclc=2.119e-005; %clearance compress volume
vcle=1.249e-005; %clearance expansion volume

b1=sqrt(L^2-(e-r)^2);
b2= sqrt((L-r)^2-e^2);
b4= sqrt((L+r)^2-e^2);
b_theta=sqrt(L^2-(e+r*cos(theta))^2);

ap=(pi/4)*(D^2-d_rod^2);% Area piston
ad=(pi/4)*D^2 ;        % Area displacer

vc_l= 2*ap*(b1-b_theta);
ve_l= ad*(b_theta-b2-r*sin(theta));

dvc=-2*ap*r*sin(theta)*(e+r*cos(theta))/b_theta ;
dve=-(dvc*ad/(2*ap))-(r*cos(theta)*ad) ;

vc=vclc+vc_l;
```

```
ve=vcle+ve_l;
```

---

rk4.m

```
function [x, y, dy] = rk4(deriv,n,x,dx,y)
%Classical fourth order Runge-Kutta method
%Integrates n first order differential equations
%dy(x,y) over interval x to x+dx
%Izzi Urieli - Jan 21, 2002
x0 = x;
y0 = y;
[y,dy1] = feval(deriv,x0,y);
for i = 1:n
      y(i) = y0(i) + 0.5*dx*dy1(i);
end
xm = x0 + 0.5*dx;
[y,dy2] = feval(deriv,xm,y);
for i = 1:n
      y(i) = y0(i) + 0.5*dx*dy2(i);
end
[y,dy3] = feval(deriv,xm,y);
for i = 1:n
      y(i) = y0(i) + dx*dy3(i);
end
x = x0 + dx;
[y,dy] = feval(deriv,x,y);
for i = 1:n
      dy(i) = (dy1(i) + 2*(dy2(i) + dy3(i)) + dy(i))/6;
      y(i) = y0(i) + dx*dy(i);
end
```

---

regsim.m

```
function [reavg,u,mu,qrloss] = regsim(var)
% Evaluate the effectiveness and performance of the regenerator
% Israel Urieli, 7/23/2002
% Arguments:
%   var(22,37) array of variable values every 10 degrees (0 - 360)
% Returned value:
%   qrloss - regenerator net enthalpy loss [J]

% Row indices of the var array:
 TC = 1;  % Compression space temperature [K]
 TE = 2;  % Expansion space temperature [K]
 QK = 3;  % Heat transferred to the cooler [J]
 QR = 4;  % Heat transferred to the regenerator [J]
 QH = 5;  % Heat transferred to the heater [J]
 WC = 6;  % Work done by the compression space [J]
 WE = 7;  % Work done by the expansion space [J]
 W  = 8;  % Total work done (WC + WE) [J]
 P  = 9;  % Pressure [Pa]
 VC = 10; % Compression space volume [m^3]
 VE = 11; % Expansion space volume [m^3]
 MC = 12; % Mass of gas in the compression space [kg]
 MK = 13; % Mass of gas in the cooler [kg]
 MR = 14; % Mass of gas in the regenerator [kg]
 MH = 15; % Mass of gas in the heater [kg]
 ME = 16; % Mass of gas in the expansion space [kg]
 TCK = 17; %Conditional temperature compression space / cooler [K]
 THE = 18; %Conditional temeprature heater / expansion space [K]
 GACK = 19; %Conditional mass flow compression space / cooler [kg/rad]
 GAKR = 20; % Conditional mass flow cooler / regenerator [kg/rad]
 GARH = 21; % Conditional mass flow regenerator / heater [kg/rad]
 GAHE = 22; % Conditional mass flow heater / expansion space [kg/rad]

global matrix_type % m)esh or f)oil
global ar % regen internal free flow area [m^2]
global awgr % regen internal wetted area [m^2]
global dr % regen hydraulic diameter [m]
global tr % regen temperature [K]
global omega % cycle frequency [rads/s]
global mu
global kgasr rho cp
global pmean

% Reynolds number over the cycle
for(i = 1:1:37)
    gar(i) = (var(GAKR,i) + var(GARH,i))*omega/2;
    gr = gar(i)/ar;
    [mu,kgasr,re(i)] = reynum_t(tr,gr,dr);
 %mu =mu ;
end

% average and maximum Reynolds number
sumre = 0;
remax = re(1);
for(i = 1:1:36)
    sumre = sumre + re(i);
    if(re(i) > remax)
        remax = re(i);
    end
end

reavg = sumre/36;
u= reavg*(1-0.697)*mu/(0.04006e-3*0.697);

% Stanton number, number of transfer units, regenerator effectiveness
if (strncmp(matrix_type,'m',1))
    [st,fr] = matrixfr(reavg);
elseif (strncmp(matrix_type,'f',1))
    [st,ht,fr] = foilfr(dr,mu,reavg);
end

nu=(1+0.99*(reavg*0.7)^0.66)*0.697^1.79;

rho =1*pmean/(2.0769*tr*1000) ;% kg/m3
ntu = st*awgr/(2*ar);
effect = ntu/(ntu + 1);

% Calculate qrloss
for (i=1:1:37)
    qreg(i) = var(QR,i);
end
qrmin = min(qreg);
qrmax = max(qreg);
```

```
qrloss =(1- effect)*(qrmax - qrmin);
fr_new=(151.2346/re(i))+(3.994/re(i)^0.103

% Regenerator simple analysis results:
fprintf('Average Reynolds number: %.1f\n', reavg);
fprintf('Maximum Reynolds number: %.1f\n', remax);
fprintf('Stanton number(Average Re): %.3f\n',st);
fprintf('Number of transfer units: %.1f\n',ntu);
fprintf('Friction Factor: =%d ,Friction Factor2 : =%d \n',fr,fr_new);
fprintf('Regenerator effectiveness : %.3f\n',effect);
```

---

worksim.m

```
function [rho,reynold,dwork] = worksim(var,dvar,res);
% call by simple line 113
% Evaluate the pressure drop available work loss [J]
% Israel Urieli, 7/23/2002
% Arguments:
%   var(22,37) array of variable values every 10 degrees (0 - 360)
%   dvar(16,37) array of derivatives every 10 degrees (0 - 360)
%   Returned value:
%   dwork - pressure drop available work loss [J]
% Row indices of the var, dvar arrays:
 TC = 1;  % Compression space temperature [K]
 TE = 2;  % Expansion space temperature [K]
 QK = 3;  % Heat transferred to the cooler [J]
 QR = 4;  % Heat transferred to the regenerator [J]
 QH = 5;  % Heat transferred to the heater [J]
 WC = 6;  % Work done by the compression space [J]
 WE = 7;  % Work done by the expansion space [J]
 W  = 8;  % Total work done (WC + WE) [J]
 P  = 9;  % Pressure [Pa]
 VC = 10; % Compression space volume [m^3]
 VE = 11; % Expansion space volume [m^3]
 MC = 12; % Mass of gas in the compression space [kg]
 MK = 13; % Mass of gas in the cooler [kg]
 MR = 14; % Mass of gas in the regenerator [kg]
 MH = 15; % Mass of gas in the heater [kg]
 ME = 16; % Mass of gas in the expansion space [kg]
 TCK = 17; %Conditional temperature compression space / cooler [K]
 THE = 18; %Conditional temeprature heater / expansion space [K]
 GACK = 19; %Conditional mass flow compression space / cooler [kg/rad]
 GAKR = 20; % Conditional mass flow cooler / regenerator [kg/rad]
 GARH = 21; % Conditional mass flow regenerator / heater [kg/rad]
 GAHE = 22; % Conditional mass flow heater / expansion space [kg/rad]
 ROWV = 22; % number of rows in the var matrix
 ROWD = 16; % number of rows in the dvar matrix
 COL = 37; % number of columns in the matrices (every 10 degrees)
%=====================================================================

global tk tr th % cooler, regenerator, heater temperatures [K]
global freq omega % cycle frequency [herz], [rads/s]
global vh % heater void volume [m^3]
global ah % heater internal free flow area [m^2]
global dh % heater hydraulic diameter [m]
global lh % heater effective length [m]
global vk % cooler void volume [m^3]
global ak % cooler internal free flow area [m^2]
global dk % cooler hydraulic diameter [m]
global lk % cooler effective length [m]
global vr % regen void volume [m^3]
global ar % regen internal free flow area [m^2]
global lr % regenerator effective length [m]
global dr % regen hydraulic diameter [m]
global matrix_type % m)esh or f)oil
global pmean
global rho
global res
global rgas
dtheta = 2*pi/36;
dwork = 0; % initialise pumping work loss
reynold =0;
porous=0.697;% porosity
T0=273.15;
fprintf(' Rgas: %.3f[K]\n', rgas);
vi=rgas*tr/(pmean);
rho=1/vi  ;% kg/m3
d_regen=22.6e-3;

Ar2=(pi/4)*d_regen^2 ;
for(i = 1:1:36)
%Pressure drop cooler
    gk = (var(GACK,i) + var(GAKR,i))*omega/(2*ak);
    [mu,kgas,re(i)] = reynum(tk,gk,dk);
    [ht,fr] = pipefr(dk,mu,re(i));
    dpkol(i) = 2*fr*mu*vk*gk*lk/(var(MK,i)*dk^2);

%Pressure drop heater
    gh = (var(GARH,i) + var(GAHE,i))*omega/(2*ah);
    [mu,kgas,re(i)] = reynum(th,gh,dh);
    [ht,fr] = pipefr(dh,mu,re(i));
    dphot(i) = 2*fr*mu*vh*gh*lh./(var(MH,i)*dh^2);

%Pressure drop regen
  gr = (var(GAKR,i) + var(GARH,i))*omega/(2*ar); %originall
  [mu,kgas,re(i)] = reynum_t(tr,(gr/rho),dr);
   reynold = re(i);
   Cf2=(151.2346/re(i))+(3.994/re(i)^0.103 ;

    if(strncmp(matrix_type,'m',1))
         [st,fr] = matrixfr(re(i));
    elseif (strncmp(matrix_type,'f',1))
         [st,ht,fr] = foilfr(dr,mu,re(i));
    end

  a0=8*pi/4*(22.6e-3)^2;
  u=gr/rho;
  n=308;
  L=22.6e-3;
  d=0.04006e-3;

    dpreg(i) =  Cf2*L*(1-0.697)*rho*u^2*(u/abs(u))/(2*0.697*d);
    dp(i) = dpkol(i) + dpreg(i) + dphot(i);
    dwork=dwork+dtheta*dp(i)*dvar(VE,i); % pumping work [J]
      pcom(i) = var(P,i);
      pexp(i) = pcom(i) + dp(i);
end

dpkol(COL) = dpkol(1);
dpreg(COL) = dpreg(1);
dphot(COL) = dphot(1);
dp(COL) = dp(1);
pcom(COL) = pcom(1);
pexp(COL) = pexp(1);

fprintf('quitting pressure plots...\n');
```

---

reynum.m

```
function [mu,kgas,re] = reynum(t,g,d)
% evaluate dynamic viscosity, thermal conductivity, Reynolds number
% Israel Urieli, 7/22/2002
% Arguments:
%   t - gas temperature [K]
%   g - mass flux [kg/m^2.s]
%   d - hydraulic diameter [m]
% Returned values:
%   mu - gas dynamic viscosity [kg.m/s]
%   kgas - gas thermal conductivity [W/m.K]
%   re - Reynolds number

global cp % specific heat capacity at constant pressure [J/kg.K]
global mu0 % dynamic viscosity at reference temp t0 [kg.m/s]
global t0 t_suth % reference temperature [K], Sutherland constant [K]
global prandtl % Prandtl number

mu = mu0*(t0 + t_suth)/(t + t_suth)*(t/t0)^1.5;
kgas = cp*mu/prandtl;
re = abs(g)*d/mu;
%disp(re)
%   fprintf(' display Reynolds number: %.3f\n', re);
if(re < 1)
   re = 1;
end
```

---

reynum_t.m

```
function [mu,kgasr,re] = reynum_t(t,g,d)
% evaluate dynamic viscosity, thermal conductivity, Reynolds number
% Israel Urieli, 7/22/2002
% Arguments:
%   t - gas temperature [K]
%   g - mass flux [kg/m^2.s]
%   d - hydraulic diameter [m]
% Returned values:
%   mu - gas dynamic viscosity [kg.m/s]
%   kgas - gas thermal conductivity [W/m.K]
%   re - Reynolds number

global cp % specific heat capacity at constant pressure [J/kg.K]
global mu0 % dynamic viscosity at reference temp t0 [kg.m/s]
global t0 t_suth % reference temperature [K], Sutherland constant [K]
global prandtl

 mu = mu0*(t0 + t_suth)/(t + t_suth)*(t/t0)^1.5;
 kgasr = cp*mu/prandtl;
 porous=0.697;
 A_reg=8*(pi/4)*(22.6e-3)^2 ;
 u= abs(g)/1;
      s0= 0.1265822e-003 ;
      w0= 0.0865822e-003;

 re=0.04006e-3*porous*u/((1-porous)*mu) ;
 if(re < 1)
   re = 1;
 end
```

ง2 โปรแกรมคำนวณกำลังบ่งชี้โดยเปลี่ยนอัตราส่วน e/L และ r/L

โปรแกรมสำหรับคำนวณกำลังบ่งชี้ เมื่อเปลี่ยนแปลงอัตราส่วน e/L และ r/L ตั้งแต่ 0.1-0.8 และ 0.1-0.5 ปรับปรุงโปรแกรมเดิมจาก ง1 ให้ทำการคำนวณโดยการวนรอบเปลี่ยนค่า e/L และ r/L โดยปรับปรุงไฟล์ cycle.m และ volume.m

cycle.m

```
clc;
clear all;
global res
global temp_h temp_k
global freq_h
global e_l  r_l
temp=[965 773 969];
tempk=[337 330 353];
rpm=[41.66 50 58.33];

for  (j = 0.4:0.05:0.4) % (j = 0.1:0.05:0.8)
  for  (k = 0.5:0.05:0.5)

        e_l =j ;%k
        r_l =k ;%j

       for(i = 1:1:3)

        if ((j+k) <= 1.00)

       temp_h=temp(i);
       temp_k=tempk(i);
       freq_h=rpm(i);
       sea(temp_h,temp_k,freq_h);
         else
       sea(0,0,0);
         end

     end
  end
```

```
end
```

---

volume.m

```
function [vc,ve,dvc,dve] = volume(theta) %call by dadiab line 48
% determine working space volume variations and derivatives
% Israel Urieli, 7/6/2002
% Argument:  theta - current cycle angle [radians]
% Returned values:
%   vc, ve - compression, expansion space volumes [m^3]
%   dvc, dve - compression, expansion space volume derivatives

global engine_type % s)inusoidal, y)oke (both alpha engines)
global e_l
global r_l

if(strncmp(engine_type,'s',1))
   [vc,ve,dvc,dve] = e_rhombic(theta); %GPU3

end
%==============================================================
function [vc,ve,dvc,dve] = sinevol(theta)
% sinusoidal drive volume variations and derivatives
% Israel Urieli, 7/6/2002
% Argument:  theta - current cycle angle [radians]
% Returned values:
%   vc, ve - compression, expansion space volumes [m^3]
%   dvc, dve - compression, expansion space volume derivatives

global vclc vcle % compression,expansion clearence vols [m^3]
global vswc vswe % compression, expansion swept volumes [m^3]
global alpha % phase angle advance of expansion space [radians]

 vc = vclc + 0.5*vswc*(1 + cos(theta));
 ve = vcle + 0.5*vswe*(1 + cos(theta + alpha));
 dvc = -0.5*vswc*sin(theta);
 dve = -0.5*vswe*sin(theta + alpha);
%==============================================================
% GPU
  function [vc,ve,dvc,dve] = e_rhombic(theta,e_l,r_l)
% sinusoidal drive volume variations and derivatives
% Israel Urieli, 7/6/2002
% Argument:  theta - current cycle angle [radians]
% Returned values:
%   vc, ve - compression, expansion space volumes [m^3]
%   dvc, dve - compression, expansion space volume derivatives
global vcd vhd % compression,expansion clearence vols [m^3]
global vswc vswe % compression, expansion swept volumes [m^3]
global alpha % phase angle advance of expansion space [radians]
global D L e r
global e_l r_l

D=0.0699;
L=0.046;
d_rod=0.00953;
e =e_l*L;
r= r_l*L;
```

```
vclc=2.119e-005; %clearance compress volume
vcle=1.249e-005; %clearance expansion volume

b1=sqrt(L^2-(e-r)^2);
b2= sqrt((L-r)^2-e^2);
b4= sqrt((L+r)^2-e^2);
b_theta=sqrt(L^2-(e+r*cos(theta))^2);

ap=(pi/4)*(D^2-d_rod^2);% Area piston
ad=(pi/4)*D^2 ;        % Area displacer

vc_l= 2*ap*(b1-b_theta);
ve_l= ad*(b_theta-b2-r*sin(theta));

dvc=-2*ap*r*sin(theta)*(e+r*cos(theta))/b_theta ;
dve=-(dvc*ad/(2*ap))-(r*cos(theta)*ad) ;

vc=vclc+vc_l;
ve=vcle+ve_l;
```