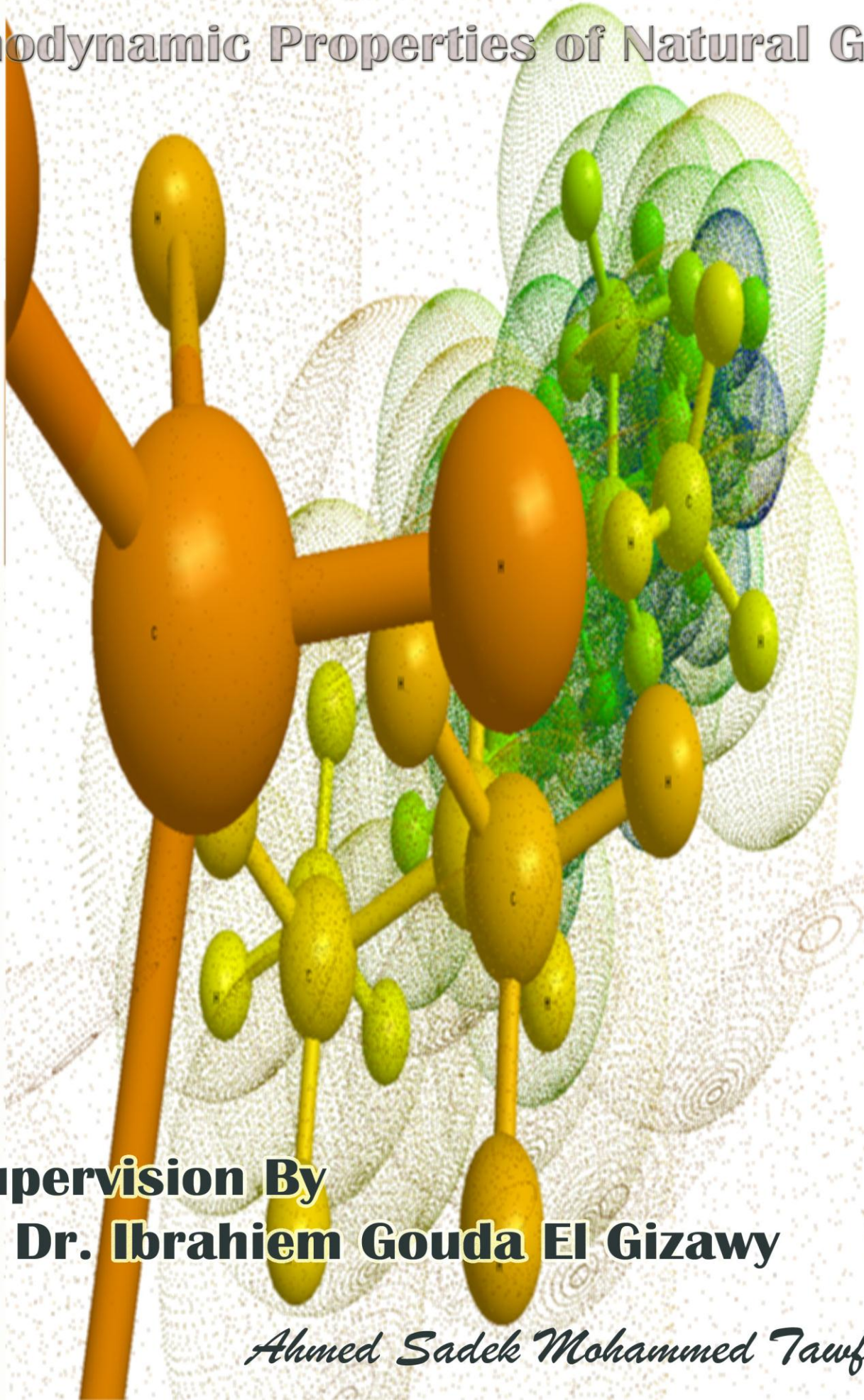
Thermodynamic Properties of Natural Gas

Graduation Project 2006



Supervision By

Dr. Ibrahiem Gouda El Gizawy

Ahmed Sadek Mohammed Tawfeek

بسم الله الرحمن الرحيم

"يَا بُنَيَّ إِنَّهَا إِن تَكُ مِثْقَالَ حَبَّةٍ مِّنْ خَرْدَلٍ فَتَكُنْ فِي صَخْرَةٍ أَوْ فِي السَّمَاوَاتِ أَوْ فِي الأَرْضِ يَأْتِ بِهَا اللهُ إِنَّ اللهَ لَطِيفٌ خَبِيرٌ"

شكر واجب

الحمد لله الذى هدانا لهذا وما كنا لنهتدى لولا أن هدانا الله.

فلقد من الله علينا بنعمة العلم واتمها بإخراج هذا المشروع الذى تحويه أيديكم الآن, تنبعث من بين جوانحه أفراح وأحزان وليال طوال, عمل دؤوب لا نبتغى به غير وجه الله سبحانه تعالى.

وليس من نافلة القول أن نتذكر ونذكر أنفسنا بكل من قام بإبداء رأى أو بإيفاد ملحوظة أو بتقديم النصيحة الغالية التى بها خرج هذا العمل الى النور وأضحى جليا واضحا.

إلى أبى ...

إلى أمى ...

إلى زملائى ...

إلى أساتذتى ...

وإلى كل من أمسك مشعلا يضئ به درب العلم للسالكين من الرائحين فيه والغاديين.

إلى راعى هذا العمل الذى لم يبخل علي بعلمه واتسع صدره لشطحاتى و افكارى, والذى لم يتوان عن المساعدة والنصح ليجعل منى, ومن كل زملائى شيئا قد يكون ذا نفع لهم ولمن بعدهم, شكراً لاشبهة فيه, لا رياء ولا نفاقا, ولا تقربا ولا ابتعادا, جزاه الله عنا كل الخير

إلى الدكتور إبراهيم الجيزاوى

Thermodynamic Properties of Natural Gas

Supervision By
Dr. Ibrahiem Gouda El Gizawy

Ahmed Sadek Mohammed Tawfeek

Mataria Faculty of Engineering
Mechanical Power Department
Helwan University

Graduation Project
2006

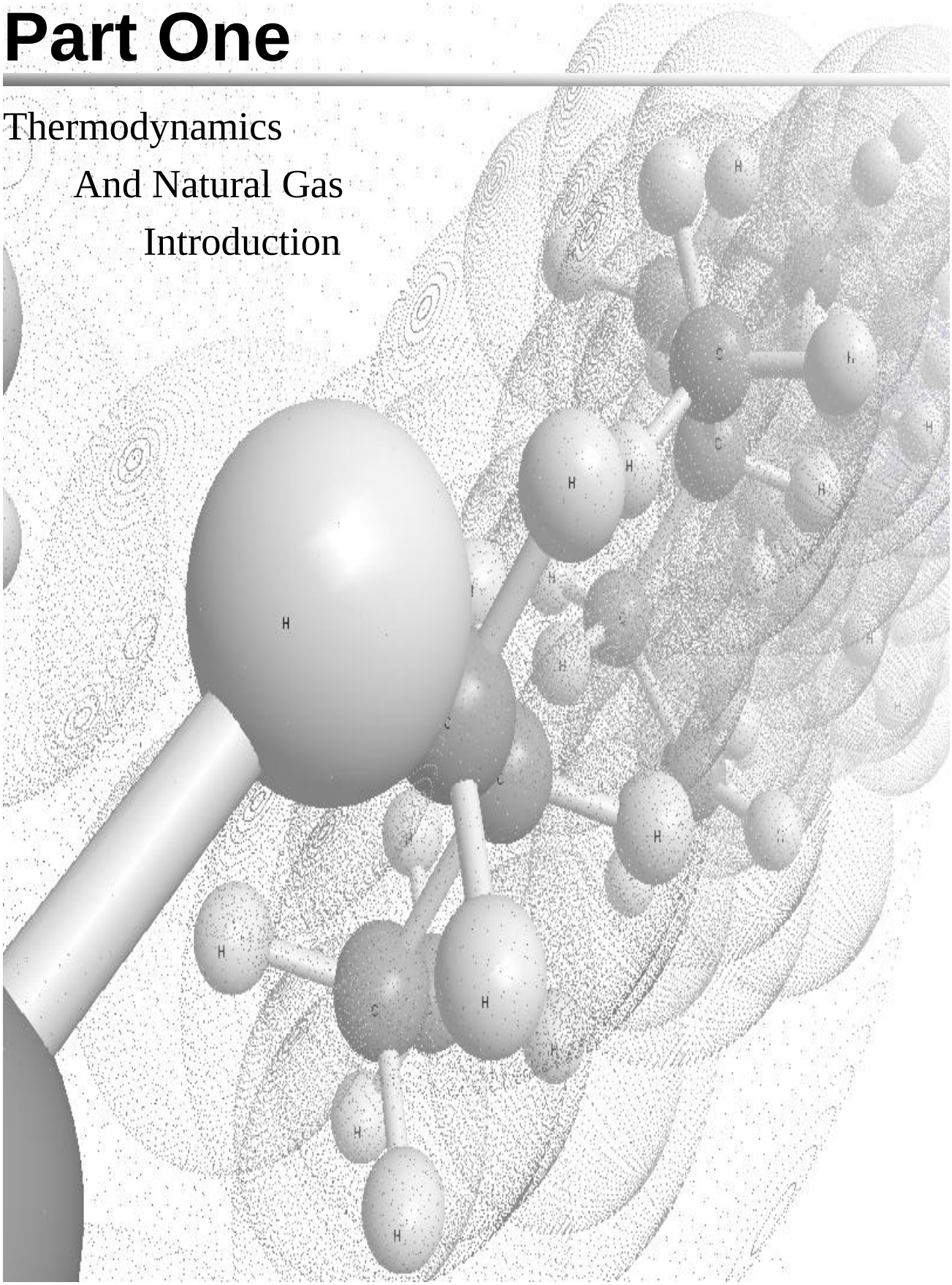
Contents

List of Figures

List of Graphs

Part One

Thermodynamics
And Natural Gas
Introduction



C h a p t e r 1

Equation of State

In physics and thermodynamics, an equation of state is a constitutive equation describing the state of matter under a given set of physical conditions. It provides a mathematical relationship between two or more state functions associated with the matter, such as its temperature, pressure, volume, or internal energy. Equations of state are useful in describing the properties of fluids, mixtures of fluids, solids, and even the interior of stars.

The most prominent use of an equation of state is to predict the state of gases and liquids. One of the simplest equations of state for this purpose is the ideal gas law, which is roughly accurate for gases at low pressures and high temperatures. However, this equation becomes increasingly inaccurate at higher pressures and lower temperatures, and fails to predict condensation from a gas to a liquid. Therefore, a number of much more accurate equations of state have been developed for gases and liquids. At present, there is no single equation of state that accurately predicts the properties of all substances under all conditions.

In addition to predicting the behavior of gases and liquids, there are also equations of state for predicting the volume of solids, including the transition of solids from one crystalline state to another. There are equations that model the interior of stars, including neutron stars. A related concept is the perfect fluid equation of state used in cosmology.

In the following equations the variables are defined as follows, any consistent set of units can be used although SI units are preferred:

P = pressure

V = volume

n = number of moles of a substance

$V_m = V/n$ = **molar volume**, the volume of 1 mole of gas or liquid

T = temperature (K)

R = ideal gas constant (8.314472 J/(mol·K))

CLASSICAL IDEAL GAS LAW

The classical ideal gas law may be written:

$$PV = nRT$$

The ideal gas law may also be expressed as follows

$$P = \rho(\gamma - 1)e$$

where ρ is the density, γ the adiabatic index, and e the internal energy. This form is purely in terms of intensive quantities and is useful when simulating the Euler equations because it expresses the relationship between internal energy and other forms of energy (such as kinetic), thus allowing simulations to obey the First Law.

VAN DER WAALS EQUATION OF STATE

$\left(P + \frac{a}{V_m^2}\right)(V_m - b) = RT$, note that V_m is molar volume.

Where a, b and R are constants that depend on the specific material. They can be calculated from the critical properties as:

$$a = 3P_cV_c^2$$

$$b = \frac{V_c}{3}$$

Proposed in 1873, the van der Waals equation of state was one of the first to perform markedly better than the ideal gas law. In this landmark equation a is called the attraction parameter and b the repulsion parameter or the effective molecular volume. While the equation is definitely superior to the ideal gas law and does predict the formation of a liquid phase, the agreement with experimental data is limited for conditions where the liquid forms. While the van der Waals equation is commonly referenced in text-books and papers for historical reasons, it is now obsolete. Other modern equations of only slightly greater complexity are much more accurate.

Van der Waals equation may be considered as the ideal gas law, "improved" due to two independent reasons:

Molecules are thought as particles with volume, not material points. Thus V cannot be too little, less than some constant. So we get (V - b) instead of V.

While ideal gas molecules do not interact, we consider molecules attracting others within a distance of several molecules' radii. It makes no effect inside material, but surface molecules attract to inside. We see this as diminishing of pressure on the outer shell (which is used in the ideal gas law), so we write (*P* + something) instead of *P*. To evaluate this 'something', let's examine addition force acting on an element of gas surface. While force acting on each surface molecule is ~ρ, the force acting on the whole element is $\sim\rho^2\sim\frac{1}{V_m^2}$

THE VIRIAL EQUATION OF STATE

$$\frac{PV_m}{RT}=1+\frac{B}{V_m}+\frac{C}{V^3{}_m}+\frac{D}{V^3{}_m}+\ldots$$

$$B=-V_c$$

$$C=\frac{V_c^2}{3}$$

Although usually not the most convenient equation of state, the virial equation is important because it can be derived directly from statistical mechanics. If appropriate assumptions are made about the mathematical form of intermolecular forces, theoretical expressions can be developed for each of the coefficients. In this case *B* corresponds to interactions between pairs of molecules, *C* to triplets, and so on.

REDLICH-KWONG EQUATION OF STATE

$$P=\frac{RT}{V_m-b}-\frac{a}{\sqrt{T}V_m(V_m+b)}$$

$$a=\frac{0.42748R^2T_c^{2.5}}{P_c}$$

$$b=\frac{0.08664RT_c}{P_c}$$

R = ideal gas constant (8.31451 J/(mol·K))

Introduced in 1949 the Redlich-Kwong equation of state was a considerable improvement over other equations of the time. It is still of interest primarily due to its relatively simple form. While superior to the van der Waals equation of state, it performs poorly with respect to the liquid phase and thus cannot be used for accurately calculating vapor-liquid equilibria. However, it can be used in conjunction with separate liquid-phase correlations for this purpose.

The Redlich-Kwong equation is adequate for calculation of gas phase properties when the ratio of the pressure to the critical pressure (reduced pressure) is less than about one-half of the ratio of the temperature to the critical temperature (reduced temperature).

THE SOAVE EQUATION OF STATE

$$P = \frac{RT}{V_m - b} - \frac{a\alpha}{V_m(V_m + b)}$$

R = ideal gas constant (8.31451 J/(mol·K))

$$a = \frac{0.42747 R^2 T_c^2}{P_c}$$

$$b = \frac{0.08664 R T_c}{P_c}$$

$$\alpha = \left(1 + \left(0.48508 + 1.55171\omega - 0.15613\omega^2\right)\left(1 - T_r^{0.5}\right)\right)^2$$

$$T_r = \frac{T}{T_c}$$

Where ω is the acentric factor for the species.

for hydrogen:

$$\alpha = 1.202 \exp(-0.30288 T_r)$$

In 1972 Soave replaced the $a/\sqrt{(T)}$ term of the Redlich-Kwong equation with a function α(T,ω) involving the temperature and the acentric factor. The α function was devised to fit the vapor pressure data of hydrocarbons and the equation does fairly well for these materials.

THE PENG-ROBINSON EQUATION OF STATE

$$P = \frac{RT}{V_m - b} - \frac{a\alpha}{V_m^2 + 2bV_m - b^2}$$

R = ideal gas constant (8.31451 J/(mol·K))

$$a = \frac{0.45724 R^2 T_c^2}{P_c}$$

$$b = \frac{0.07780 R T_c}{P_c}$$

$$\alpha = \left(1 + \left(0.37464 + 1.54226\omega - 0.26992\omega^2\right)\left(1 - T_r^{0.5}\right)\right)^2$$

$$T_r = \frac{T}{T_c}$$

Where ω is the acentric factor for the species.

The Peng-Robinson equation was developed in 1976 in order to satisfy the following goals:

The parameters should be expressible in terms of the critical properties and the acentric factor.

The model should provide reasonable accuracy near the critical point, particularly for calculations of the Compressibility factor and liquid density.

The mixing rules should not employ more than a single binary interaction parameter, which should be independent of temperature pressure and composition.

The equation should be applicable to all calculations of all fluid properties in natural gas processes.

For the most part the Peng-Robinson equation exhibits performance similar to the Soave equation, although it is generally superior in predicting the liquid densities of many materials, especially nonpolar ones.

THE BWRS EQUATION OF STATE

$$P = \rho RT + \left(B_\circ RT - A_\circ - \frac{C_\circ}{T^2} + \frac{D_\circ}{T^3} - \frac{E_\circ}{T^4} \right)\rho^2 + \left(bRT - a - \frac{d}{T} \right)\rho^3 + \alpha\left(a + \frac{d}{T} \right)\rho^6$$

$$+ \frac{c\rho^3}{T^2}\left(1 + \gamma\rho^2\right)\exp\left(-\gamma\rho^2\right)$$

ρ = the molar density

Values of the various parameters for 15 substances can be found in:

K.E. Starling, *Fluid Properties for Light Petroleum Systems.* Gulf Publishing Company (1973).

ELLIOTT, SURESH, DONOHUE

The Elliott, Suresh, and Donohue (ESD) equation of state (EOS) was proposed in 1990. The equation seeks to correct a shortcoming in the Peng-Robinson EOS in that there was an inaccuracy in the van der Waals repulsive term. The EOS accounts for the effect of the shape of a non-polar molecule and can be extended to polymers with the addition of an extra term (not shown). The EOS itself was developed through modeling computer simulations and should capture the essential physics of the size, shape, and hydrogen bonding.

$$\frac{PV_m}{RT} = Z = 1 + \frac{4\langle c\eta \rangle}{1 - 1.9\eta} - \frac{9.5\langle qY\eta \rangle}{1 + 1.7745\langle Y\eta \rangle}$$

Where:

c = a "`shape factor"'

η = bρ

q = 1 + 1.90476(c − 1)

$$Y = \exp\left(\frac{\in}{kT} \right) - 1.0617$$

Reference: Elliott & Lira, Introductory Chemical Engineering Thermodynamics, 1999, Prentice Hall.

STIFFENED EQUATION OF STATE

When considering water under very high pressures (typical applications are underwater nuclear explosions, sonic shock lithotripsy, and sonoluminescence) the stiffened equation of state is often used:

$$p = \rho(\gamma - 1)e - \gamma p^0$$

Where e is the internal energy per unit mass, γ is an empirically determined constant typically taken to be about 6.1, and p^0 is another constant, representing the molecular attraction between water molecules. The magnitude of the correction is about 2 gigapascals (20000 atmospheres).

The equation is stated in this form because the speed of sound in water is given by $c^2 = \gamma(p + p^0) / \rho$.

Thus water behaves as though it is an ideal gas that is *already* under about 20000 atmospheres (2 GPa) pressure, and explains why water is commonly assumed to be incompressible: when the external pressure changes from 1 atmosphere to 2 atmospheres (100 kPa to 200 kPa), the water behaves as an ideal gas would do when changing from 20001 to 20002 atmospheres (200.01 MPa to 201.02 MPa).

This equation mispredicts the specific heat capacity of water but few alternatives are available for severely nonisentropic processes such as strong shocks.

ULTRARELATIVISTIC EQUATION OF STATE

An ultrarelativistic fluid has equation of state

$$p = c_s^2 \mu$$

Where p is the pressure, μ is the energy density, and c_s is a constant referred to as the sound speed.

IDEAL BOSE EQUATION OF STATE

The equation of state for an ideal Bose gas is

$$PV_m = RT \frac{Li_{\alpha+1}(z)}{\zeta(\alpha)} \left(\frac{T}{T_c} \right)^{\alpha}$$

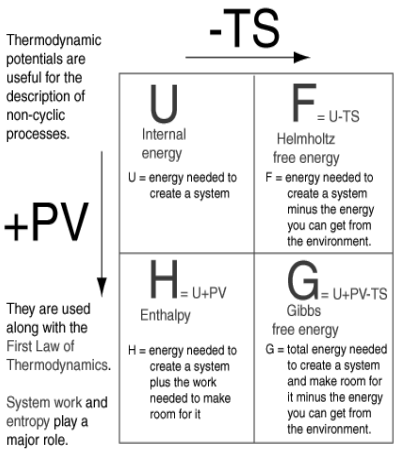
where α is an exponent specific to the system (e.g. in the absence of a potential field, $\alpha=3/2$), z is $\exp(\mu/kT)$ where μ is the chemical potential, Li is the polylogarithm, ζ is the Riemann zeta function, and T_c is the critical temperature at which a Bose-Einstein condensate begins to form.

C h a p t e r 2

Thermodynamic Potentials

In thermodynamics, thermodynamic potentials are parameters associated with a thermodynamic system and have the dimensions of energy. They are called "potentials" because in a sense, they describe the amount of potential energy in a thermodynamic system when it is subjected to certain constraints. The different potentials correspond to different constraints to which the system may be subjected. The four most common thermodynamic potentials are:

Name	Formula	Natural variables
Internal energy	U	$S, V, \{N_i\}$
Helmholtz free energy	$A = U - TS$	$T, V, \{N_i\}$
Enthalpy	$H = U + PV$	$S, P, \{N_i\}$
Gibbs free energy	$G = U + PV - TS$	$T, P, \{N_i\}$



Where T = temperature, S = entropy, P = pressure, V = volume. The Helmholtz free energy is often denoted by the symbol F, but the use of A is preferred by IUPAC (See Alberty, 2001). Ni is the number of particles of type i in the system. For the sake of completeness, the set of all Ni are also included as natural variables, although they are sometimes ignored.

DESCRIPTION AND INTERPRETATION

Thermodynamic potentials are very useful when calculating the expected results of a chemical reaction, or when measuring the properties of materials in a chemical reaction. The chemical reactions usually take place under some simple constraints such as constant pressure and temperature, or constant entropy and volume, and when this is true, there is a corresponding thermodynamic potential which comes into play. Just as in mechanics, the system will tend towards lower values of potential and at equilibrium, under these constraints, the potential will take on an unchanging minimum value. The thermodynamic potentials can also be used to estimate the total amount of energy available from a thermodynamic system under constraint.

NATURAL VARIABLES

The variables that are held constant in this process are termed the **natural variables** of that potential. The natural variables are important not only for the above mentioned reason, but also because if a thermodynamic potential can be determined as a function of its natural variables, all of the thermodynamic properties of the system can be found by taking partial derivatives of that potential with respect to its natural variables and this is true for no other combination of variables.

CONJUGATE VARIABLES

Just as a small increment of energy in a mechanical system is the product of a force times a small displacement, so an increment in the energy of a thermodynamic system can be expressed as the sum of the products of certain generalized "forces" which, when unbalanced, cause certain generalized "displacements" to occur, with their product being the energy transferred as a result. These forces and their associated displacements are called conjugate variables. For example, consider the PV conjugate pair. The pressure P acts as a generalized force: Pressure differences force a change in volume dV, and their product is the energy lost by the system due to work. Here pressure is the driving force, volume is the associated displacement, and the two form a pair of conjugate variables. In a similar way, temperature differences drive changes in entropy, and their product is the energy transferred by heat transfer. The thermodynamic force is always an intensive variable and the displacement is always an extensive variable, yielding an extensive energy.

MORE CONJUGATE VARIABLES - THE CHEMICAL POTENTIAL

The theory of thermodynamic potentials is not complete until we consider the number of particles in a system as a variable on par with the other extensive quantities such as volume and entropy. The number of particles is, like volume and entropy, the displacement variable in a conjugate pair. The generalized force component of this pair is the chemical potential. The chemical potential may be thought of as a force which, when imbalanced, pushes an exchange of particles, either with the surroundings, or between phases inside the system. In cases where there are a mixture of chemicals and phases, this is a useful concept. For example if a container holds liquid water and water vapor, there will be a chemical potential (which is negative) for the liquid which pushes the water molecules into the vapor (evaporation) and a chemical potential for the vapor, pushing vapor molecules into the liquid (condensation). Only when these "forces" equilibrate and the chemical potentials of each phase is equal, is equilibrium obtained.

MORE THERMODYNAMIC POTENTIALS

Notice that the set of natural variables for the above four potentials are formed from every combination of the T-S and P-V variables, as long as two conjugate variables are not used. There is no reason to ignore the $\mu i - Ni$ conjugate pairs, and in fact we may define four additional potentials for each species. Using IUPAC notation in which the brackets contain the natural variables (other than the main four), we have:

Formula	Natural variables
$U[\mu_j] = U - \mu_j N_j$	$S, V, \{N_{i\neq j}\}, \mu_j$
$A[\mu_j] = U - TS - \mu_j N_j$	$T, V, \{N_{i\neq j}\}, \mu_j$
$H[\mu_j] = U + PV - \mu_j N_j$	$S, P, \{N_{i\neq j}\}, \mu_j$
$G[\mu_j] = U + PV - TS - \mu_j N_j$	$T, P, \{N_{i\neq j}\}, \mu_j$

If there is only one species, then we are done, but if there are, say two species, then there will be additional potentials such as U[μ1,μ2] = U − μ1N1 − μ2N2 and so on. If there are D dimensions to the thermodynamic space, then there are 2D unique thermodynamic potentials. For the most simple case, a single phase ideal gas, there will be three dimensions, yielding eight thermodynamic potentials.

THE FUNDAMENTAL EQUATIONS

The definitions of the thermodynamic potentials may be differentiated and, along with the first and second law of thermodynamics, a set of differential equations known as the fundamental equations may be derived. By the first law of thermodynamics, any differential change in the internal energy U of a system can be written as the sum of heat flowing into the system and work done by the system on the environment, along with any change due to the addition of new particles to the system:

$$dU = \delta Q - \delta W + \sum_i \mu_i dN_i$$

where δQ is the infinitesimal heat flow into the system, and δW is the infinitesimal work done by the system, μi is the chemical potential of particle type i and Ni is the number of type i particles. (Note that neither δQ nor δW are exact differentials. Small changes in these variables are therefore represented with δ rather than d.)

By the second law of thermodynamics, we can express the internal energy change in terms of state functions and their differentials:

$$\delta Q \leq TdS$$

$$\delta W = PdV$$

Where T is temperature, S is entropy, P is pressure, and V is volume, and the equality holds for reversible processes.

This leads to the standard differential form of the internal energy:

$$dU \leq TdS - PdV + \sum_i \mu_i dN_i$$

Applying Legendre transforms repeatedly, the following differential relations hold for the four potentials:

$$dU \leq TdS - PdV + \sum_i \mu_i dN_i$$

$$dA \leq -SdT - PdV + \sum_i \mu_i dN_i$$

$$dH \le TdS + VdP + \sum_i \mu_i dN_i$$

$$dG \le -sdT + VdP + \sum_i \mu_i dN_i$$

Note that the infinitesimals on the right hand side of each of the above equations are of the natural variables of the potential on the left hand side. The above relations illustrate that when the natural variables of each potential are held constant, the potential decreases in value in an irreversible way, approaching its constant, minimum value at equilibrium.

Similar equations can be developed for all of the other thermodynamic potentials of the system. There will be one fundamental equation for each thermodynamic potential, resulting in a total of 2D fundamental equations.

THE EQUATIONS OF STATE

We can use the above equations to derive some differential definitions of some thermodynamic parameters. If we define Φ to stand for any of the thermodynamic potentials, then the above equations are of the form:

$$d\Phi = \sum_i x_i dy_i$$

Where xi and yi are conjugate pairs, and the yi are the natural variables of the potential Φ. From the chain rule it follows that:

$$x_i = \left(\frac{\partial \Phi}{\partial y_j} \right)_{\{y_{i \neq j}\}}$$

Where $\{y_{i \neq j}\}$ is the set of all natural variables of Φ except yj. This yields expressions for various thermodynamic parameters in terms of the derivatives of the potentials with respect to their natural variables. These equations are known as equations of state since they specify parameters of the thermodynamic state. If we restrict ourselves to the potentials U,A,H and G, then we have:

$$+T = \left(\frac{\partial U}{\partial S} \right)_{V,\{N_i\}} = \left(\frac{\partial H}{\partial S} \right)_{P,\{N_i\}}$$

$$-P = \left(\frac{\partial U}{\partial V} \right)_{S,\{N_i\}} = \left(\frac{\partial A}{\partial V} \right)_{T,\{N_i\}}$$

$$+V = \left(\frac{\partial H}{\partial P} \right)_{S,\{N_i\}} = \left(\frac{\partial G}{\partial P} \right)_{T,\{N_i\}}$$

$$-S = \left(\frac{\partial G}{\partial T} \right)_{P,\{N_i\}} = \left(\frac{\partial A}{\partial T} \right)_{V,\{N_i\}}$$

$$\mu_j = \left(\frac{\partial \phi}{\partial N_j} \right)_{X,Y,\{N_{j \neq i}\}}$$

Where, in the last equation, φ is any of the thermodynamic potentials U,A,H, G and $X, Y, \{N_{j\neq i}\}$ are the set of natural variables for that potential, excluding Ni. If we use all potentials, then we will have more equations of state such as

$$-N_j = \left(\frac{\partial U\left[\mu_j\right]}{\partial \mu_j}\right)_{S,V,\{N_{i\neq j}\}}$$

and so on. In all, there will be D equations for each potential resulting in a total of D2D equations of state.

THE MAXWELL RELATIONS

Again, define xi and yi to be conjugate pairs, and the yi to be the natural variables of some potential Φ. We may take the "cross differentials" of the state equations, which obey the following relationship:

$$\left(\frac{\partial}{\partial x_i}\left(\frac{\partial \Phi}{\partial y_k}\right)_{\{y_{i\neq k}\}}\right)_{\{x_{i\neq j}\}} = \left(\frac{\partial}{\partial y_k}\left(\frac{\partial \Phi}{\partial x_j}\right)_{\{x_{i\neq j}\}}\right)_{\{y_{i\neq k}\}}$$

From these we get the Maxwell relations. There will be (D-1)/2 of them for each potential giving a total of D(D-1)/2 equations in all. If we restrict ourselves the U, A, H, G

$$\left(\frac{\partial T}{\partial V}\right)_{S,\{N_i\}} = -\left(\frac{\partial P}{\partial S}\right)_{V,\{N_i\}}$$

$$\left(\frac{\partial T}{\partial P}\right)_{S,\{N_i\}} = +\left(\frac{\partial V}{\partial S}\right)_{P,\{N_i\}}$$

$$\left(\frac{\partial S}{\partial V}\right)_{T,\{N_i\}} = +\left(\frac{\partial P}{\partial T}\right)_{V,\{N_i\}}$$

$$\left(\frac{\partial S}{\partial P}\right)_{T,\{N_i\}} = -\left(\frac{\partial V}{\partial T}\right)_{P,\{N_i\}}$$

Using the equations of state involving the chemical potential we get equations such as:

$$\left(\frac{\partial T}{\partial N_j}\right)_{V,S,\{N_{i\neq j}\}} = -\left(\frac{\partial \mu_j}{\partial S}\right)_{V,\{N_i\}}$$

and using the other potentials we can get equations such as:

$$\left(\frac{\partial N_j}{\partial V}\right)_{S,\mu_j,\{N_{i\neq j}\}} = -\left(\frac{\partial P}{\partial \mu_j}\right)_{S,V,\{N_{i\neq j}\}}$$

$$\left(\frac{\partial N_j}{\partial N_k}\right)_{S,V,\mu_j,\{N_{i\neq j},k\}} = -\left(\frac{\partial \mu_k}{\partial \mu_j}\right)_{S,V,\{N_{i\neq j}\}}$$

EULER INTEGRALS

Again, define xi and yi to be conjugate pairs, and the yi to be the natural variables of the internal energy. Since all of the natural variables of the internal energy U are extensive quantities

$$U(\{\alpha y_i\}) = \alpha U(\{y_i\})$$

it follows from Euler's homogeneous function theorem that the internal energy can be written as:

$$U(\{y_i\}) = \sum_j y_i \left(\frac{\partial U}{\partial y_j}\right)_{\{y_{i\neq j}\}}$$

From the equations of state, we then have:

$$U = TS - PV - \sum_i \mu_i N_i$$

Substituting into the expressions for the other main potentials we have:

$$A = -PV + \sum_i \mu_i N_i$$

$$H = TS + \sum_i \mu_i N_i$$

$$G = \sum_i \mu_i N_i$$

As in the above sections, this process can be carried out on all of the other thermodynamic potentials.

THE GIBBS-DUHEM RELATION

We may differentiate the above Euler equation for the internal energy:

$$dU = TdS + SdT - PdV - VdP + \sum_i \left(\mu_i dN_i + N_i d\mu_i\right)$$

But the fundamental equation for U states that

$$dU = TdS - PdV + \sum_i \mu_i dN_i$$

Subtracting yields the Gibbs-Duhem relation:

$$0 = SdT - VdP + \sum_i N_i d\mu_i$$

The Gibbs-Duhem is a relationship among the intensive parameters of the system. It follows that for a simple system with r components, there will be r+1 independent parameters, or degrees of freedom. For example, a simple system with a single component will have two degrees of freedom, and may be specified by only two parameters, such as pressure and volume for example. The law is named after Josiah Gibbs and Pierre Duhem.

C h a p t e r 3

Natural Gas

Commonly referred to as gas, is a gaseous fossil fuel consisting primarily of methane. It is found in oil fields and natural gas fields, and in coal beds. When methane-rich gases are produced by the anaerobic decay of non-fossil organic material, these are referred to as biogas. Sources of biogas include swamps, which produce swamp gas; marshes, which produce marsh gas; landfills, which produce landfill gas, as well as sewage sludge and manure, by way of anaerobic digesters, in addition to Enteric fermentation particularly in cattle.

CHEMICAL COMPOSITION

The primary component of natural gas is methane (CH4), the shortest and lightest hydrocarbon molecule. It also contains heavier gaseous hydrocarbons such as ethane (C2H6), propane (C3H8) and butane (C4H10), as well as other sulphur containing gases, in varying amounts, see also natural gas condensate. Natural gas also contains and is the primary market source of helium.

Organosulfur compounds and hydrogen sulfide are common contaminants which must be removed prior to most uses. Gas with a significant amount of sulfur impurities is termed "sour" and often referred to as "acid gas". Processed Natural gas that is available to end-users is tasteless and odorless, however, before gas is distributed to end-users, it is odorized by adding small amounts of thiols, to assist in leak detection. Processed Natural gas is, in itself, harmless to the human body, however, natural gas is a simple asphyxiant and can kill if it displaces air to the point where the oxygen content will not support life.

Natural gas can also be hazardous to life and property through an explosion. Natural gas is lighter than air, and so tends to dissipate into the atmosphere. But when natural gas is confined, such as within a house, gas concentrations can reach explosive mixtures and, if ignited, result in blasts that could destroy buildings. Methane has a lower explosive limit of 5% in air, and an upper explosive limit of 15%.

Explosive concerns with compressed natural gas used in vehicles are almost non-existent, due to the escaping nature of the gas, and the need to maintain concentrations between 5% and 15% to trigger explosions.

ENERGY CONTENT AND STATISTICS

Combustion of one cubic metre of commercial quality natural gas yields 38 megajoules (10.6 kWh). Equivalently, one cubic foot of natural gas produces 1031 British Thermal Units (BTUs).

In the USA, at retail, natural gas is often sold in units of therms (th); 1 therm = 100,000 BTU. Wholesale transactions are generally done in decatherms (Dth), or in thousand decatherms (MDth), or in million decatherms (MMDth). A million decatherms is roughly a billion cubic feet of natural gas.

STORAGE AND TRANSPORT

The major difficulty in the use of natural gas is transportation and storage. Natural gas pipelines are economical, but are impractical across oceans. Many existing pipelines in North America are close to reaching their capacity, prompting some politicians in colder climates to speak publicly of potential shortages.

LNG carriers can be used to transport liquefied natural gas (LNG) across oceans, while tank trucks can carry liquefied or compressed natural gas (CNG) over shorter distances. They may transport natural gas directly to end-users, or to distribution points such as pipelines for further transport. These may have a higher cost, requiring additional facilities for liquefaction or compression at the production point, and then gasification or decompression at end-use facilities or into a pipeline.

In the past, the natural gas which was recovered in the course of recovering petroleum could not be profitably sold, and was simply burned at the oil field (known as flaring). This wasteful practice is now illegal in many countries, especially since it adds greenhouse gas pollution to the earth's atmosphere. Additionally, companies now recognize that value for the gas may be achieved with LNG, CNG, or other transportation methods to end-users in the future. The gas is now re-injected back into the formation for later recovery. This also assists oil pumping by keeping underground pressures higher. In Saudi Arabia, in the late 1970s, a "Master Gas System" was created, ending the need for flaring. The natural gas is used to generate electricity and heat for desalinization. Similarly, some land-fills that also discharge methane gases have been set-up to capture the methane and generate electricity.

Natural gas is often stored in underground caverns formed inside depleted gas reservoirs from previous gas wells, salt domes, or in tanks as liquefied natural gas. The gas is injected during periods of low demand and extracted during periods of higher demand. Storage near the ultimate end-users helps to best meet volatile demands, but this may not always be practicable.

NATURAL GAS CRISIS

Many politicians and prominent figures in North America have spoken publicly about a possible natural gas crisis. This includes former Secretary of Energy Spencer Abraham, former Chairman of the Federal Reserve Alan Greenspan, and Ontario Minister of Energy Dwight Duncan.

The natural gas crisis is typically described by the increasing price of natural gas in the U.S. over the last few years, due to the decline in indigenous supply and the increase in demand for electricity generation. Indigenous supply has not truly fallen -- but it has leveled off (no matter how many new straws are put into the ground, about the same amount of natural gas is recovered each year). But because of the continuing growth in demand, and the temporary but dramatic hit to production that came from Hurricanes Katrina and Rita, the price has become so high that many industrial users, mainly in the petrochemical industry, have closed their plants causing loss of jobs. Greenspan has suggested that a solution to the natural gas crisis is the import of LNG.

This solution is both capital intensive and politically charged due to the NIMBY syndrome and the public perception that LNG terminals are explosive risks, especially in the wake of the 9/11 terrorist attacks in the United States. The U.S. Department of Homeland Security is responsible for maintaining their security, and the security arrangements during the 2004 Democratic Convention in Boston, Massachusetts, home to one of only six LNG terminals in the United States, were extraordinarily tight.

New or expanded LNG terminals create tough infrastructure problems and require high capital spending. LNG terminals require a very spacious—at least 40 feet (12.2 m) deep[1]—harbor, as well as being sheltered from wind and waves. These "suitable" sites are thus deep in well-populated seaports, which are also burdened with right-of-way concerns for LNG pipelines, or conversely, required to also host the LNG expansion plant facilities and end use (petrochemical) plants amidst the high population densities of major cities, with the associated fumes, multiple serious risks to safety.

Typically, to attain "well-sheltered" waters, suitable harbor sites are well up rivers or estuaries, which are unlikely to be dredged deep enough. Since these very large vessels must move slowly and ponderously in restricted waters, the transit times to and from the terminal become costly, as multiple tugboats and security boats shelter and safeguard the large vessels. Operationally, LNG tankers are (for example, in Boston) effectively given sole use of the harbor, forced to arrive and depart during non-peak hours, and precluded from occupying the same harbor until the first is well-departed. These factors increase operating costs and make capital investment less attractive.

To substantially increase the amount of LNG used to supply natural gas to North America, not only must "re-gasification" plants be built on North American shores -- difficult for the reasons stated above -- someone also must put substantial, new liquefaction stations in Indonesia, the Middle East, and Africa, in order to concentrate the gas generally associated with oil production in those areas. A substantial expansion of the fleet of LNG carriers also must occur, to move the huge amount of fuel needed to make up for the coming shortfall in North America.

USES

Power generation

Natural gas is a major source for electricity generation through the use of gas turbines and steam turbines. Particularly high efficiencies can be achieved through combining gas turbines with a steam turbine in combined cycle mode. Natural gas burns cleaner than other fossil fuels, such as oil and coal, and produces less greenhouse gas per unit energy released. For an equivalent amount of heat, burning natural gas produces about 30% less carbon dioxide than burning petroleum and about 45% less than burning coal. [1] Combined cycle power generation using natural gas is thus the cleanest source of power available using

fossil fuels, and this technology is widely used wherever gas can be obtained at a reasonable cost. Fuel cell technology may eventually provide cleaner options for converting natural gas into electricity, but as yet it is not price-competitive. Also, the natural gas supply is said to peak around the year 2030, 20 years after the peak of oil. It is also projected that the world's supply of natural gas should be exhausted around the year 2085.

Hydrogen

Natural gas can be used to produce hydrogen that can be used in hydrogen vehicles.

Natural gas vehicles

Compressed natural gas (and LPG) is used as a clean alternative to other automobile fuels. As of 2003, the countries with the largest number of natural gas vehicles were Argentina, Brazil, Pakistan, Italy, and India. The energy efficiency is generally equal to that of gasoline engines, but lower compared with modern diesel engines, partially due to the fact that natural gas engines function using the Otto Cycle, but research is on its way to improve the process (Westport Cycle).

Residential domestic use

Natural gas is supplied to homes, where it is used for such purposes as cooking and heating/cooling. CNG is used in rural homes without connections to piped-in public utility services, or with portable grills.

Fertilizer

Natural gas is a major feedstock for the production of ammonia, via the Haber process, for use in fertilizer production.

Other

Natural gas is also used in the manufacture of fabrics, glass, steel, plastics, paint, and other products.

SOURCES

Natural gas is commercially produced from oil fields and natural gas fields. Gas produced from oil wells is called casinghead gas or associated gas. The largest two natural gas fields are probably South Pars Gas Field in Iran and Urengoy gas field in Russia, with reserves on the order of 1013 m^3. See also List of natural gas fields. Qatar also has 25 trillion cubic meters of natural gas (5% of the world's proven supply), enough to last 250 years at current production levels.

Town gas is a mixture of methane and other gases which can be used in a similar way to natural gas and can be produced by treating coal chemically. This is a historic technology, still used as 'best solution' in some local circumstances, although coal gasification is not usually economic at current gas prices, depending upon infrastructure considerations.

Methanogenic archaea are responsible for all biological sources of methane, some in symbiotic relationships with other life forms, including termites, ruminants, and cultivated crops. Methane released directly into the atmosphere would be considered a pollutant, however, methane in the atmosphere is oxidised, producing carbon dioxide and water.

Possible future sources

Future sources of methane, the principal component of natural gas, include landfill gas, biogas and methane hydrate. Biogas, and especially landfill gas, are already used in some areas, but their use could be greatly expanded. Landfill gas is a type of biogas, but biogas usually refers to gas produced from organic material that has not been mixed with other waste.

Landfill gas is created from the decomposition of waste in landfills. If the gas is not removed, the pressure may get so high that it works its way to the surface, causing damage to the landfill structure, unpleasant

odor, vegetation die-off and an explosion hazard. The gas can be vented to the atmosphere, flared or burned to produce electricity or heat.

Once water vapor is removed, about half of landfill gas is methane. Almost all of the rest is carbon dioxide, but there are also small amounts of nitrogen, oxygen and hydrogen. There are usually trace amounts of hydrogen sulfide and siloxanes, but their concentration varies widely. Landfill gas cannot be distributed through natural gas pipelines unless it is cleaned up to the same quality. It is usually more economical to combust the gas on site or within a short distance of the landfill using a dedicated pipeline. Water vapor is often removed, even if combusting the gas on site. Other non-methane components may also be removed in order to meet emissions standards, to prevent fouling of the equipment or for environmental considerations. Co-firing landfill gas with natural gas improves combustion, which lowers emissions.

Biogas is usually produced using agricultural waste materials, such as unmerchantable parts of plants and manure. Biogas can also be produced by separating organic materials from waste that otherwise goes to landfills, which is more efficient than just capturing the landfill gas it produces. Using materials that would otherwise generate no income, or even cost money to get rid of, improves the profitability and energy balance of biogas production.

Anaerobic lagoons are used to produce biogas from manure, while biogas reactors can be used for manure or plant parts. Like landfill gas, biogas is mostly methane and carbon dioxide, with small amounts of nitrogen, oxygen and hydrogen. However, with the exception of pesticides, there are usually lower levels of contaminants.

A speculative source of enormous quantities of methane is from methane hydrate, found under sediments in the oceans. However, as of 2006 no technology has been developed to recover it economically.

SAFETY

In any form, a minute amount of odorant such as t-butyl mercaptan, with a rotting-cabbage-like smell, is added to the otherwise colorless and odorless gas, so that leaks can be detected before a fire or explosion occurs. Sometimes a related compound, thiophane is used, with a rotten-egg smell. Adding odorant to natural gas began in the United States after the 1937 New London School explosion. The buildup of gas in the school went unnoticed, killing three hundred students and faculty when it ignited. Odorants are considered non-toxic in the extremely low concentrations occurring in natural gas delivered to the end user.

In mines, where methane seeping from rock formations has no odor, sensors are used, and mining apparatus has been specifically developed to avoid ignition sources, e.g., the Davy lamp.

Explosions caused by natural gas leaks occur a few times each year. Individual homes, small businesses and boats are most frequently affected when an internal leak builds up gas inside the structure. Frequently, the blast will be enough to significantly damage a building but leave it standing. In these cases, the people inside tend to have minor to moderate injuries. Occasionally, the gas can collect in high enough quantities to cause a deadly explosion, disintegrating one or more buildings in the process. The gas usually dissipates readily outdoors, but can sometimes collect in dangerous quantities if weather conditions are right. Also, considering the tens of millions of structures that use the fuel, the individual risk of using natural gas is very low.

Some gas fields yield sour gas containing hydrogen sulfide. This untreated gas is toxic.

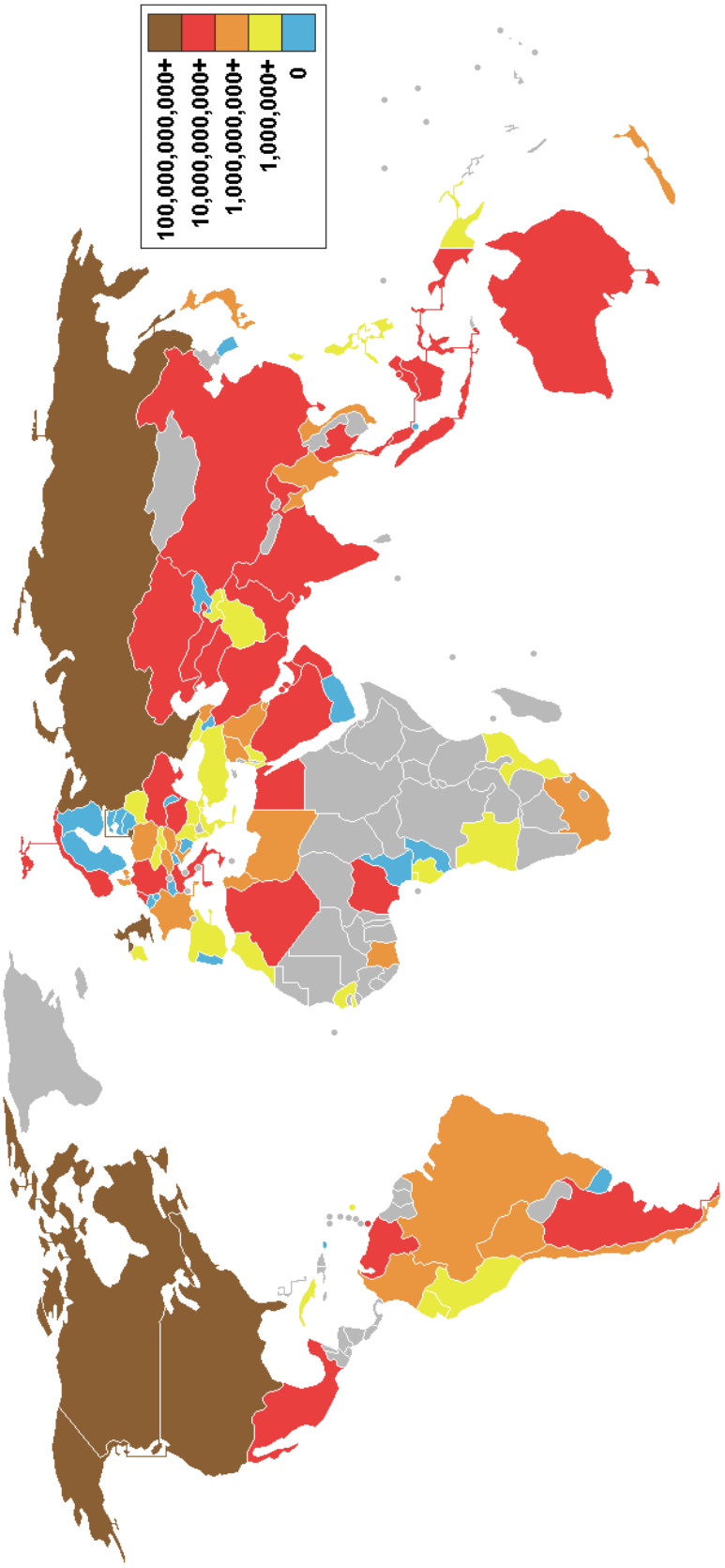
Extraction of natural gas (or oil) leads to decrease in pressure in the reservoir. This in turn may lead to subsidence at ground level. Subsidence may affect ecosystems, waterways, sewer and water supply systems, foundations, etc.

NATURAL GAS CONSUMPTION

This is a list of countries by natural gas consumption mostly based on The World Factbook [1] accessed in June 2006. as it seen in the list Egypt is Occupying the 24th place in consumption of natural gas which made it one of the heaviest country in consumpion

Rank by sovereign state	Country	Natural Gas- consumption (m^3)	Date of Information
1	World	2673000000000	2003 est.
2	United States	633600000000	2003 est.
3	European Union	465600000000	2001
4	Russia	402100000000	2004 est.
5	United Kingdom	95150000000	2003 est.
6	Germany	93880000000	2003 est.
7	Canada	90950000000	2003 est.
8	Japan	86510000000	2003 est.
9	Iran	79000000000	2003 est.
10	Italy	76880000000	2003 est.
11	Ukraine	75800000000	2004
12	Saudi Arabia	60060000000	2003 est.
13	Mexico	55100000000	2004 est.
14	Netherlands	50400000000	2003 est.
15	Uzbekistan	49300000000	2004
16	France	43740000000	2003 est.
17	United Arab Emirates	37880000000	2003 est.
18	Argentina	34580000000	2003 est.
19	China	33440000000	2003 est.
20	Venezuela	29700000000	2003 est.
21	Thailand	29150000000	2003 est.
22	Malaysia	28530000000	2003 est.
23	India	27100000000	2003 est.
24	Egypt	27000000000	2003 est.
25	Australia	25080000000	2003 est.
26	South Korea	24090000000	2003 est.
27	Pakistan	23800000000	2003 est.
28	Spain	23270000000	2003 est.
29	Turkey	22600000000	2005 est.
30	Indonesia	22500000000	2005 est.

Figure 1: Natural Gas Consumption



C h a p t e r 4

Properties Equations And Derivation

In this chapter we will investigate the equations which will be used in the software implementation. We will only mention the sequence of derivations regardless of expressing the final derivation form.

This will hopefully give the reader the ability to trace the sequence of the application that were made so he can get back to this chapter and to be able to grasp the whole idea of the research at any time during the walking through the this report.

However equations will be re deduced again in Part three of this report followed by the final form of the equation and the code implementation.

IDEAL ISOBARIC SPECIFIC HEAT

Dealing with a the natural gas is the same like dealing with gas mixtures we have to know the essential properties of every gas included in the mixture such as molecular weight, mole fraction, and the constants of ideal specific heat of the desired equation.

In this project we have used two methods in calculating the Ideal Specific Heat mainly Cp isobaric specific heat.

First Method is the traditional method which rely on polynomial Equation

$$Cp_{ideal}(T) = M(a + bT + cT^2 + dT^3)$$

Where

M is the molecular weight.

T is the Temperature in Kelvin.

Cp is the ideal isobaric specific heat.

a, b, c and d are the constants of pure substances contained in natural gas.

Second Method is obtained from Schely and Maeschke[I] the equation that were mentioned in their paper is

$$\frac{Cp_{ideal}(T)}{R_u} = b + c\left\{\frac{d/T}{\sinh(d/T)}\right\}^2 + e\left\{\frac{f/T}{\cosh(f/T)}\right\}^2 + g\left\{\frac{h/T}{\sinh(h/T)}\right\}^2 + i\left\{\frac{j/T}{\cosh(j/T)}\right\}^2$$

Where

R_u is the universal gas constant (8.314 $kJ/kmol.K$)

b, c, d, e, f, g, h, i and j are constants.

Then the ideal isobaric specific heat for mixture could be calculated from the equation

$$Cp_{ideal} = \sum_{i=1}^{i=n} y_i Cp_{ideal}$$

ACTUAL ISOBARIC SPECIFIC HEAT

After getting the ideal specific heat for the mixture we will calculate the actual isobaric specific heat.

Equation

Assuming we have the equation of state that govern the behavior of natural gas or by using one of the equations described in chapter one we can know the Cp departure by the following equation

$$(Cp_{actual} - Cp_{ideal}) = -T\int_1^p (\partial^2 v/\partial T^2)_p \, dp$$

Then we can proceed to calculate the Actual isobaric specific heat by

$$Cp_{actual} = Cp_{ideal} + (Cp_{actual} - Cp_{ideal})$$

ISOCHORIC SPECIFIC HEAT

Equation

Calculating the Isochoric specific heat *Cv* is done by calculating the difference between the Isochoric and Isobaric specific heat. This can be done by

$$Cp - Cv = T\left(\frac{\partial v}{\partial T}\right)_{p=c}\left(\frac{\partial p}{\partial T}\right)_{v}$$

After that we sum up this difference by the Actual *Cp* and then we can get the *Cv* value.

JOULE THOMSON EFFECT

In physics, the Joule-Thomson effect, or Joule-Kelvin effect, is a process in which the temperature of a real gas is either decreased or increased by letting the gas expand freely at constant enthalpy (which means that no heat is transferred to or from the gas, and no external work is extracted).

It is named for James Prescott Joule and William Thomson, 1st Baron Kelvin who established the effect in 1852 following earlier work by Joule on Joule expansion in which a gas expands at constant internal energy.

Description

The relationship between temperature, pressure and volume of a gas is simply described by the gas laws. When volume is increased in an irreversible process, the gas laws do not uniquely determine what happens to the pressure and temperature of the gas. Reversible adiabatic expansion, in which the gas does positive work in the process of expansion, always causes a decrease in temperature.

However, when a real gas (as differentiated from an ideal gas) expands freely at constant enthalpy, the temperature may either decrease or increase, depending on the initial temperature and pressure. For any given pressure, a real gas has a Joule-Thomson (Kelvin) inversion temperature, above which expansion at constant enthalpy causes the temperature to rise, and below which expansion at constant enthalpy causes cooling. For most gases at atmospheric pressure, the inversion temperature is fairly high (above room temperature), and so most gases at those temperature and pressure conditions are cooled by isenthalpic expansion.

Physical mechanism

As a gas expands, the average distance between molecules grows. Because of intermolecular attractive forces, expansion causes an increase in the potential energy of the gas. If no external work is extracted in the process ("free expansion") and no heat is transferred, the total energy of the gas remains the same because of the conservation of energy. The increase in potential energy thus means a decrease in kinetic energy and therefore in temperature.

A second mechanism has the opposite effect. During gas molecule collisions, kinetic energy is temporarily converted into potential energy. As the average intermolecular distance increases, there is a drop in the number of collisions per time unit, which causes a decrease in average potential energy. Again, total energy is conserved, so this leads to an increase in kinetic energy (temperature). Below the Joule-Thompson inversion temperature, the former effect (work done internally against intermolecular attractive forces) dominates, and free expansion causes an decrease in temperature. Above the inversion temperature, the

latter effect (increased potential energy associated with collisions) dominates, and free expansion causes a temperature increase.

The derivation which will be used to determine this property is

$$\mu = \left(\frac{\partial T}{\partial p}\right)_h = \frac{1}{Cp}\left[T\left(\frac{\partial v}{\partial T}\right) - v\right]$$

INTERNAL ENERGY DERIVATION

Equation

Internal Energy is calculated by the following equation

$$\int_n^{n+1} du = \int_{T1}^{T2}[C_v]dT + \int_{v1}^{v2}\left[T\left(\frac{\partial P}{\partial T}\right)_v - P\right]dv$$

ENTHALPY

In thermodynamics, the quantity enthalpy, symbolized by H, also called heat content, is the sum of the internal energy of a thermodynamic system plus the energy associated with work done by the system on the atmosphere which is the product of the pressure times the volume. The term enthalpy is composed of the prefix en-, meaning to "put into", plus the Greek suffix -thalpein, meaning "to heat".

Overview

Enthalpy is a quantifiable state function, and the total enthalpy of a system cannot be measured directly; the enthalpy change of a system is measured instead. A possible interpretation of enthalpy is as follows. Imagine we are to create the system out of nothing, then, in addition to supplying the internal energy U for the system, we need to do work to push the atmosphere away in order to make room for the system. Assuming the environment is at some constant pressure P, this mechanical work required is just PV where V is the volume of the system. Therefore, colloquially, enthalpy is the total amount of energy one needs to provide to create the system and then place it in the atmosphere. Conversely, if the system is annihilated, the energy extracted is not just U, but also the work done by the atmosphere as it collapses to fill the space previously occupied by the system, which is PV.

Enthalpy is a thermodynamic potential, and is useful particularly for nearly-constant pressure processes, where any energy input to the system must go into internal energy or the mechanical work of expanding the system. For systems at constant pressure, the change in enthalpy is the heat received by the system plus the non-mechanical work that has been done. In other words, when considering change in enthalpy, one can ignore the compression/expansion mechanical work. Therefore, for a simple system, with a constant number of particles, the difference in enthalpy is the maximum amount of thermal energy derivable from a thermodynamic process in which the pressure is held constant.

Standard enthalpy

The standard enthalpy change of reaction (denoted H° or Ho) is the enthalpy change that occurs in a system when 1 equivalent of matter is transformed by a chemical reaction under standard conditions.

A common standard enthalpy change is the standard enthalpy change of formation, which has been determined for a vast number of substances. The enthalpy change of any reaction under any conditions can be computed, given the standard enthalpy change of formation of all of the reactants and products. Other reactions with standard enthalpy change values include combustion (standard enthalpy change of combustion) and neutralisation (standard enthalpy change of neutralisation).

Specific enthalpy

The specific enthalpy of a working mass is a property of that mass used in thermodynamics, defined as h = u + P * v where u is the specific internal energy, P is the pressure, and v is specific volume. In other words, h = H / m where m is the mass of the system. The SI unit for specific enthalpy is joules/kilogram.

Equation

The enthalpy equation derivative is as follows

$$\int_{n}^{n+1} dh = \int_{T_n}^{T_{n+1}} [C_p] dT + \int_{P_n}^{P_{n+1}} \left[v - T\left(\frac{\partial v}{\partial T}\right)_P \right] dp$$

ENTROPY

In physics and thermodynamics, entropy is a differential term "dQ/T", where dQ is the amount of heat absorbed reversibly by a thermodynamic system at a temperature T. German physicist Rudolf Clausius introduced the mathematical concept of entropy in the early 1860s to account for the dissipation of energy in thermodynamic systems that produce work. He coined the term based on the Greek entrepein meaning "to turn inward". Although the concept of entropy is primarily a thermodynamic construct, it has given rise to ideas in many disparate fields of study, including statistical mechanics, thermal physics, information theory, psychodynamics, economics, and evolution.

Overview

In a system made up of quantities of matter, its pressure differences, density differences, and temperature differences all tend to equalize over time. The system's entropy, which increases with this process, is a measure of how far the equalization has progressed. For example, take a system consisting of a cup of hot water in a cool room. Over time the water will tend to cool and evaporate, and the room will warm up slightly. The system's heat has become more evenly distributed, and thus the entropy of the cup of water and the room has increased.

Entropy is often described as "a measure of the disorder of a thermodynamic system" or "how mixed-up the system is". This has led to a popular misinterpretation of the meaning of entropy because order and disorder are not well-defined terms in science, and their use with entropy does not match the usual use of the words. However, "disorder" can be formally defined in a way that is consistent with the realities of entropy: a system that is more "disordered" or more "mixed up" (on a molecular scale) is equivalently also "a system with a lower amount of energy available to do work" or "a system in a macroscopically more probable state". These definitions, however, are not standard and the word "disorder" can not be used to describe entropy in any clear way.

The entropy of a thermodynamic system can be interpreted in two distinct, but compatible, ways:

From a macroscopic perspective, in classical thermodynamics the entropy is interpreted simply as a state function of a thermodynamic system: that is, a property depending only on the current state of the system, independent of how that state came to be achieved. The state function has the important property that, when multiplied by a reference temperature, it can be understood as a measure of the amount of energy in a physical system that cannot be used to do thermodynamic work; i.e., work mediated by thermal energy.

More precisely, in any process where the system gives up energy ΔE, and its entropy falls by ΔS, a quantity at least TR ΔS of that energy must be given up to the system's surroundings as unusable heat (TR is the temperature of the system's external surroundings). Otherwise the process will not go forward.

From a microscopic perspective, in statistical thermodynamics the entropy is envisioned as a measure of the number of microscopic configurations that are capable of yielding the observed macroscopic description of the thermodynamic system. A more "disordered" or "mixed up" system can thus be formally defined as one which has more microscopic states compatible with the macroscopic description. It can be shown that this definition of entropy, sometimes referred to as Boltzmann's postulate, reproduces all of the properties of the entropy of classical thermodynamics.

An important law of physics, the second law of thermodynamics, states that the total entropy of any isolated thermodynamic system tends to increase over time, approaching a maximum value. Unlike almost all other laws of physics, this associates thermodynamics with a definite arrow of time. However, for a universe of infinite size, which cannot be regarded as an isolated system, the second law does not apply.

The second law

An important law of physics, the second law of thermodynamics, states that the total entropy of any isolated thermodynamic system tends to increase over time, approaching a maximum value; and so, by implication, the entropy of the universe (i.e. the system and its surroundings), assumed as an isolated system, tends to increase. We will consider the meaning of the "second law" further in a subsequent section. Two important consequences are that heat cannot of itself pass from a colder to a hotter body: i.e., it is impossible to transfer heat from a cold to a hot reservoir without at the same time converting a certain amount of work to heat. It is also impossible for any device that operates on a cycle to receive heat from a single reservoir and produce a net amount of work; it can only get useful work out of the heat if heat is at the same time transferred from a hot to a cold reservoir. This means that there is no possibility of a 'perpetuum mobile' which is isolated. Also, from this it follows, that a reduction in the increase of entropy in a specified process, such as a chemical reaction, means that it is energetically more efficient.

Entropy and cosmology

We have previously mentioned that a finite universe may be considered an isolated system. As such, it may be subject to the Second Law of Thermodynamics, so that its total entropy is constantly increasing. It has been speculated that the universe is fated to a heat death in which all the energy ends up as a homogeneous distribution of thermal energy, so that no more work can be extracted from any source.

If the universe can be considered to have increasing entropy, then, as Roger Penrose has pointed out, an important role in the disordering process is played by gravity, which causes dispersed matter to accumulate into stars, which collapse eventually into black holes. Jacob Bekenstein and Stephen Hawking have shown that black holes have the maximum possible entropy of any object of equal size. This makes them likely end points of all entropy-increasing processes, if they are totally effective matter and energy traps. Hawking has, however, recently changed his stance on this aspect.

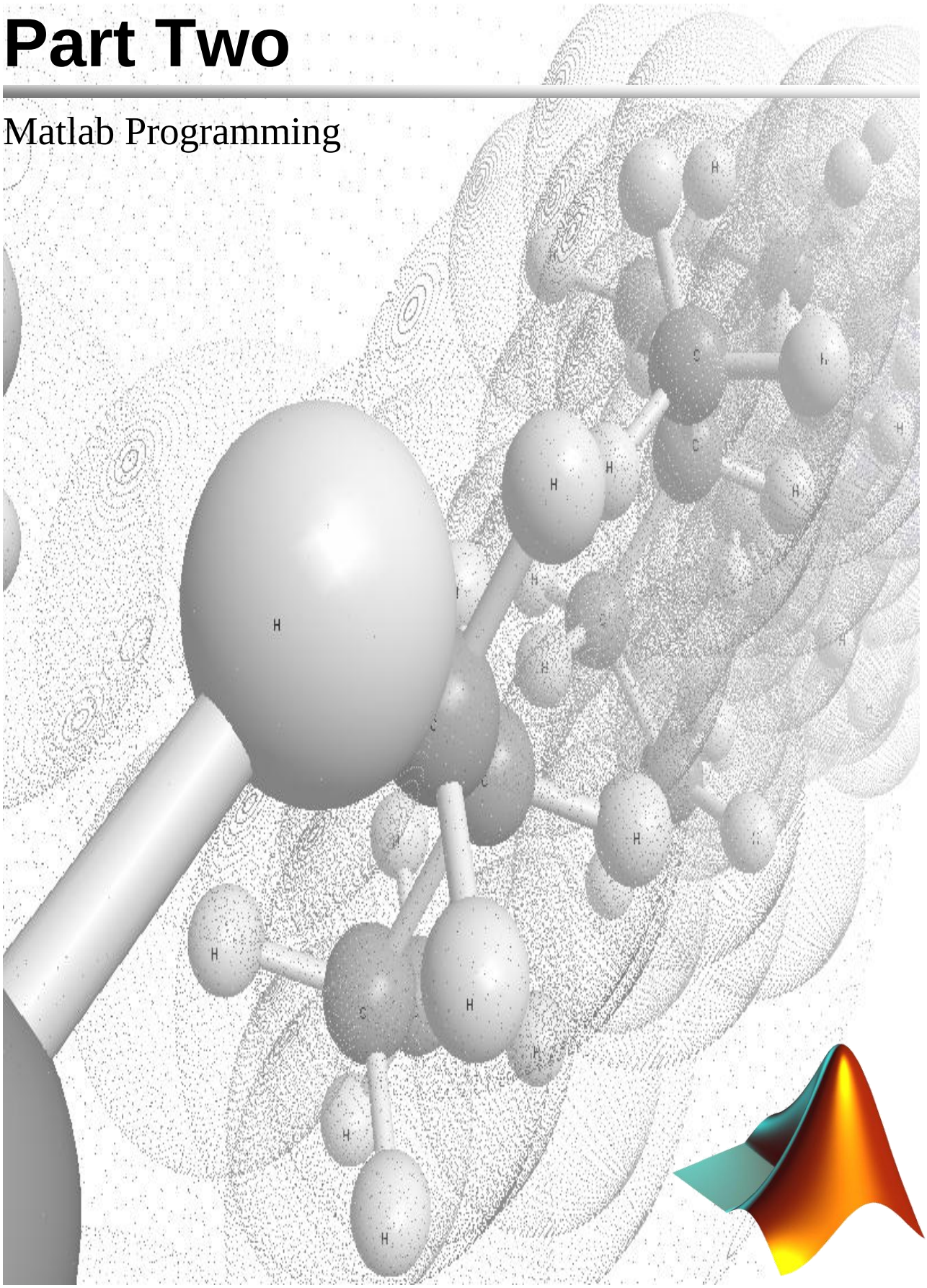
The role of entropy in cosmology remains a controversial subject. Recent work has cast extensive doubt on the heat death hypothesis and the applicability of any simple thermodynamic model to the universe in general. Although entropy does increase in the model of an expanding universe, the maximum possible entropy rises much more rapidly and leads to an "entropy gap," thus pushing the system further away from equilibrium with each time increment. Complicating factors, such as the energy density of the vacuum and macroscopic quantum effects, are difficult to reconcile with thermodynamical models, making any predictions of large-scale thermodynamics extremely difficult.

Equation

$$\int_{n}^{n+1} ds = \int_{T_n}^{T_{n+1}} \left[\frac{C_p}{T} \right] dt + \int_{P_n}^{P_{n+1}} \left[\left(\frac{\partial v}{\partial T} \right)_P \right] dp$$

Part Two

Matlab Programming



C h a p t e r 5

Variables are Matrices

One of the most important things to master any programming language is to know your variables data types that are used in the language. In old good language like Basic there were only two types of variables one is of numeric type and string type, as long as you investigate further in some higher languages like C, and C++ the variables data types are extended in detailed way such integer types, float, double, unsigned, and signed types. Beside the famous C data Type "char".

MATRICES

Here in matlab the situation is going completely different if you have a background programming skills you know for sure that the variable is intended to hold one value such as if you wrote

x = 100;

The *x* variable is storing the value 100 but in matlab this assumption is a wrong assumption which leads the first learners to make errors easily.

So what is the correct thinking about *x* variable? The fact is that x is a matrix which its 1 x 1 element.

MATLAB® is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include Math and computation Algorithm development Data acquisition Modeling, simulation, and prototyping Data analysis, exploration, and visualization Scientific and engineering graphics Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

The fact that matlab is using matrices as its only way to represent variables is the mean reason which makes scientists and mathematicians use it with ease.

ENTERING MATRICES

The best way for you to get started with MATLAB is to learn how to handle matrices. Start MATLAB and follow along with each example.

You can enter matrices into MATLAB in several different ways: Enter an explicit list of elements. Load matrices from external data files. Generate matrices using built-in functions. Create matrices with your own functions in M-files.

Start by entering Dürer's matrix as a list of its elements. You only have to follow a few basic conventions: Separate the elements of a row with blanks or commas. Use a semicolon, ; , to indicate the end of each row. Surround the entire list of elements with square brackets, [ ].

To enter Dürer's matrix, simply type in the Command Window

A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]

MATLAB displays the matrix you just entered:

```
A =
   16    3    2   13
    5   10   11    8
    9    6    7   12
    4   15   14    1
```

This matrix matches the numbers in the engraving. Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A. Now that you have A in the workspace, take a look at what makes it so interesting. Why is it magic?

SUM, TRANSPOSE, AND DIAG

You are probably already aware that the special properties of a magic square have to do with the various ways of summing its elements. If you take the sum along any row or column, or along either of the two main diagonals, you will always get the same number. Let us verify that using MATLAB. The first statement to try is

```
sum(A)
```

MATLAB replies with

```
ans =
   34   34   34   34
```

When you do not specify an output variable, MATLAB uses the variable ans, short for answer, to store the results of a calculation. You have computed a row vector containing the sums of the columns of A. Sure enough, each of the columns has the same sum, the magic sum, 34.

How about the row sums? MATLAB has a preference for working with the columns of a matrix, so the easiest way to get the row sums is to transpose the matrix, compute the column sums of the transpose, and then transpose the result. The transpose operation is denoted by an apostrophe or single quote, '. It flips a matrix about its main diagonal and it turns a row vector into a column vector.

So

```
A'
```

produces

```
ans =
   16    5    9    4
    3   10    6   15
    2   11    7   14
   13    8   12    1
```

and

```
sum(A')'
```

produces a column vector containing the row sums

```
ans =
   34
   34
   34
```

```
   34
```

The sum of the elements on the main diagonal is obtained with the sum and the diag functions:

```
diag(A)
```

produces

```
ans =
   16
   10
    7
    1
```

and

```
sum(diag(A))
```

produces

```
ans =
   34
```

The other diagonal, the so-called antidiagonal, is not so important mathematically, so MATLAB does not have a ready-made function for it. But a function originally intended for use in graphics, fliplr, flips a matrix from left to right:

```
sum(diag(fliplr(A)))
```

```
ans =
   34
```

You have verified that the matrix in Dürer's engraving is indeed a magic square and, in the process, have sampled a few MATLAB matrix operations. The following sections continue to use this matrix to illustrate additional MATLAB capabilities.

SUBSCRIPTS

The element in row i and column j of A is denoted by A(i,j). For example, A(4,2) is the number in the fourth row and second column. For our magic square, A(4,2) is 15. So to compute the sum of the elements in the fourth column of A, type

```
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

This produces

```
ans =
   34
```

but is not the most elegant way of summing a single column.

It is also possible to refer to the elements of a matrix with a single subscript, A(k). This is the usual way of referencing row and column vectors. But it can also apply to a fully two-dimensional matrix, in which case the array is regarded as one long column vector formed from the columns of the original matrix. So, for our magic square, A(8) is another way of referring to the value 15 stored in A(4,2).

If you try to use the value of an element outside of the matrix, it is an error:

```
t = A(4,5)
```

Index exceeds matrix dimensions.

On the other hand, if you store a value in an element outside of the matrix, the size increases to accommodate the newcomer:

```
X = A;

X(4,5) = 17

X =

    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4    15    14     1    17
```

THE COLON OPERATOR

The colon, :, is one of the most important MATLAB operators. It occurs in several different forms. The expression

```
1:10
```

is a row vector containing the integers from 1 to 10:

```
1     2     3     4     5     6     7     8     9    10
```

To obtain nonunit spacing, specify an increment. For example,

```
100:-7:50
```

is

```
100    93    86    79    72    65    58    51
```

and

```
0:pi/4:pi
```

is

```
0    0.7854    1.5708    2.3562    3.1416
```

Subscript expressions involving colons refer to portions of a matrix:

```
A(1:k,j)
```

is the first k elements of the jth column of A. So

```
sum(A(1:4,4))
```

computes the sum of the fourth column. But there is a better way. The colon by itself refers to all the elements in a row or column of a matrix and the keyword end refers to the last row or column. So

```
sum(A(:,end))
```

computes the sum of the elements in the last column of A:

```
ans =

    34
```

Why is the magic sum for a 4-by-4 square equal to 34? If the integers from 1 to 16 are sorted into four groups with equal sums, that sum must be

```
sum(1:16)/4
```

which, of course, is

```
ans =

    34
```

MORE ABOUT MATRICES AND ARRAYS

Linear Algebra

Informally, the terms matrix and array are often used interchangeably. More precisely, a matrix is a two-dimensional numeric array that represents a linear transformation. The mathematical operations defined on matrices are the subject of linear algebra.

Dürer's magic square

```
A = [16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1 ]
```

provides several examples that give a taste of MATLAB matrix operations. You have already seen the matrix transpose, A'. Adding a matrix to its transpose produces a symmetric matrix:

```
A + A'

ans =

    32     8    11    17
     8    20    17    23
    11    17    14    26
    17    23    26     2
```

The multiplication symbol, *, denotes the matrix multiplication involving inner products between rows and columns. Multiplying the transpose of a matrix by the original matrix also produces a symmetric matrix:

```
A'*A


ans =

   378   212   206   360
   212   370   368   206
   206   368   370   212
   360   206   212   378
```

The determinant of this particular matrix happens to be zero, indicating that the matrix is singular:

```
d = det(A)


d =

     0
```

The reduced row echelon form of A is not the identity:

```
R = rref(A)


R =

     1     0     0     1
     0     1     0    -3
     0     0     1     3
     0     0     0     0
```

Since the matrix is singular, it does not have an inverse. If you try to compute the inverse with

```
X = inv(A)
```

you will get a warning message:

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 9.796086e-018.
```

Roundoff error has prevented the matrix inversion algorithm from detecting exact singularity. But the value of rcond, which stands for reciprocal condition estimate, is on the order of eps, the floating-point relative precision, so the computed inverse is unlikely to be of much use.

The eigenvalues of the magic square are interesting:

```
e = eig(A)

e =

   34.0000
    8.0000
    0.0000
   -8.0000
```

One of the eigenvalues is zero, which is another consequence of singularity. The largest eigenvalue is 34, the magic sum. That is because the vector of all ones is an eigenvector:

```
v = ones(4,1)

v =

     1
     1
     1
     1

A*v

ans =

    34
    34
    34
    34
```

Arrays

When they are taken away from the world of linear algebra, matrices become two-dimensional numeric arrays. Arithmetic operations on arrays are done element by element. This means that addition and subtraction are the same for arrays and matrices, but that multiplicative operations are different. MATLAB uses a dot, or decimal point, as part of the notation for multiplicative array operations.

The list of operators includes

.+ Addition .- Subtraction .* Element-by-element multiplication./Element-by-element division.\Element-by-element left division.^Element-by-element power.' Unconjugated array transpose

If the Dürer magic square is multiplied by itself with array multiplication

```
A.*A
```

the result is an array containing the squares of the integers from 1 to 16, in an unusual order:

```
ans =
  256    9    4   169
   25  100  121    64
   81   36   49   144
   16  225  196     1
```

Building Tables

Array operations are useful for building tables. Suppose n is the column vector

```
n = (0:9)';
```

Then

```
pows = [n  n.^2  2.^n]
```

builds a table of squares and powers of 2:

```
pows =
    0    0    1
    1    1    2
    2    4    4
    3    9    8
    4   16   16
    5   25   32
    6   36   64
    7   49  128
    8   64  256
    9   81  512
```

The elementary math functions operate on arrays element by element. So

```
format short g
x = (1:0.1:2)';
logs = [x log10(x)]
```

builds a table of logarithms.

```
 logs =
    1.0         0
    1.1    0.04139
    1.2    0.07918
    1.3    0.11394
    1.4    0.14613
    1.5    0.17609
    1.6    0.20412
    1.7    0.23045
    1.8    0.25527
    1.9    0.27875
    2.0    0.30103
```

C h a p t e r 6

Matlab Programming

FLOW CONTROL

if, else, and elseif

The if statement evaluates a logical expression and executes a group of statements when the expression is true. The optional elseif and else keywords provide for the execution of alternate groups of statements. An end keyword, which matches the if, terminates the last group of statements. The groups of statements are delineated by the four keywords -- no braces or brackets are involved.

The MATLAB algorithm for generating a magic square of order n involves three different cases: when n is odd, when n is even but not divisible by 4, or when n is divisible by 4. This is described by

```
if rem(n,2) ~= 0
   M = odd_magic(n)
elseif rem(n,4) ~= 0
   M = single_even_magic(n)
else
   M = double_even_magic(n)
end
```

In this example, the three cases are mutually exclusive, but if they weren't, the first true condition would be executed.

It is important to understand how relational operators and if statements work with matrices. When you want to check for equality between two variables, you might use

```
if A == B, ...
```

This is valid MATLAB code, and does what you expect when A and B are scalars. But when A and B are matrices, A == B does not test if they are equal, it tests where they are equal; the result is another matrix of 0's and 1's showing element-by-element equality. (In fact, if A and B are not the same size, then A == B is an error.)

```
A = magic(4);     B = A;     B(1,1) = 0;
A == B
ans =
    0    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1
```

The proper way to check for equality between two variables is to use the isequal function,

```
if isequal(A, B), ...
```

isequal returns a scalar logical value of 1 (representing true) or 0 (false), instead of a matrix, as the expression to be evaluated by the if function. Using the A and B matrices from above, you get

```
isequal(A, B)
ans =
```

```
   0
```

Here is another example to emphasize this point. If A and B are scalars, the following program will never reach the "unexpected situation". But for most pairs of matrices, including our magic squares with interchanged columns, none of the matrix conditions A > B, A < B, or A == B is true for all elements and so the else clause is executed.

```
if A > B
  'greater'
elseif A < B
  'less'
elseif A == B
  'equal'
else
  error('Unexpected situation')
end
```

Several functions are helpful for reducing the results of matrix comparisons to scalar conditions for use with if, including

isequal

isempty

all

any

switch and case

The switch statement executes groups of statements based on the value of a variable or expression. The keywords case and otherwise delineate the groups. Only the first matching case is executed. There must always be an end to match the switch.

The logic of the magic squares algorithm can also be described by

```
 switch (rem(n,4)==0) + (rem(n,2)==0)
   case 0
     M = odd_magic(n)
   case 1
     M = single_even_magic(n)
   case 2
     M = double_even_magic(n)
   otherwise
     error('This is impossible')
 end
```

for

The for loop repeats a group of statements a fixed, predetermined number of times. A matching end delineates the statements.

```
for n = 3:32
```

```
  r(n) = rank(magic(n));
end
r
```

The semicolon terminating the inner statement suppresses repeated printing, and the r after the loop displays the final result.

It is a good idea to indent the loops for readability, especially when they are nested.

```
for i = 1:m
  for j = 1:n
    H(i,j) = 1/(i+j);
  end
end
```

while

The while loop repeats a group of statements an indefinite number of times under control of a logical condition. A matching end delineates the statements.

Here is a complete program, illustrating while, if, else, and end, that uses interval bisection to find a zero of a polynomial.

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
  x = (a+b)/2;
  fx = x^3-2*x-5;
  if sign(fx) == sign(fa)
    a = x; fa = fx;
  else
    b = x; fb = fx;
  end
end
x
```

The result is a root of the polynomial x3 - 2x - 5, namely

```
x =
  2.09455148154233
```

The cautions involving matrix comparisons that are discussed in the section on the if statement also apply to the while statement

continue

The continue statement passes control to the next iteration of the for loop or while loop in which it appears, skipping any remaining statements in the body of the loop. In nested loops, continue passes control to the next iteration of the for loop or while loop enclosing it.

The example below shows a continue loop that counts the lines of code in the file magic.m, skipping all blank lines and comments. A continue statement is used to advance to the next line in magic.m without incrementing the count whenever a blank line or comment line is encountered.

```
fid = fopen('magic.m','r');
count = 0;
while ~feof(fid)
   line = fgetl(fid);
   if isempty(line) | strncmp(line,'%',1)
      continue
   end
   count = count + 1;
end
disp(sprintf('%d lines',count));
```

break

The break statement lets you exit early from a for loop or while loop. In nested loops, break exits from the innermost loop only.

Here is an improvement on the example from the previous section. Why is this use of break a good idea?

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
  x = (a+b)/2;
  fx = x^3-2*x-5;
  if fx == 0
    break
  elseif sign(fx) == sign(fa)
    a = x; fa = fx;
  else
    b = x; fb = fx;
  end
end
x
```

try - catch

The general form of a try-catch statement sequence is

```
try
  statement
  ...
  statement
```

```
catch
  statement
  ...
  statement
end
```

In this sequence the statements between try and catch are executed until an error occurs. The statements between catch and end are then executed. Use lasterr to see the cause of the error. If an error occurs between catch and end, MATLAB terminates execution unless another try-catch sequence has been established.

This statement is used to handle the errors of the program like division by 0.

return

return terminates the current sequence of commands and returns control to the invoking function or to the keyboard. return is also used to terminate keyboard mode. A called function normally transfers control to the function that invoked it when it reaches the end of the function. You can insert a return statement within the called function to force an early termination and to transfer control to the invoking function.

OTHER DATA STRUCTURES

Multidimensional Arrays

Multidimensional arrays in MATLAB are arrays with more than two subscripts. One way of creating a multidimensional array is by calling zeros, ones, rand, or randn with more than two arguments. For example,

```
R = randn(3,4,5);
```

creates a 3-by-4-by-5 array with a total of 3x4x5 = 60 normally distributed random elements.

A three-dimensional array might represent three-dimensional physical data, say the temperature in a room, sampled on a rectangular grid. Or it might represent a sequence of matrices, A(k), or samples of a time-dependent matrix, A(t). In these latter cases, the (i, j)th element of the kth matrix, or the tkth matrix, is denoted by A(i,j,k).

MATLAB and Dürer's versions of the magic square of order 4 differ by an interchange of two columns. Many different magic squares can be generated by interchanging columns. The statement

```
p = perms(1:4);
```

generates the 4! = 24 permutations of 1:4. The kth permutation is the row vector p(k,:). Then

```
A = magic(4);
M = zeros(4,4,24);
for k = 1:24
  M(:,:,k) = A(:,p(k,:));
end
```

stores the sequence of 24 magic squares in a three-dimensional array, M. The size of M is

```
size(M)
ans =
   4    4   24
```

The statement

sum(M,d)

computes sums by varying the dth subscript. So

sum(M,1)

is a 1-by-4-by-24 array containing 24 copies of the row vector

34 34 34 34

and

sum(M,2)

is a 4-by-1-by-24 array containing 24 copies of the column vector

34

34

34

34

Finally,

S = sum(M,3)

adds the 24 matrices in the sequence. The result has size 4-by-4-by-1, so it looks like a 4-by-4 array.

S =

204 204 204 204

204 204 204 204

204 204 204 204

204 204 204 204

Cell Arrays

Cell arrays in MATLAB are multidimensional arrays whose elements are copies of other arrays. A cell array of empty matrices can be created with the cell function. But, more often, cell arrays are created by enclosing a miscellaneous collection of things in curly braces, {}. The curly braces are also used with subscripts to access the contents of various cells. For example,

C = {A sum(A) prod(prod(A))}

produces a 1-by-3 cell array. The three cells contain the magic square, the row vector of column sums, and the product of all its elements. When C is displayed, you see

C =

[4x4 double] [1x4 double] [20922789888000]

This is because the first two cells are too large to print in this limited space, but the third cell contains only a single number, 16!, so there is room to print it.

Here are two important points to remember. First, to retrieve the contents of one of the cells, use subscripts in curly braces. For example, C{1} retrieves the magic square and C{3} is 16. Second, cell arrays contain copies of other arrays, not pointers to those arrays. If you subsequently change A, nothing happens to C.

You can use three-dimensional arrays to store a sequence of matrices of the same size. Cell arrays can be used to store a sequence of matrices of different sizes. For example,

```
M = cell(8,1);
for n = 1:8
  M{n} = magic(n);
end
M
```

produces a sequence of magic squares of different order.

```
M =
   [     1]
   [ 2x2  double]
   [ 3x3  double]
   [ 4x4  double]
   [ 5x5  double]
   [ 6x6  double]
   [ 7x7  double]
   [ 8x8  double]
```

You can retrieve the 4-by-4 magic square matrix with

```
M{4}
```

Characters and Text

Enter text into MATLAB using single quotes. For example,

```
s = 'Hello'
```

The result is not the same kind of numeric matrix or array you have been dealing with up to now. It is a 1-by-5 character array.

Internally, the characters are stored as numbers, but not in floating-point format. The statement

```
a = double(s)
```

converts the character array to a numeric matrix containing floating-point representations of the ASCII codes for each character. The result is

```
a =
   72   101   108   108   111
```

The statement

```
s = char(a)
```

reverses the conversion.

Converting numbers to characters makes it possible to investigate the various fonts available on your computer. The printable characters in the basic ASCII character set are represented by the integers 32:127. (The integers less than 32 represent nonprintable control characters.) These integers are arranged in an appropriate 6-by-16 array with

```
F = reshape(32:127,16,6)';
```

The printable characters in the extended ASCII character set are represented by F+128. When these integers are interpreted as characters, the result depends on the font currently being used. Type the statements

char(F)

char(F+128)

and then vary the font being used for the Command Window. Select Preferences from the File menu to change the font. If you include tabs in lines of code, use a fixed-width font, such as Monospaced, to align the tab positions on different lines.

Concatenation with square brackets joins text variables together into larger strings. The statement

h = [s, ' world']

joins the strings horizontally and produces

h =

  Hello world

The statement

v = [s; 'world']

joins the strings vertically and produces

v =

  Hello

  world

Note that a blank has to be inserted before the 'w' in h and that both words in v have to have the same length. The resulting arrays are both character arrays; h is 1-by-11 and v is 2-by-5.

To manipulate a body of text containing lines of different lengths, you have two choices -- a padded character array or a cell array of strings. When creating a character array, you must make each row of the array the same length. (Pad the ends of the shorter rows with spaces.) The char function does this padding for you. For example,

S = char('A','rolling','stone','gathers','momentum.')

produces a 5-by-9 character array.

S =

A

rolling

stone

gathers

momentum.

Alternatively, you can store the text in a cell array. For example,

C = {'A';'rolling';'stone';'gathers';'momentum.'}

creates a 5-by-1 cell array that requires no padding because each row of the array can have a different length.

C =

  'A'

  'rolling'

  'stone'

  'gathers'

'momentum.'

You can convert a padded character array to a cell array of strings with

C = cellstr(S)

and reverse the process with

S = char(C)

Structures

Structures are multidimensional MATLAB arrays with elements accessed by textual field designators. For example,

S.name = 'Ed Plum';

S.score = 83;

S.grade = 'B+'

creates a scalar structure with three fields.

S =

name: 'Ed Plum'

score: 83

grade: 'B+'

Like everything else in MATLAB, structures are arrays, so you can insert additional elements. In this case, each element of the array is a structure with several fields. The fields can be added one at a time,

S(2).name = 'Toni Miller';

S(2).score = 91;

S(2).grade = 'A-';

or an entire element can be added with a single statement.

S(3) = struct('name','Jerry Garcia',...

'score',70,'grade','C')

Now the structure is large enough that only a summary is printed.

S =

1x3 struct array with fields:

name

score

grade

There are several ways to reassemble the various fields into other MATLAB arrays. They are all based on the notation of a comma-separated list. If you type

S.score

it is the same as typing

S(1).score, S(2).score, S(3).score

This is a comma-separated list. Without any other punctuation, it is not very useful. It assigns the three scores, one at a time, to the default variable ans and dutifully prints out the result of each assignment. But when you enclose the expression in square brackets,

[S.score]

it is the same as

[S(1).score, S(2).score, S(3).score]

which produces a numeric row vector containing all the scores.

```
ans =
   83    91    70
```

Similarly, typing

S.name

just assigns the names, one at a time, to ans. But enclosing the expression in curly braces,

{S.name}

creates a 1-by-3 cell array containing the three names.

```
ans =
   'Ed Plum'    'Toni Miller'    'Jerry Garcia'
```

And

char(S.name)

calls the char function with three arguments to create a character array from the name fields,

```
ans =
Ed Plum
Toni Miller
Jerry Garcia
```

Dynamic Field Names

The most common way to access the data in a structure is by specifying the name of the field that you want to reference. Another means of accessing structure data is to use dynamic field names. These names express the field as a variable expression that MATLAB evaluates at run-time. The dot-parentheses syntax shown here makes expression a dynamic field name:

structName.(expression)

Index into this field using the standard MATLAB indexing syntax. For example, to evaluate expression into a field name and obtain the values of that field at columns 1 through 25 of row 7, use

structName.(expression)(7,1:25)

Dynamic Field Names Example. The avgscore function shown below computes an average test score, retrieving information from the testscores structure using dynamic field names:

```
function avg = avgscore(testscores, student, first, last)
for k = first:last
  scores(k) = testscores.(student).week(k);
```

```
end
avg = sum(scores)/(last - first + 1);
```

You can run this function using different values for the dynamic field student:

```
avgscore(testscores, 'Ann Lane', 1, 20)
ans =
   83.5000
avgscore(testscores, 'William King', 1, 20)
ans =
   92.1000
```

FUNCTIONS

Functions are M-files that can accept input arguments and return output arguments. The names of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

A good example is provided by rank. The M-file rank.m is available in the directory

toolbox/matlab/matfun

You can see the file with

type rank

Here is the file.

```
function r = rank(A,tol)
%   RANK Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
%   RANK(A,tol) is the number of singular values of A
%   that are larger than tol.
%   RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

s = svd(A);
if nargin==1
   tol = max(size(A)') * max(s) * eps;
end
r = sum(s > tol);
```

The first line of a function M-file starts with the keyword function. It gives the function name and order of arguments. In this case, there are up to two input arguments and one output argument.

The next several lines, up to the first blank or executable line, are comment lines that provide the help text. These lines are printed when you type

help rank

The first line of the help text is the H1 line, which MATLAB displays when you use the lookfor command or request help on a directory.

The rest of the file is the executable MATLAB code defining the function. The variable s introduced in the body of the function, as well as the variables on the first line, r, A and tol, are all local to the function; they are separate from any variables in the MATLAB workspace.

This example illustrates one aspect of MATLAB functions that is not ordinarily found in other programming languages -- a variable number of arguments. The rank function can be used in several different ways.

rank(A)

r = rank(A)

r = rank(A,1.e-6)

Many M-files work this way. If no output argument is supplied, the result is stored in ans. If the second input argument is not supplied, the function computes a default value. Within the body of the function, two quantities named nargin and nargout are available which tell you the number of input and output arguments involved in each particular use of the function. The rank function uses nargin, but does not need to use nargout.

TYPES OF FUNCTIONS

MATLAB offers several different types of functions to use in your programming.

Anonymous Functions

An anonymous function is a simple form of MATLAB function that does not require an M-file. It consists of a single MATLAB expression and any number of input and output arguments. You can define an anonymous function right at the MATLAB command line, or within an M-file function or script. This gives you a quick means of creating simple functions without having to create M-files each time.

The syntax for creating an anonymous function from an expression is

f = @(arglist)expression

The statement below creates an anonymous function that finds the square of a number. When you call this function, MATLAB assigns the value you pass in to variable x, and then uses x in the equation x.^2:

sqr = @(x) x.^2;

To execute the sqr function defined above, type

a = sqr(5)

a = 25

Primary and Subfunctions

All functions that are not anonymous must be defined within an M-file. Each M-file has a required primary function that appears first in the file, and any number of subfunctions that follow the primary. Primary functions have a wider scope than subfunctions. That is, primary functions can be invoked from outside of

their M-file (from the MATLAB command line or from functions in other M-files) while subfunctions cannot. Subfunctions are visible only to the primary function and other subfunctions within their own M-file.

The rank function shown in the section on Functions is an example of a primary function.

Private Functions

A private function is a type of primary M-file function. Its unique characteristic is that it is visible only to a limited group of other functions. This type of function can be useful if you want to limit access to a function, or when you choose not to expose the implementation of a function.

Private functions reside in subdirectories with the special name private. They are visible only to functions in the parent directory. For example, assume the directory newmath is on the MATLAB search path. A subdirectory of newmath called private can contain functions that only the functions in newmath can call.

Because private functions are invisible outside the parent directory, they can use the same names as functions in other directories. This is useful if you want to create your own version of a particular function while retaining the original in another directory. Because MATLAB looks for private functions before standard M-file functions, it will find a private function named test.m before a nonprivate M-file named test.m.

Nested Functions

You can define functions within the body of any MATLAB M-file function. These are said to be nested within the outer function. A nested function contains any or all of the components of any other M-file function. In this example, function B is nested in function A:

```
function x = A(p1, p2)
...
B(p2)
  function y = B(p3)
  ...
  end
...
end
```

Like other functions, a nested function has its own workspace where variables used by the function are stored. But it also has access to the workspaces of all functions in which it is nested. So, for example, a variable that has a value assigned to it by the primary function can be read or overwritten by a function nested at any level within the primary. Similarly, a variable that is assigned in a nested function can be read or overwritten by any of the functions containing that function.

Function Overloading

Overloaded functions act the same way as overloaded functions in most computer languages. Overloaded functions are useful when you need to create a function that responds to different types of inputs accordingly. For instance, you might want one of your functions to accept both double-precision and integer input, but to handle each type somewhat differently. You can make this difference invisible to the user by creating two separate functions having the same name, and designating one to handle double types and one to handle integers. When you call the function, MATLAB chooses which M-file to dispatch to based on the type of the input arguments.

FUNCTION HANDLES

You can create a handle to any MATLAB function and then use that handle as a means of referencing the function. A function handle is typically passed in an argument list to other functions, which can then execute, or evaluate, the function using the handle.

Construct a function handle in MATLAB using the at sign, @, before the function name. The following example creates a function handle for the sin function and assigns it to the variable fhandle.

```
fhandle = @sin;
```

You can call a function by means of its handle in the same way that you would call the function using its name. The syntax is

```
fhandle(arg1, arg2, ...);
```

The function plot_fhandle, shown below, receives a function handle and data, generates y-axis data using the function handle, and plots it:

```
function x = plot_fhandle(fhandle, data)

plot(data, fhandle(data))
```

When you call plot_fhandle with a handle to the sin function and the argument shown below, the resulting evaluation produces a sine wave plot.

```
plot_fhandle(@sin, -pi:0.01:pi)
```

C h a p t e r 7

Object Oriented Programming

(OOP) or Object oriented programming is the most modern technique used in programming inspired by the idea that the programmer shouldn't invent the wheel every time he make a program. Otherwise he build his program as blocks of code which can interact and dealing with each other independently the idea as I believe is the same idea based on the thermodynamics System.

As you can found in System properties and processes, you can found in OOP Class attributes and methods.

You can represent many thermodynamic systems in classes such as Internal Combustion Engines which have properties of RPM, Torque, Bore, Length, ..., and All Power related stuff.

By dividing your program into classes you get the maximum efficiency from your code by reusing it many times as you want.

The mythology used in writing this research code is the Object Oriented Programming.

In the next pages we will describe OOP and how matlab is implementing it in its workspace.

CLASSES AND OBJECTS: AN OVERVIEW

You can view classes as new data types having specific behaviors defined for the class. For example, a polynomial class might redefine the addition operator (+) so that it correctly performs the operation of addition on polynomials. Operations defined to work with objects of a particular class are known as methods of that class.

You can also view classes as new items that you can treat as single entities. An example is an arrow object that MATLAB can display on graphs (perhaps composed of MATLAB line and patch objects) and that has properties like a Handle Graphics object. You can create an arrow simply by instantiating the arrow class.

You can add classes to your MATLAB environment by specifying a MATLAB structure that provides data storage for the object and creating a class directory containing M-files that operate on the object. These M-files contain the methods for the class. The class directory can also include functions that define the way various MATLAB operators, including arithmetic operations, subscript referencing, and concatenation, apply to the objects. Redefining how a built-in operator works for your class is known as overloading the operator.

FEATURES OF OBJECT-ORIENTED PROGRAMMING

When using well-designed classes, object-oriented programming can significantly increase code reuse and make your programs easier to maintain and extend. Programming with classes and objects differs from ordinary structured programming in these important ways:

- Function and operator overloading. You can create methods that override existing MATLAB functions. When you call a function with a user-defined object as an argument, MATLAB first checks to see if there is a method defined for the object's class. If there is, MATLAB calls it, rather than the normal MATLAB function.
- Encapsulation of data and methods. Object properties are not visible from the command line; you can access them only with class methods. This protects the object properties from operations that are not intended for the object's class.
- Inheritance. You can create class hierarchies of parent and child classes in which the child class inherits data fields and methods from the parent. A child class can inherit from one parent (single inheritance) or many parents (multiple inheritance). Inheritance can span one or more generations. Inheritance enables sharing common parent functions and enforcing common behavior amongst all child classes.
- Aggregation. You can create classes using aggregation, in which an object contains other objects. This is appropriate when an object type is part of another object type. For example, a savings account object might be a part of a financial portfolio object.

CREATING OBJECTS

You create an object by calling the class constructor and passing it the appropriate input arguments. In MATLAB, constructors have the same name as the class name. For example, the statement,

```
p = polynom([1 0 -2 -5]);
```

creates an object named p belonging to the class polynom. Once you have created a polynom object, you can operate on the object using methods that are defined for the polynom class. See Example -- A Polynomial Class for a description of the polynom class.

INVOKING METHODS ON OBJECTS

Class methods are M-file functions that take an object as one of the input arguments. The methods for a specific class must be placed in the class directory for that class (the @classname directory). This is the first place that MATLAB looks to find a class method.

The syntax for invoking a method on an object is similar to a function call. Generally, it looks like

[out1,out2,...] = methodName(object,arg1,arg2, ...);

For example, suppose a user-defined class called polynom has a char method defined for the class. This method converts a polynom object to a character string and returns the string. This statement calls the char method on the polynom object p.

s = char(p);

Using the class function, you can confirm that the returned value s is a character string.

class(s)

ans =

   char

s

s =

   x^3-2*x-5

You can use the methods command to produce a list of all of the methods that are defined for a class.

PRIVATE METHODS

Private methods can be called only by other methods of their class. You define private methods by placing the associated M-files in a private subdirectory of the @classname directory. In the example,

@classname/private/updateObj.m

the method updateObj has scope only within the classname class. This means that updateObj can be called by any method that is defined in the @classname directory, but it cannot be called from the MATLAB command line or by methods outside of the class directory, including parent methods.

Private methods and private functions differ in that private methods (in fact all methods) have an object as one of their input arguments and private functions do not. You can use private functions as helper functions, such as described in the next section.

HELPER FUNCTIONS

In designing a class, you may discover the need for functions that perform support tasks for the class, but do not directly operate on an object. These functions are called helper functions. A helper function can be a subfunction in a class method file or a private function. When determining which version of a particular function to call, MATLAB looks for these functions in the order listed above. For more information about the order in which MATLAB calls functions and methods, see How MATLAB Determines Which Method to Call.

SETTING UP CLASS DIRECTORIES

The M-files defining the methods for a class are collected together in a directory referred to as the class directory. The directory name is formed with the class name preceded by the character @. For example, one of the examples used in this chapter is a class involving polynomials in a single variable. The name of

the class, and the name of the class constructor, is polynom. The M-files defining a polynomial class would be located in directory with the name @polynom.

The class directories are subdirectories of directories on the MATLAB search path, but are not themselves on the path. For instance, the new @polynom directory could be a subdirectory of the MATLAB working directory or your own personal directory that has been added to the search path.

ADDING THE CLASS DIRECTORY TO THE MATLAB PATH

After creating the class directory, you need to update the MATLAB path so that MATLAB can locate the class source files. The class directory should not be directly on the MATLAB path. Instead, you should add the parent directory to the MATLAB path. For example, if the @polynom class directory is located at

c:\myClasses\@polynom

you add the class directory to the MATLAB path with the addpath command

addpath c:\myClasses;

USING MULTIPLE CLASS DIRECTORIES

A MATLAB class can access methods in multiple @classname directories if all such directories are visible to MATLAB (i.e., the parent directories are on the MATLAB path or in the current directory). When you attempt to use a method of the class, MATLAB searches all the visible directories named @classname for the appropriate method.

For more information, see How MATLAB Determines Which Method to Call.

DATA STRUCTURE

One of the first steps in the design of a new class is the choice of the data structure to be used by the class. Objects are stored in MATLAB structures. The fields of the structure, and the details of operations on the fields, are visible only within the methods for the class. The design of the appropriate data structure can affect the performance of the code.

CLASS SAMPLE FROM THE PROJECT CODE

@Gas Class

The @Gass class is responsible of storing pure gas information which will be used later in the program.

Every class should implement known functions to be used as an object after instantiating it, the function should reside in the class directory, you should notice the directory which not start with '@' is not recognized as a class but as a regular folder.

@Gas\Gas.m

```
function ng = Gas(Name,MW)
% Gas Class
% for Calculating numerous heat capacities
% actually by two method
% polynomial method
% method mentioned in Jaeschke and Schley paper.
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com
```

```
g.Name=cellstr(Name);

g.PolyConstants = [0 0 0 0];  % for a b c d constants

g.SchleyConstants = [0 0 0 0 0 0 0 0 0];

g.MolecularWeight = MW;

ng=class(g,'Gas');
```

@Gas\Display.m

```
function display(ng)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

disp ('Gas Class');

disp ('------------');



gnum = length(ng);

if gnum == 1
   disp ('       ');
   disp ('Gas Molecular Weight:');
   disp (ng.MolecularWeight);
   disp('a = ');
   disp(ng.PolyConstants(1));
   disp('b = ');
   disp(ng.PolyConstants(2));
   disp('c = ');
   disp(ng.PolyConstants(3));
   disp('d = ');
   disp(ng.PolyConstants(4));
   disp('Jaeschke and Schley constants (Savidge and Shen)');
   disp(ng.SchleyConstants);
else
   disp('Poly Constants');
   disp(get(ng,'PolyConst'));
   disp('Jaeschke and Schley constants (Savidge and Shen)');
   disp(get(ng,'SchleyConst'));
end
```

@Gas\get.m

```
function val = get(g, propName)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

gnum = length(g);
if gnum == 1
```

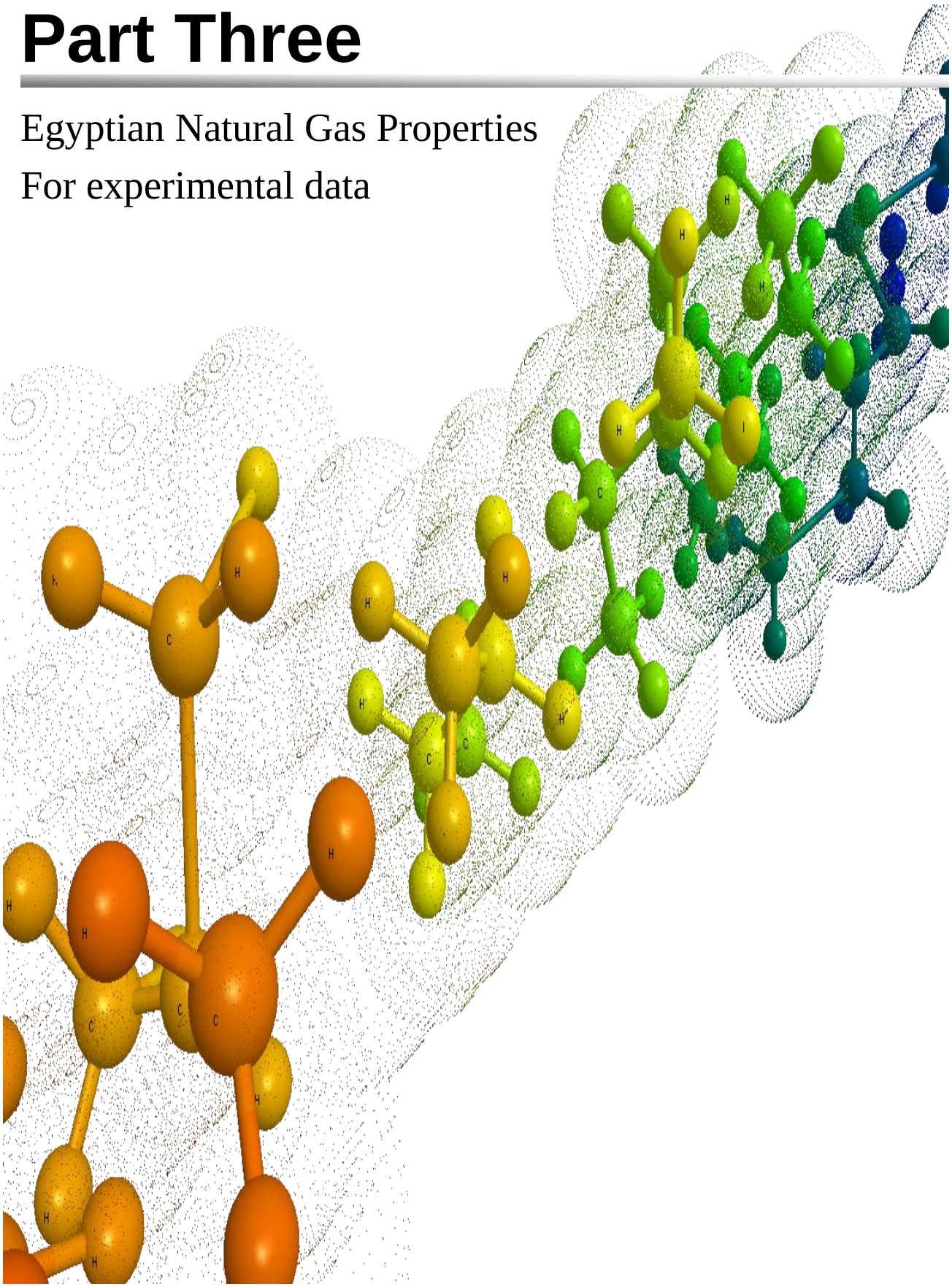
```
    switch (propName)
        case 'PolyConst'
            val = g.PolyConstants;
        case 'SchleyConst'
            val = g.SchleyConstants;
        case 'Name'
            val = g.Name;
        case 'MW'
            val = g.MolecularWeight;
    end
else
    switch (propName)
        case 'PolyConst'
            val= vertcat(g(:).PolyConstants);
        case 'SchleyConst'
            val = vertcat(g(:).SchleyConstants);
        case 'Name'
            val = vertcat(g(:).Name);
        case 'MW'
            val = vertcat(g(:).MolecularWeight);
    end
end
```

However this was a sample from the Gas class. All code and derivations will be mentioned in part 3 of this report to illustrate the solution way to of finding the natural gas properties.

Part Three

Egyptian Natural Gas Properties

For experimental data



C h a p t e r 8

Code Preparation

And @Gas Class

We are now going to prepare our code, this is a necessary step for the development cycle which short development time and efforts, yields also to rapid programming with ease and accuracy, also to have a well organized program structure.

UML CLASS DIAGRAM

The UML class diagram is a method demonstrating the classes included in our program. The UML diagrams have a wide reputation in the OOP programming.

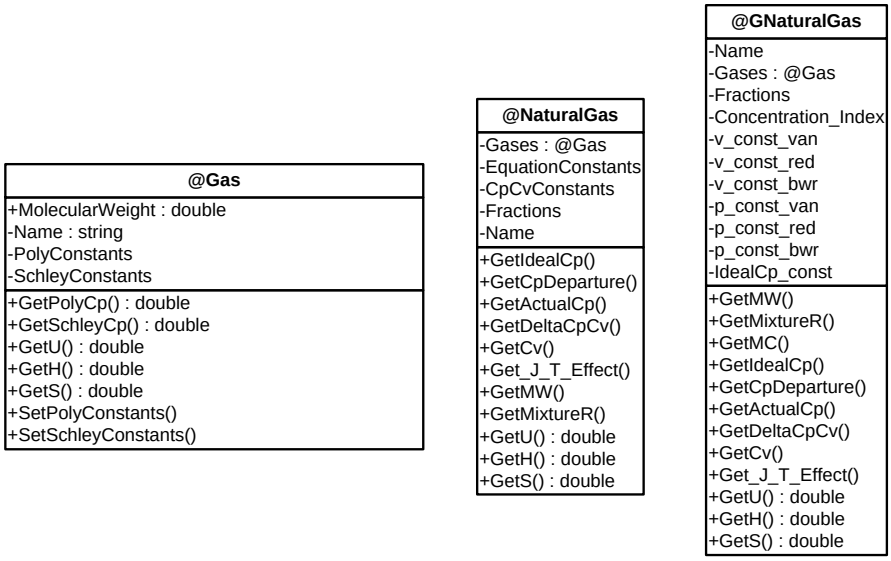


Figure 2: UML Diagram demonstrating classes used in the project

The reader can notice that every block have three distinct squares, the above is for class name, the middle is for attributes of the class, and the last is containing the class methods, (Functions), or (Procedures).

We will show the code of every function shown I the third square of every block in the next chapters along with their graph outputs.

EQUATION OF STATE

The equation of state used in this project is obtained from the research paper by El. Gizawy[II].

The Equation of state is

$$v = a + b/p + cT + \mathrm{d}/p^2 + eT^2 + fT/p + g/p^3 + hT^3 + iT^2/p + jT/p^2$$

$$p = a_1 + \frac{b_1}{v} + c_1 T + \frac{d_1}{v^2} + e_1 T^2 + f_1 \frac{T}{v} + \frac{g_1}{v^3} + h_1 T^3 + i_1 \frac{T^2}{v} + j_1 \frac{T}{v^2}$$

One of the advantages of using those equations is their linearity forms which have a great help in deducing the derivations of the equations.

It should be noticed that those equation were correlated from the Van der Waals, Redlich-Kwong, and BWR Equations besides the PVT data obtained from the experimental data from lab.

EXCEL FILES

Excel files are used extensively in our project for input and output data, a set of tools were used during project to maintain the excel data into graphs or into data fitting applications has been made.

@GAS CLASS

Now for some real programming, we are going to show my dear reader the @Gas class implementation, how the data go in and out.

So please fast your seat belt.

First we meet the constructor function

@Gas\Gas.m

```
function ng = Gas(Name,MW)
% Gas Class
% for Calculating numerous heat capacities
% actually by two method
% polynomial method
% method mentioned in Jaeschke and Schley paper.
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

g.Name=cellstr(Name);

g.PolyConstants = [0 0 0 0];  % for a b c d constants

g.SchleyConstants = [0 0 0 0 0 0 0 0 0];

g.MolecularWeight = MW;


ng=class(g,'Gas');
```

Function to set the gas polynomial constants

@Gas\SetPolyConstants.m

```
function ng=SetPolyConstants(g,const)
%Set the constants for Polynmoial a+bT+cT^2+dT^3
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

g.PolyConstants=const;
ng=g;

end
```

Function to set gas Schley constants

@Gas\SetSchleyConstants.m

```
function ng=SetSchleyConstants(g,const)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com
```

```
g.SchleyConstants=const;
ng=g;

end
```

Function to retrieve the ideal Cp from the polynomial equation of

$$\overline{C}_p = a + bT + cT^2 + dT^3$$

@Gas\GetPolyCp.m

```
function cp = GetPolyCp(ng,Temperature)
%cp = a+bT+cT^2+dT^3
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

% a= ng.PolyConstants(1);
% b= ng.PolyConstants(2);
% c= ng.PolyConstants(3);
% d= ng.PolyConstants(4);

PolyMat = get(ng,'PolyConst');
a= PolyMat(:,1);
b= PolyMat(:,2);
c= PolyMat(:,3);
d= PolyMat(:,4);

% a, b, c, and d   are now vectors

%cp = a+(b.*Temperature)+(c.* Temperature.^2)+(d.* Temperature.^3);
gnum = length(ng);

for ll=1 : gnum
   cp(ll,:) = a(ll) + (b(ll) .* Temperature) + (c(ll) .* Temperature .^2) + (d(ll) .* Temperature .^3);
end
```

Function to get Schley Ideal Cp for mixtures

$$\frac{Cp_{ideal}(T)}{R_u} = b + c\left\{\frac{d/T}{\sinh(d/T)}\right\}^2 + e\left\{\frac{f/T}{\cosh(f/T)}\right\}^2 + g\left\{\frac{h/T}{\sinh(h/T)}\right\}^2 + i\left\{\frac{j/T}{\cosh(j/T)}\right\}^2$$

@Gas\GetSchleyCp.m

```
function cp=GetSchleyCp(ng,Temperature)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

SchleyMat = get(ng,'SchleyConst');

B=SchleyMat(:,1);
C=SchleyMat(:,2);
D=SchleyMat(:,3);
```

```
E=SchleyMat(:,4);
F=SchleyMat(:,5);
G=SchleyMat(:,6);
H=SchleyMat(:,7);
I=SchleyMat(:,8);
J=SchleyMat(:,9);

R = 8.31451;

gnum = length(ng);
for ll=1 : gnum
   cp(ll,:) = B(ll)  + C(ll) .* ((D(ll)./Temperature)./sinh(D(ll)./Temperature)).^2;
   cp(ll,:) = cp(ll,:) + E(ll) .* ((F(ll)./Temperature)./cosh(F(ll)./Temperature)).^2;
   cp(ll,:) = cp(ll,:) + G(ll) .* ((H(ll)./Temperature)./sinh(H(ll)./Temperature)).^2;
   cp(ll,:) = cp(ll,:) + I(ll) .* ((J(ll)./Temperature)./cosh(J(ll)./Temperature)).^2;
   cp(ll,:) =cp(ll,:) .* R;
end
```

Function to get internal energy

$$\int_{n}^{n+1} du = \int_{0}^{T} [C_v] dT = \int_{0}^{T} [C_p -] dt = \left[aT + \frac{b}{2}T^2 + \frac{c}{3}T^3 + \frac{d}{4}T^4 \right]_0^T$$

@Gas\GetU.m

```
function cu = GetU(Gas, T)
%u(Gas, T) Summary of this function goes her
%  Calculating the ideal internal Energy at chosen temperature
% the relation of
% u=Integrate ( Cv dt ) = Integrate (Cp-R)

%the following constants is absolute means it is not divided by Molecular
%Weight

PolyMat = get(Gas,'PolyConst');
a= PolyMat(:,1);
b= PolyMat(:,2);
c= PolyMat(:,3);
d= PolyMat(:,4);

   function nu = gu(T)
      if T<=0
         nu = 0;
      else
         nu = (a .* T) + ((b./2) .* T.^2) + ((c./3) .* T.^3) + ...
             ((d./4) .* T.^4) - (8.314472 .* T) - gu(0);
      end
   end

   cu = gu(T);
end
```

Function to get enthalpy

$$\int_{n}^{n+1} dh = \int_{0}^{T} [C_p] dT = \left[aT + \frac{b}{2}T^2 + \frac{c}{3}T^3 + \frac{d}{4}T^4 \right]_0^T$$

@Gas\GetH.m

```
function cu = GetH(Gas, T)
%u(Gas, T) Summary of this function goes her
%  Calculating the ideal internal Energy at chosen temperature
% the relation of
% u=Integrate ( Cp dt )

%the following constants is absolute means it is not divided by Molecular
%Weight

PolyMat = get(Gas,'PolyConst');
a= PolyMat(:,1);
b= PolyMat(:,2);
c= PolyMat(:,3);
d= PolyMat(:,4);

   function nu = gu(T)
      if T<=0
         nu = 0;
      else
         nu = (a .* T) + ((b./2) .* T.^2) + ((c./3) .* T.^3) + ...
             ((d./4) .* T.^4) - gu(0);
      end
   end

   cu = gu(T);
end
```

Function to get entropy

$$\int_{0}^{T} ds = \int_{0}^{T} \left[\frac{C_p}{T} \right] dt = \left[a \ln T + bT + \frac{c}{2}T^2 + \frac{d}{3}T^3 \right]_0^T$$

@Gas\GetS.m

```
function cu = GetS(Gas, T)
%S(Gas, T) Summary of this function goes her
%  Calculating the ideal internal Energy at chosen temperature
% the relation of
% u=Integrate ( Cp/T dt )

%the following constants is absolute means it is not divided by Molecular
%Weight

PolyMat = get(Gas,'PolyConst');
a= PolyMat(:,1);
b= PolyMat(:,2);
```

```
c= PolyMat(:,3);
d= PolyMat(:,4);

   function nu = gu(T)
      if T<=0
         nu = 0;
      else
         nu = ((a .* log(T)) + (b .* T) + ((c./2) .* T.^2) + ...
             ((d./3) .* T.^3)) - gu(0);
      end
   end

   cu = gu(T);
end
```

The function which load the Gas data into @Gas object is as follows

@Gas

```
function mGas = LoadGas( gasname )

[gasdata,gasnames] = xlsread('GasConstants','schley');
[gaspolydata] =xlsread('GasConstants','poly');
[gasmw] = xlsread('GasConstants','mweight');


for i=1:length(gasnames)
   if strcmpi(gasnames(i) , gasname)
      mGas = Gas(gasnames(i),gasmw(i));
      mGas = SetSchleyConstants (mGas,gasdata(i,:));
      mGas = SetPolyConstants(mGas,gaspolydata(i,:));
   end
end
```

The data is loaded from file 'GasConstants.xls' are

Mole Fraction table

Methane	0.79044
Ethane	0.11419
Propane	0.04717
Isobutane	0.00366
n-butane	0.00361
Isopentane	0.00037
n-Pentane	0.00032
n-Hexane	0.0002
Nitrogen	0.0037
CarbonDioxide	0.03664

Polynomial Constants

Methane	19.89	5.02E-02	1.27E-05	-1.10E-08
Ethane	6.9	1.73E-01	-6.41E-05	7.29E-09
Propane	-4.04	3.05E-01	-1.57E-04	3.17E-08
Isobutane	-7.913	4.16E-01	-2.30E-04	4.99E-08
n-butane	3.96	3.72E-01	-1.83E-04	3.50E-08
Isopentane	6.774	4.54E-01	-2.25E-04	4.23E-08
n-Pentane	6.774	4.54E-01	-2.25E-04	4.23E-08
n-Hexane	6.938	5.52E-01	-2.87E-04	5.77E-08
Nitrogen	28.9	-1.57E-03	8.08E-06	-2.87E-09
CarbonDioxide	22.26	5.98E-02	-3.50E-05	7.47E-09

Schley Constants

Methane	4.0009	0.7632	820.659	0.0046	178.41	8.7443	1.06E+03
Ethane	4.00263	4.33939	559.314	1.23722	223.284	13.1974	1031.38
Propane	4.02939	6.60569	479.856	3.197	200.893	19.1921	955.312
Isobutane	4.06714	8.97575	438.27	5.25156	198.018	25.1423	1905.02
n-butane	4.33944	9.44893	468.27	6.89406	183.636	24.4618	1914.1
Isopentane	4	11.7618	292.503	20.1101	910.237	33.1688	1919.37
n-Pentane	4	8.95043	178.67	21.836	840.538	33.4032	1774.25
n-Hexane	4	11.6977	182.326	26.8142	859.207	38.6164	1826.59
Nitrogen	3.50031	0.13732	662.738	-0.1466	-680.562	0.90066	1740.06
CarbonDioxide	3.50002	2.04452	919.306	-1.06044	-865.07	2.03366	483.553

Molecular weight

Methane	16.043
Ethane	30.07
Propane	44.094
Isobutane	58.124
n-butane	58.124
Isopentane	72.151
n-Pentane	72
n-Hexane	86.178
Nitrogen	28
CarbonDioxide	44

EXAMPLE

```
>> n = LoadGas('Nitrogen')
Gas Class
------------

Gas Molecular Weight:
   28

a =
  28.9000

b =
```

```
   -0.0016

c =
  8.0810e-006

d =
 -2.8730e-009

Jaeschke and Schley constants (Savidge and Shen)
  1.0e+003 *

    0.0035    0.0001    0.6627   -0.0001   -0.6806    0.0009    1.7401         0         0

>> GetPolyCp(n,500)

ans =

   29.7756
```

C h a p t e r 9

Actual Cp

Now we are proceeding to investigate the Actual Cp for the natural gas mixture.

First we will implement the @NaturalGas class global functions then proceeding with Ideal Cp , DepartureCp, and ActualCp functions.

@NATURALGAS CLASS IMPLEMENTAION

Constructor

@NaturalGas\NaturalGas.m

```
function ngas = NaturalGas(Name,Gases,Fractions,EqConstants,CpCvConst)
%NaturalGas Class
%   (Name,Gases,Fractions,EqConstants,CpCvConst)
% Holding Array of gases and their corresponding mole fractions
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

ng.Name = cellstr(Name);
ng.Gases = Gases;
ng.Fractions = Fractions;

% the constants of Vander, Redlish, BWR, ..., etc.
ng.EquationConstants =  EqConstants; % this is the v(P,T) constants
ng.CpCvConstants = CpCvConst;       % this is the P(v,T) constants


ngas = class(ng,'NaturalGas');
```

Display Function

@NaturalGas\display.m

```
function  display(NGas)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

disp('Natural Gas Class');
ngnum = length(NGas);

if ngnum == 1
   disp('Gas and Fraction');
   gs = [get(NGas.Gases,'Name') cellstr(num2str(NGas.Fractions))];
   disp(gs);
   disp('Total Fraction =');
   disp(sum(NGas.Fractions));
else
  disp ('Group of natural gases');
end
```

Molecular Weight function

@NaturalGas\GetMW.m

```
function MW = GetMW(NGas)
%GetMW (NGas)
%  Get the molecular weight for the natural gas
%  By   SUM(Fractions * Molecular Weight for every component)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

MW = sum(NGas.Fractions .* get(NGas.Gases,'MW'));
```

Mixture R Function

@NaturalGas\GetMixtureR.m

```
function R = GetMixtureR(NGas)
%GetMixtureR(NGas)
%  Mixture Gas Constant
%  By   8.314 * SUM(Fractions / Molecular Weight for every component)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

R = 8.314472  ./ GetMW(NGas);
```

The function which load the Gas data into @NaturalGas object is as follows

LoadEgyptNaturalGas.m

```
function E_NG = LoadEgyptNaturalGas()
%LoadEgyptNaturalGas
%  will load the natural gas of egyptian natural gas
%  and with mole fraction

[gasdata,gasnames] = xlsread('GasConstants','schley');
[gaspolydata] =xlsread('GasConstants','poly');
[fractions] = xlsread('GasConstants','fraction');
[gasmw] = xlsread('GasConstants','mweight');


[equconst] = xlsread('GasConstants','equconst'); % this is the v(P,T) constants
[cpcvconst] = xlsread('GasConstants','cpcvconst'); % this is the P(v,T) constants


for i=1:length(gasnames)
       mGas(i) = Gas(gasnames(i),gasmw(i));
       mGas(i) = SetSchleyConstants (mGas(i),gasdata(i,:));
       mGas(i) = SetPolyConstants(mGas(i),gaspolydata(i,:));
end

E_NG = NaturalGas('EgyptNaturalGas',mGas,fractions,equconst,cpcvconst);
```

Example

```
>> eng = LoadEgyptNaturalGas()
Natural Gas Class
Gas and Fraction
   'Methane'         '0.79044'
   'Ethane'          '0.11419'
   'Propane'         '0.04717'
   'Isobutane'       '0.00366'
   'n-butane'        '0.00361'
   'Isopentane'      '0.00037'
   'n-Pentane'       '0.00032'
   'n-Hexane'        ' 0.0002'
   'Nitrogen'        ' 0.0037'
   'CarbonDioxide'    '0.03664'
```

```
Total Fraction =
   1.0003

>> GetIdealCp(eng,1,500)

ans =

   2.6587
```

CALCULATING IDEAL CP

By using the equation

$$C_p = \sum_{i=1}^{i=n} y_i M_i C_{p_i}$$

Where n is number of included gases in natural gas, and M is mole fraction

@NaturalGas\GetIdealCp.m

```
function cp = GetIdealCp(NGas, MethodNum, Temperature)
%GetIdealCp (NGas, MethodNum, Temperature)
%  Get the Ideal isobaric specific heat for the mixture
%  Parameter either 1 or 2
%  1 == poly function
%  2 == Schley function
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

tempnum = length(Temperature);

switch(MethodNum)
   case 1
      tcp =  GetPolyCp(NGas.Gases, Temperature);
      mfr = repmat(NGas.Fractions,1,tempnum);
      cp = sum (mfr .* tcp);
   case 2
      tcp = GetSchleyCp(NGas.Gases, Temperature);
      mfr = repmat(NGas.Fractions,1,tempnum);
      cp = sum (mfr .* tcp);
end
cp = cp ./ GetMW(NGas);
```

As noted from code the first parameter is expressing which method will be used in calculating the Ideal Cp.

CALCULATING DEPARTURE CP

$$(Cp_{NG\,actual} - Cp_{NG\,ideal}) = -T\int_1^p (\partial^2 v/\partial T^2)_p \, dp$$

$$v = a + b/p + cT + \mathrm{d}/p^2 + eT^2 + fT/p + g/p^3 + hT^3 + iT^2/p + jT/p^2$$

$$(Cp_{NG\,actual} - Cp_{NG\,ideal}) = -T\int_2^p \left(2e + bhT + \frac{2i}{p}\right) dp$$

$$(Cp_{NG\,actual} - Cp_{NG\,ideal}) = -T\left[(2ep + 6hTp + 2i\ln p) - (405.2e + 1215.6hT + 2i\ln 202.6)\right]$$

@NaturalGas\GetCpDeparture.m

```
function DeltaCp = GetCpDeparture(NGas,Method,P,T)
%GetCpDeparture (NGas,Method,Pressure,Temperature)
%  Get Heat Capacity Departure   At certain Pressure and Temperature
%  ((Cp)ideal  - (Cp) Actual)= - T * integerate(d^2v/dt^2)dp
%  NGas:  @NaturalGas Object
%  Method Parameter:
%   1- Vanderval Constants
%   2- Redlish Constants
%   3- BWR Constatnts
%   4- Expiremental Constants
%  P: Pressure      in kPa
%  T: Temperature   in Kelvin
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

a = NGas.EquationConstants(1,Method);
b = NGas.EquationConstants(2,Method);
c = NGas.EquationConstants(3,Method);
d = NGas.EquationConstants(4,Method);
e = NGas.EquationConstants(5,Method);
f = NGas.EquationConstants(6,Method);
g = NGas.EquationConstants(7,Method);
h = NGas.EquationConstants(8,Method);
i = NGas.EquationConstants(9,Method);
j = NGas.EquationConstants(10,Method);

%check for method if == 4 then use the equation for experimental

% if Method == 4
%     %experimental equation
%     %new equation of v
%     %v = a+b/T+c/P+d/T^2+e/P^2+f/(T*P)+g/T^3+h/P^3+i/(T*P^2)+j/(T^2*P)
%     % final equation -T * integerate(d^2v/dt^2)dp
%     i_d2vdt2_p = ...
%     ( ...
%         (2 .* b .* P ./ T.^3)  + (6 .* d .* P ./ T.^4)  ...
%         + (2 .* f .* log2(P) ./ T .^3) + (12 .* g .* P ./ T.^5) ...
%         - (2 .* i ./ (T.^3 .* P)) + (6 .* j .* log2(P) ./ T.^4) ...
%     ) ...
%     - ...
%     ( ...
%         (2 .* b .* (2 * 101.3) ./ T.^3)  + (6 .* d .* (2 * 101.3) ./ T.^4)  ...
%         + (2 .* f .* log2((2 * 101.3)) ./ T .^3) + (12 .* g .* (2 * 101.3) ./ T.^5) ...
%         - (2 .* i ./ (T.^3 .* (2 * 101.3))) + (6 .* j .* log2((2 * 101.3)) ./ T.^4) ...
%     );
%
%     DeltaCp = - T .* i_d2vdt2_p;
%
% else

    tmp1 = 2 .* e .* P;
    tmp1 = tmp1 + (6 .* h .* T .* P);
    tmp1 = tmp1 + (2 .* i .* log(P));
    tmp2 = 405.2 .* e;
    tmp2 = tmp2 + (1215.6 .* h .* T);
    tmp2 = tmp2 + (2 .* i .* log(202.6));
```

```
    DeltaCp = - T .* (tmp1 - tmp2);

end
```

CALCULATING ACTUAL CP

@NaturalGas\GetActualCp.m

```
function cp = GetActualCp(NGas, IdealMethod, DepartureMethod, Pressure, Temperature)
%GetActualCp(NGas, IdealMethod, DepartureMethod, Pressure, Temperature)
%  Get the actual cp for the given natural gas based on arguments
%  IdealMethod
%   1- Polynomial
%   2- Schley
%  DepartureMethod
%   1- Vander, 2-Redlish, 3- BWR, 4- Experimental
% Pressure, Temperature    P,T  ;)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

cp = GetIdealCp(NGas, IdealMethod, Temperature);
cp = cp + GetCpDeparture(NGas, DepartureMethod, Pressure, Temperature);
```

C h a p t e r 10

Isochoric Specific Heat Cv

And Joule Thomson Effect

Continuing from the last chapter we will now investigate the Isochoric specific heat and Joule Thomson Effect.

CALCULATING CP – CV

$$Cp - Cv = T\left(\frac{\partial v}{\partial T}\right)_{p=c}\left(\frac{\partial p}{\partial T}\right)_{v}$$

$$v = a + b/p + cT + \mathrm{d}/p^2 + eT^2 + fT/p + g/p^3 + hT^3 + iT^2/p + jT/p^2$$

$$p = a_1 + \frac{b_1}{v} + c_1 T + \frac{d_1}{v^2} + e_1 T^2 + f_1\frac{T}{v} + \frac{g_1}{v^3} + h_1 T^3 + i_1\frac{T^2}{v} + j_1\frac{T}{v^2}$$

$$\left(\frac{\partial v}{\partial T}\right)_{p=c} = \left(c + 2eT + \frac{f}{p} + 3hT^2 + 2i\frac{T}{p} + \frac{j}{p^2}\right)$$

$$\left(\frac{\partial p}{\partial T}\right)_{v} = \left(C_1 + 2e_1 T + \frac{f_1}{v} + 3h_1 T^2 + 2i_1\frac{T}{v} + \frac{j_1}{v^2}\right)_{v}$$

$$(Cp_{NG})_{p,T} - (Cv_{NG})_{v,T} = T\left(c + 2eT + \frac{f}{p} + 3hT^2 + 2i\frac{T}{p} + \frac{j}{p^2}\right)_{p=c}\left(C_1 + 2e_1 T + \frac{f_1}{v} + 3h_1 T^2 + 2i_1\frac{T}{v} + \frac{j_1}{v^2}\right)_{T}$$

@NaturalGas\GetDeltaCpCv.m

```
function [dCpCv] = GetDeltaCpCv(NGas,Method, P, v, T)
%GetDeltaCpCv(NGas,Method,Pressure, Volume, Temperature)
%  Delta Cp Cv = Temperature * dv^2/dT dp/dT
%  where v=F(Temperature) and p=F(Temperature)
%  Method Parameter
%  1- Vanderval Constants
%  2- Redlish Constants
%  3- BWR Constatnts
%  4- Expiremental Constants
%  then enter parameters as  P, V, T   ;)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

A = NGas.EquationConstants(1,Method);
B = NGas.EquationConstants(2,Method);
C = NGas.EquationConstants(3,Method);
D = NGas.EquationConstants(4,Method);
E = NGas.EquationConstants(5,Method);
F = NGas.EquationConstants(6,Method);
G = NGas.EquationConstants(7,Method);
H = NGas.EquationConstants(8,Method);
I = NGas.EquationConstants(9,Method);
J = NGas.EquationConstants(10,Method);

A1 = NGas.CpCvConstants(1,Method);
B1 = NGas.CpCvConstants(2,Method);
C1 = NGas.CpCvConstants(3,Method);
D1 = NGas.CpCvConstants(4,Method);
E1 = NGas.CpCvConstants(5,Method);
```

```
F1 = NGas.CpCvConstants(6,Method);
G1 = NGas.CpCvConstants(7,Method);
H1 = NGas.CpCvConstants(8,Method);
I1 = NGas.CpCvConstants(9,Method);
J1 = NGas.CpCvConstants(10,Method);

%check for method if == 4 then use the equation for experimental

% if Method == 4
%     %(Cp-Cv) = T * (dv/dt)^2 * dp/dt
%
%     %note: constants A1..J1  are those to the equation of P=F(T,v)
%     dp_dt = B1 + 2 .* D1 .* T + F1 ./ v + 3 .* G1 * T.^2 + I1 ./ v.^2 ...
%         + 2 .* J1./v .*T;
%     dv_dt = -B .* T.^-2 - 2 .* D .* T.^-3 - F ./ P * T.^-2 - 3 .* G .* T.^-4 ...
%         - I ./ P.^2 .* T.^-2 - 2 .* J ./ P .* T.^-3;
%
%     dCpCv = T .* dv_dt.^2 .* dp_dt;
% else

    %calculating the second term
    ft = C1 + 2 .* E1 .* T + F1 ./ v;
    ft = ft + 3 .* H1 .* T.^2;
    ft = ft + 2 .* I1 .* T./ v;
    ft = ft + J1 ./ v.^2;

    %calculating the first term
    st = C + 2 .* E .* T + F ./ P + 3 .* H .* T.^2;
    st = st + 2 .* I .* T ./ P + J ./ P.^2;


    dCpCv = (ft .* st) .* T;

end
```

@NaturalGas\GetDeltaCpCvDeviation.m

```
function [ dvCpCv ] = GetDeltaCpCvDeviation ...
    ( NGas,Method,Pressure, Volume,Temperature )
%GetDeltaCpCvDeviation ( NGas,Method,Pressure, Volume,Temperature )
%  getting the deviation between the expiremental data and theoritical
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

exp = GetDeltaCpCv(NGas,4,Pressure, Volume,Temperature);
thr = GetDeltaCpCv(NGas,Method,Pressure, Volume,Temperature);

dvCpCv = ((exp - thr) ./ thr) .* 100;
```

CALCULATING JOULE THOMSON EFFECT

$$\mu = \left(\frac{\partial T}{\partial p}\right)_h = \frac{1}{Cp}\left[T\left(\frac{\partial v}{\partial T}\right) - v\right]$$

$$v = a + b/p + cT + \mathrm{d}/p^2 + eT^2 + fT/p + g/p^3 + hT^3 + iT^2/p + jT/p^2$$

$$\left(\frac{\partial v}{\partial T}\right)_{p=c} = \left(c + 2eT + \frac{f}{p} + 3hT^2 + 2i\frac{T}{p} + \frac{j}{p^2}\right)$$

@NaturalGas\.m

```
function [ mue ] = Get_J_T_Effect ...
    ( NGas, IdealMethod, ActualMethod, P, V, T )
%Get_J_T_Effect
%  ( NGas, IdealMethod, ActualMethod, Pressure, Volume, Temperature )
%  getting Joule-Thomson Effect
%  In physics, the Joule-Thomson effect, or Joule-Kelvin effect, is
%  a process in which the temperature of a gas is decreased by letting
%  the gas expand adiabatically.
%  It is named for James Prescott Joule and William Thomson,
%  1st Baron Kelvin who established the effect in 1852 following earlier
%  work by Joule on Joule expansion in which a gas expands at constant
%  internal energy.
% ------------------------------------------------------------------------
% IdealMethod
% 1:= poly
% 2:= schley
% ActualMethod (1, 2, 3, 4)
% Van, redlish, bwr, expiremental
% and P V T
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com


A = NGas.EquationConstants(1,IdealMethod);
B = NGas.EquationConstants(2,IdealMethod);
C = NGas.EquationConstants(3,IdealMethod);
D = NGas.EquationConstants(4,IdealMethod);
E = NGas.EquationConstants(5,IdealMethod);
F = NGas.EquationConstants(6,IdealMethod);
G = NGas.EquationConstants(7,IdealMethod);
H = NGas.EquationConstants(8,IdealMethod);
I = NGas.EquationConstants(9,IdealMethod);
J = NGas.EquationConstants(10,IdealMethod);




% if ActualMethod == 4
%     dv_dt = -B .* T.^-2 - 2 .* D .* T.^-3 - F ./ P * T.^-2 - 3 .* G .* T.^-4 ...
%       - I ./ P.^2 .* T.^-2 - 2 .* J ./ P .* T.^-3;
```

```
%
%     tmp = dv_dt .* T;
%     tmp = tmp - V;
%
%     mue = tmp ./ GetActualCp_F(NGas, P, T);
% else
    tmp = C + 2 .* E .* T + F ./ P + 3 .* H .* T.^2;
    tmp = tmp + 2 .* I .* T ./ P + J ./ P.^2;
    tmp = tmp .* T;
    tmp = tmp - V;
    mue = ...
    tmp ./ GetActualCp(NGas,IdealMethod, ActualMethod, P, T);

end
```

EXPORT FUNCTION FOR THE PREVIOUS PROPERTIES

The following function is responsible to export all the required data for graphing and fitting to excel files into the desired folder.

@NaturalGas\ExportResults.m

```
function ExportResults( NGas , fl)
%Export Results to excell files
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

cd (fl)
mkdir ('DataFitting')
mkdir ('DataGraphing')
mkdir ('DataGraphing\Actual Cp')
mkdir ('DataGraphing\Actual Cv')
mkdir ('DataGraphing\Actual Joule Thomson')



p = @(x) x .* 101.3;

%PVT Values
pressure=p([2 5:5:30]);
t=[300:25:800];
v=[0.5885 0.30977 0.14714 0.13078 0.11205];

%Experimental data range
exp_p=p([2 5 10]);  %pressure ranges we want
t_2_5_10 = [300:25:800];  %temperature range for pressures less than 15 bar
p_15 = [p(15)]; t_15 = [480:6.1905:610];
p_20 = [p(20)]; t_20 = [650:1.9048:690];

disp('1- Calculating Ideal Cp:');
%for grapher
iCp1 = [t' GetIdealCp(NGas,1,t)'];
iCp2 = [t' GetIdealCp(NGas,2,t)'];
iCp = [t' GetIdealCp(NGas,1,t)' GetIdealCp(NGas,2,t)'];
```

```
%write to excell sheets
xlswrite ('DataGraphing\IdealCp.xls',iCp,'Poly_Schley');
%----------------------


disp('2- Calculating Poly-Schley error:')
pserr = ((iCp1(:,2) - iCp2(:,2)) ./ iCp2(:,2)) .* 100 ;

%for grapher
pserr_graph = [t' pserr];

%for datafit
pserr_dfit = [pserr t'];

%write to excell sheets
xlswrite ('DataGraphing\Poly_Schley_Error_Percentage.xls',pserr_graph,'ErrorPercentage');
xlswrite ('DataFitting\Poly_Schley_Error_Percentage.xls',pserr_dfit,'ErrorPercentage');
%----------------------

disp('3- Cp Departure Percentage');
%with all set of constants van,red,BWR,expiermental
% and pressure 1, 30 bar
% the data format is for grapher as follows
% Temperature, Departure p=[2 5 10 15 20] bar

% 1- for poly base
%for van
dp1 = [t' GetCpDeparturePercentage(NGas,1,1,p(2),t)'];
dp1 = [dp1 GetCpDeparturePercentage(NGas,1,1,p(5),t)'];
dp1 = [dp1 GetCpDeparturePercentage(NGas,1,1,p(10),t)'];
dp1 = [dp1 GetCpDeparturePercentage(NGas,1,1,p(15),t)'];
dp1 = [dp1 GetCpDeparturePercentage(NGas,1,1,p(20),t)'];
dp1 = [dp1 GetCpDeparturePercentage(NGas,1,1,p(25),t)'];
dp1 = [dp1 GetCpDeparturePercentage(NGas,1,1,p(30),t)'];
dp1 = num2cell(dp1);
dp1 = [{'T', 'P=2','P=5','P=10', 'P=15', 'P=20', 'P=25', 'P=30'} ; dp1];
%for red
dp2 = [t' GetCpDeparturePercentage(NGas,1,2,p(2),t)'];
dp2 = [dp2 GetCpDeparturePercentage(NGas,1,2,p(5),t)'];
dp2 = [dp2 GetCpDeparturePercentage(NGas,1,2,p(10),t)'];
dp2 = [dp2 GetCpDeparturePercentage(NGas,1,2,p(15),t)'];
dp2 = [dp2 GetCpDeparturePercentage(NGas,1,2,p(20),t)'];
dp2 = [dp2 GetCpDeparturePercentage(NGas,1,2,p(25),t)'];
dp2 = [dp2 GetCpDeparturePercentage(NGas,1,2,p(30),t)'];
dp2 = num2cell(dp2);
dp2 = [{'T', 'P=2','P=5','P=10', 'P=15', 'P=20', 'P=25', 'P=30'} ; dp2];
%for BWR
dp3 = [t' GetCpDeparturePercentage(NGas,1,3,p(2),t)'];
dp3 = [dp3 GetCpDeparturePercentage(NGas,1,3,p(5),t)'];
dp3 = [dp3 GetCpDeparturePercentage(NGas,1,3,p(10),t)'];
dp3 = [dp3 GetCpDeparturePercentage(NGas,1,3,p(15),t)'];
dp3 = [dp3 GetCpDeparturePercentage(NGas,1,3,p(20),t)'];
dp3 = [dp3 GetCpDeparturePercentage(NGas,1,3,p(25),t)'];
```

```
dp3 = [dp3 GetCpDeparturePercentage(NGas,1,3,p(30),t)'];
dp3 = num2cell(dp3);
dp3 = [{'T', 'P=2','P=5','P=10', 'P=15', 'P=20', 'P=25', 'P=30'} ; dp3];

%for expierment
dp4 = [t' GetCpDeparturePercentage(NGas,1,4,p(2),t)'];
dp4 = [dp4 t' GetCpDeparturePercentage(NGas,1,4,p(5),t)'];
dp4 = [dp4 t' GetCpDeparturePercentage(NGas,1,4,p(10),t)'];
dp4 = [dp4 t' GetCpDeparturePercentage(NGas,1,4,p(15),t)'];
dp4 = [dp4 t' GetCpDeparturePercentage(NGas,1,4,p(20),t)'];
dp4 = [dp4 t' GetCpDeparturePercentage(NGas,1,4,p(25),t)'];
dp4 = [dp4 t' GetCpDeparturePercentage(NGas,1,4,p(30),t)'];
dp4 = num2cell(dp4);
dp4 = [{'T', 'P=2','T', 'P=5','T', 'P=10','T', 'P=15','T', 'P=20','T', 'P=25','T', 'P=30'} ; dp4];
%-------------------------------------------------------



%write to excell sheets
%poly
xlswrite ('DataGraphing\CpDeparturePercentage.xls',dp1,'poly_van');
xlswrite ('DataGraphing\CpDeparturePercentage.xls',dp2,'poly_red');
xlswrite ('DataGraphing\CpDeparturePercentage.xls',dp3,'poly_BWR');
xlswrite ('DataGraphing\CpDeparturePercentage.xls',dp4,'poly_Expiremental');
%----------------------

disp('4- Actual Cp for data fitting');

%van
poly(1) = {GetActualCpData(NGas,1,1,pressure,t)};
schley(1) =  {GetActualCpData(NGas,2,1,pressure,t)};
%red
poly(2) = {GetActualCpData(NGas,1,2,pressure,t)};
schley(2) =  {GetActualCpData(NGas,2,2,pressure,t)};
%BWR
poly(3) = {GetActualCpData(NGas,1,3,pressure,t)};
schley(3) =  {GetActualCpData(NGas,2,3,pressure,t)};

tmpel = GetActualCpData(NGas,1,4,exp_p,t);
tmpel = [tmpel; GetActualCpData(NGas,1,4,p_15,t)];
tmpel = [tmpel; GetActualCpData(NGas,1,4,p_20,t)];

poly(4) = {tmpel};

tmpel = GetActualCpData(NGas,2,4,exp_p,t);
tmpel = [tmpel; GetActualCpData(NGas,2,4,p_15,t)];
tmpel = [tmpel; GetActualCpData(NGas,2,4,p_20,t)];
schley(4) =  {tmpel};

%write to excell sheets
xlswrite ('DataFitting\ActualCpData.xls',poly{1},'polyvan');
xlswrite ('DataFitting\ActualCpData.xls',schley{1},'schleyvan');
xlswrite ('DataFitting\ActualCpData.xls',poly{2},'polyRedlish');
xlswrite ('DataFitting\ActualCpData.xls',schley{2},'Schleyredlish');
xlswrite ('DataFitting\ActualCpData.xls',poly{3},'polyBWR');
```

```
xlswrite ('DataFitting\ActualCpData.xls',schley{3},'SchleyBWR');
xlswrite ('DataFitting\ActualCpData.xls',poly{4},'poly_Exper');
xlswrite ('DataFitting\ActualCpData.xls',schley{4},'Schley_Exper');
%----------------------

disp('4a- Actual Cp For Grapher directly from the program');

%van
poly(1) = {GetActualCpForGrapher(NGas,1,1,pressure,t)};
schley(1) =  {GetActualCpForGrapher(NGas,2,1,pressure,t)};
%red
poly(2) = {GetActualCpForGrapher(NGas,1,2,pressure,t)};
schley(2) =  {GetActualCpForGrapher(NGas,2,2,pressure,t)};
%BWR
poly(3) = {GetActualCpForGrapher(NGas,1,3,pressure,t)};
schley(3) =  {GetActualCpForGrapher(NGas,2,3,pressure,t)};

%Experimental
tmp = [GetActualCpForGrapher(NGas,1,4,p(2),t)];
tmp = [tmp GetActualCpForGrapher(NGas,1,4,p(5),t)];
tmp = [tmp GetActualCpForGrapher(NGas,1,4,p(10),t)];
tmp = [tmp GetActualCpForGrapher(NGas,1,4,p_15,t)];
tmp = [tmp GetActualCpForGrapher(NGas,1,4,p_20,t)];
tmp = [tmp GetActualCpForGrapher(NGas,1,4,p(25),t)];
tmp = [tmp GetActualCpForGrapher(NGas,1,4,p(30),t)];
tmp = num2cell(tmp);
tmp = [{'T', 'P=2','T', 'P=5','T', 'P=10','T', 'P=15','T', 'P=20','T', 'P=25','T', 'P=30'} ; tmp];

poly(4) = {tmp};

%actual Cp from the fitted equation
%exp = [t' GetActualCp_F(NGas,p(2),t)' GetActualCp_F(NGas,p(5),t)'
GetActualCp_F(NGas,p(10),t)' GetActualCp_F(NGas,p(15),t)' GetActualCp_F(NGas,p(20),t)'
GetActualCp_F(NGas,p(30),t)' ];
%exp = num2cell(exp);
%exp = [{'T', 'P=2','P=5', 'P=10', 'P=15', 'P=20', 'P=30'} ; exp];

%schley(4) =  {GetActualCpForGrapher(NGas,2,4,pressure,t)};

%write to excell sheets
xlswrite ('DataGraphing\Actual Cp\ActualCpData.xls',poly{1},'polyvan');
%xlswrite ('DataGraphing\ActualCpData.xls',schley{1},'schleyvan');
xlswrite ('DataGraphing\Actual Cp\ActualCpData.xls',poly{2},'polyRedlish');
%xlswrite ('DataGraphing\ActualCpData.xls',schley{2},'Schleyredlish');
xlswrite ('DataGraphing\Actual Cp\ActualCpData.xls',poly{3},'polyBWR');
%xlswrite ('DataGraphing\ActualCpData.xls',schley{3},'SchleyBWR');

xlswrite ('DataGraphing\Actual Cp\ActualCpData.xls',poly{4},'poly_Exper');
%xlswrite ('DataGraphing\ActualCpData.xls',schley{4},'Schley_Exper');
%xlswrite ('DataGraphing\Actual Cp\ActualCpData.xls',exp,'fitted_equ_Exper');
%----------------------

disp('4b- Experimental Cp to Theoritical Cp For Grapher by fitted');
```

```
%van
tmp = [GetExperimentalCpPercentageForGrapher(NGas,1,1,p(2),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,1,p(5),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,1,p(10),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,1,p_15,t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,1,p_20,t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,1,p(25),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,1,p(30),t)];
tmp = num2cell(tmp);
tmp = [{'T', 'P=2','T', 'P=5','T', 'P=10','T', 'P=15','T', 'P=20','T', 'P=25','T', 'P=30'} ; tmp];
poly(1) = {tmp};
%schley(1) =  {GetExperimentalCpPercentageForGrapher(NGas,2,1,pressure,t)};
%red
tmp = [GetExperimentalCpPercentageForGrapher(NGas,1,2,p(2),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,2,p(5),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,2,p(10),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,2,p_15,t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,2,p_20,t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,2,p(25),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,2,p(30),t)];
tmp = num2cell(tmp);
tmp = [{'T', 'P=2','T', 'P=5','T', 'P=10','T', 'P=15','T', 'P=20','T', 'P=25','T', 'P=30'} ; tmp];
poly(2) = {tmp};
%schley(2) =  {GetExperimentalCpPercentageForGrapher(NGas,2,2,pressure,t)};
%BWR
tmp = [GetExperimentalCpPercentageForGrapher(NGas,1,3,p(2),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,3,p(5),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,3,p(10),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,3,p_15,t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,3,p_20,t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,3,p(25),t)];
tmp = [tmp GetExperimentalCpPercentageForGrapher(NGas,1,3,p(30),t)];
tmp = num2cell(tmp);
tmp = [{'T', 'P=2','T', 'P=5','T', 'P=10','T', 'P=15','T', 'P=20','T', 'P=25','T', 'P=30'} ; tmp];
poly(3) = {tmp};
%schley(3) =  {GetExperimentalCpPercentageForGrapher(NGas,2,3,pressure,t)};

%write to excell sheets
xlswrite ('DataGraphing\Actual Cp\DevPercentCpExpCpTh.xls',poly{1},'polyvan');
%xlswrite ('DataGraphing\DevPercentCpExpCpTh.xls',schley{1},'schleyvan');
xlswrite ('DataGraphing\Actual Cp\DevPercentCpExpCpTh.xls',poly{2},'polyRedlish');
%xlswrite ('DataGraphing\DevPercentCpExpCpTh.xls',schley{2},'Schleyredlish');
xlswrite ('DataGraphing\Actual Cp\DevPercentCpExpCpTh.xls',poly{3},'polyBWR');
%xlswrite ('DataGraphing\DevPercentCpExpCpTh.xls',schley{3},'SchleyBWR');
%----------------------



disp('5- Joule Thomson effect actual');
%van
poly_jt(1) = {Get_J_T_Data(NGas,1,1)};
schley_jt(1) = {Get_J_T_Data(NGas,2,1)};
%redlish
poly_jt(2) = {Get_J_T_Data(NGas,1,2)};
schley_jt(2) = {Get_J_T_Data(NGas,2,2)};
%BWR
```

```
poly_jt(3) = {Get_J_T_Data(NGas,1,3)};
schley_jt(3) = {Get_J_T_Data(NGas,2,3)};
%expiermental
poly_jt(4) = {Get_J_T_Data(NGas,1,4)};
schley_jt(4) = {Get_J_T_Data(NGas,2,4)};

%write to excell sheets
xlswrite ('DataFitting\ActualJouleThomson.xls',poly_jt{1},'polyvan');
xlswrite ('DataFitting\ActualJouleThomson.xls',schley_jt{1},'schleyvan');
xlswrite ('DataFitting\ActualJouleThomson.xls',poly_jt{2},'polyRedlish');
xlswrite ('DataFitting\ActualJouleThomson.xls',schley_jt{2},'Schleyredlish');
xlswrite ('DataFitting\ActualJouleThomson.xls',poly_jt{3},'polyBWR');
xlswrite ('DataFitting\ActualJouleThomson.xls',schley_jt{3},'SchleyBWR');
xlswrite ('DataFitting\ActualJouleThomson.xls',poly_jt{4},'polyExp');
xlswrite ('DataFitting\ActualJouleThomson.xls',schley_jt{4},'SchleyExp');
%----------------------

%two sheet for method with two volumes
vol=[v(1) v(5)];

disp('5b- Joule Thomson deviation');
p_mue(1) = {Get_J_T_Dev_Data(NGas,1,1)};
s_mue(1) = {Get_J_T_Dev_Data(NGas,2,1)};
p_mue(2) = {Get_J_T_Dev_Data(NGas,1,2)};
s_mue(2) = {Get_J_T_Dev_Data(NGas,2,2)};
p_mue(3) = {Get_J_T_Dev_Data(NGas,1,3)};
s_mue(3) = {Get_J_T_Dev_Data(NGas,2,3)};



%write to excell sheets
xlswrite ('DataFitting\mueDeviation.xls',p_mue{1},'poly_van');
xlswrite ('DataFitting\mueDeviation.xls',s_mue{1},'schley_van');
xlswrite ('DataFitting\mueDeviation.xls',p_mue{2},'poly_red');
xlswrite ('DataFitting\mueDeviation.xls',s_mue{2},'schley_red');
xlswrite ('DataFitting\mueDeviation.xls',p_mue{3},'poly_bwr');
xlswrite ('DataFitting\mueDeviation.xls',s_mue{3},'schley_bwr');

%----------------------
disp('5c- Effect Of v on mue');
v_max = 0.594;
v_min = 0.119;

van_jte = [t' Get_J_T_Effect(NGas,1,1,20*101.3,v_max,t)'
Get_J_T_Effect(NGas,1,1,20*101.3,v_min,t)'];
red_jte = [t' Get_J_T_Effect(NGas,1,2,20*101.3,v_max,t)'
Get_J_T_Effect(NGas,1,2,20*101.3,v_min,t)'];
bwr_jte = [t' Get_J_T_Effect(NGas,1,3,20*101.3,v_max,t)'
Get_J_T_Effect(NGas,1,3,20*101.3,v_min,t)'];
exp_jte = [t' Get_J_T_Effect(NGas,1,4,20*101.3,v_max,t)'
Get_J_T_Effect(NGas,1,4,20*101.3,v_min,t)'];

%converting to cells to be able to write charachters into excell

van_jte = [{'T', 'v max=0.594', 'v min = 0.119'} ; num2cell(van_jte)];
```

```
red_jte = [{'T', 'v max=0.594', 'v min = 0.119'} ; num2cell(red_jte)];
bwr_jte = [{'T', 'v max=0.594', 'v min = 0.119'} ; num2cell(bwr_jte)];
exp_jte = [{'T', 'v max=0.594', 'v min = 0.119'} ; num2cell(exp_jte)];

xlswrite ('DataGraphing\Actual Joule Thomson\EffectOf_v_on_mue.xls',van_jte,'VAN');
xlswrite ('DataGraphing\Actual Joule Thomson\EffectOf_v_on_mue.xls',red_jte,'Redlich');
xlswrite ('DataGraphing\Actual Joule Thomson\EffectOf_v_on_mue.xls',bwr_jte,'BWR');
xlswrite ('DataGraphing\Actual Joule Thomson\EffectOf_v_on_mue.xls',exp_jte,'Experimental');
%-----------------------------------------------

disp('5c- mue data of minimum v  and p=[2 5 10 15 20 25 30]');

% mv function for simplifing as because we will change only Pressure variable
mv = @(m, p) Get_J_T_Effect(NGas,1,m,p*101.3,v_min,t);
van_mv = [t' mv(1,2)' mv(1,5)' mv(1,10)' mv(1,15)' mv(1,20)' mv(1,25)' mv(1,30)'];
red_mv = [t' mv(2,2)' mv(2,5)' mv(2,10)' mv(2,15)' mv(2,20)' mv(2,25)' mv(2,30)'];
bwr_mv = [t' mv(3,2)' mv(3,5)' mv(3,10)' mv(3,15)' mv(3,20)' mv(3,25)' mv(3,30)'];
exp_mv = [t' mv(4,2)' mv(4,5)' mv(4,10)' mv(4,15)' mv(4,20)' mv(4,25)' mv(4,30)'];

%converting to cells to be able to write charachters into excell
van_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(van_mv)];
red_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(red_mv)];
bwr_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(bwr_mv)];
exp_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(exp_mv)];

xlswrite ('DataGraphing\Actual Joule Thomson\mue_Min_V.xls',van_mv,'poly VAN');
xlswrite ('DataGraphing\Actual Joule Thomson\mue_Min_V.xls',red_mv,'Poly Redlish');
xlswrite ('DataGraphing\Actual Joule Thomson\mue_Min_V.xls',bwr_mv,'Poly BWR');
xlswrite ('DataGraphing\Actual Joule Thomson\mue_Min_V.xls',exp_mv,'Poly Experimental');
%----------------------

disp('5d- mue deviation data of minimum v  and p=[2 5 10 15 20 25 30]');
% mv function for simplifing as because we will change only Pressure variable
mv = @(m, p) GetMueDev(NGas,1,m,p*101.3,v_min,t);
van_mv = [t' mv(1,2)' mv(1,5)' mv(1,10)' mv(1,15)' mv(1,20)' mv(1,25)' mv(1,30)'];
red_mv = [t' mv(2,2)' mv(2,5)' mv(2,10)' mv(2,15)' mv(2,20)' mv(2,25)' mv(2,30)'];
bwr_mv = [t' mv(3,2)' mv(3,5)' mv(3,10)' mv(3,15)' mv(3,20)' mv(3,25)' mv(3,30)'];

%converting to cells to be able to write charachters into excell
van_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(van_mv)];
red_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(red_mv)];
bwr_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(bwr_mv)];

xlswrite ('DataGraphing\Actual Joule Thomson\mue_deviation_MIN_v.xls',van_mv,'Poly VAN');
xlswrite ('DataGraphing\Actual Joule Thomson\mue_deviation_MIN_v.xls',red_mv,'Poly Redlish');
xlswrite ('DataGraphing\Actual Joule Thomson\mue_deviation_MIN_v.xls',bwr_mv,'Poly BWR');

%----------------------------------------

disp('6- Delta CpCv ');
%van
dcpv_van_maxvol = GetDCpCvData(NGas,1,vol(1));
dcpv_van_minvol = GetDCpCvData(NGas,1,vol(2));
%redlish
```

```
dcpv_red_maxvol = GetDCpCvData(NGas,2,vol(1));
dcpv_red_minvol = GetDCpCvData(NGas,2,vol(2));
%BWR
dcpv_bwr_maxvol = GetDCpCvData(NGas,3,vol(1));
dcpv_bwr_minvol = GetDCpCvData(NGas,3,vol(2));
%BWR
dcpv_exp_maxvol = GetDCpCvData(NGas,4,vol(1));
dcpv_exp_minvol = GetDCpCvData(NGas,4,vol(2));

%write to excell sheets
xlswrite ('DataGraphing\DeltaCpCv_R.xls',dcpv_van_maxvol,'van_maxv');
xlswrite ('DataGraphing\DeltaCpCv_R.xls',dcpv_van_minvol,'van_minv');
xlswrite ('DataGraphing\DeltaCpCv_R.xls',dcpv_red_maxvol,'redlish_maxv');
xlswrite ('DataGraphing\DeltaCpCv_R.xls',dcpv_red_minvol,'redlish_minv');
xlswrite ('DataGraphing\DeltaCpCv_R.xls',dcpv_bwr_maxvol,'BWR_maxv');
xlswrite ('DataGraphing\DeltaCpCv_R.xls',dcpv_bwr_minvol,'BWR_minv');
xlswrite ('DataGraphing\DeltaCpCv_R.xls',dcpv_bwr_maxvol,'Exp_maxv');
xlswrite ('DataGraphing\DeltaCpCv_R.xls',dcpv_bwr_minvol,'Exp_minv');

%----------------------
disp('6a- %Cv Data at minimum v  and p=[2 5 10 15 20 25 30]');
% mv function for simplifing as because we will change only Pressure variable
mv = @(m, p) GetCvDev(NGas,1,m,p*101.3,v_min,t);
van_mv = [t' mv(1,2)' mv(1,5)' mv(1,10)' mv(1,15)' mv(1,20)' mv(1,25)' mv(1,30)'];
red_mv = [t' mv(2,2)' mv(2,5)' mv(2,10)' mv(2,15)' mv(2,20)' mv(2,25)' mv(2,30)'];
bwr_mv = [t' mv(3,2)' mv(3,5)' mv(3,10)' mv(3,15)' mv(3,20)' mv(3,25)' mv(3,30)'];

%converting to cells to be able to write charachters into excell
van_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(van_mv)];
red_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(red_mv)];
bwr_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(bwr_mv)];

xlswrite ('DataGraphing\Actual Cv\Cv_Percentage_MIN_V.xls',van_mv,'Poly VAN');
xlswrite ('DataGraphing\Actual Cv\Cv_Percentage_MIN_V.xls',red_mv,'Poly Redlich');
xlswrite ('DataGraphing\Actual Cv\Cv_Percentage_MIN_V.xls',bwr_mv,'Poly BWR');
%----------------------

disp('6b- Cv data of minimum v  and p=[2 5 10 15 20 25 30]');

% mv function for simplifing as because we will change only Pressure variable
mv = @(m, p) GetCv(NGas,1,m,p*101.3,v_min,t);
van_mv = [t' mv(1,2)' mv(1,5)' mv(1,10)' mv(1,15)' mv(1,20)' mv(1,25)' mv(1,30)'];
red_mv = [t' mv(2,2)' mv(2,5)' mv(2,10)' mv(2,15)' mv(2,20)' mv(2,25)' mv(2,30)'];
bwr_mv = [t' mv(3,2)' mv(3,5)' mv(3,10)' mv(3,15)' mv(3,20)' mv(3,25)' mv(3,30)'];
exp_mv = [t' mv(4,2)' mv(4,5)' mv(4,10)' mv(4,15)' mv(4,20)' mv(4,25)' mv(4,30)'];

%converting to cells to be able to write charachters into excell
van_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(van_mv)];
red_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(red_mv)];
bwr_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(bwr_mv)];
exp_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(exp_mv)];

xlswrite ('DataGraphing\Actual Cv\Cv_Min_V.xls',van_mv,'poly VAN');
xlswrite ('DataGraphing\Actual Cv\Cv_Min_V.xls',red_mv,'Poly Redlish');
```

```
xlswrite ('DataGraphing\Actual Cv\Cv_Min_V.xls',bwr_mv,'Poly BWR');
xlswrite ('DataGraphing\Actual Cv\Cv_Min_V.xls',exp_mv,'Poly Experimental');
%----------------------
disp('6c- Cv data of minimum v  and p=[2 5 10 15 20 25 30]');

% mv function for simplifing as because we will change only Pressure variable
mv = @(m, p) GetCv(NGas,1,m,p*101.3,v_max,t);
van_mv = [t' mv(1,2)' mv(1,5)' mv(1,10)' mv(1,15)' mv(1,20)' mv(1,25)' mv(1,30)'];
red_mv = [t' mv(2,2)' mv(2,5)' mv(2,10)' mv(2,15)' mv(2,20)' mv(2,25)' mv(2,30)'];
bwr_mv = [t' mv(3,2)' mv(3,5)' mv(3,10)' mv(3,15)' mv(3,20)' mv(3,25)' mv(3,30)'];
exp_mv = [t' mv(4,2)' mv(4,5)' mv(4,10)' mv(4,15)' mv(4,20)' mv(4,25)' mv(4,30)'];

%converting to cells to be able to write charachters into excell
van_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(van_mv)];
red_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(red_mv)];
bwr_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(bwr_mv)];
exp_mv = [{'T', 'P = 2','P = 5', 'P = 10', 'P = 15', 'P = 20', 'P = 25', 'P = 30'} ; num2cell(exp_mv)];

xlswrite ('DataGraphing\Actual Cv\Cv_Max_V.xls',van_mv,'poly VAN');
xlswrite ('DataGraphing\Actual Cv\Cv_Max_V.xls',red_mv,'Poly Redlish');
xlswrite ('DataGraphing\Actual Cv\Cv_Max_V.xls',bwr_mv,'Poly BWR');
xlswrite ('DataGraphing\Actual Cv\Cv_Max_V.xls',exp_mv,'Poly Experimental');
%----------------------
disp('7- Cv data fit');
%van
poly_cv(1) = {GetCvData(NGas,1,1)};
schley_cv(1) = {GetCvData(NGas,2,1)};
%redlish
poly_cv(2) = {GetCvData(NGas,1,2)};
schley_cv(2) = {GetCvData(NGas,2,2)};
%BWR
poly_cv(3) = {GetCvData(NGas,1,3)};
schley_cv(3) = {GetCvData(NGas,2,3)};
%Expieremntal
poly_cv(4) = {GetCvData(NGas,1,4)};
schley_cv(4) = {GetCvData(NGas,2,4)};

%write to excell sheets
xlswrite ('DataFitting\CvData.xls',poly_cv{1},'polyvan');
xlswrite ('DataFitting\CvData.xls',schley_cv{1},'schleyvan');
xlswrite ('DataFitting\CvData.xls',poly_cv{2},'polyRedlish');
xlswrite ('DataFitting\CvData.xls',schley_cv{2},'schleyRedlish');
xlswrite ('DataFitting\CvData.xls',poly_cv{3},'polyBWR');
xlswrite ('DataFitting\CvData.xls',schley_cv{3},'schleyBWR');
xlswrite ('DataFitting\CvData.xls',poly_cv{4},'poly_Exp');
xlswrite ('DataFitting\CvData.xls',schley_cv{4},'schley_Exp');
%----------------------

disp('7a- Cv deviation');
p_mue(1) = {GetCv_Dev_Data(NGas,1,1)};
s_mue(1) = {GetCv_Dev_Data(NGas,2,1)};
p_mue(2) = {GetCv_Dev_Data(NGas,1,2)};
s_mue(2) = {GetCv_Dev_Data(NGas,2,2)};
p_mue(3) = {GetCv_Dev_Data(NGas,1,3)};
s_mue(3) = {GetCv_Dev_Data(NGas,2,3)};
```

```
%write to excell sheets
xlswrite ('DataFitting\CvDeviationData.xls',p_mue{1},'poly_van');
xlswrite ('DataFitting\CvDeviationData.xls',s_mue{1},'schley_van');
xlswrite ('DataFitting\CvDeviationData.xls',p_mue{2},'poly_red');
xlswrite ('DataFitting\CvDeviationData.xls',s_mue{2},'schley_red');
xlswrite ('DataFitting\CvDeviationData.xls',p_mue{3},'poly_bwr');
xlswrite ('DataFitting\CvDeviationData.xls',s_mue{3},'schley_bwr');
```

```
Command Window

>> ExportResults (eng,'X:\College Project\Theoritical Cp,Cv,mue')
Warning: Directory already exists.
> In NaturalGas.ExportResults at 6
Warning: Directory already exists.
> In NaturalGas.ExportResults at 7
Warning: Directory already exists.
> In NaturalGas.ExportResults at 8
Warning: Directory already exists.
> In NaturalGas.ExportResults at 9
Warning: Directory already exists.
> In NaturalGas.ExportResults at 10
1- Calculating Ideal Cp:
2- Calculating Poly-Schley error:
3- Cp Departure Percentage
4- Actual Cp for data fitting
4a- Actual Cp For Grapher directly from the program
4b- Experimental Cp to Theoritical Cp For Grapher by fitted
5- Joule Thomson effect actual
5b- Joule Thomson deviation
5c- Effect Of v on mue
5c- mue data of minimum v  and p=[2 5 10 15 20 25 30]
5d- mue deviation data of minimum v  and p=[2 5 10 15 20 25 30]
6- Delta CpCv
6a- %Cv Data at minimum v  and p=[2 5 10 15 20 25 30]
6b- Cv data of minimum v  and p=[2 5 10 15 20 25 30]
6c- Cv data of minimum v  and p=[2 5 10 15 20 25 30]
7- Cv data fit
7a- Cv deviation
>>
```

Figure 3: A snapshot of the function during exporting the values.

C h a p t e r 11

Calculating Thermodynamic Properties

CALCULATING INTERNAL ENERGY 'U'

$$\int_{n}^{n+1} du = \int_{0}^{T} [C_v] dT + \int_{1.990}^{v} \left[T\left(\frac{\partial P}{\partial T}\right)_v - P \right] dv$$

The first term is for the ideal state so we will get it from the ideal equations of Cv but we don't have a direct equation for C_v so we will express C_v as $C_v = C_p - R$ which R is the Gas Constant

R could be expressed as $R = \frac{\overline{R}}{M}$ where M is the molecular weight and $\overline{R}$ is global gas constant

R = 8.314472 kJ/(kmol.K)

The program has a function "GetMixtureR" for obtaining the R directly

$$\int_{T_n}^{T_{n+1}} [C_v] dT = \int_{T_n}^{T_{n+1}} \left[\frac{\overline{C_p}}{M} - \frac{\overline{R}}{M} \right] dt = \frac{\left[aT + \frac{b}{2}T^2 + \frac{c}{3}T^3 + \frac{d}{4}T^4 - \overline{R}T \right]_n^{n+1}}{M}$$

Our fitted Cp is already fitted with the division on molecular weight

So

$$C_p = \frac{\overline{C_p}}{M} = \frac{a + bT + cT^2 + dT^3}{M}$$

Final Form of integration:

$$\int_{0}^{T} [C_v] dT = \int_{0}^{T} \left[C_p - \frac{\overline{R}}{M} \right] dt = \left[aT + \frac{b}{2}T^2 + \frac{c}{3}T^3 + \frac{d}{4}T^4 - \frac{\overline{R}T}{M} \right]_0^T$$

Calculating the Second Term:

$$\int_{1.990}^{v} \left[T\left(\frac{\partial P}{\partial T}\right)_v - P \right] dv$$

$$T\left(\frac{\partial P}{\partial T}\right)_v = cT + 2eT^2 + \frac{f}{v}T + 3hT^3 + 2\frac{i}{v}T^2 + \frac{j}{v^2}$$

$$T\left(\frac{\partial P}{\partial T}\right)_v - P = eT^2 + 2hT^3 + \frac{i}{v}T^2 - a - \frac{b}{v} - \frac{d}{v^2} - \frac{g}{v^3}$$

$$\int_{1.990}^{v} \left[T\left(\frac{\partial P}{\partial T}\right)_v - P \right] dv = \left[eT^2 v + 2hT^3 v + iT^2 \ln v - av - b\ln v + \frac{d}{v} + \frac{1}{2}\frac{g}{v^2} \right]_{1.990}^{v}$$

@NaturalGas\GetU.m

```
function cu = GetU(NGas, Method, v, T)
%u(Gas, Method, v, T) Summary of this function goes her
%  Calculating the ideal internal Energy at chosen temperature
% the relation of

% the constants from fitted equation of ideal mixture poly cp
% the constants are for the whole Cp not

a = 0.844916445395786;
b = 3.88545432275805E-03;
c = -3.7399494251652E-07;
d = -2.83571472845517E-10;



   function nu = gu(T)
      if T<=0
         nu = 0;
      else
         Tb=0;
         nu = (a .* T) + (b./2 .* T.^2) + (c./3 .* T.^3) + ...
              (d./4 .* T.^4) - (GetMixtureR(NGas) .* T) - ...
          ((a .* Tb) + (b./2 .* Tb.^2) + (c./3 .* Tb.^3) + ...
          (d./4 .* Tb.^4) - (GetMixtureR(NGas) .* Tb));
     end
   end

a1 = NGas.CpCvConstants(1,Method);
b1 = NGas.CpCvConstants(2,Method);
c1 = NGas.CpCvConstants(3,Method);
d1 = NGas.CpCvConstants(4,Method);
e1 = NGas.CpCvConstants(5,Method);
f1 = NGas.CpCvConstants(6,Method);
g1 = NGas.CpCvConstants(7,Method);
h1 = NGas.CpCvConstants(8,Method);
i1 = NGas.CpCvConstants(9,Method);
j1 = NGas.CpCvConstants(10,Method);

   function nua = gua(v)
      vbase = 0.5885;

      tmp1 = ((e1 .* T.^2 .* v) + (2 .* h1 .* T.^3 .* v) + ...
           (i1 .* T.^2 .* log(v)) - (a1 .* v) - (b1 .* log(v)) + ...
           (d1 ./ v) + (0.5 .* g1 ./ v.^2));

      tmp2 = (e1 .* T.^2 .* vbase) + (2 .* h1 .* T.^3 .* vbase) + ...
           (i1 .* T.^2 .* log(vbase)) - (a1 .* vbase) - (b1 .* log(vbase)) + ...
           (d1 ./ vbase) + (0.5 .* g1 ./ vbase.^2);

      nua = tmp1 - tmp2;
   end
%cu = gu(T) + gua(v);
cu = GetUFromH(NGas, Method, v, T);
end
```

CALCULATING ENTHALPY 'H'

$$\int_{n}^{n+1} dh = \int_{T_n}^{T_{n+1}} [C_p]dT + \int_{P_n}^{P_{n+1}} \left[v - T\left(\frac{\partial v}{\partial T}\right)_P \right] dp$$

$$\int_{0}^{T} [C_p]dT = \int_{0}^{T} [C_p]dT = \left[aT + \frac{b}{2}T^2 + \frac{c}{3}T^3 + \frac{d}{4}T^4 \right]_0^T$$

Calculating second term:

$$T\left(\frac{\partial v}{\partial T}\right) = cT + 2eT^2 + \frac{f}{P}T + 3hT^3 + 2\frac{i}{P}T^2 + \frac{j}{P^2}T$$

$$v - T\left(\frac{\partial v}{\partial T}\right)_P = a + \frac{b}{P} + \frac{d}{P^2} - eT^2 + \frac{g}{P^3} - 2hT^3 - \frac{i}{P}T^2$$

$$\int_{0}^{P} \left[v - T\left(\frac{\partial v}{\partial T}\right)_P \right] dp = aP + b\ln P - \frac{d}{P} - eT^2P - \frac{1}{2}\frac{g}{P^2} - 2hT^3P - iT^2 \ln P$$

@NaturalGas\GetH.m

```
function cu = GetH(NGas, Method, P, T)
%h(Gas, Method, P, v, T) Summary of this function goes her
%  Calculating the ideal internal Energy at chosen temperature
% the relation of



% the constants from fitted equation of ideal mixture poly cp
% the constants are for the whole Cp not

a = 0.844916445395786;
b = 3.88545432275805E-03;
c = -3.7399494251652E-07;
d = -2.83571472845517E-10;

% specify interval for 200 iteration
deltaT = T/200;

   %making a function which will be called many times recusrsively
   function nh = gh(T)
      if T<=0
         nh = 0;
      else
         nh = (a .* T) + ((b./2) .* T.^2) + ((c./3) .* T.^3) + ...
             ((d./4) .* T.^4) - gh(0);
     end
   end
```

```
a1 = NGas.EquationConstants(1,Method);
b1 = NGas.EquationConstants(2,Method);
c1 = NGas.EquationConstants(3,Method);
d1 = NGas.EquationConstants(4,Method);
e1 = NGas.EquationConstants(5,Method);
f1 = NGas.EquationConstants(6,Method);
g1 = NGas.EquationConstants(7,Method);
h1 = NGas.EquationConstants(8,Method);
i1 = NGas.EquationConstants(9,Method);
j1 = NGas.EquationConstants(10,Method);

   function nha = gha(P)
      Pbase = 202.6;
      tmp1 = (a1 .* P) + (b1 .* log(P)) - (d1 ./ P) - (e1 .* T.^2 .* P) ...
          - (0.5 .* g1 ./ P.^2) - (2 .* h1 .* T.^3 .* P) ...
          - (i1.* T.^2 .* log(P));

      tmp2 = (a1 .* Pbase) + (b1 .* log(Pbase)) - (d1 ./ Pbase) - (e1 .* T.^2 .* Pbase) ...
          - (0.5 .* g1 ./ Pbase.^2) - (2 .* h1 .* T.^3 .* Pbase) ...
          - (i1.* T.^2 .* log(Pbase));

          nha  =  tmp1 - tmp2;

   end

cu = gh(T) + gha(P);
%cu = gha(P);
end
```

CALCULATING ENTROPY 'S'

$$\int_{n}^{n+1} ds = \int_{T_n}^{T_{n+1}} \left[\frac{C_p}{T}\right] dt + \int_{P_n}^{P_{n+1}} \left[\left(\frac{\partial v}{\partial T}\right)_P\right] dp$$

$$\int_{0}^{T} ds = \int_{0}^{T} \left[\frac{C_p}{T}\right] dt = \left[a \ln T + bT + \frac{c}{2}T^2 + \frac{d}{3}T^3\right]_0^T$$

$$\int_{0}^{P} \left[\left(\frac{\partial v}{\partial T}\right)_P\right] dp = cP + 2eTP + f \ln P + 3hT^2P + 2iT \ln P - \frac{j}{P}$$

@NaturalGas\GetS.m

```
function cu = GetS(NGas, Method, P, T)
%h(Gas, Method, P, v, T) Summary of this function goes her
%  Calculating the ideal internal Energy at chosen temperature
% the relation of
```

```
% the constants from fitted equation of ideal mixture poly cp
% the constants are for the whole Cp not

a = 0.844916445395786;
b = 3.88545432275805E-03;
c = -3.7399494251652E-07;
d = -2.83571472845517E-10;

   %making a function which will be called many times recusrsively
   function ns = gs(T)
      if T<=0
         ns = 0;
      else
         ns = (a .* log(T)) + (b .* T) + ((c./2) .* T.^2) ...
           + ((d./3) .* T.^3) - gs(0);
     end
   end


a1 = NGas.EquationConstants(1,Method);
b1 = NGas.EquationConstants(2,Method);
c1 = NGas.EquationConstants(3,Method);
d1 = NGas.EquationConstants(4,Method);
e1 = NGas.EquationConstants(5,Method);
f1 = NGas.EquationConstants(6,Method);
g1 = NGas.EquationConstants(7,Method);
h1 = NGas.EquationConstants(8,Method);
i1 = NGas.EquationConstants(9,Method);
j1 = NGas.EquationConstants(10,Method);

   function nsa = gsa(P)
      Pbase=202.6;

      tmp1 = (c1 .* P) + (2 .* e1 .* T .* P) + (f1 .* log(P)) ...
          + (3 .* h1 .* T.^2 .* P) + (2 .* i1 .* T .* log(P)) ...
          - (j1 ./ P);

       tmp2 = (c1 .* Pbase) + (2 .* e1 .* T .* Pbase) + (f1 .* log(Pbase)) ...
          + (3 .* h1 .* T.^2 .* Pbase) + (2 .* i1 .* T .* log(Pbase)) ...
          - (j1 ./ Pbase);

      nsa = tmp1 - tmp2;
   end

cu = gs(T) + gsa(P);
end
```

EXPORT FUNCTION FOR THERMODYNAMIC PROPERTIES

The following function is responsible to export all the required data for graphing and fitting to excel files into the desired folder.

@NaturalGas\.m

```
function ExportProps(NGas , Folder)
%ExportProps(NGas , Folder)
%  Exporting the results of H, U, S

ExpFolder = strcat(Folder,'\ThermoProperties');
mkdir(ExpFolder);



pBar = @(x) x .* 101.3;

v = [1.99 0.5885 0.30977 0.14714 0.13078 0.1177];
T = [300:25:800];
P = pBar([2 5:5:30]);



disp('1) Exporting u(v,T)');

for i=[1:4]
dp1 = [T' GetU(NGas,i,v(1),T)'];
dp1 = [dp1 GetU(NGas,i,v(2),T)'];
dp1 = [dp1 GetU(NGas,i,v(3),T)'];
dp1 = [dp1 GetU(NGas,i,v(4),T)'];
dp1 = [dp1 GetU(NGas,i,v(5),T)'];
dp1 = [dp1 GetU(NGas,i,v(6),T)'];
dp1 = num2cell(dp1);
dp1 = [{'T', strcat('v=',num2str(v(1))),strcat('v=',num2str(v(2))),strcat('v=',num2str(v(3))), ...
    strcat('v=',num2str(v(4))), strcat('v=',num2str(v(5))), strcat('v=',num2str(v(6)))} ; dp1];

dp2 = [T' repmat(v(1),length(T),1) GetU(NGas,i,v(1),T)'];
dp2 = [dp2; T' repmat(v(2),length(T),1) GetU(NGas,i,v(2),T)'];
dp2 = [dp2; T' repmat(v(3),length(T),1) GetU(NGas,i,v(3),T)'];
dp2 = [dp2; T' repmat(v(4),length(T),1) GetU(NGas,i,v(4),T)'];
dp2 = [dp2; T' repmat(v(5),length(T),1) GetU(NGas,i,v(5),T)'];
dp2 = [dp2; T' repmat(v(6),length(T),1) GetU(NGas,i,v(6),T)'];
dp2 = num2cell(dp2);
dp2 = [{'T', 'v', 'u'}; dp2];

    switch i
        case 1
            sheetname = 'van';
        case 2
            sheetname = 'red';
        case 3
            sheetname = 'bwr';
        case 4
            sheetname = 'exp';
```

```
    end

    xlswrite (strcat(ExpFolder,'\ng_u.xls'),dp1,sheetname);
    xlswrite (strcat(ExpFolder,'\ng_u_fit.xls'),dp2,sheetname);
end



disp('2) Exporting h(P,T)');

for i=[1:4]
dp1 = [T' GetH(NGas,i,P(1),T)'];
dp1 = [dp1 GetH(NGas,i,P(2),T)'];
dp1 = [dp1 GetH(NGas,i,P(3),T)'];
dp1 = [dp1 GetH(NGas,i,P(4),T)'];
dp1 = [dp1 GetH(NGas,i,P(5),T)'];
dp1 = [dp1 GetH(NGas,i,P(6),T)'];
dp1 = [dp1 GetH(NGas,i,P(7),T)'];
dp1 = num2cell(dp1);
dp1 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))), ...
    strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
    strcat('P=',num2str(P(7)))} ; dp1];

dp2 = [T' repmat(P(1),length(T),1) GetH(NGas,i,P(1),T)'];
dp2 = [dp2; T' repmat(P(2),length(T),1) GetH(NGas,i,P(2),T)'];
dp2 = [dp2; T' repmat(P(3),length(T),1) GetH(NGas,i,P(3),T)'];
dp2 = [dp2; T' repmat(P(4),length(T),1) GetH(NGas,i,P(4),T)'];
dp2 = [dp2; T' repmat(P(5),length(T),1) GetH(NGas,i,P(5),T)'];
dp2 = [dp2; T' repmat(P(6),length(T),1) GetH(NGas,i,P(6),T)'];
dp2 = num2cell(dp2);
dp2 = [{'T', 'P', 'h'}; dp2];

    switch i
        case 1
            sheetname = 'van';
        case 2
            sheetname = 'red';
        case 3
            sheetname = 'bwr';
        case 4
            sheetname = 'exp';
    end

    xlswrite (strcat(ExpFolder,'\ng_h.xls'),dp1,sheetname);
    xlswrite (strcat(ExpFolder,'\ng_h_fit.xls'),dp2,sheetname);
end



disp('3) Exporting s(P,T)');

for i=[1:4]
dp1 = [T' GetS(NGas,i,P(1),T)'];
```

```
dp1 = [dp1 GetS(NGas,i,P(2),T)'];
dp1 = [dp1 GetS(NGas,i,P(3),T)'];
dp1 = [dp1 GetS(NGas,i,P(4),T)'];
dp1 = [dp1 GetS(NGas,i,P(5),T)'];
dp1 = [dp1 GetS(NGas,i,P(6),T)'];
dp1 = [dp1 GetS(NGas,i,P(7),T)'];
dp1 = num2cell(dp1);
dp1 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))), ...
   strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
   strcat('P=',num2str(P(7)))} ; dp1];

dp2 = [T' repmat(P(1),length(T),1) GetS(NGas,i,P(1),T)'];
dp2 = [dp2; T' repmat(P(2),length(T),1) GetS(NGas,i,P(2),T)'];
dp2 = [dp2; T' repmat(P(3),length(T),1) GetS(NGas,i,P(3),T)'];
dp2 = [dp2; T' repmat(P(4),length(T),1) GetS(NGas,i,P(4),T)'];
dp2 = [dp2; T' repmat(P(5),length(T),1) GetS(NGas,i,P(5),T)'];
dp2 = [dp2; T' repmat(P(6),length(T),1) GetS(NGas,i,P(6),T)'];
dp2 = num2cell(dp2);
dp2 = [{'T', 'P', 's'}; dp2];

   switch i
      case 1
         sheetname = 'van';
      case 2
         sheetname = 'red';
      case 3
         sheetname = 'bwr';
      case 4
         sheetname = 'exp';
   end

   xlswrite (strcat(ExpFolder,'\ng_s.xls'),dp1,sheetname);
   xlswrite (strcat(ExpFolder,'\ng_s_fit.xls'),dp2,sheetname);
end

% %internal functions to get u, h, and s  departures
% %-------------------------------------------------
function dep = GetDepU(NGas, Method, v, T)
   actual = GetU(NGas, Method, v, T);
   ideal = GetU(NGas, Method, 1.990, T);
   dep = ((actual - ideal)./ideal).*100;
end

function dep=GetDepH(NGas, Method, P, T)
   actual = GetH(NGas, Method, P, T);
   ideal = GetH(NGas, Method, 202.6, T);
   dep = ((actual - ideal)./ideal).*100;
end

function dep=GetDepS(NGas, Method, P, T)
   actual = GetS(NGas, Method, P, T);
   ideal = GetS(NGas, Method, 202.6, T);
   dep = ((actual - ideal)./ideal).*100;
end
%-------------------------------------------------
```

```
disp('4) Exporting Departure u(v,T)');

for i=[1:4]
dp1 = [T' GetDepU(NGas,i,v(1),T)'];
dp1 = [dp1 GetDepU(NGas,i,v(2),T)'];
dp1 = [dp1 GetDepU(NGas,i,v(3),T)'];
dp1 = [dp1 GetDepU(NGas,i,v(4),T)'];
dp1 = [dp1 GetDepU(NGas,i,v(5),T)'];
dp1 = [dp1 GetDepU(NGas,i,v(6),T)'];
dp1 = num2cell(dp1);
dp1 = [{'T', strcat('v=',num2str(v(1))),strcat('v=',num2str(v(2))),strcat('v=',num2str(v(3))), ...
   strcat('v=',num2str(v(4))), strcat('v=',num2str(v(5))), strcat('v=',num2str(v(6)))} ; dp1];
   switch i
      case 1
         sheetname = 'van';
      case 2
         sheetname = 'red';
      case 3
         sheetname = 'bwr';
      case 4
         sheetname = 'exp';
   end

   xlswrite (strcat(ExpFolder,'\ng_u_departure.xls'),dp1,sheetname);
end



disp('5) Exporting Departure h(P,T)');

for i=[1:4]
dp1 = [T' GetDepH(NGas,i,P(1),T)'];
dp1 = [dp1 GetDepH(NGas,i,P(2),T)'];
dp1 = [dp1 GetDepH(NGas,i,P(3),T)'];
dp1 = [dp1 GetDepH(NGas,i,P(4),T)'];
dp1 = [dp1 GetDepH(NGas,i,P(5),T)'];
dp1 = [dp1 GetDepH(NGas,i,P(6),T)'];
dp1 = [dp1 GetDepH(NGas,i,P(7),T)'];
dp1 = num2cell(dp1);
dp1 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))), ...
   strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
   strcat('P=',num2str(P(7)))} ; dp1];

   switch i
      case 1
         sheetname = 'van';
      case 2
         sheetname = 'red';
      case 3
         sheetname = 'bwr';
      case 4
         sheetname = 'exp';
   end
```

```
    xlswrite (strcat(ExpFolder,'\ng_h_departure.xls'),dp1,sheetname);
end



disp('6) Exporting Departure s(P,T)');

for i=[1:4]
dp1 = [T' GetDepS(NGas,i,P(1),T)'];
dp1 = [dp1 GetDepS(NGas,i,P(2),T)'];
dp1 = [dp1 GetDepS(NGas,i,P(3),T)'];
dp1 = [dp1 GetDepS(NGas,i,P(4),T)'];
dp1 = [dp1 GetDepS(NGas,i,P(5),T)'];
dp1 = [dp1 GetDepS(NGas,i,P(6),T)'];
dp1 = [dp1 GetDepS(NGas,i,P(7),T)'];
dp1 = num2cell(dp1);
dp1 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))), ...
    strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
    strcat('P=',num2str(P(7)))} ; dp1];

    switch i
        case 1
            sheetname = 'van';
        case 2
            sheetname = 'red';
        case 3
            sheetname = 'bwr';
        case 4
            sheetname = 'exp';
    end

    xlswrite (strcat(ExpFolder,'\ng_s_departure.xls'),dp1,sheetname);
end

end
```

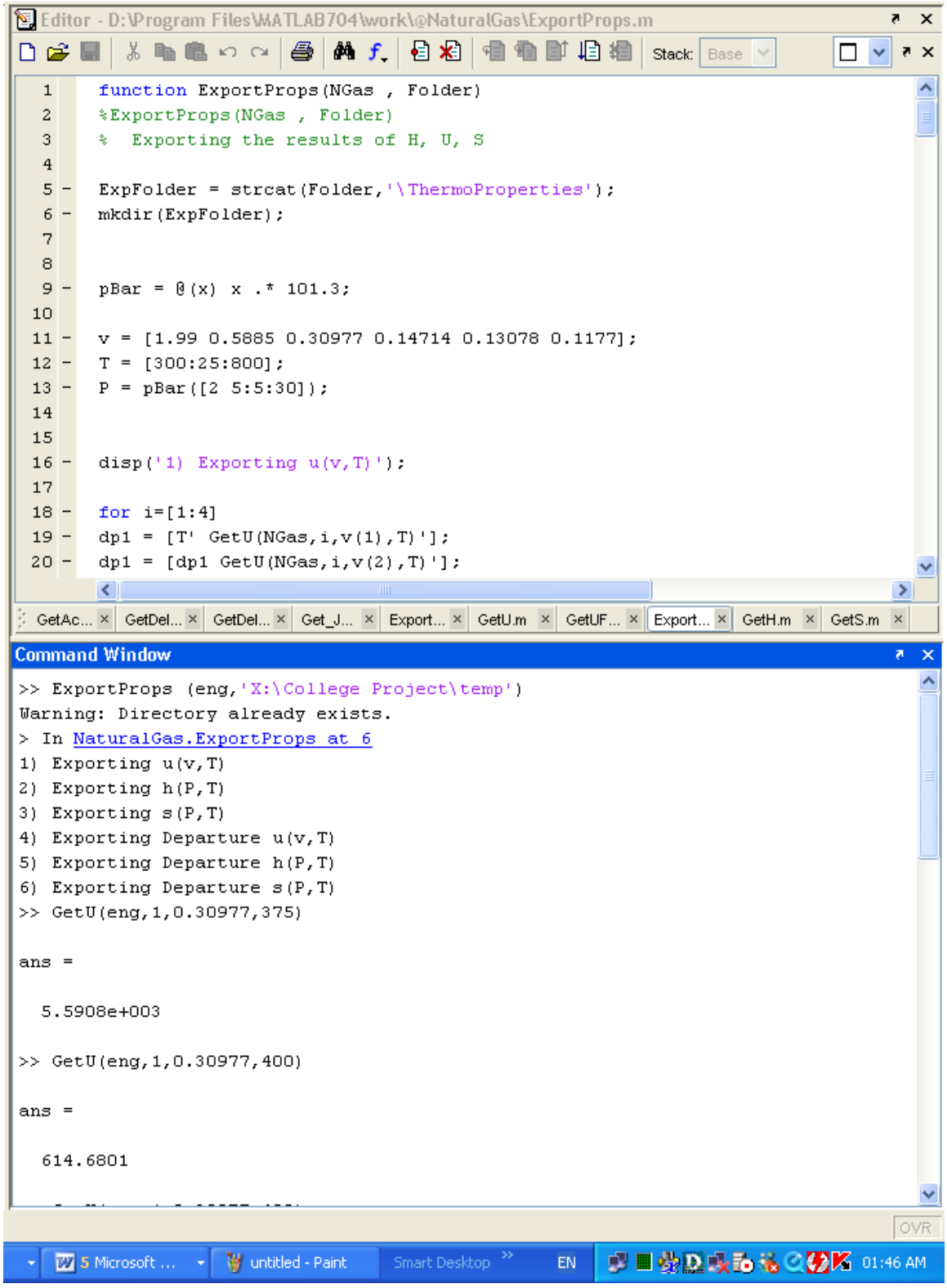
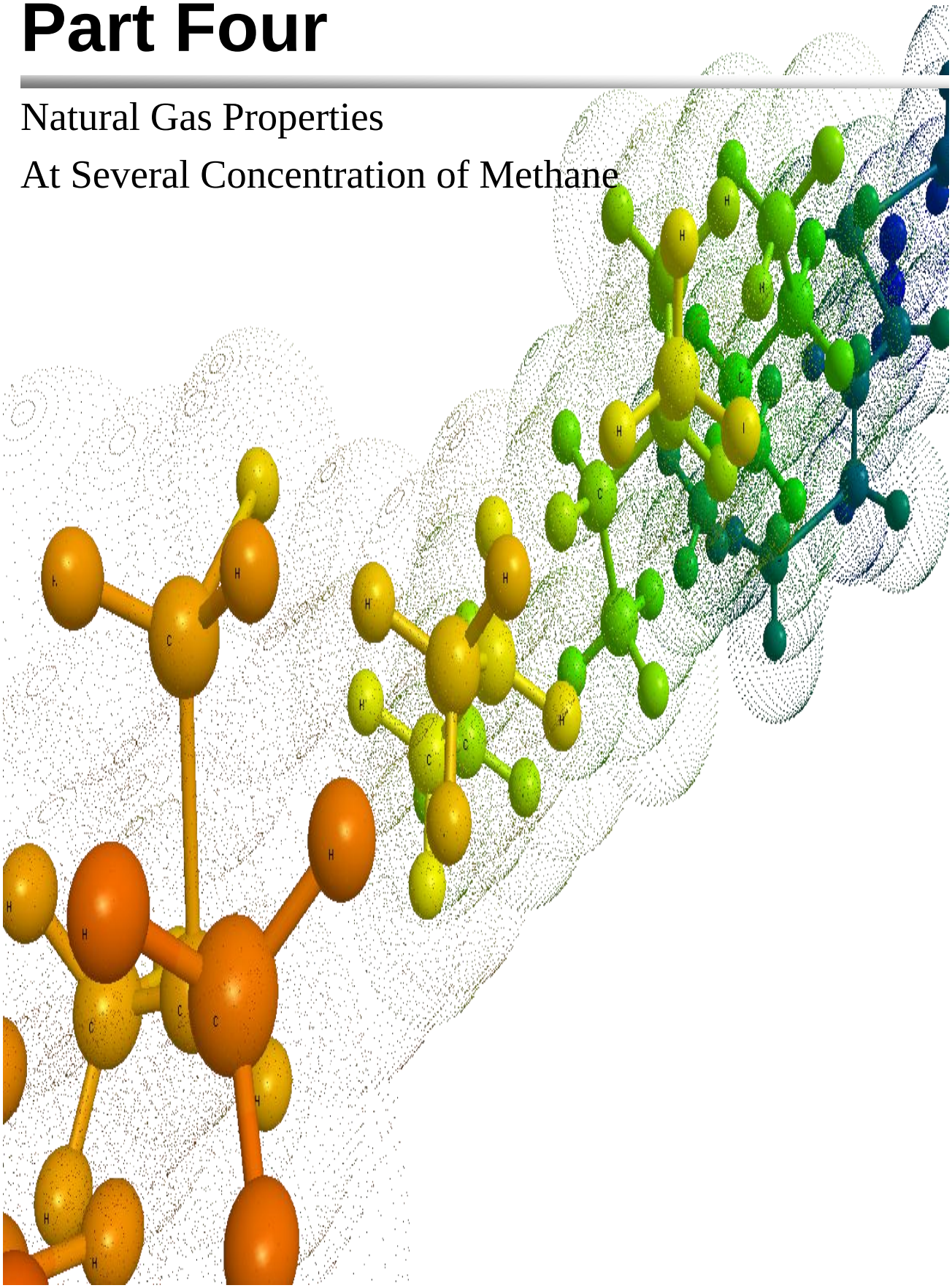


Figure 4: A snapshot of the function during exporting properties.

Part Four

Natural Gas Properties

At Several Concentration of Methane



C h a p t e r 12

GNaturalGas Class

This class is for calculating the natural gas properties at specific Methane concentrations.

The concentrations used in the data are 9 different concentrations for Methane.

The mole fractions are

Methane	0.6005	0.76	0.79044	0.8309	0.8545	0.8811	0.9201	0.9329	1
Ethane	0.2272	0.1201	0.11419	0.0999	0.0661	0.0713	0.0456	0.0457	0
Propane	0.1498	0.067	0.04717	0.0232	0.0245	0.0138	0.0086	0.009	0
Isobutane	0.0023	0.0087	0.00366	0.0012	0.005	0.0008	0.0003	0.0004	0
n-butane	0.0038	0.0166	0.00361	0.0011	0.0062	0.0007	0.0002	0.0002	0
Isopentane	0.0001	0.0044	0.00037	0	0.002	0.0001	0.0006	0	0
n-Pentane	0.0001	0.0039	0.00032	0	0.0012	0.0001	0.0004	0	0
n-Hexane	0	0	0.0002	0.0003	0.0015	0.0001	0	0	0
Nitrogen	0.0044	0.00244	0.0037	0.0032	0.0056	0.0265	0.0034	0.0028	0
CarbonDioxide	0.0119	0.0087	0.03664	0.0405	0.0334	0.03664	0.0207	0.0092	0

@GNATURALGAS IMPLEMENTATION

Constructor

@GNaturalGas\GNaturalGas.m

```
function ngas = GNaturalGas(Name,Concentration_Index)
%NaturalGas Class
%   (Name,Gases,Fractions)
% Holding Array of gases and their corresponding mole fractions
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

[gaspolydata,gasnames] =xlsread('Concentrations','poly');
[fractions] = xlsread('Concentrations','Fractions');
[gasmw] = xlsread('Concentrations','mweight');



for i=1:length(gasnames)
       mGas(i) = Gas(gasnames(i),gasmw(i));
       mGas(i) = SetPolyConstants(mGas(i),gaspolydata(i,:));
end

ng.Name = cellstr(Name);
ng.Gases = mGas;
ng.Fractions = fractions(:,Concentration_Index);
ng.Concentration_Index=Concentration_Index;

ng.v_const_van = xlsread('v_const','van');
ng.v_const_red = xlsread('v_const','red');
ng.v_const_bwr = xlsread('v_const','bwr');

ng.p_const_van = xlsread('p_const','van');
ng.p_const_red = xlsread('p_const','red');
ng.p_const_bwr = xlsread('p_const','bwr');

ng.IdealCp_const = xlsread('idealcp_const','IdealCp_Const');

ngas = class(ng,'GNaturalGas');
```

@GNaturalGas\display.m

```
function  display(NGas)
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

disp('Natural Gas Class');
ngnum = length(NGas);

if ngnum == 1
   disp('Gas and Fraction');
   gs = [get(NGas.Gases,'Name') cellstr(num2str(NGas.Fractions))];
   disp(gs);
   disp('Total Fraction =');
   disp(sum(NGas.Fractions));
else
```

```
   disp ('Group of natural gases');
end
```

To load @GNaturalGas object with specific concentration you refer to the desired concentration with numbers between [1 – 9].

Example

```
>> gn=GNaturalGas('anyname',3)
Natural Gas Class
Gas and Fraction
    'Methane'          '0.79044'
    'Ethane'          '0.11419'
    'Propane'          '0.04717'
    'Isobutane'        '0.00366'
    'n-butane'         '0.00361'
    'Isopentane'       '0.00037'
    'n-Pentane'        '0.00032'
    'n-Hexane'          ' 0.0002'
    'Nitrogen'         ' 0.0037'
    'CarbonDioxide'    '0.03664'

Total Fraction =
    1.0003

To get the Cv at specific PVT you can use this
>> GetCv(gn,2,500,0.5,500)

ans =

    2.3191
```

C h a p t e r 13

Calculating Properties

All properties will be calculated but with taken the concentration of methane into consideration so we check if the change of concentration have an impact of the natural gas behaviour.

All calculations are based on theoretical Equation of States.

IDEAL CP

@GNaturalGas\GetIdealCp.m

```
function cp = GetIdealCp(NGas, Temperature)
%GetIdealCp (NGas, MethodNum, Temperature)
%  Get the Ideal isobaric specific heat for the mixture
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

%disp(NGas.Fractions);
tempnum = length(Temperature);

tcp = GetPolyCp(NGas.Gases, Temperature);
mfr = repmat(NGas.Fractions,1,tempnum);
cp = sum (mfr .* tcp);

cp = cp ./ GetMW(NGas);
```

ACTUAL CP

@GNaturalGas\GetCpDeparture.m

```
function [ dcp ] = GetCpDeparture(GNG, Method, P, T)
%GetCpDeparture Summary of this function goes here
%  Detailed explanation goes here

switch Method
   case 1
      const = GNG.v_const_van; %constants of v of vanderval
   case 2
      const = GNG.v_const_red; %constants of v of redlish
   case 3
      const = GNG.v_const_bwr; %constants of v of bwr
end

a = const(1,GNG.Concentration_Index);
b = const(2,GNG.Concentration_Index);
c = const(3,GNG.Concentration_Index);
d = const(4,GNG.Concentration_Index);
e = const(5,GNG.Concentration_Index);
f = const(6,GNG.Concentration_Index);
g = const(7,GNG.Concentration_Index);
h = const(8,GNG.Concentration_Index);
i = const(9,GNG.Concentration_Index);
j = const(10,GNG.Concentration_Index);


% dcp = -a .* b .* (b-1) .* (T.^(b-1)).* (1./(c+1));
% dcp = dcp .* ((P.^(c+1)) - ((101.3).^(c+1)));


tmp1 = 2 .* e .* P;
tmp1 = tmp1 + (6 .* h .* T .* P);
tmp1 = tmp1 + (2 .* i .* log(P));
tmp2 = 405.2 .* e;
```

```
tmp2 = tmp2 + (1215.6 .* h .* T);
tmp2 = tmp2 + (2 .* i .* log(202.6));
dcp = - T .* (tmp1 - tmp2);
```

@GNaturalGas\GetActualCp.m

```
function [ cp ] = GetActualCp( GNG, Method, P, T )
%GetActualCp Summary of this function goes here
%  Detailed explanation goes here

cp = GetIdealCp(GNG, T);

cp = cp + GetCpDeparture(GNG, Method, P, T);
```

ACTUAL CV

@GNaturalGas\GetDeltaCpCv.m

```
function [dCpCv] = GetDeltaCpCv(GNG,Method, P, v, T)
%GetDeltaCpCv(NGas,Method,Pressure, Volume, Temperature)
%  Delta Cp Cv = Temperature * dv^2/dT dp/dT
%  where v=F(Temperature) and p=F(Temperature)
%  Method Parameter
%  1- Vanderval Constants
%  2- Redlish Constants
%  3- BWR Constatnts
%  then enter parameters as  P, V, T   ;)
% Copyright 2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com

switch Method
   case 1
      const = GNG.v_const_van; %constants of v of vanderval
      const1 = GNG.p_const_van; %constants of v of vanderval
   case 2
      const = GNG.v_const_red; %constants of v of redlish
      const1 = GNG.p_const_red; %constants of v of redlish
   case 3
      const = GNG.v_const_bwr; %constants of v of bwr
      const1 = GNG.p_const_bwr; %constants of v of bwr
end

a = const(1,GNG.Concentration_Index);
b = const(2,GNG.Concentration_Index);
c = const(3,GNG.Concentration_Index);
d = const(4,GNG.Concentration_Index);
e = const(5,GNG.Concentration_Index);
f = const(6,GNG.Concentration_Index);
g = const(7,GNG.Concentration_Index);
h = const(8,GNG.Concentration_Index);
i = const(9,GNG.Concentration_Index);
j = const(10,GNG.Concentration_Index);

a1 = const1(1,GNG.Concentration_Index);
b1 = const1(2,GNG.Concentration_Index);
```

```
c1 = const1(3,GNG.Concentration_Index);
d1 = const1(4,GNG.Concentration_Index);
e1 = const1(5,GNG.Concentration_Index);
f1 = const1(6,GNG.Concentration_Index);
g1 = const1(7,GNG.Concentration_Index);
h1 = const1(8,GNG.Concentration_Index);
i1 = const1(9,GNG.Concentration_Index);
j1 = const1(10,GNG.Concentration_Index);



   %calculating the second term
   ft = c1 + 2 .* e1 .* T + f1 ./ v;
   ft = ft + 3 .* h1 .* T.^2;
   ft = ft + 2 .* i1 .* T./ v;
   ft = ft + j1 ./ v.^2;

   %calculating the first term
   st = c + 2 .* e .* T + f ./ P + 3 .* h .* T.^2;
   st = st + 2 .* i .* T ./ P + j ./ P.^2;


   dCpCv = (ft .* st) .* T;

end
```

@GNaturalGas\GetCv.m

```
function [ Cv ] = ...
   GetCv(NGas, ActualMethod, Pressure, Volume, Temperature)
%GetCv(NGas, IdealMethod, ActualMethod, Pressure, Volume, Temperature)
%  Specific heat for constant volume :)
%  GetCv(NGas,  ActualMethod, Pressure, Volume, Temperature)
% ActualMethod (1, 2, 3)
% Van, redlish, bwr
% and P V T
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com


Cp = GetActualCp(NGas, ActualMethod, Pressure, Temperature);


dCpCv = GetDeltaCpCv(NGas, ActualMethod, Pressure, Volume, Temperature);

Cv = Cp - dCpCv;
```

JOULE THOMSON EFFECT

@GNaturalGas\Get_J_T_Effect.m

```
function [ mue ] = Get_J_T_Effect ...
    ( GNG, Method, P, V, T )
%Get_J_T_Effect
%  ( NGas, IdealMethod, ActualMethod, Pressure, Volume, Temperature )
%  getting Joule-Thomson Effect
%  In physics, the Joule-Thomson effect, or Joule-Kelvin effect, is
%  a process in which the temperature of a gas is decreased by letting
%  the gas expand adiabatically.
%  It is named for James Prescott Joule and William Thomson,
%  1st Baron Kelvin who established the effect in 1852 following earlier
%  work by Joule on Joule expansion in which a gas expands at constant
%  internal energy.
% ---------------------------------------------------------------------
% ActualMethod (1, 2, 3, 4)
% Van, redlish, bwr, expiremental
% and P V T
% Copyright 2005-2006 By Ahmed Sadek Mohammed Tawfeek ahmed.amara@gmail.com



switch Method
    case 1
        const = GNG.v_const_van; %constants of v of vanderval
    case 2
        const = GNG.v_const_red; %constants of v of redlish
    case 3
        const = GNG.v_const_bwr; %constants of v of bwr
end

a = const(1,GNG.Concentration_Index);
b = const(2,GNG.Concentration_Index);
c = const(3,GNG.Concentration_Index);
d = const(4,GNG.Concentration_Index);
e = const(5,GNG.Concentration_Index);
f = const(6,GNG.Concentration_Index);
g = const(7,GNG.Concentration_Index);
h = const(8,GNG.Concentration_Index);
i = const(9,GNG.Concentration_Index);
j = const(10,GNG.Concentration_Index);




tmp = c + 2 .* e .* T + f ./ P + 3 .* h .* T.^2;
tmp = tmp + 2 .* i .* T ./ P + j ./ P.^2;
tmp = tmp .* T;
tmp = tmp - V;
mue = tmp ./ GetActualCp(GNG, Method, P, T);

end
```

CALCULATING U

@GNaturalGas\GetU.m

```
function cu = GetU(GNG, Method, v, T)
%u(Gas, Method, v, T) Summary of this function goes her
%  Calculating the ideal internal Energy at chosen temperature
% the relation of

% the constants from fitted equation of ideal mixture poly cp
% the constants are for the whole Cp not

a = GNG.IdealCp_const(1,GNG.Concentration_Index);
b = GNG.IdealCp_const(2,GNG.Concentration_Index);
c = GNG.IdealCp_const(3,GNG.Concentration_Index);
d = GNG.IdealCp_const(4,GNG.Concentration_Index);

   function nu = gu(T)
      if T<=0
         nu = 0;
      else
         nu = (a .* T) + (b./2 .* T.^2) + (c./3 .* T.^3) + ...
             (d./4 .* T.^4) - (GetMixtureR(GNG) .* T) - gu(0);
     end
   end



switch Method
   case 1
      const1 = GNG.p_const_van; %constants of v of vanderval
   case 2
      const1 = GNG.p_const_red; %constants of v of redlish
   case 3
      const1 = GNG.p_const_bwr; %constants of v of bwr
end

a1 = const1(1,GNG.Concentration_Index);
b1 = const1(2,GNG.Concentration_Index);
c1 = const1(3,GNG.Concentration_Index);
d1 = const1(4,GNG.Concentration_Index);
e1 = const1(5,GNG.Concentration_Index);
f1 = const1(6,GNG.Concentration_Index);
g1 = const1(7,GNG.Concentration_Index);
h1 = const1(8,GNG.Concentration_Index);
i1 = const1(9,GNG.Concentration_Index);
j1 = const1(10,GNG.Concentration_Index);



   function nua = gua(v)
      vbase = 1.990;

      tmp1 = ((e1 .* T.^2 .* v) + (2 .* h1 .* T.^3 .* v) + ...
           (i1 .* T.^2 .* log(v)) - (a1 .* v) - (b1 .* log(v)) + ...
           (d1 ./ v) + (0.5 .* g1 ./ v.^2));
```

```
        tmp2 = (e1 .* T.^2 .* vbase) + (2 .* h1 .* T.^3 .* vbase) + ...
            (i1 .* T.^2 .* log(vbase)) - (a1 .* vbase) - (b1 .* log(vbase)) + ...
            (d1 ./ vbase) + (0.5 .* g1 ./ vbase.^2);

        nua = tmp1 - tmp2;
    end

%cu = gu(T) + gua(v);
cu = GetUFromH(GNG ,Method, v, T);
end
```

ENTHALPY

@GNaturalGas\GetH.m

```
function cu = GetH(GNG, Method, P, T)
%h(Gas, Method, P, v, T) Summary of this function goes her
%  Calculating the ideal internal Energy at chosen temperature
% the relation of




% the constants from fitted equation of ideal mixture poly cp
% the constants are for the whole Cp not

a = GNG.IdealCp_const(1,GNG.Concentration_Index);
b = GNG.IdealCp_const(2,GNG.Concentration_Index);
c = GNG.IdealCp_const(3,GNG.Concentration_Index);
d = GNG.IdealCp_const(4,GNG.Concentration_Index);




% specify interval for 200 iteration
deltaT = T/200;

    %making a function which will be called many times recusrsively
    function nh = gh(T)
        if T<=0
            nh = 0;
        else
            nh = (a .* T) + ((b./2) .* T.^2) + ((c./3) .* T.^3) + ...
                ((d./4) .* T.^4) - gh(0);
      end
    end




switch Method
    case 1
        const = GNG.v_const_van; %constants of v of vanderval
    case 2
```

```
        const = GNG.v_const_red; %constants of v of redlish
    case 3
        const = GNG.v_const_bwr; %constants of v of bwr
end

a1 = const(1,GNG.Concentration_Index);
b1 = const(2,GNG.Concentration_Index);
c1 = const(3,GNG.Concentration_Index);
d1 = const(4,GNG.Concentration_Index);
e1 = const(5,GNG.Concentration_Index);
f1 = const(6,GNG.Concentration_Index);
g1 = const(7,GNG.Concentration_Index);
h1 = const(8,GNG.Concentration_Index);
i1 = const(9,GNG.Concentration_Index);
j1 = const(10,GNG.Concentration_Index);


    function nha = gha(P)
        Pbase = 202.6;
        tmp1 = (a1 .* P) + (b1 .* log(P)) - (d1 ./ P) - (e1 .* T.^2 .* P) ...
            - (0.5 .* g1 ./ P.^2) - (2 .* h1 .* T.^3 .* P) ...
            - (i1.* T.^2 .* log(P));

        tmp2 = (a1 .* Pbase) + (b1 .* log(Pbase)) - (d1 ./ Pbase) - (e1 .* T.^2 .* Pbase) ...
            - (0.5 .* g1 ./ Pbase.^2) - (2 .* h1 .* T.^3 .* Pbase) ...
            - (i1.* T.^2 .* log(Pbase));

            nha  =  tmp1 - tmp2;

    end

cu = gh(T) + gha(P);
end
```

ENTROPY

@GNaturalGas\GetS.m

```
function cu = GetS(GNG, Method, P, T)
%h(Gas, Method, P, v, T) Summary of this function goes her
%  Calculating the ideal internal Energy at chosen temperature
% the relation of




% the constants from fitted equation of ideal mixture poly cp
% the constants are for the whole Cp not

a = GNG.IdealCp_const(1,GNG.Concentration_Index);
b = GNG.IdealCp_const(2,GNG.Concentration_Index);
c = GNG.IdealCp_const(3,GNG.Concentration_Index);
d = GNG.IdealCp_const(4,GNG.Concentration_Index);



   %making a function which will be called many times recusrsively
   function ns = gs(T)
      if T<=0
         ns = 0;
      else
         ns = (a .* log(T)) + (b .* T) + ((c./2) .* T.^2) ...
           + ((d./3) .* T.^3) - gs(0);
     end
   end



switch Method
   case 1
      const = GNG.v_const_van; %constants of v of vanderval
   case 2
      const = GNG.v_const_red; %constants of v of redlish
   case 3
      const = GNG.v_const_bwr; %constants of v of bwr
end

a1 = const(1,GNG.Concentration_Index);
b1 = const(2,GNG.Concentration_Index);
c1 = const(3,GNG.Concentration_Index);
d1 = const(4,GNG.Concentration_Index);
e1 = const(5,GNG.Concentration_Index);
f1 = const(6,GNG.Concentration_Index);
g1 = const(7,GNG.Concentration_Index);
h1 = const(8,GNG.Concentration_Index);
i1 = const(9,GNG.Concentration_Index);
j1 = const(10,GNG.Concentration_Index);

   function nsa = gsa(P)
      Pbase=202.6;
```

```
        tmp1 = (c1 .* P) + (2 .* e1 .* T .* P) + (f1 .* log(P)) ...
            + (3 .* h1 .* T.^2 .* P) + (2 .* i1 .* T .* log(P)) ...
            - (j1 ./ P);

         tmp2 = (c1 .* Pbase) + (2 .* e1 .* T .* Pbase) + (f1 .* log(Pbase)) ...
            + (3 .* h1 .* T.^2 .* Pbase) + (2 .* i1 .* T .* log(Pbase)) ...
            - (j1 ./ Pbase);

        nsa = tmp1 - tmp2;
    end

cu = gs(T) + gsa(P);
end
```

C h a p t e r 14

Export Function of @GNaturalGas

The function is load the nine concentrations and proceed with calculations

The outpur folder shoud be specified by user.

ExportGNG.m

```
function ExportGNG(Folder)

%load all Compositions
disp('Loading Natural Gases of Several Compositions:');

nn=9;

for iy=1:nn
   disp(num2str(iy));
   gn(iy)=GNaturalGas('gn',iy);
end

T = [300:25:800];
p = @(x) x .* 101.3;

N = [0.600500 0.760000 0.790440 0.830900 0.854500 0.881100 0.920100 0.932900 1.000000];

%pressure=p([50 80 130 210 340 550 890 1440]);
%pressure=p([1 2 5:5:30]);
pressure=p([2 50 100 150 200 250 300 400]);
v=[0.5885 0.30977 0.14714 0.13078 0.11205];
```

```
ExportIdealCp();
ExportDepartureCp();
ExportActualCp();
ExportCv();
ExportMue();
ExportProps();


   function ExportIdealCp()
      disp('Exporting idea; Cp for grapher and datafit');
      Graph = [T' GetIdealCp(gn(1),T)'];

      %T Cp N
      Fitting = [T' repmat(GetMC(gn(1)),length(T),1) GetIdealCp(gn(1),T)'];

      %calculate ideal cp for all concentrations
      for iy=1:nn
         Graph = [Graph GetIdealCp(gn(iy),T)'];
         Fitting = [Fitting ; T' repmat(GetMC(gn(iy)),length(T),1)  GetIdealCp(gn(iy),T)'];
      end

      Graph = num2cell(Graph);
      Graph = [{'T', ...
         strcat('N=',num2str(GetMC(gn(1)))), strcat('N=',num2str(GetMC(gn(2)))), ...
         strcat('N=',num2str(GetMC(gn(3)))), strcat('N=',num2str(GetMC(gn(4)))), ...
         strcat('N=',num2str(GetMC(gn(5)))), strcat('N=',num2str(GetMC(gn(6)))), ...
         strcat('N=',num2str(GetMC(gn(7)))), strcat('N=',num2str(GetMC(gn(8)))), ...
         strcat('N=',num2str(GetMC(gn(9))))}; Graph];

         xlswrite(strcat(Folder,'\Fitting\IdealCp\IdealCp.xls'),Fitting,'IdealCp');
         xlswrite(strcat(Folder,'\Graphs\IdealCp\IdealCp.xls'),Graph,'IdealCp');
   end

   function ExportDepartureCp()
      disp('Exporting Departure Cp for grapher');
      for iy=1:nn
         for ix=1:8
            if ix==1
               dcp = (GetCpDeparture(gn(iy), 1, pressure(ix),T)./GetIdealCp(gn(iy),T)).*100;
               graph = [T'  dcp'];
               dcp2 = (GetCpDeparture(gn(iy), 2, pressure(ix),T)./GetIdealCp(gn(iy),T)).*100;
               graph2 = [T'  dcp'];
               dcp3 = (GetCpDeparture(gn(iy), 3, pressure(ix),T)./GetIdealCp(gn(iy),T)).*100;
               graph3 = [T'  dcp3'];
            else
               dcp = (GetCpDeparture(gn(iy), 1, pressure(ix), T)./GetIdealCp(gn(iy),T)).*100;
               graph = [graph dcp'];
               dcp2 = (GetCpDeparture(gn(iy), 2, pressure(ix), T)./GetIdealCp(gn(iy),T)).*100;
               graph2 = [graph2 dcp2'];
               dcp3 = (GetCpDeparture(gn(iy), 3, pressure(ix), T)./GetIdealCp(gn(iy),T)).*100;
               graph3 = [graph3 dcp3'];
            end
         end
```

```
        %store the graph data
        ns=strcat('N=',num2str(N(iy)));

        graph = num2cell(graph);
        graph = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
            strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph];
        graph2 = num2cell(graph2);
        graph2 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
            strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph2];
        graph3 = num2cell(graph3);
        graph3 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
            strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph3];

        xlswrite(strcat(Folder,'\Graphs\CpDeparture\CpDeparture_van.xls'),graph,ns)
        xlswrite(strcat(Folder,'\Graphs\CpDeparture\CpDeparture_red.xls'),graph2,ns)
        xlswrite(strcat(Folder,'\Graphs\CpDeparture\CpDeparture_bwr.xls'),graph3,ns)
        graph=[];
        graph2=[];
        graph3=[];
    end
end

function ExportActualCp()
    disp('Exporting Actual Cp for grapher and datafit programs');
    for iy=1:nn
        for ix=1:8
            if ix==1 && iy==1
                result = [repmat(pressure(ix),length(T),1) T' ...
                    repmat(N(iy),length(T),1) ...
                    GetActualCp(gn(iy),1,pressure(ix),T)'];
                result2 = [repmat(pressure(ix),length(T),1) T' ...
                    repmat(N(iy),length(T),1) ...
                    GetActualCp(gn(iy),2,pressure(ix),T)'];
                result3 = [repmat(pressure(ix),length(T),1) T' ...
                    repmat(N(iy),length(T),1) ...
                    GetActualCp(gn(iy),3,pressure(ix),T)'];
            else
                result = [result ; repmat(pressure(ix),length(T),1) T' ...
```

```
                repmat(N(iy),length(T),1) ...
                GetActualCp(gn(iy),1,pressure(ix),T)'];
            result2 = [result2 ; repmat(pressure(ix),length(T),1) T' ...
                repmat(N(iy),length(T),1) ...
                GetActualCp(gn(iy),2,pressure(ix),T)'];
            result3 = [result3 ; repmat(pressure(ix),length(T),1) T' ...
                repmat(N(iy),length(T),1) ...
                GetActualCp(gn(iy),3,pressure(ix),T)'];
        end
        if ix==1
            graph = [T'  GetActualCp(gn(iy),1,pressure(ix),T)'];
            graph2 = [T'  GetActualCp(gn(iy),2,pressure(ix),T)'];
            graph3 = [T'  GetActualCp(gn(iy),3,pressure(ix),T)'];
        else
            graph = [graph GetActualCp(gn(iy),1,pressure(ix),T)'];
            graph2 = [graph2 GetActualCp(gn(iy),2,pressure(ix),T)'];
            graph3 = [graph3 GetActualCp(gn(iy),3,pressure(ix),T)'];
        end
    end
    %store the graph data
    ns=strcat('N=',num2str(N(iy)));

    graph = num2cell(graph);
    graph = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
        strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph];
    graph2 = num2cell(graph2);
    graph2 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
        strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph2];
    graph3 = num2cell(graph3);
    graph3 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
        strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph3];

    xlswrite(strcat(Folder,'\Graphs\ActualCp\AcutalCp_van.xls'),graph,ns)
    xlswrite(strcat(Folder,'\Graphs\ActualCp\AcutalCp_red.xls'),graph2,ns)
    xlswrite(strcat(Folder,'\Graphs\ActualCp\AcutalCp_bwr.xls'),graph3,ns)
    graph=[];
    graph2=[];
    graph3=[];
```

```
    end
    xlswrite(strcat(Folder,'\Fitting\ActualCp\AcutalCp.xls'),result,'van');
    xlswrite(strcat(Folder,'\Fitting\ActualCp\AcutalCp.xls'),result2,'red');
    xlswrite(strcat(Folder,'\Fitting\ActualCp\AcutalCp.xls'),result3,'bwr');
end


function ExportCv()
    disp('Exporting CpCv-R for grapher and Cv for datafit programs');
    for iy=1:nn              % for concentrations
       for iz=1:5           % for specific volumes
          for ix=1:8        % for pressure
            if iy==1
                result = [ repmat(v(iz),length(T),1) ...
                           repmat(pressure(ix),length(T),1) ...
                           T' ...
                           repmat(N(iy),length(T),1) ...
                           GetCv(gn(iy),1,pressure(ix),v(iz),T)' ...
                          ];

                result2 = [repmat(v(iz),length(T),1) ...
                           repmat(pressure(ix),length(T),1) ...
                           T' ...
                           repmat(N(iy),length(T),1) ...
                           GetCv(gn(iy),2,pressure(ix),v(iz),T)' ...
                          ];

                result3 = [repmat(v(iz),length(T),1) ...
                           repmat(pressure(ix),length(T),1) ...
                           T' ...
                           repmat(N(iy),length(T),1) ...
                           GetCv(gn(iy),3,pressure(ix),v(iz),T)' ...
                          ];
            else
                result = [result ; repmat(v(iz),length(T),1) ...
                           repmat(pressure(ix),length(T),1) ...
                           T' ...
                           repmat(N(iy),length(T),1) ...
                           GetCv(gn(iy),1,pressure(ix),v(iz),T)' ...
                          ];

                result2 = [result2 ; repmat(v(iz),length(T),1) ...
                           repmat(pressure(ix),length(T),1) ...
                           T' ...
                           repmat(N(iy),length(T),1) ...
                           GetCv(gn(iy),2,pressure(ix),v(iz),T)' ...
                          ];

                result3 = [result3 ; repmat(v(iz),length(T),1) ...
                           repmat(pressure(ix),length(T),1) ...
                           T' ...
                           repmat(N(iy),length(T),1) ...
                           GetCv(gn(iy),3,pressure(ix),v(iz),T)' ...
                          ];
            end
          end
```

```
        end
    end

    xlswrite(strcat(Folder,'\Fitting\Cv\AcutalCv.xls'),result,'van');
    xlswrite(strcat(Folder,'\Fitting\Cv\AcutalCv.xls'),result2,'red');
    xlswrite(strcat(Folder,'\Fitting\Cv\AcutalCv.xls'),result3,'bwr');

    disp('Graphing  Cv for minimum v');
    for iy=1:nn
        for ix=1:8
            if ix==1
                graph = [T'  GetCv(gn(iy),1,pressure(ix),v(5),T)'];
                graph2 = [T'  GetCv(gn(iy),2,pressure(ix),v(5),T)'];
                graph3 = [T'  GetCv(gn(iy),3,pressure(ix),v(5),T)'];
            else
                graph = [graph GetCv(gn(iy),1,pressure(ix),v(5),T)'];
                graph2 = [graph2 GetCv(gn(iy),2,pressure(ix),v(5),T)'];
                graph3 = [graph3 GetCv(gn(iy),3,pressure(ix),v(5),T)'];
            end
        end
        %store the graph data
        ns=strcat('N=',num2str(N(iy)));

        graph = num2cell(graph);
        graph = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
            strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph];
        graph2 = num2cell(graph2);
        graph2 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
            strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph2];
        graph3 = num2cell(graph3);
        graph3 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
            strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph3];

        xlswrite(strcat(Folder,'\Graphs\Cv\Cv_van_min.xls'),graph,ns)
        xlswrite(strcat(Folder,'\Graphs\Cv\Cv_red_min.xls'),graph2,ns)
        xlswrite(strcat(Folder,'\Graphs\Cv\Cv_bwr_min.xls'),graph3,ns)
        graph=[];
        graph2=[];
```

```
        graph3=[];
    end

    disp('Graphing  CpCv-R for minimum v');
    for iy=1:nn
        for ix=1:8
            if ix==1
                graph = [T'  (GetDeltaCpCv(gn(iy),1,pressure(ix),v(5),T)./GetMixtureR(gn(iy)))'];
                graph2 = [T'  (GetDeltaCpCv(gn(iy),2,pressure(ix),v(5),T)./GetMixtureR(gn(iy)))'];
                graph3 = [T'  (GetDeltaCpCv(gn(iy),3,pressure(ix),v(5),T)./GetMixtureR(gn(iy)))'];
            else
                graph =  [graph  (GetDeltaCpCv(gn(iy),1,pressure(ix),v(5),T)./GetMixtureR(gn(iy)))'];
                graph2 = [graph2 (GetDeltaCpCv(gn(iy),2,pressure(ix),v(5),T)./GetMixtureR(gn(iy)))'];
                graph3 = [graph3 (GetDeltaCpCv(gn(iy),3,pressure(ix),v(5),T)./GetMixtureR(gn(iy)))'];
            end
        end
        %store the graph data
        ns=strcat('N=',num2str(N(iy)));

        graph = num2cell(graph);
        graph = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
            strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph];
        graph2 = num2cell(graph2);
        graph2 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
            strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph2];
        graph3 = num2cell(graph3);
        graph3 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
            strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph3];

        xlswrite(strcat(Folder,'\Graphs\Cp-Cv Over R\van_min.xls'),graph,ns)
        xlswrite(strcat(Folder,'\Graphs\Cp-Cv Over R\red_min.xls'),graph2,ns)
        xlswrite(strcat(Folder,'\Graphs\Cp-Cv Over R\bwr_min.xls'),graph3,ns)
        graph=[];
        graph2=[];
        graph3=[];
    end

end
```

```
function ExportMue()
    disp('Exporting mue for grapher and datafit program');
    for iy=1:nn             % for concentrations
        for iz=1:5          % for specific volumes
            for ix=1:8      % for pressure
              if ix==1 && iy==1
                  result = [ repmat(v(iz),length(T),1) ...
                             repmat(pressure(ix),length(T),1) ...
                             T' ...
                             repmat(N(iy),length(T),1) ...
                             Get_J_T_Effect(gn(iy),1,pressure(ix),v(iz),T)' ...
                            ];

                  result2 = [repmat(v(iz),length(T),1) ...
                              repmat(pressure(ix),length(T),1) ...
                              T' ...
                              repmat(N(iy),length(T),1) ...
                              Get_J_T_Effect(gn(iy),2,pressure(ix),v(iz),T)' ...
                             ];

                  result3 = [repmat(v(iz),length(T),1) ...
                              repmat(pressure(ix),length(T),1) ...
                              T' ...
                              repmat(N(iy),length(T),1) ...
                              Get_J_T_Effect(gn(iy),3,pressure(ix),v(iz),T)' ...
                             ];
              else
                  result = [result ; repmat(v(iz),length(T),1) ...
                             repmat(pressure(ix),length(T),1) ...
                             T' ...
                             repmat(N(iy),length(T),1) ...
                             Get_J_T_Effect(gn(iy),1,pressure(ix),v(iz),T)' ...
                            ];

                  result2 = [result2 ; repmat(v(iz),length(T),1) ...
                              repmat(pressure(ix),length(T),1) ...
                              T' ...
                              repmat(N(iy),length(T),1) ...
                              Get_J_T_Effect(gn(iy),2,pressure(ix),v(iz),T)' ...
                             ];

                  result3 = [result3 ; repmat(v(iz),length(T),1) ...
                              repmat(pressure(ix),length(T),1) ...
                              T' ...
                              repmat(N(iy),length(T),1) ...
                              Get_J_T_Effect(gn(iy),3,pressure(ix),v(iz),T)' ...
                             ];
              end
            end
        end
    end

    xlswrite(strcat(Folder,'\Fitting\mue\mue.xls'),result,'van');
    xlswrite(strcat(Folder,'\Fitting\mue\mue.xls'),result2,'red');
    xlswrite(strcat(Folder,'\Fitting\mue\mue.xls'),result3,'bwr');
```

```
        disp('Graphing  mue for minimum v');
        for iy=1:nn
            for ix=1:8
              if ix==1
                 graph = [T'  Get_J_T_Effect(gn(iy),1,pressure(ix),v(5),T)'];
                 graph2 = [T'  Get_J_T_Effect(gn(iy),2,pressure(ix),v(5),T)'];
                 graph3 = [T'  Get_J_T_Effect(gn(iy),3,pressure(ix),v(5),T)'];
              else
                 graph = [graph Get_J_T_Effect(gn(iy),1,pressure(ix),v(5),T)'];
                 graph2 = [graph2 Get_J_T_Effect(gn(iy),2,pressure(ix),v(5),T)'];
                 graph3 = [graph3 Get_J_T_Effect(gn(iy),3,pressure(ix),v(5),T)'];
              end
            end
            %store the graph data
            ns=strcat('N=',num2str(N(iy)));

            graph = num2cell(graph);
            graph = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
                 strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph];
            graph2 = num2cell(graph2);
            graph2 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
                 strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph2];
            graph3 = num2cell(graph3);
            graph3 = [{'T', ...

strcat('P=',num2str(pressure(1))),strcat('P=',num2str(pressure(2))),strcat('P=',num2str(pressure(3)
)), ...

strcat('P=',num2str(pressure(4))),strcat('P=',num2str(pressure(5))),strcat('P=',num2str(pressure(6)
)), ...
                 strcat('P=',num2str(pressure(7))),strcat('P=',num2str(pressure(8)))}; graph3];

            xlswrite(strcat(Folder,'\Graphs\mue\mue_van_min.xls'),graph,ns)
            xlswrite(strcat(Folder,'\Graphs\mue\mue_red_min.xls'),graph2,ns)
            xlswrite(strcat(Folder,'\Graphs\mue\mue_bwr_min.xls'),graph3,ns)
            graph=[];
            graph2=[];
            graph3=[];
        end
    end

    function ExportProps()
        disp('Exporting Thermo Properties for grapher and datafit');
```

```
    P=pressure;

    disp('1) Exporting u(T,v,N)');
    for iy=1:nn             % for concentrations
       for iz=1:5         % for specific volumes
          if iz==1
             %-------- grapher
             gp1 = [T' GetU(gn(iy),1,v(iz),T)'];
             gp2 = [T' GetU(gn(iy),2,v(iz),T)'];
             gp3 = [T' GetU(gn(iy),3,v(iz),T)'];
          else
             %-------- grapher
             gp1 = [gp1 GetU(gn(iy),1,v(iz),T)'];
             gp2 = [gp2 GetU(gn(iy),2,v(iz),T)'];
             gp3 = [gp3 GetU(gn(iy),3,v(iz),T)'];
          end
          if iy==1 && iz==1
             %-------- fittin
             dp1 = [T' repmat(v(iz),length(T),1) repmat(N(iy),length(T),1) GetU(gn(iy),1,v(iz),T)'];
             dp2 = [T' repmat(v(iz),length(T),1) repmat(N(iy),length(T),1) GetU(gn(iy),2,v(iz),T)'];
             dp3 = [T' repmat(v(iz),length(T),1) repmat(N(iy),length(T),1) GetU(gn(iy),3,v(iz),T)'];
          else
             %-------- fittin
             dp1 = [dp1; T' repmat(v(iz),length(T),1) repmat(N(iy),length(T),1)
GetU(gn(iy),1,v(iz),T)'];
             dp2 = [dp2; T' repmat(v(iz),length(T),1) repmat(N(iy),length(T),1)
GetU(gn(iy),2,v(iz),T)'];
             dp3 = [dp3; T' repmat(v(iz),length(T),1) repmat(N(iy),length(T),1)
GetU(gn(iy),3,v(iz),T)'];
          end
       end
       %here storing the u for several v and the current concentration
       %N sheet
       ns=strcat('N=',num2str(N(iy)));

       gp1 = num2cell(gp1);
       gp2 = num2cell(gp2);
       gp3 = num2cell(gp3);

       gp1 = [{'T', strcat('v=',num2str(v(1))),strcat('v=',num2str(v(2))),strcat('v=',num2str(v(3))), ...
       strcat('v=',num2str(v(4))), strcat('v=',num2str(v(5)))} ; gp1];
       gp2 = [{'T', strcat('v=',num2str(v(1))),strcat('v=',num2str(v(2))),strcat('v=',num2str(v(3))), ...
       strcat('v=',num2str(v(4))), strcat('v=',num2str(v(5)))} ; gp2];
       gp3 = [{'T', strcat('v=',num2str(v(1))),strcat('v=',num2str(v(2))),strcat('v=',num2str(v(3))), ...
       strcat('v=',num2str(v(4))), strcat('v=',num2str(v(5)))} ; gp3];

       xlswrite(strcat(Folder,'\Graphs\u\van.xls'),gp1,ns)
       xlswrite(strcat(Folder,'\Graphs\u\red.xls'),gp2,ns)
       xlswrite(strcat(Folder,'\Graphs\u\bwr.xls'),gp3,ns)
       clear gp1;
       clear gp2;
       clear gp3;

    end

    dp1 = num2cell(dp1);
```

```
        dp2 = num2cell(dp2);
        dp3 = num2cell(dp3);
        dp1 = [{'T', 'v', 'N', 'u'}; dp1];
        dp2 = [{'T', 'v', 'N', 'u'}; dp2];
        dp3 = [{'T', 'v', 'N', 'u'}; dp3];
        xlswrite (strcat(Folder,'\Fitting\u\ng_u_fit.xls'),dp1,'van');
        xlswrite (strcat(Folder,'\Fitting\u\ng_u_fit.xls'),dp2,'red');
        xlswrite (strcat(Folder,'\Fitting\u\ng_u_fit.xls'),dp3,'bwr');



        disp('2) Exporting h(T,P,N)');

        for iy=1:nn             % for concentrations
            for ix=1:8          % for pressures
                if ix==1
                    %-------- grapher
                    gp1 = [T' GetH(gn(iy),1,pressure(ix),T)'];
                    gp2 = [T' GetH(gn(iy),2,pressure(ix),T)'];
                    gp3 = [T' GetH(gn(iy),3,pressure(ix),T)'];
                else
                    %-------- grapher
                    gp1 = [gp1 GetH(gn(iy),1,pressure(ix),T)'];
                    gp2 = [gp2 GetH(gn(iy),2,pressure(ix),T)'];
                    gp3 = [gp3 GetH(gn(iy),3,pressure(ix),T)'];
                end
                if iy==1 && ix==1
                    %-------- fittin
                    dp1 = [T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetH(gn(iy),1,pressure(ix),T)'];
                    dp2 = [T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetH(gn(iy),2,pressure(ix),T)'];
                    dp3 = [T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetH(gn(iy),3,pressure(ix),T)'];
                else
                    %-------- fittin
                    dp1 = [dp1; T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetH(gn(iy),1,pressure(ix),T)'];
                    dp2 = [dp2; T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetH(gn(iy),2,pressure(ix),T)'];
                    dp3 = [dp3; T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetH(gn(iy),3,pressure(ix),T)'];
                end
            end
            %here storing the u for several v and the current concentration
            %N sheet
            ns=strcat('N=',num2str(N(iy)));


             gp1 = num2cell(gp1);
            gp2 = num2cell(gp2);
            gp3 = num2cell(gp3);

            gp1 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
            strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
            strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp1];
            gp2 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
```

```
...
        strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
        strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp2];
        gp3 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
        strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
        strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp3];


        xlswrite(strcat(Folder,'\Graphs\h\van.xls'),gp1,ns)
        xlswrite(strcat(Folder,'\Graphs\h\red.xls'),gp2,ns)
        xlswrite(strcat(Folder,'\Graphs\h\bwr.xls'),gp3,ns)
        clear gp1;
        clear gp2;
        clear gp3;

    end

    dp1 = num2cell(dp1);
    dp2 = num2cell(dp2);
    dp3 = num2cell(dp3);
    dp1 = [{'T', 'P', 'N', 'h'}; dp1];
    dp2 = [{'T', 'P', 'N', 'h'}; dp2];
    dp3 = [{'T', 'P', 'N', 'h'}; dp3];
    xlswrite (strcat(Folder,'\Fitting\h\ng_h_fit.xls'),dp1,'van');
    xlswrite (strcat(Folder,'\Fitting\h\ng_h_fit.xls'),dp2,'red');
    xlswrite (strcat(Folder,'\Fitting\h\ng_h_fit.xls'),dp3,'bwr');


    disp('3) Exporting s(P,T,N)');

    for iy=1:nn             % for concentrations
       for ix=1:8          % for pressures
          if ix==1
             %-------- grapher
             gp1 = [T' GetS(gn(iy),1,pressure(ix),T)'];
             gp2 = [T' GetS(gn(iy),2,pressure(ix),T)'];
             gp3 = [T' GetS(gn(iy),3,pressure(ix),T)'];
          else
             %-------- grapher
             gp1 = [gp1 GetS(gn(iy),1,pressure(ix),T)'];
             gp2 = [gp2 GetS(gn(iy),2,pressure(ix),T)'];
             gp3 = [gp3 GetS(gn(iy),3,pressure(ix),T)'];
          end
          if iy==1 && ix==1
             %-------- fittin
             dp1 = [T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetS(gn(iy),1,pressure(ix),T)'];
             dp2 = [T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetS(gn(iy),2,pressure(ix),T)'];
             dp3 = [T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetS(gn(iy),3,pressure(ix),T)'];
          else
             %-------- fittin
             dp1 = [dp1; T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
```

```
GetS(gn(iy),1,pressure(ix),T)'];
                dp2 = [dp2; T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetS(gn(iy),2,pressure(ix),T)'];
                dp3 = [dp3; T' repmat(pressure(ix),length(T),1) repmat(N(iy),length(T),1)
GetS(gn(iy),3,pressure(ix),T)'];
              end
          end
          %here storing the u for several v and the current concentration
          %N sheet
          ns=strcat('N=',num2str(N(iy)));

          gp1 = num2cell(gp1);
          gp2 = num2cell(gp2);
          gp3 = num2cell(gp3);

          gp1 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
          strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
          strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp1];
          gp2 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
          strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
          strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp2];
          gp3 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
          strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
          strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp3];



          xlswrite(strcat(Folder,'\Graphs\s\van.xls'),gp1,ns)
          xlswrite(strcat(Folder,'\Graphs\s\red.xls'),gp2,ns)
          xlswrite(strcat(Folder,'\Graphs\s\bwr.xls'),gp3,ns)
          clear gp1;
          clear gp2;
          clear gp3;

      end

      dp1 = num2cell(dp1);
      dp2 = num2cell(dp2);
      dp3 = num2cell(dp3);
      dp1 = [{'T', 'P', 'N', 's'}; dp1];
      dp2 = [{'T', 'P', 'N', 's'}; dp2];
      dp3 = [{'T', 'P', 'N', 's'}; dp3];
      xlswrite (strcat(Folder,'\Fitting\s\ng_s_fit.xls'),dp1,'van');
      xlswrite (strcat(Folder,'\Fitting\s\ng_s_fit.xls'),dp2,'red');
      xlswrite (strcat(Folder,'\Fitting\s\ng_s_fit.xls'),dp3,'bwr');



      % %internal functions to get u, h, and s  departures
      % %-----------------------------------------------
      function dep = GetDepU(NGas, Method, v, T)
          actual = GetU(NGas, Method, v, T);
          ideal = GetU(NGas, Method, 1.990, T);
          dep = ((actual - ideal)./ideal).*100;
```

```
end

function dep=GetDepH(NGas, Method, P, T)
    actual = GetH(NGas, Method, P, T);
    ideal = GetH(NGas, Method, 202.6, T);
    dep = ((actual - ideal)./ideal).*100;
end

function dep=GetDepS(NGas, Method, P, T)
    actual = GetS(NGas, Method, P, T);
    ideal = GetS(NGas, Method, 202.6, T);
    dep = ((actual - ideal)./ideal).*100;
end
%-------------------------------------------------

disp('4) Exporting Departure u(v,T)');

for iy=1:nn             % for concentrations
    for iz=1:5          % for specific volumes
        if iz==1
            %-------- grapher
            gp1 = [T' GetDepU(gn(iy),1,v(iz),T)'];
            gp2 = [T' GetDepU(gn(iy),2,v(iz),T)'];
            gp3 = [T' GetDepU(gn(iy),3,v(iz),T)'];
        else
            %-------- grapher
            gp1 = [gp1 GetDepU(gn(iy),1,v(iz),T)'];
            gp2 = [gp2 GetDepU(gn(iy),2,v(iz),T)'];
            gp3 = [gp3 GetDepU(gn(iy),3,v(iz),T)'];
        end
    end
    %here storing the u for several v and the current concentration
    %N sheet
    ns=strcat('N=',num2str(N(iy)));

    gp1 = num2cell(gp1);
    gp2 = num2cell(gp2);
    gp3 = num2cell(gp3);

    gp1 = [{'T', strcat('v=',num2str(v(1))),strcat('v=',num2str(v(2))),strcat('v=',num2str(v(3))), ...
    strcat('v=',num2str(v(4))), strcat('v=',num2str(v(5)))} ; gp1];
    gp2 = [{'T', strcat('v=',num2str(v(1))),strcat('v=',num2str(v(2))),strcat('v=',num2str(v(3))), ...
    strcat('v=',num2str(v(4))), strcat('v=',num2str(v(5)))} ; gp2];
    gp3 = [{'T', strcat('v=',num2str(v(1))),strcat('v=',num2str(v(2))),strcat('v=',num2str(v(3))), ...
    strcat('v=',num2str(v(4))), strcat('v=',num2str(v(5)))} ; gp3];

    xlswrite(strcat(Folder,'\Graphs\u\van_departure.xls'),gp1,ns)
    xlswrite(strcat(Folder,'\Graphs\u\red_departure.xls'),gp2,ns)
    xlswrite(strcat(Folder,'\Graphs\u\bwr_departure.xls'),gp3,ns)
    clear gp1;
    clear gp2;
    clear gp3;

end
```

```
    disp('5) Exporting Departure h(P,T)');
    for iy=1:nn             % for concentrations
       for ix=1:8          % for pressures
          if ix==1
             %-------- grapher
             gp1 = [T' GetDepH(gn(iy),1,pressure(ix),T)'];
             gp2 = [T' GetDepH(gn(iy),2,pressure(ix),T)'];
             gp3 = [T' GetDepH(gn(iy),3,pressure(ix),T)'];
          else
             %-------- grapher
             gp1 = [gp1 GetDepH(gn(iy),1,pressure(ix),T)'];
             gp2 = [gp2 GetDepH(gn(iy),2,pressure(ix),T)'];
             gp3 = [gp3 GetDepH(gn(iy),3,pressure(ix),T)'];
          end
       end
       %here storing the u for several v and the current concentration
       %N sheet
       ns=strcat('N=',num2str(N(iy)));

       gp1 = num2cell(gp1);
       gp2 = num2cell(gp2);
       gp3 = num2cell(gp3);

       gp1 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
       strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
       strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp1];
       gp2 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
       strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
       strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp2];
       gp3 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
       strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
       strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp3];



       xlswrite(strcat(Folder,'\Graphs\h\van_departure.xls'),gp1,ns)
       xlswrite(strcat(Folder,'\Graphs\h\red_departure.xls'),gp2,ns)
       xlswrite(strcat(Folder,'\Graphs\h\bwr_departure.xls'),gp3,ns)
       clear gp1;
       clear gp2;
       clear gp3;

    end

    disp('6) Exporting Departure s(P,T)');
    for iy=1:nn             % for concentrations
       for ix=1:8          % for pressures
          if ix==1
             %-------- grapher
             gp1 = [T' GetDepS(gn(iy),1,pressure(ix),T)'];
             gp2 = [T' GetDepS(gn(iy),2,pressure(ix),T)'];
             gp3 = [T' GetDepS(gn(iy),3,pressure(ix),T)'];
```

```
            else
                %-------- grapher
                gp1 = [gp1 GetDepS(gn(iy),1,pressure(ix),T)'];
                gp2 = [gp2 GetDepS(gn(iy),2,pressure(ix),T)'];
                gp3 = [gp3 GetDepS(gn(iy),3,pressure(ix),T)'];
            end
        end
        %here storing the u for several v and the current concentration
        %N sheet
        ns=strcat('N=',num2str(N(iy)));

        gp1 = num2cell(gp1);
        gp2 = num2cell(gp2);
        gp3 = num2cell(gp3);

        gp1 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
        strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
        strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp1];
        gp2 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
        strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
        strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp2];
        gp3 = [{'T', strcat('P=',num2str(P(1))),strcat('P=',num2str(P(2))),strcat('P=',num2str(P(3))),
...
        strcat('P=',num2str(P(4))), strcat('P=',num2str(P(5))), strcat('P=',num2str(P(6))), ...
        strcat('P=',num2str(P(7))), strcat('P=',num2str(P(8)))} ; gp3];



        xlswrite(strcat(Folder,'\Graphs\s\van_departure.xls'),gp1,ns)
        xlswrite(strcat(Folder,'\Graphs\s\red_departure.xls'),gp2,ns)
        xlswrite(strcat(Folder,'\Graphs\s\bwr_departure.xls'),gp3,ns)
        clear gp1;
        clear gp2;
        clear gp3;

    end
  end
end
```

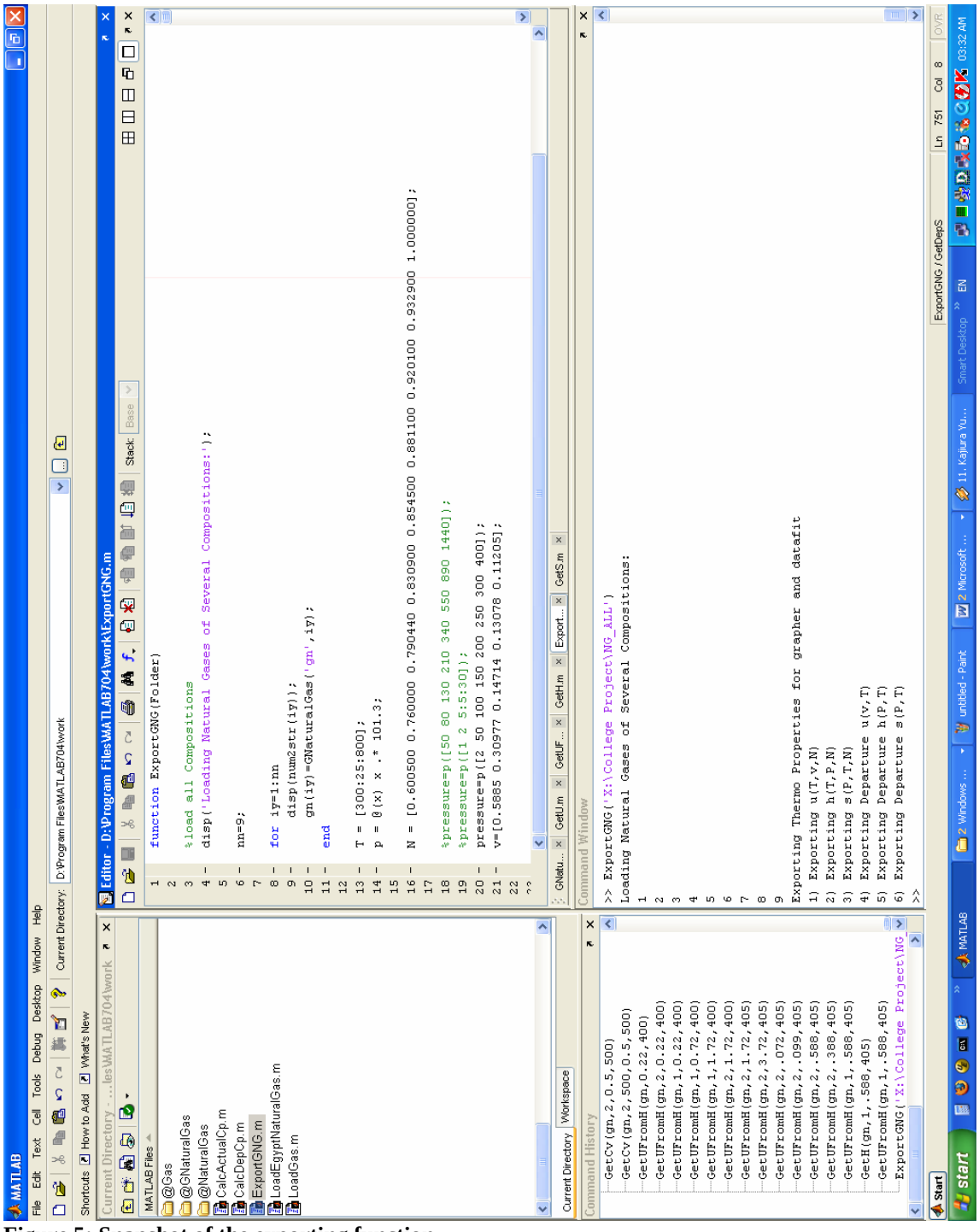
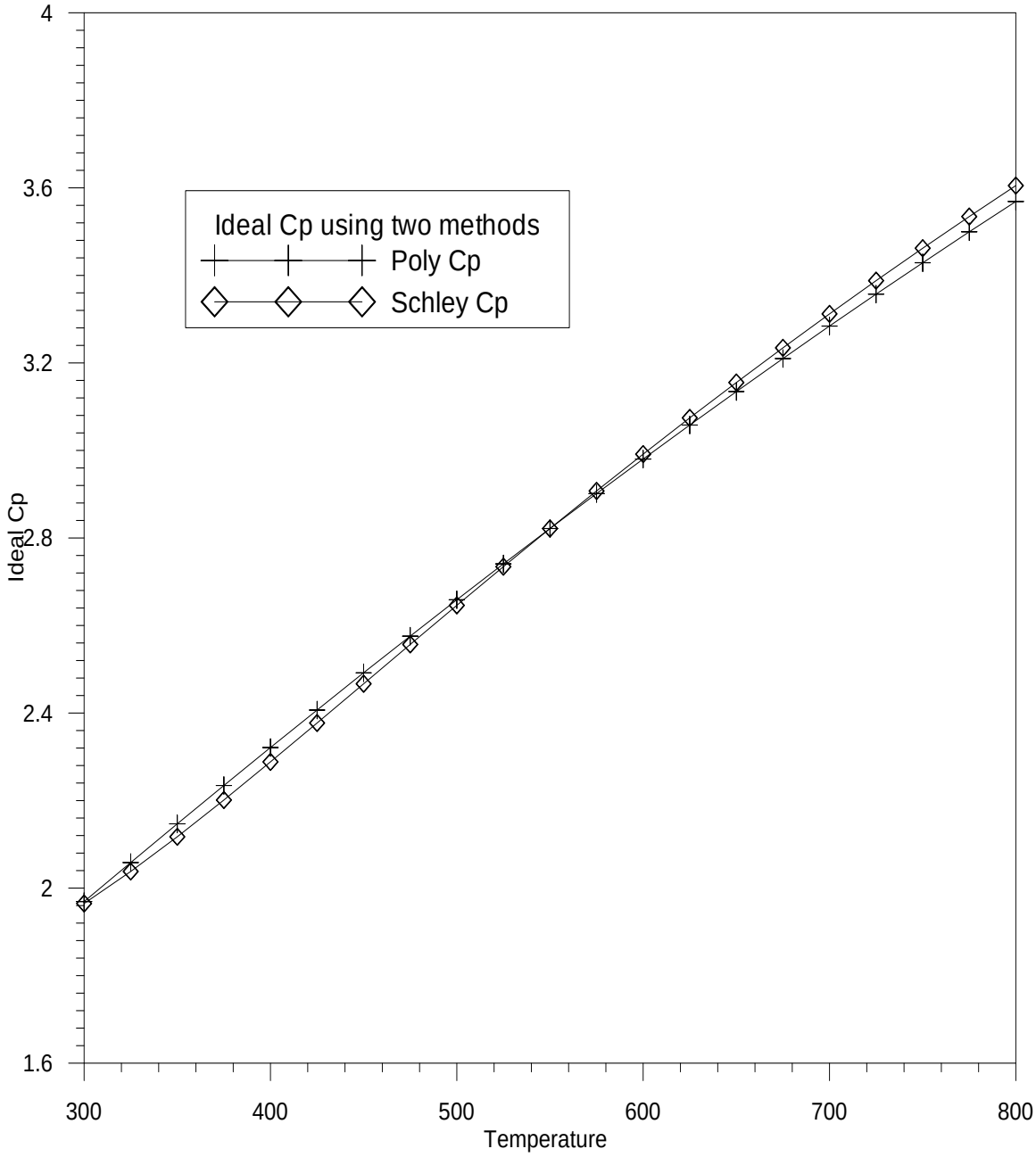
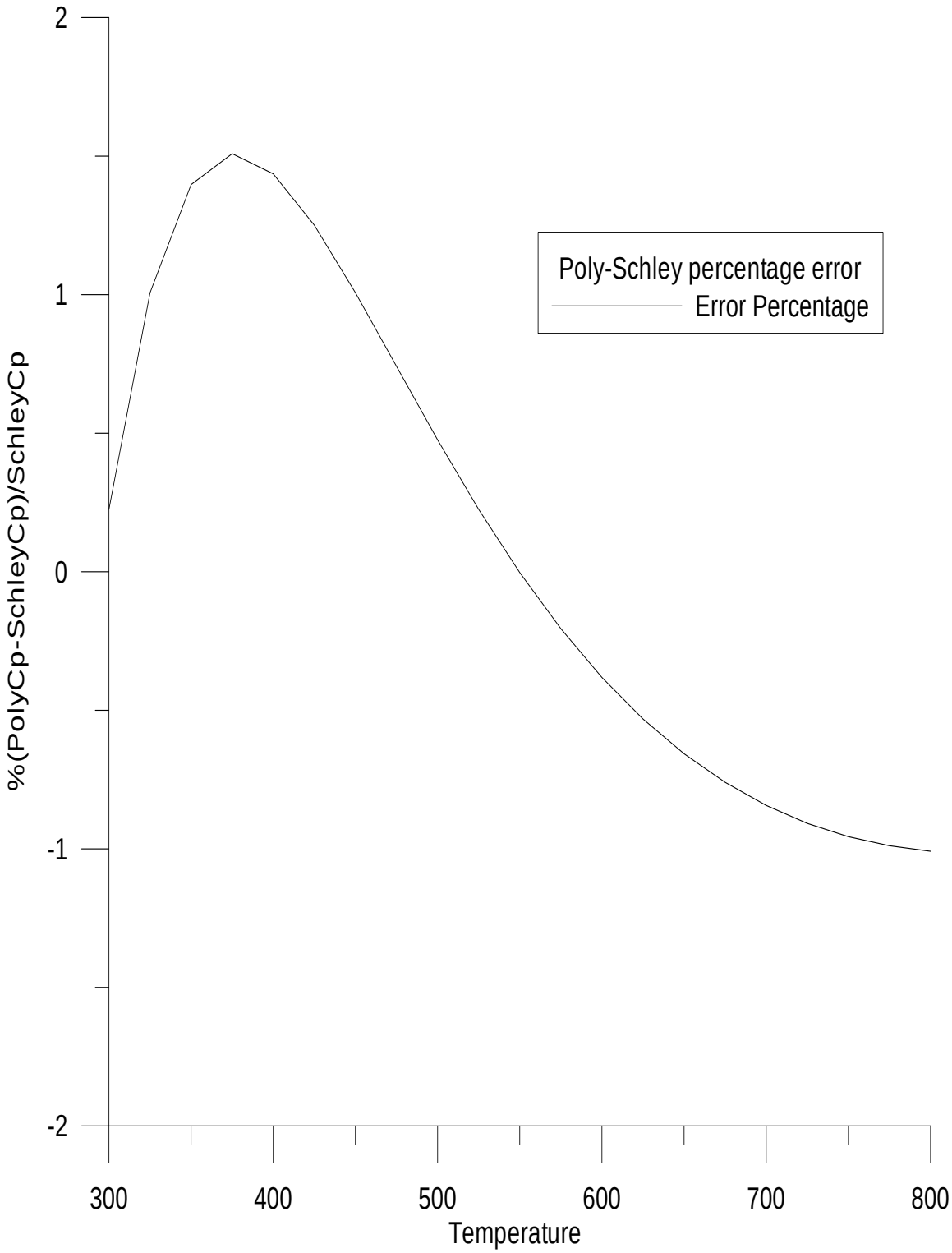


Figure 5: Snapshot of the exporting function

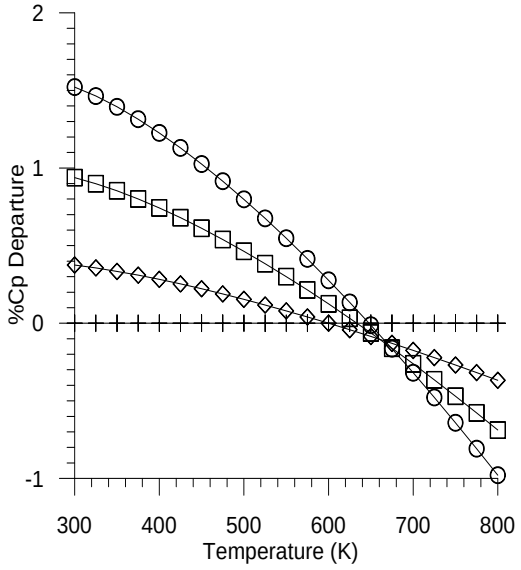
Results



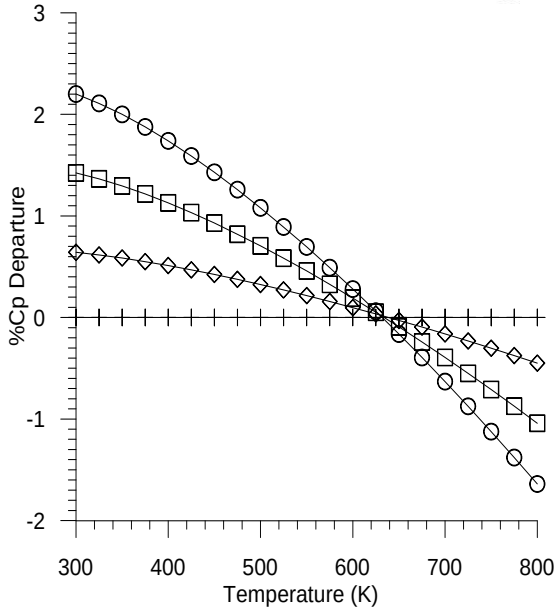
Graph 1: Ideal Cp using Poly and Schley methods



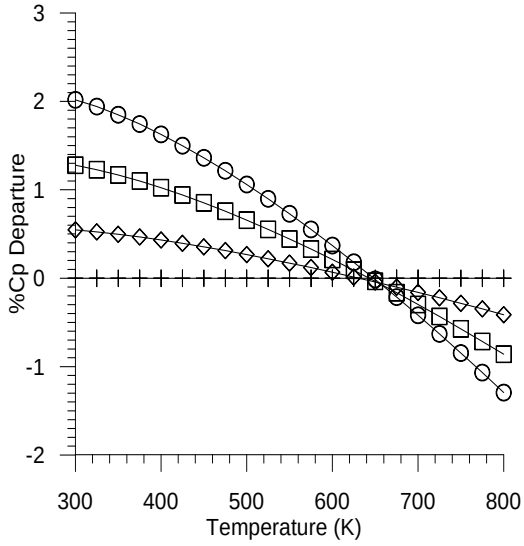
Graph 2: The error percentage in the two methods



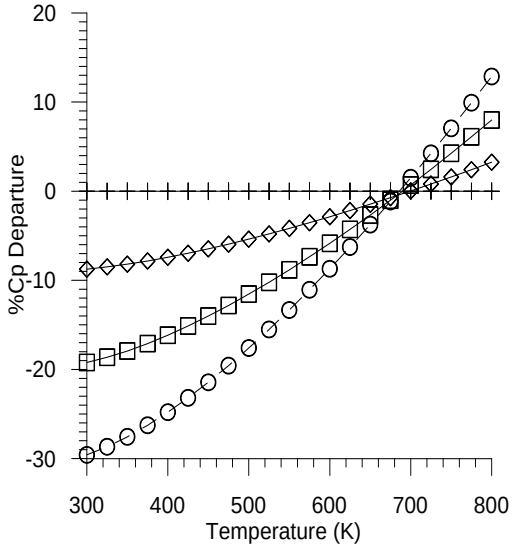
a) Van Der Wall



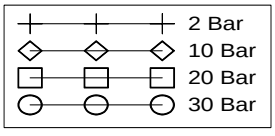
b) Redlich



c) BWR

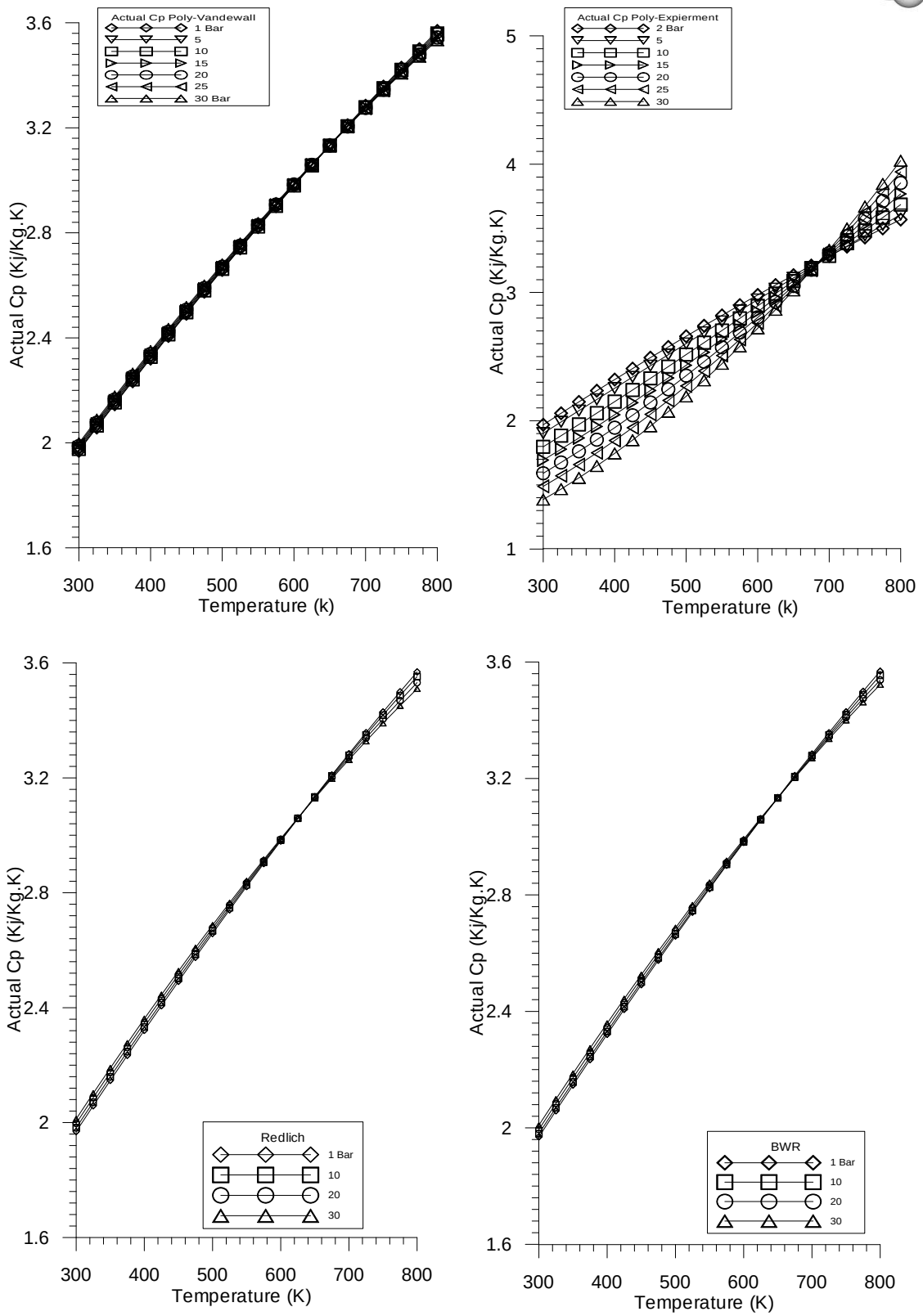


d) Expiermental



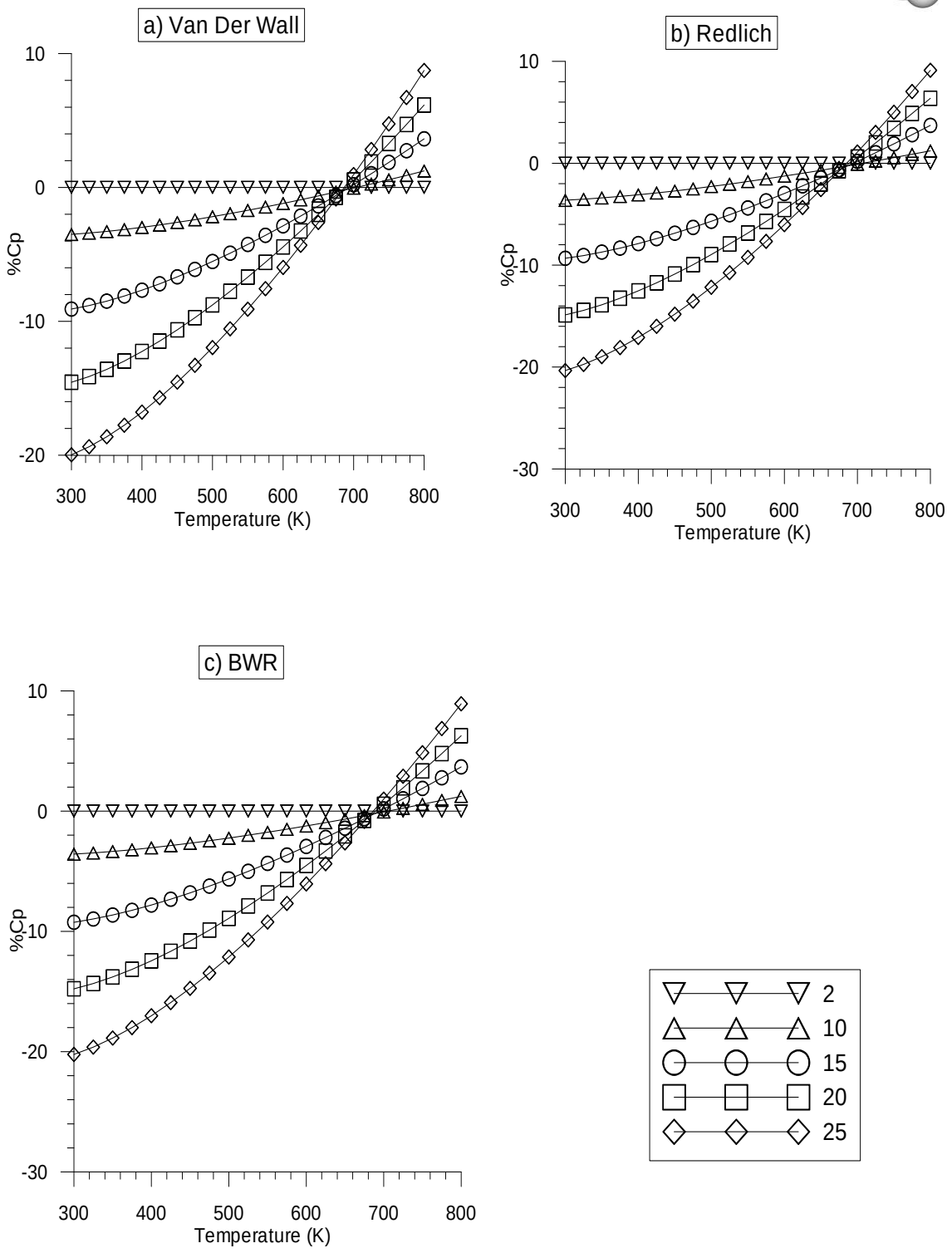
Percentage departure of actual specifc heat
from an ideal specific heat for natural gas at different pressures
%c_p Departure = ((Actual c_p- Ideal c_p)/Actual c_p)*100

Graph 3



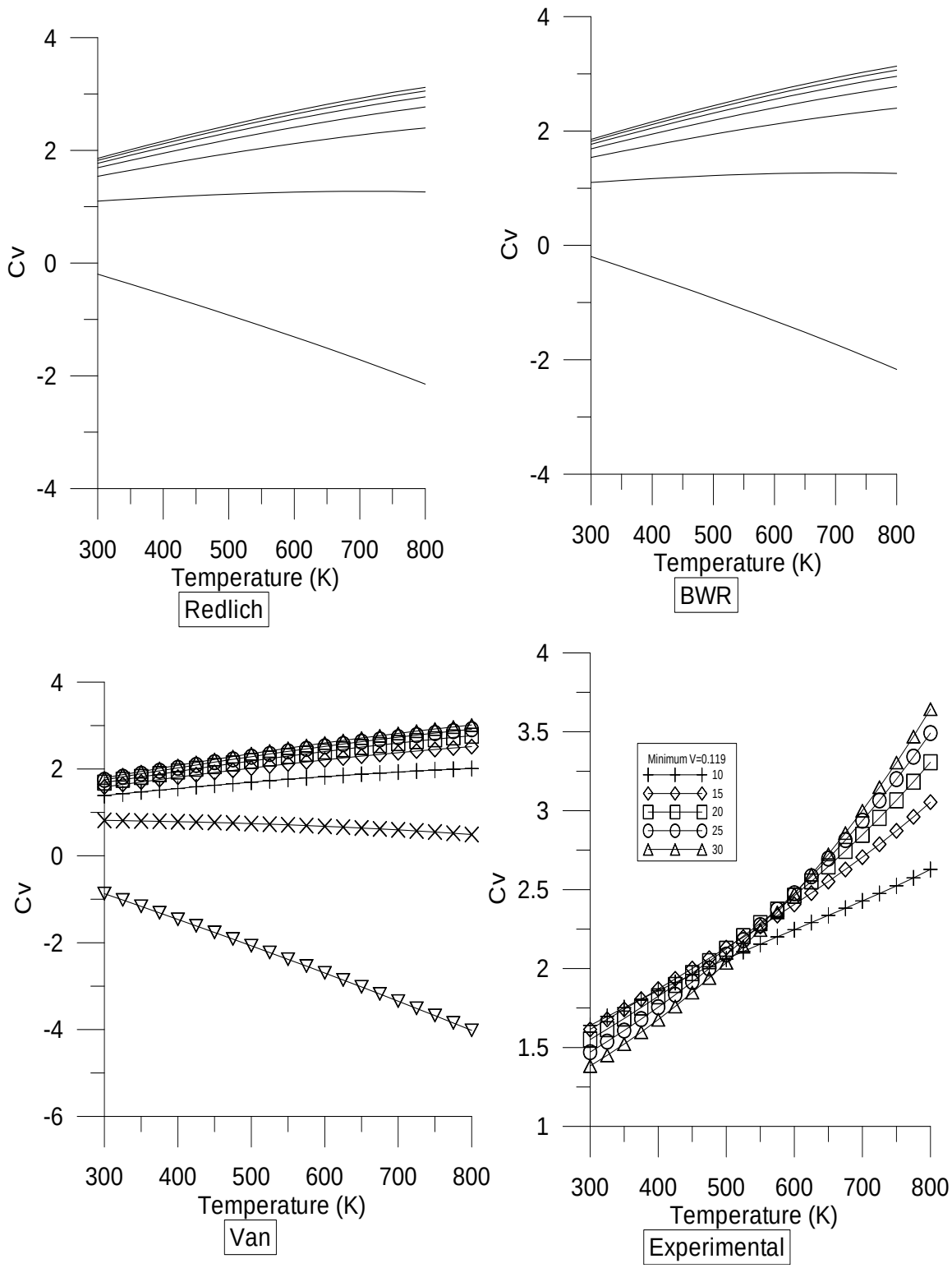
Effect of pressure and temperature on the actual isobaric specific heats for
the PVT data obtained from Van Der Waal, Experimental, Redlich and BWR equations of state.

Graph 4

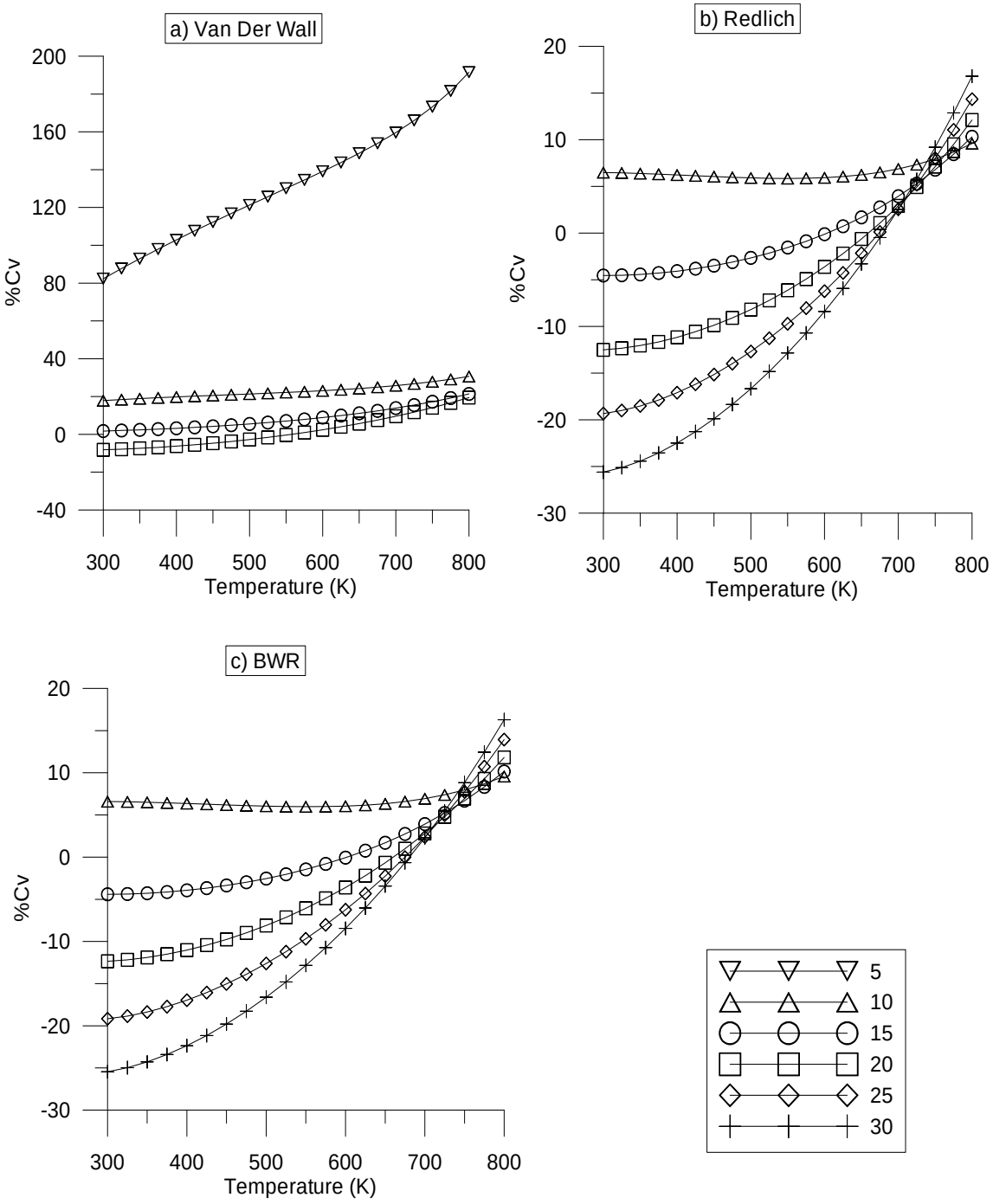


Deviation of experimental Cp from theoritical Cp

Graph 5

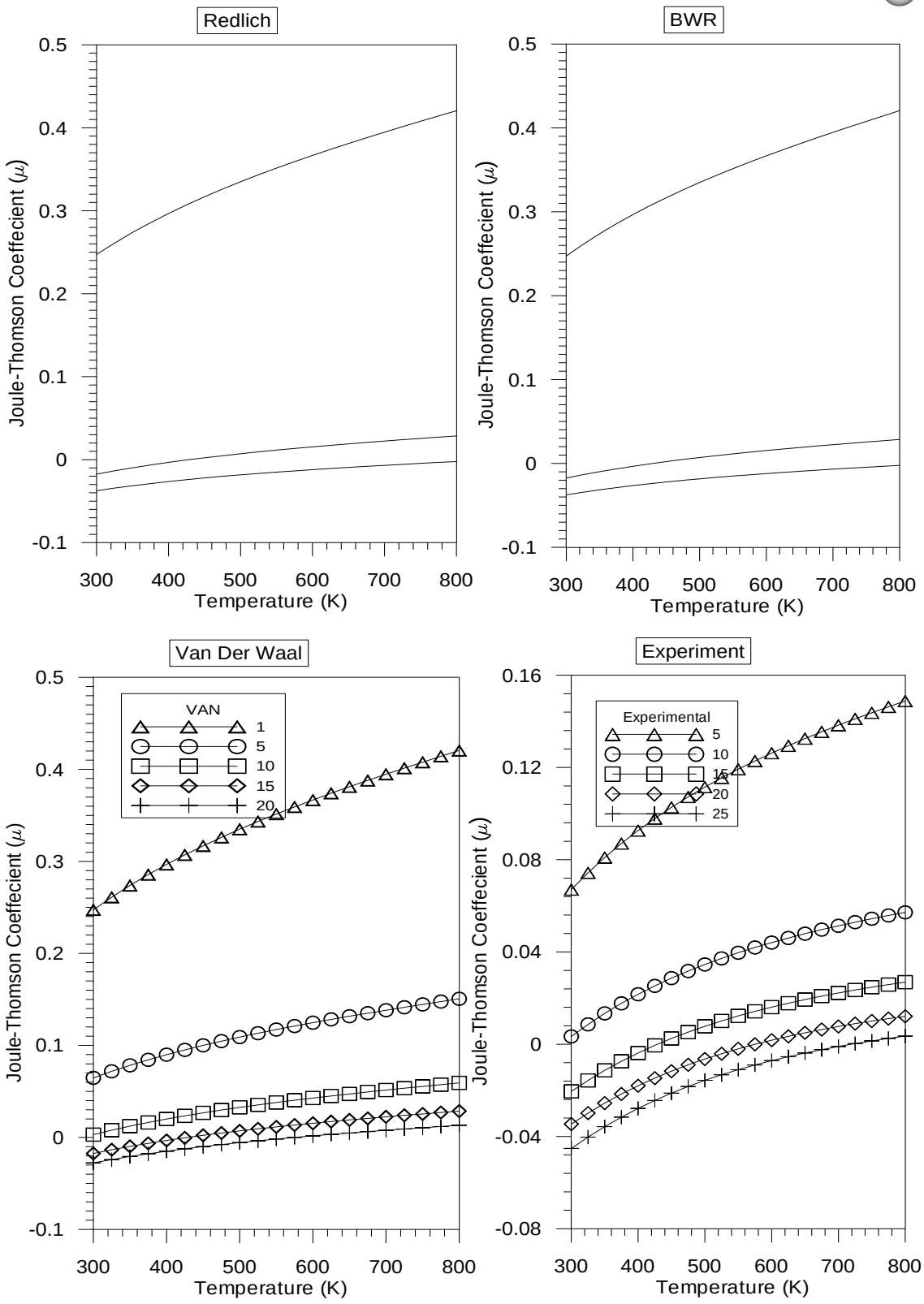


Graph 6: Cv of Van Der Waal, Redlish, BWR, and Experimental



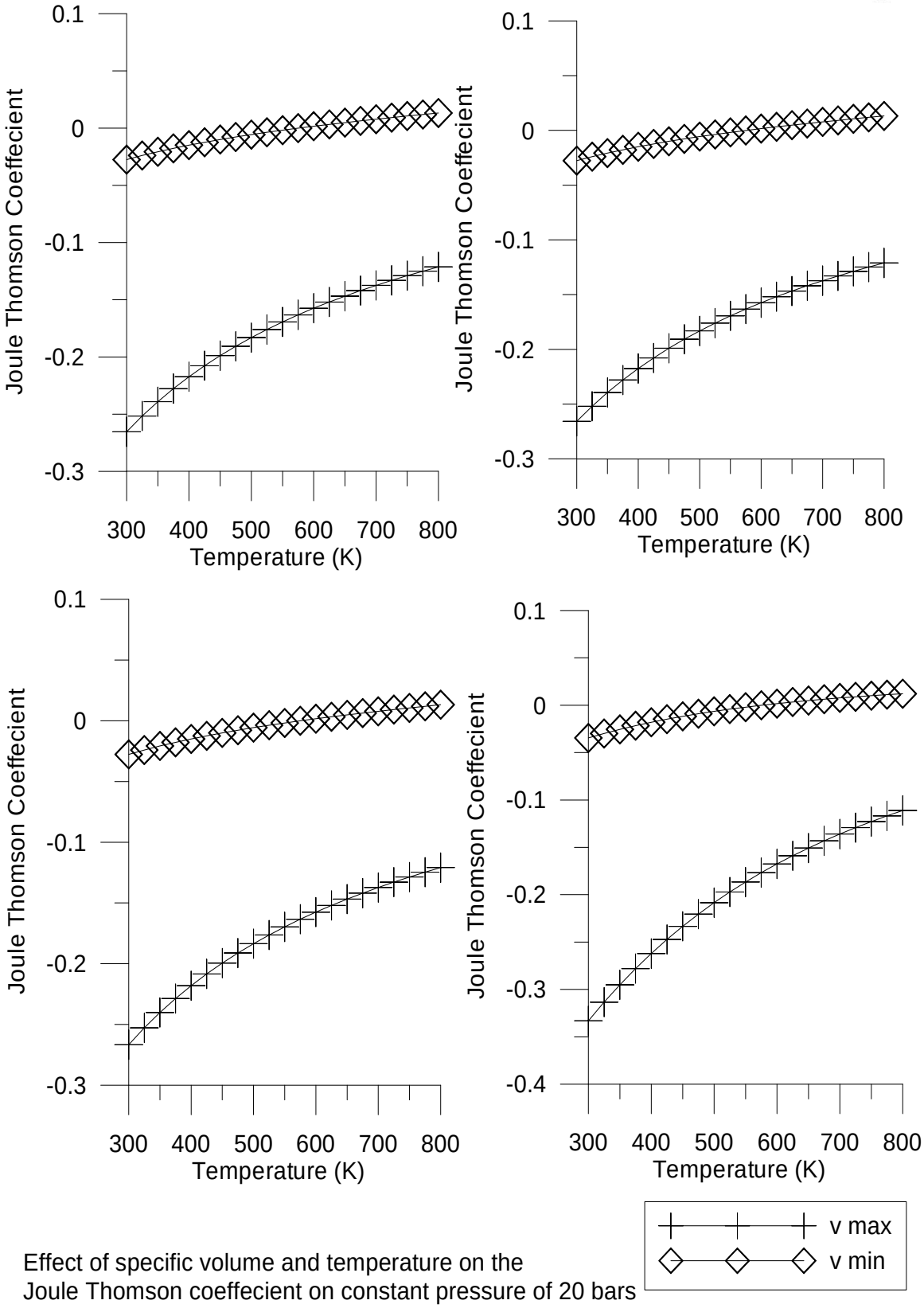
Percentage Cv of experimental Cv from theoritical Cv at minimum (v)

Graph 7: %Cv between theoretical and experimental



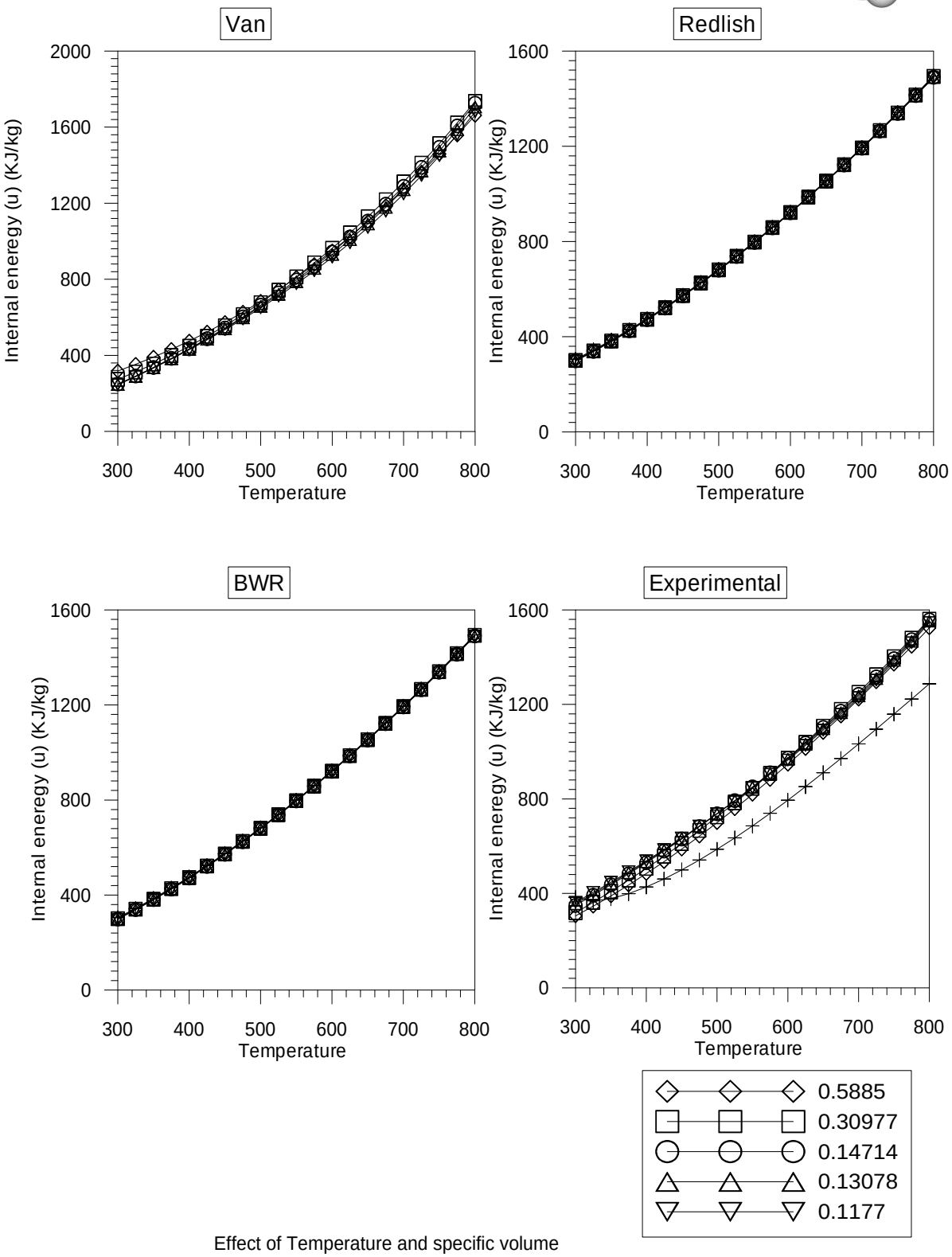
Joule-Thomson Coeffecient (μ) at (P_{max}) and (v_{min}) for PVT data

Graph 8



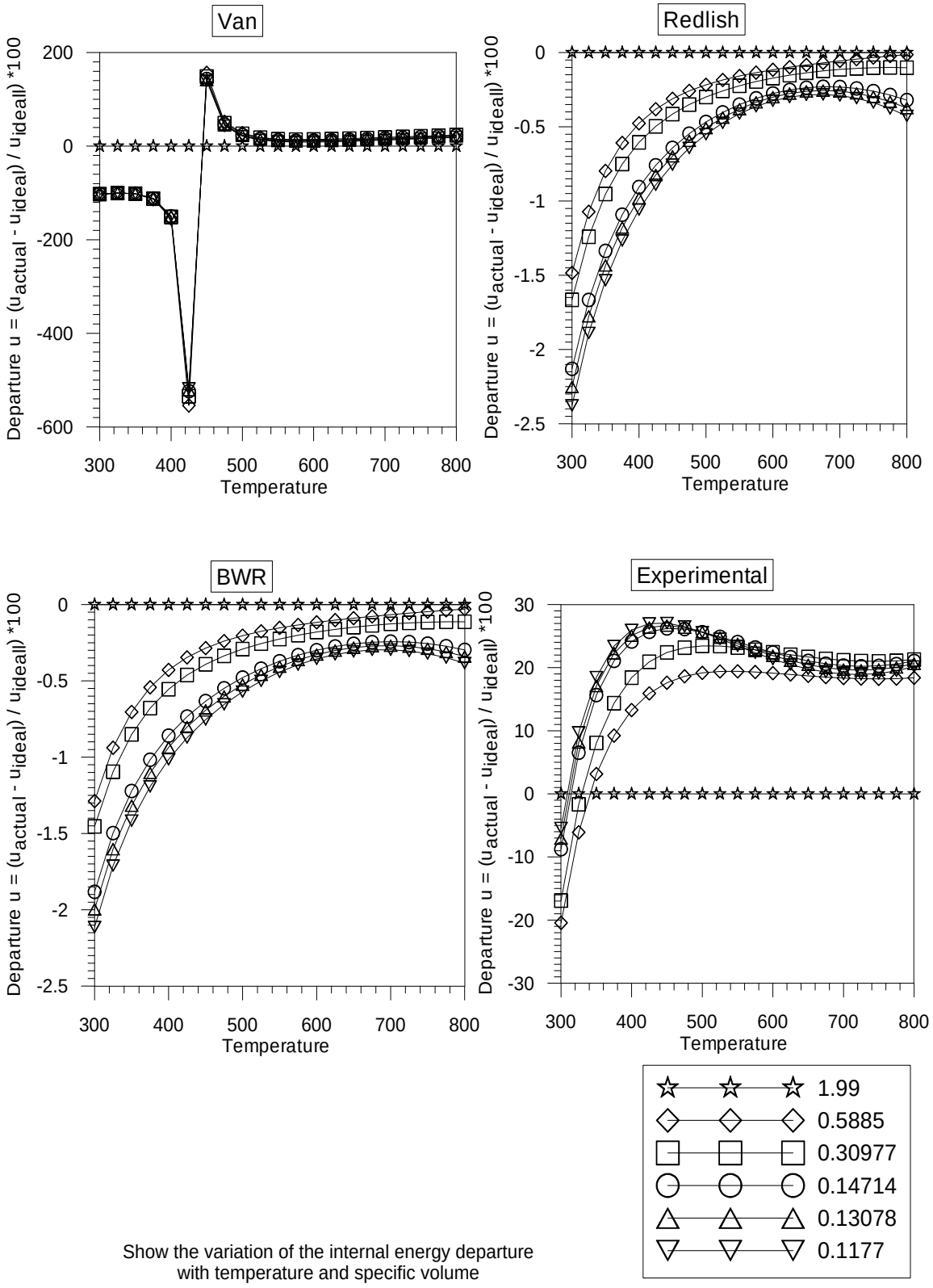
Effect of specific volume and temperature on the
Joule Thomson coeffecient on constant pressure of 20 bars

Graph 9



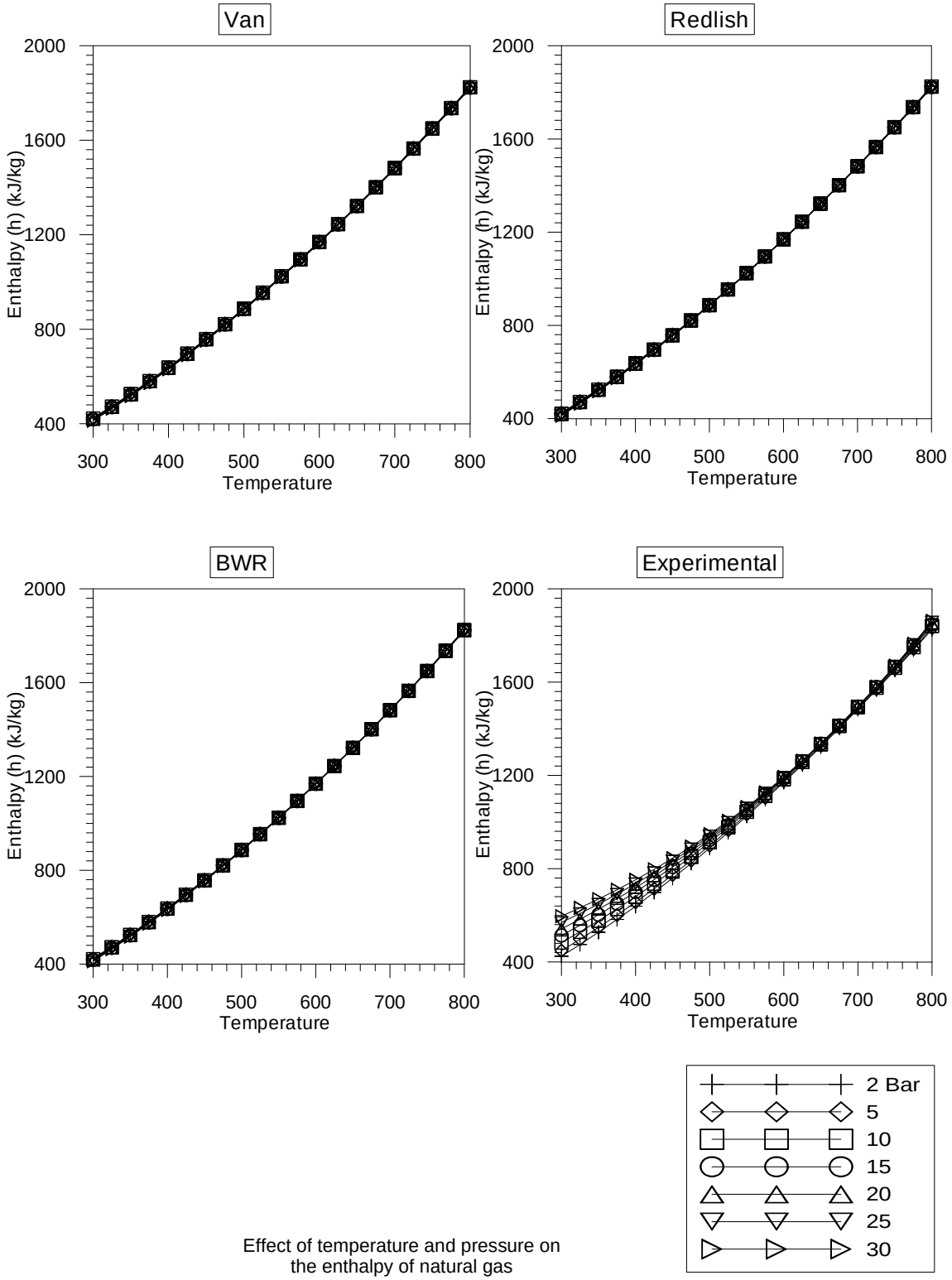
Effect of Temperature and specific volume
on internal energy (u) of natural gas

Graph 10: Internal Energy



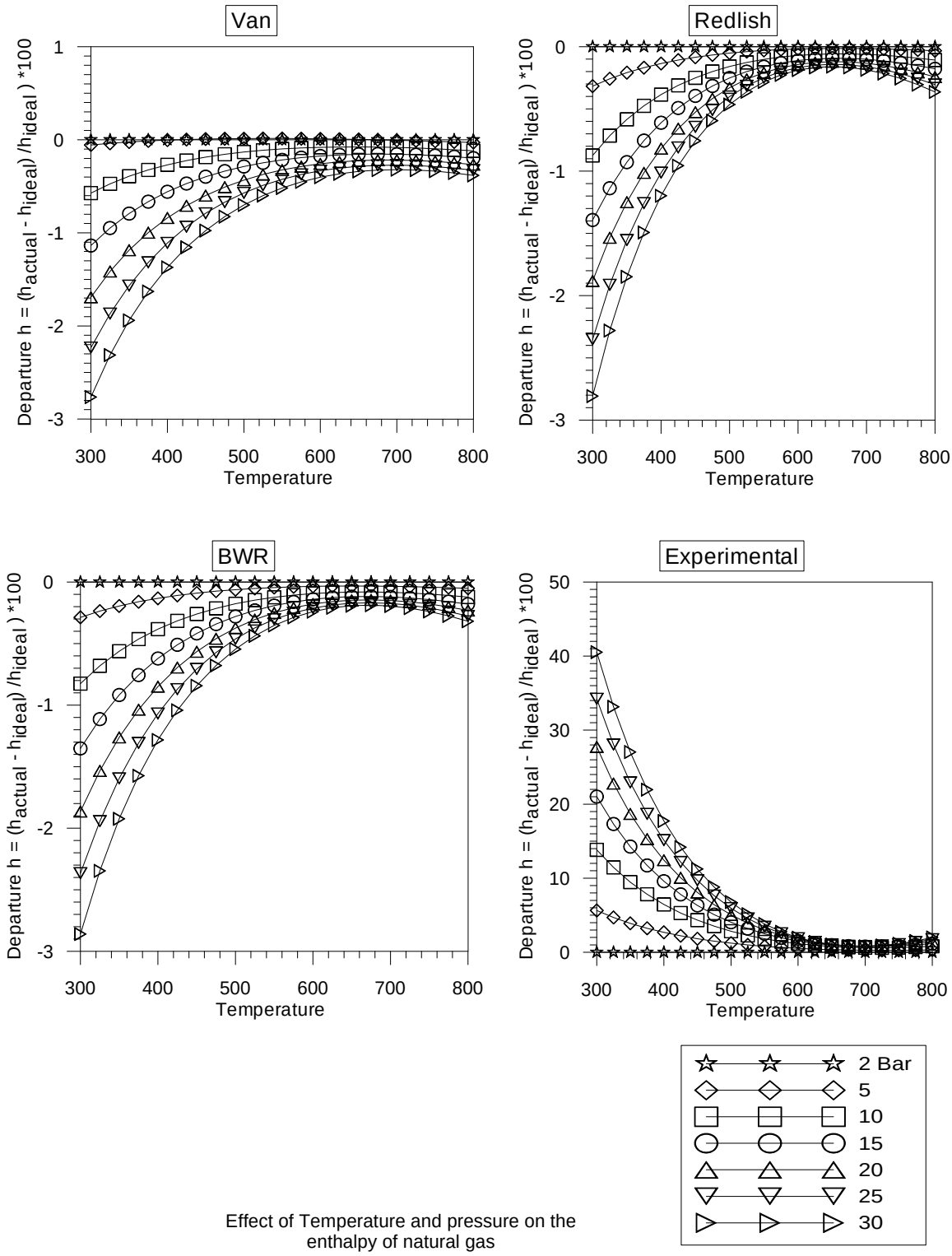
Show the variation of the internal energy departure with temperature and specific volume

Graph 11: Internal Energy Departure



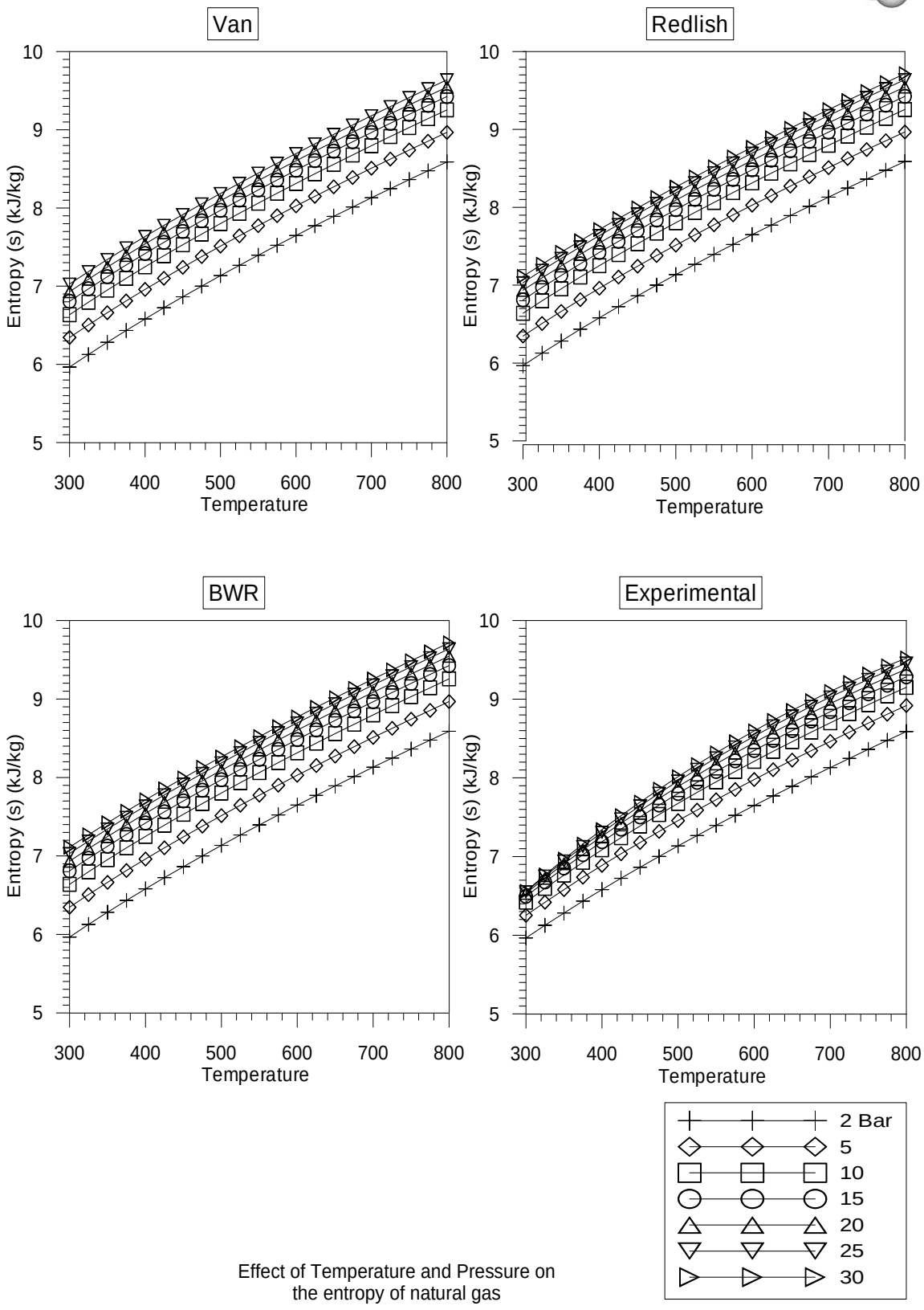
Effect of temperature and pressure on the enthalpy of natural gas

Graph 12: Enthalpy



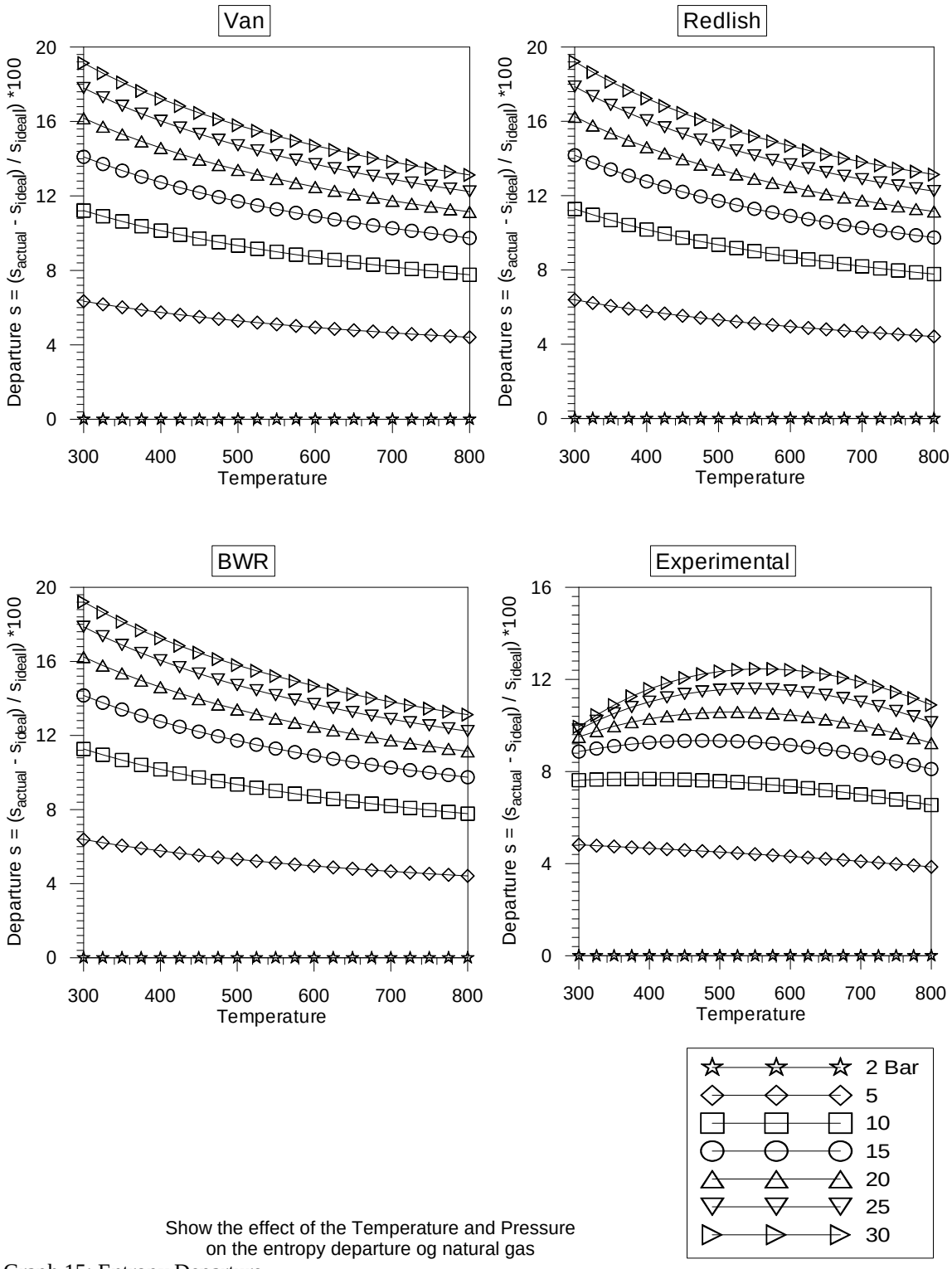
Effect of Temperature and pressure on the enthalpy of natural gas

Graph 13: Enthalpy Departure



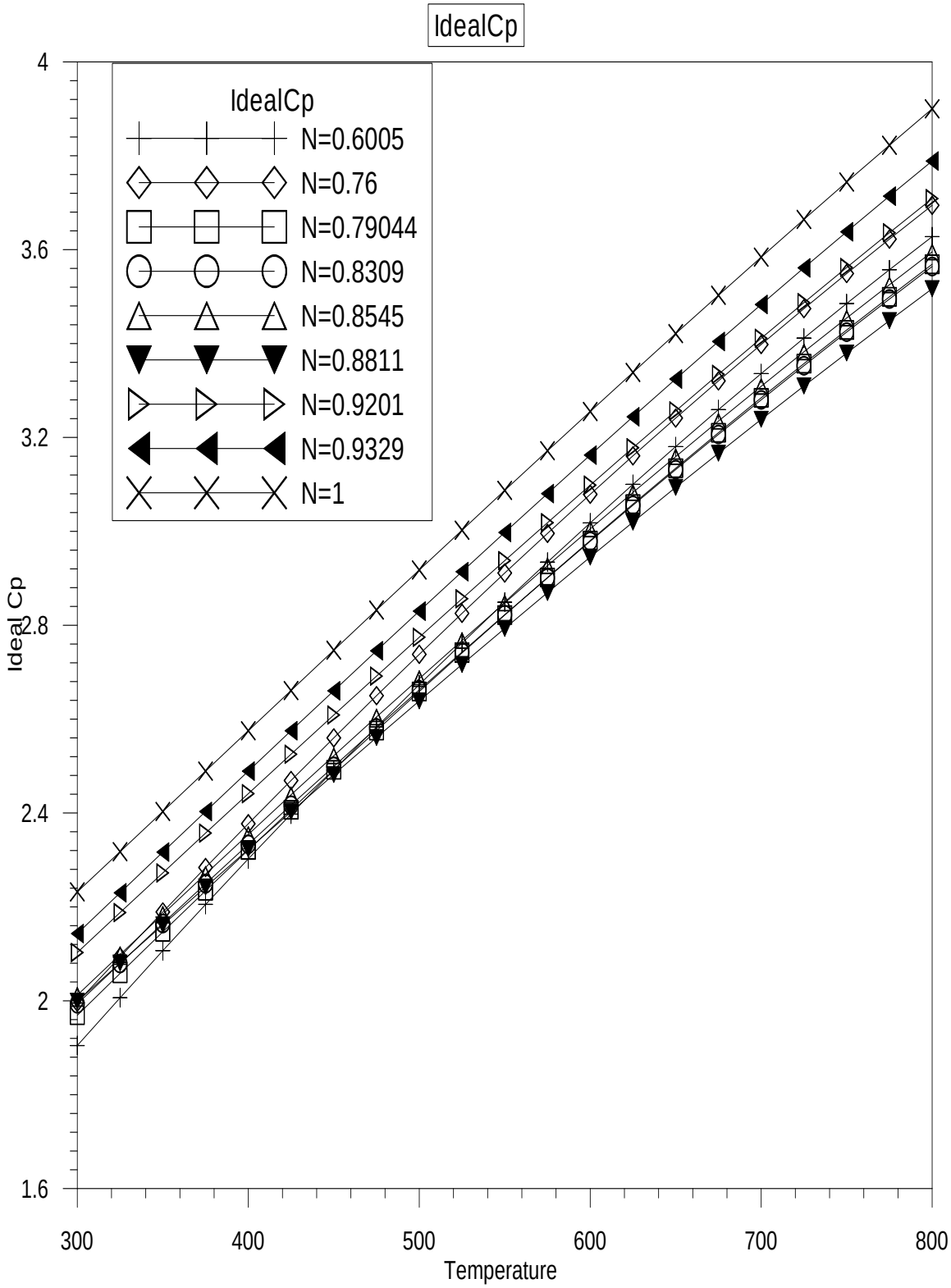
Effect of Temperature and Pressure on the entropy of natural gas

Graph 14: Entropy

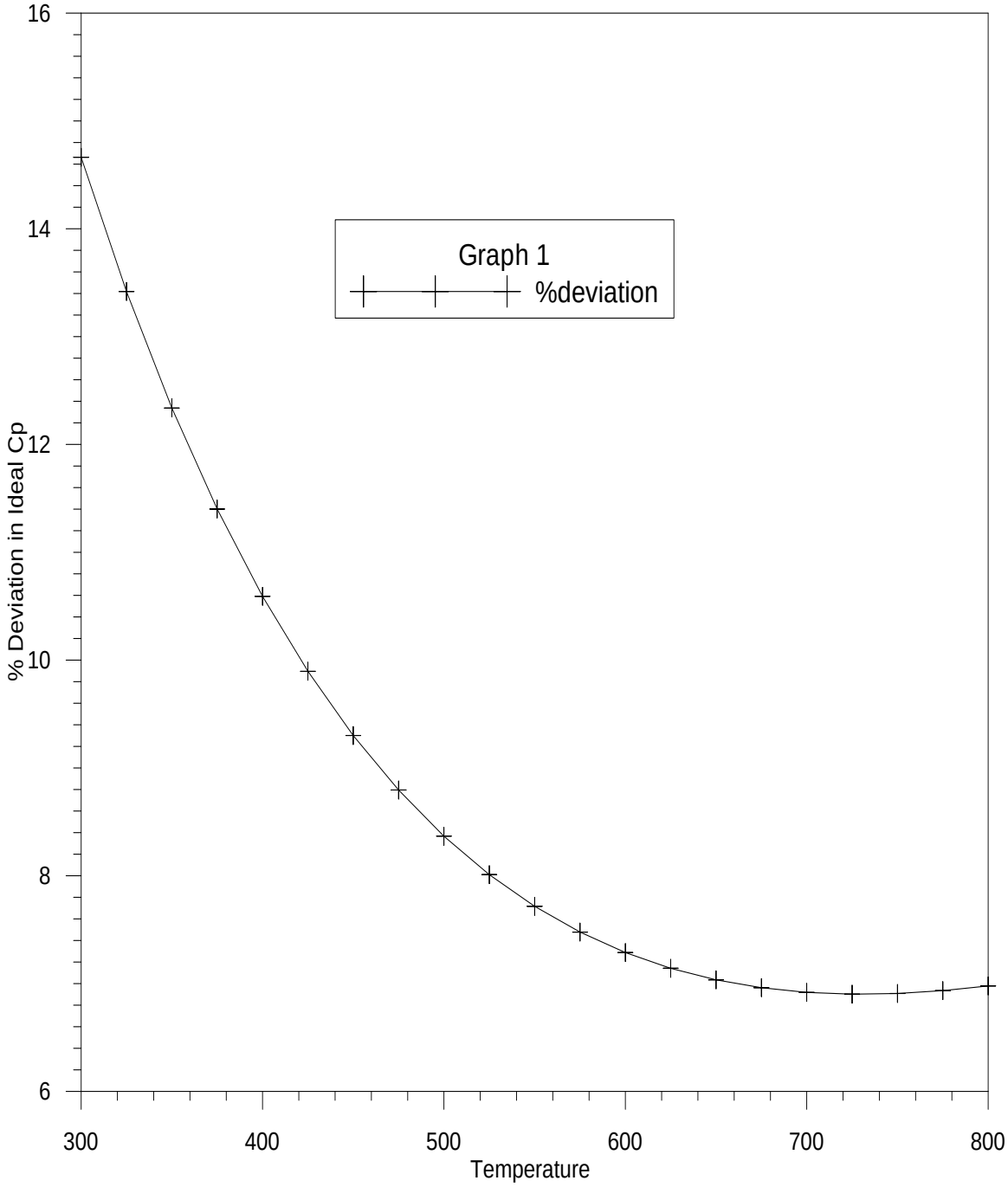


Show the effect of the Temperature and Pressure on the entropy departure og natural gas

Graph 15: Entropy Departure

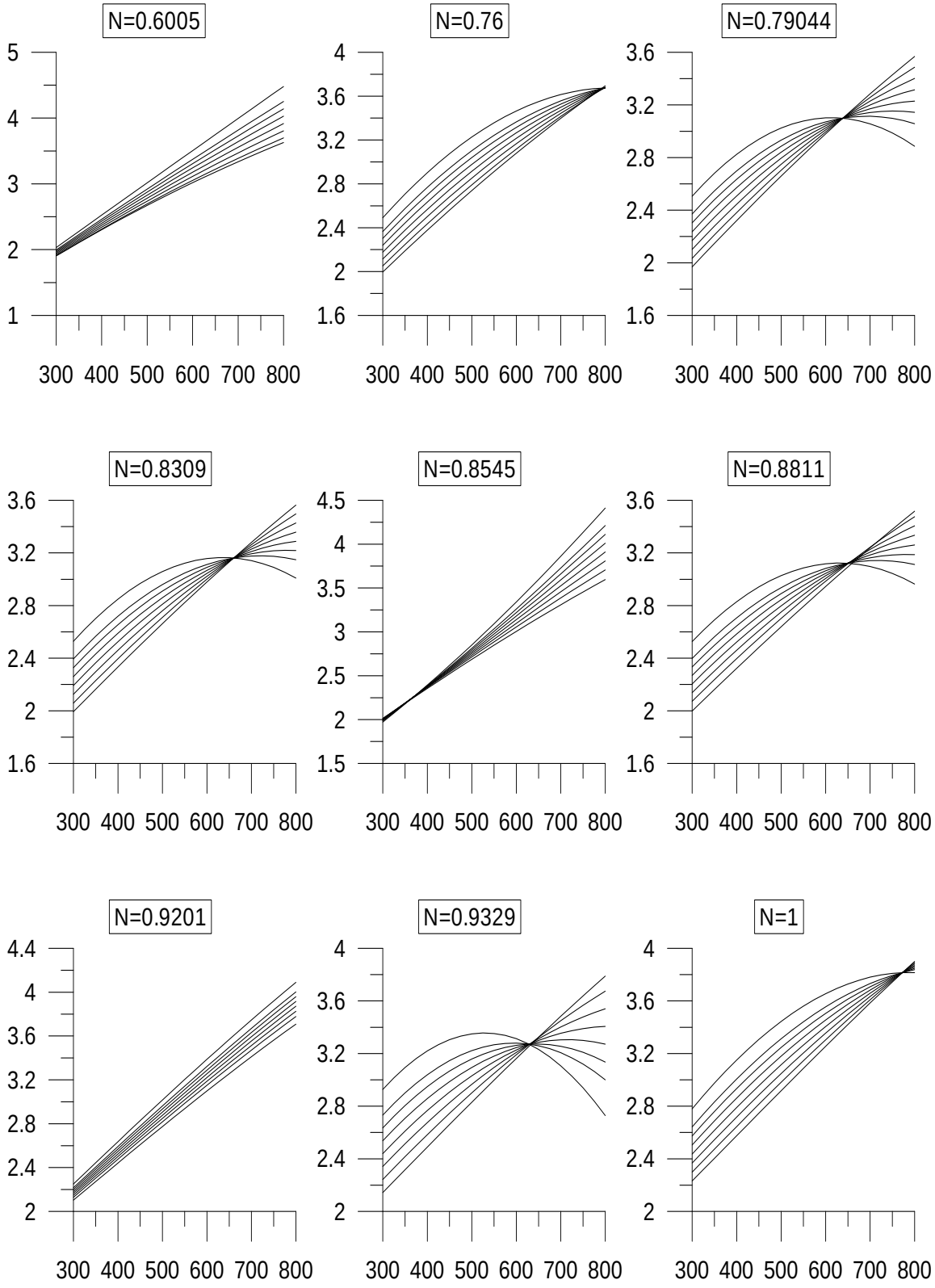


Graph 16: Ideal Cp in several concentrations of methane

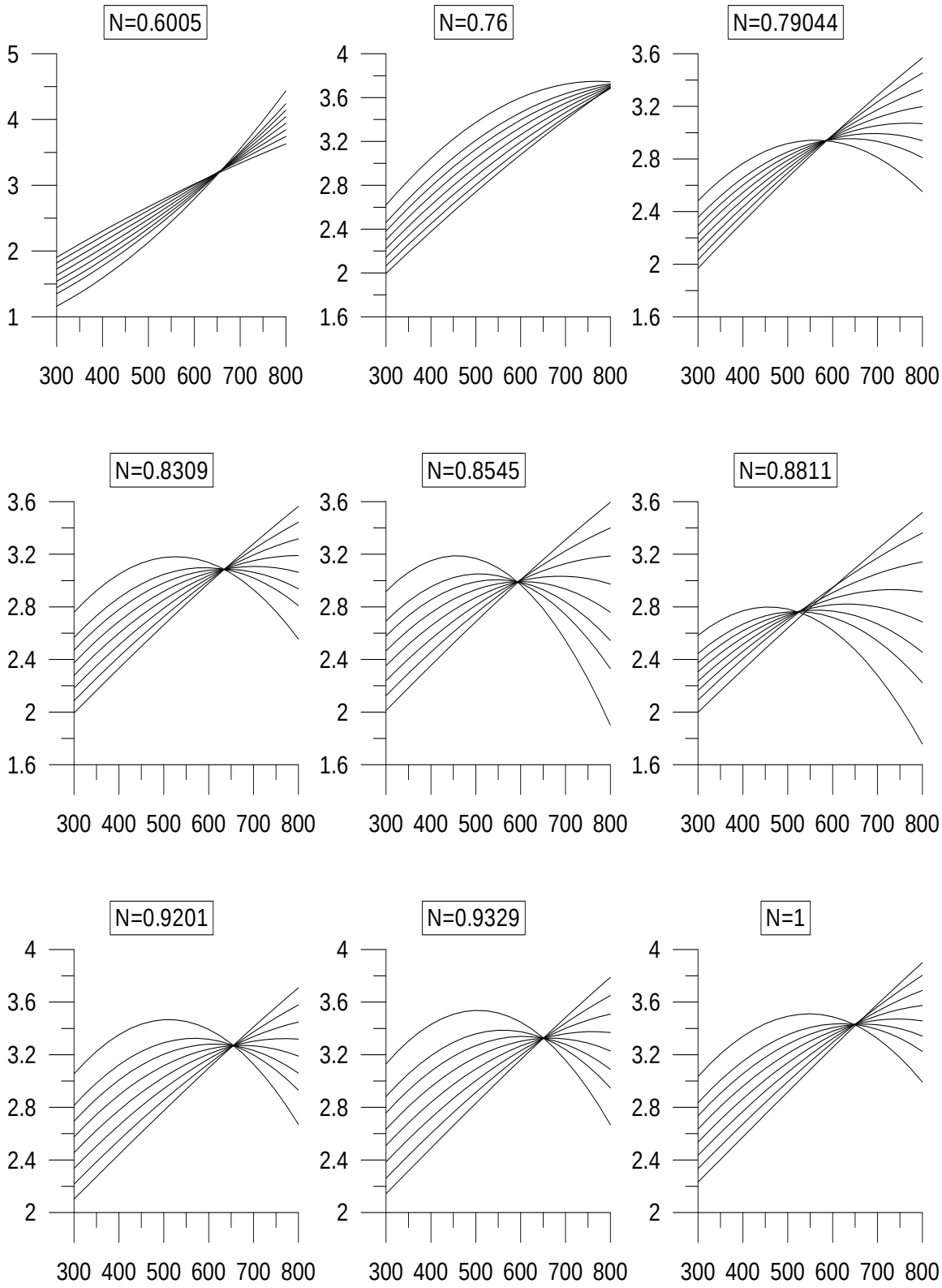


Deviation of ideal Cp between max N and min N

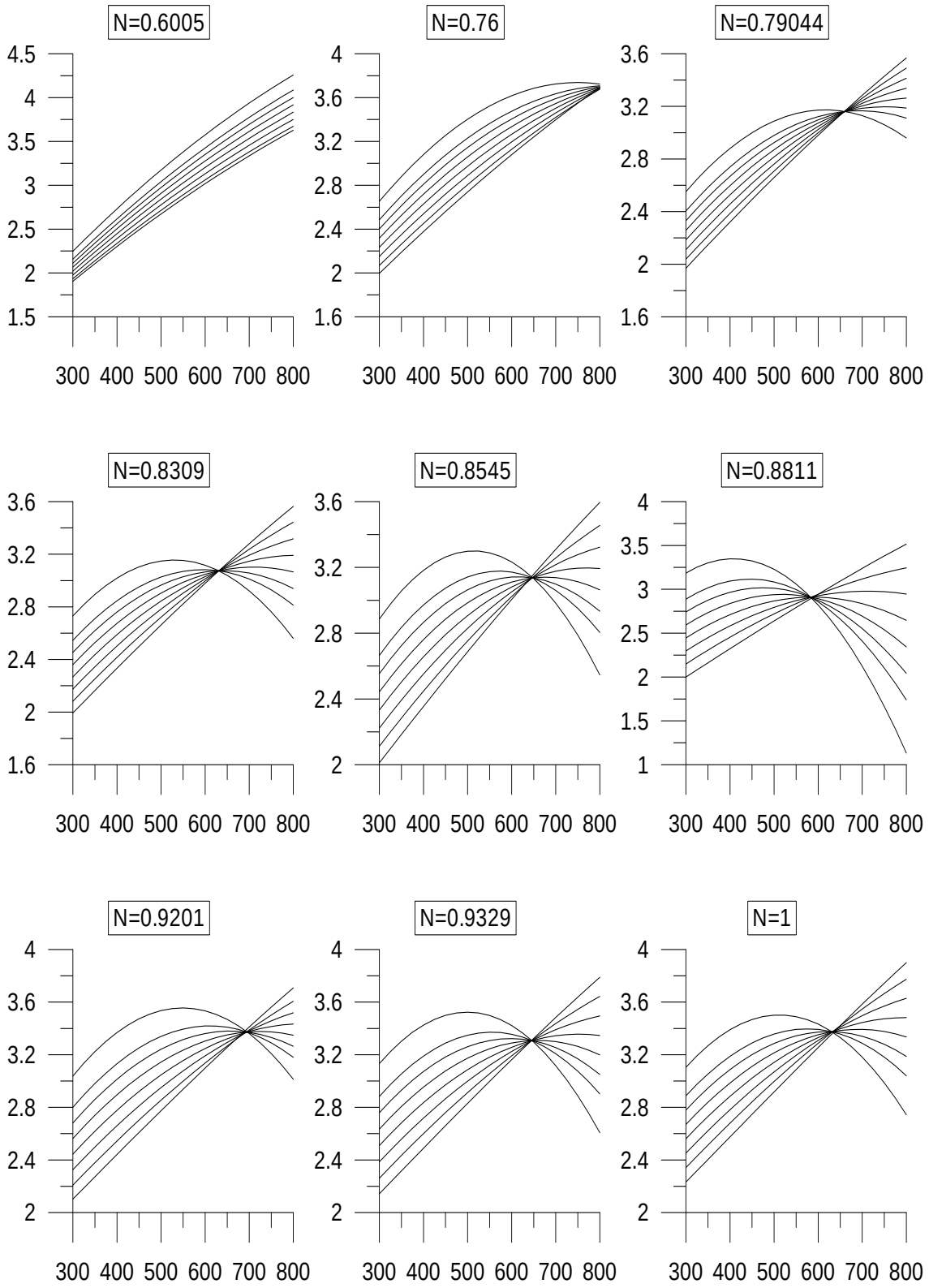
Graph 17



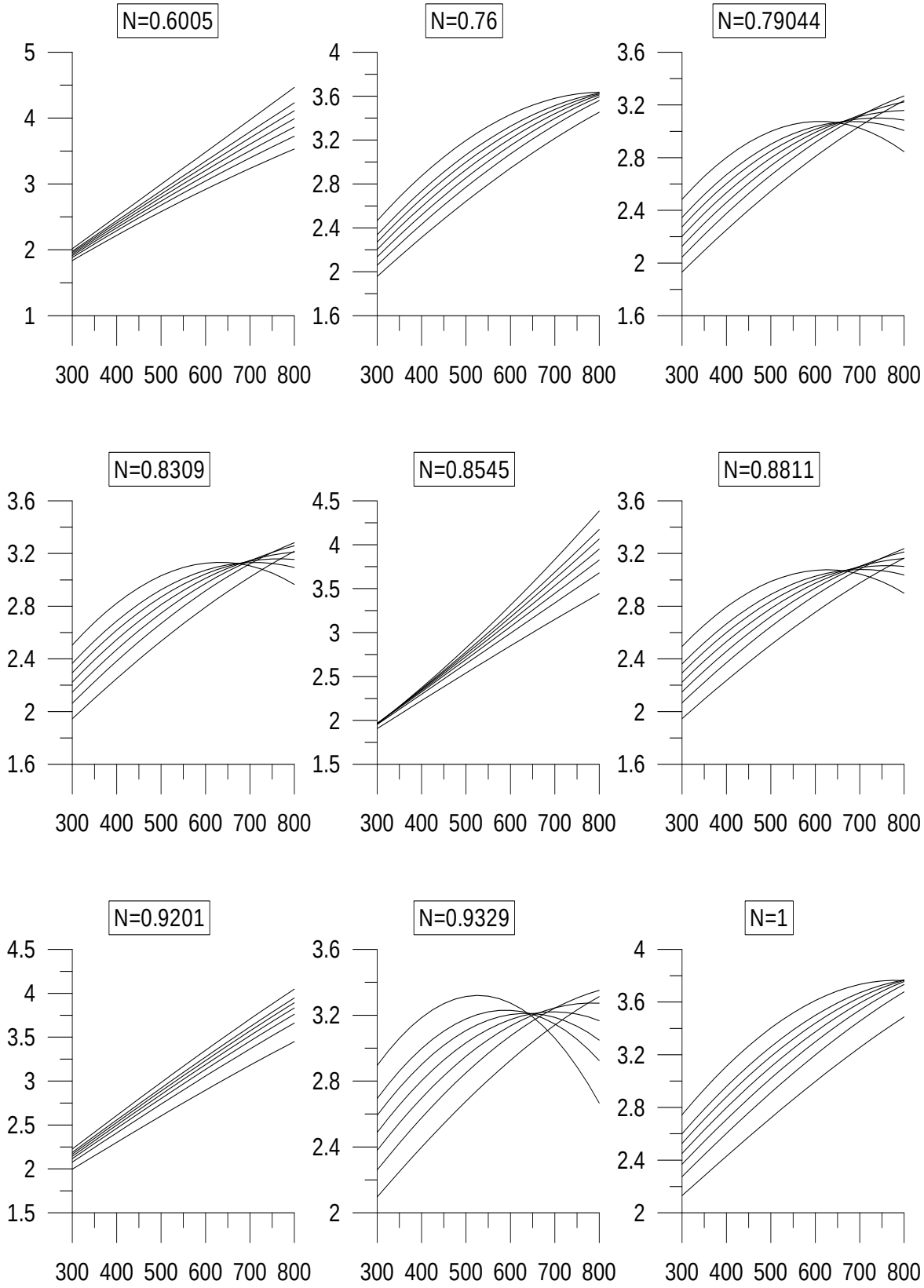
Graph 18 : Van Der Waals Actual Cp



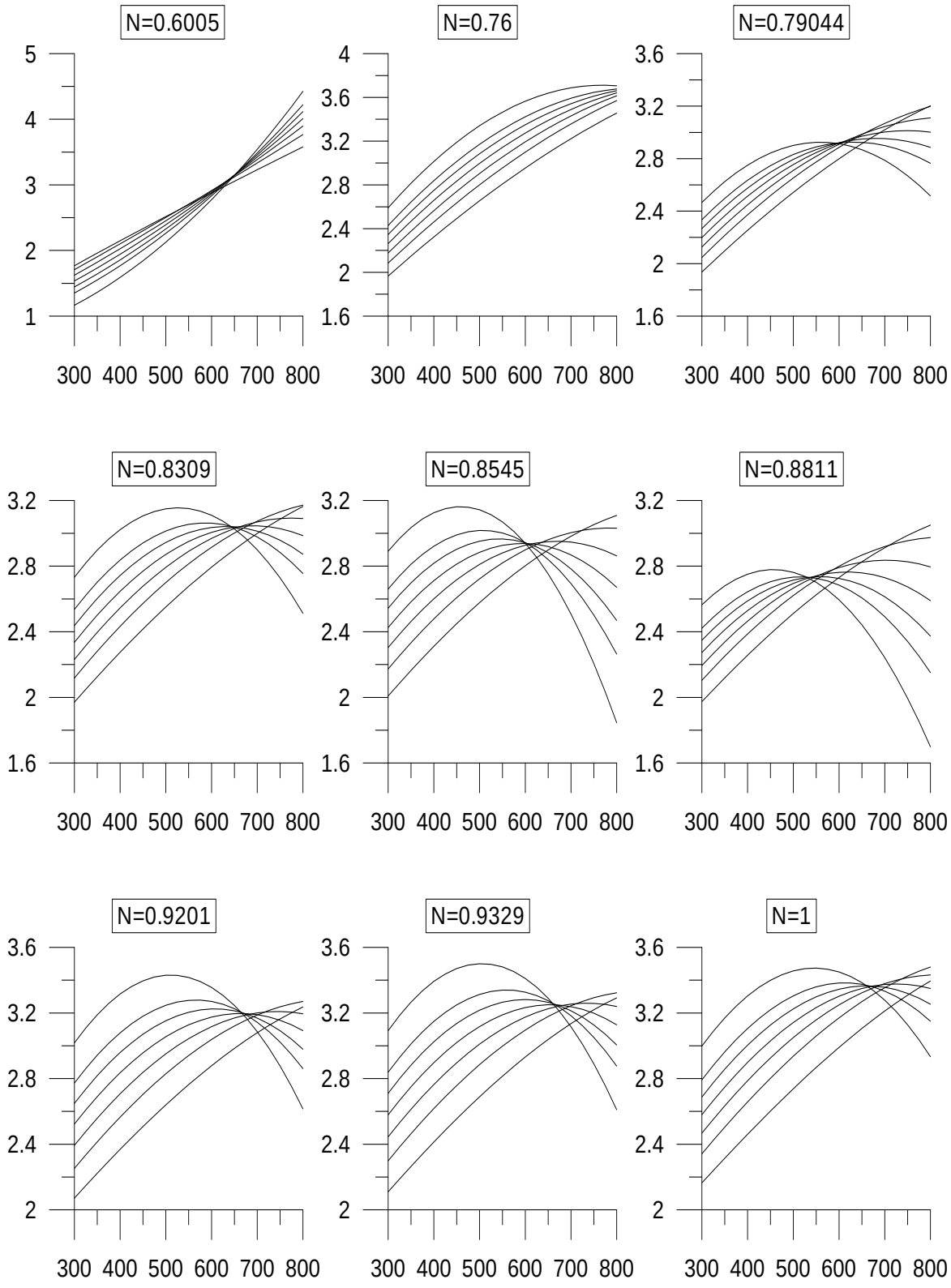
Graph 19: Redlish Actual Cp



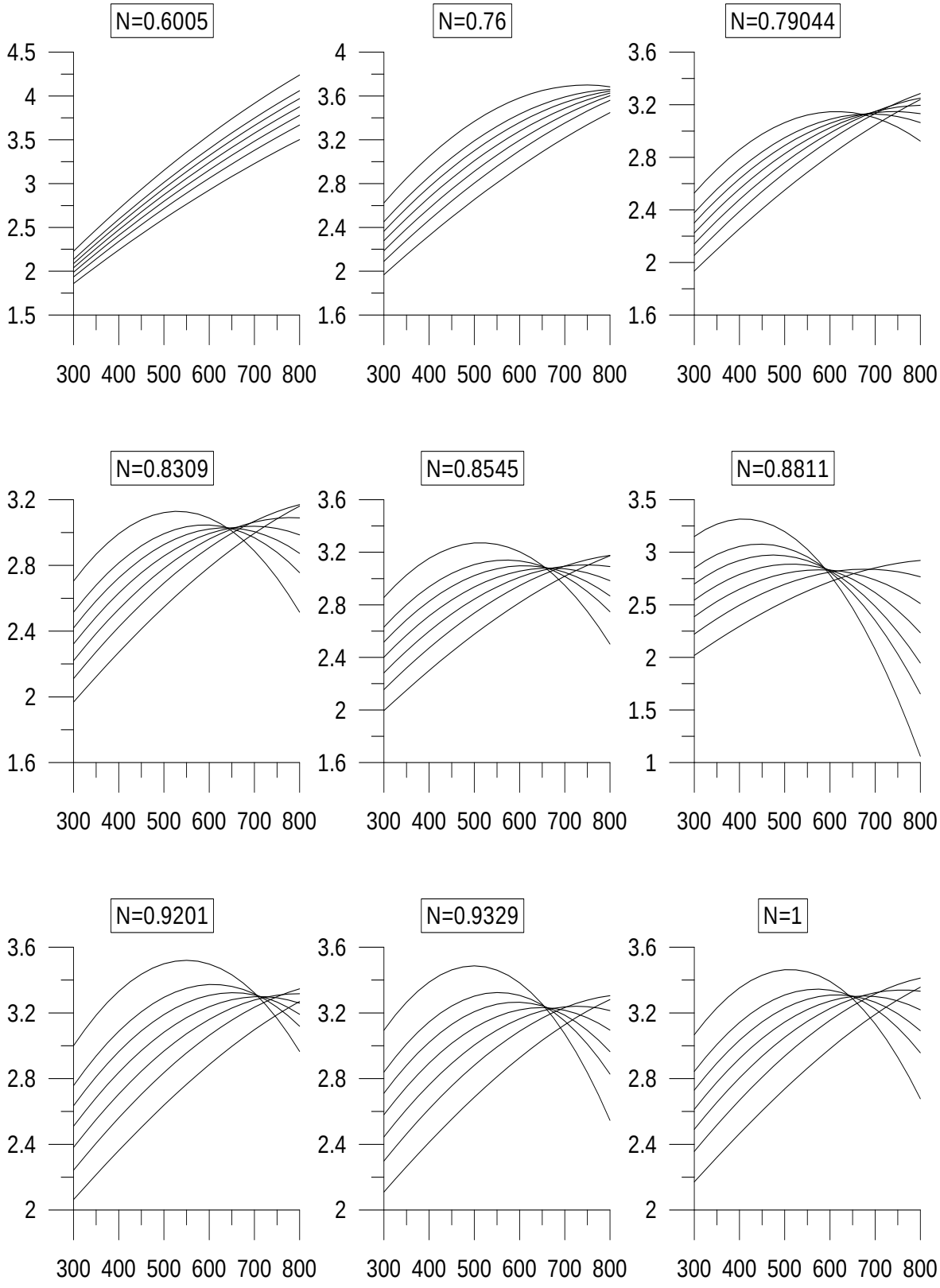
Graph 20: BWR Acutal Cp



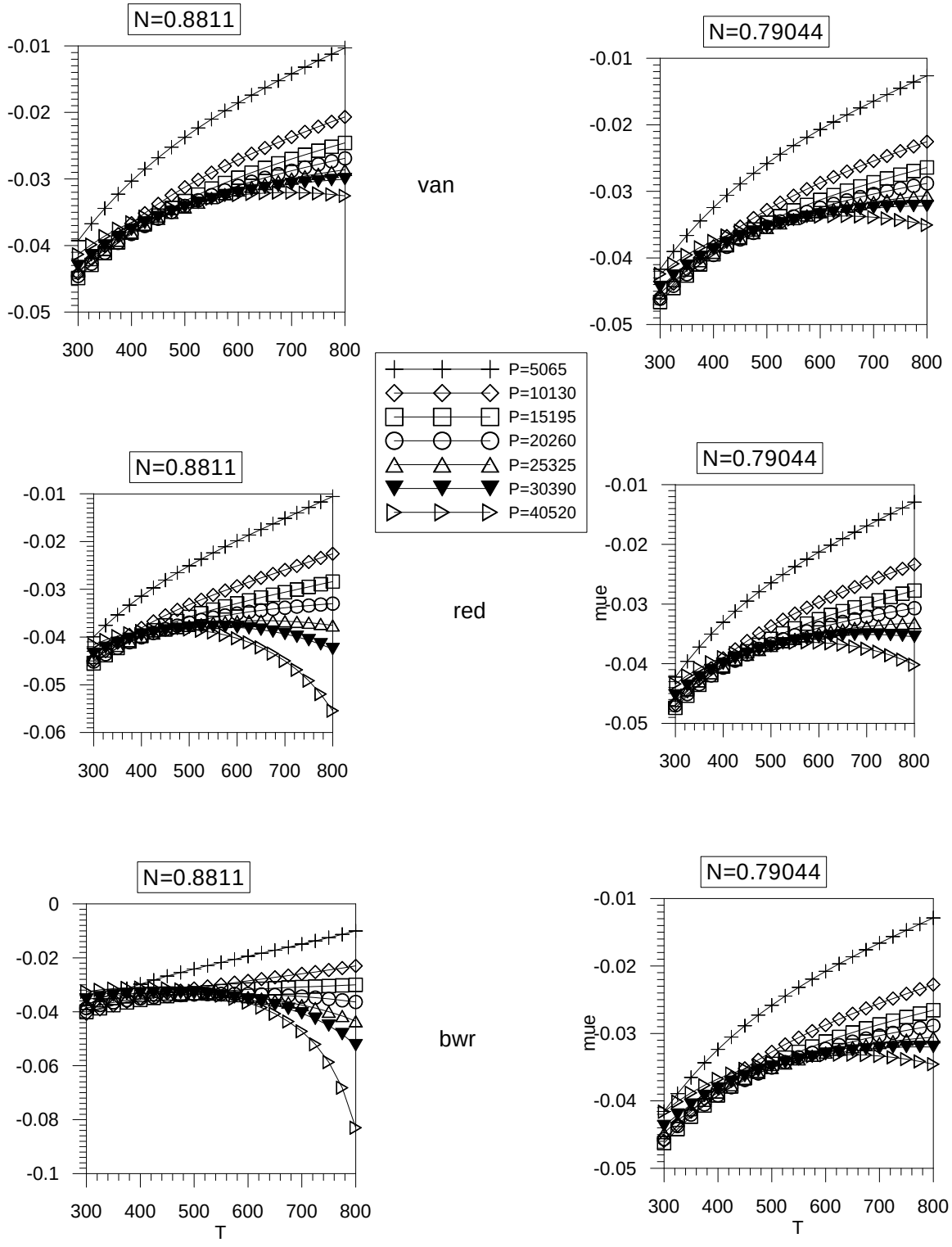
Graph 21: Van Der Waal Cv



Graph 22: Redlish Cv

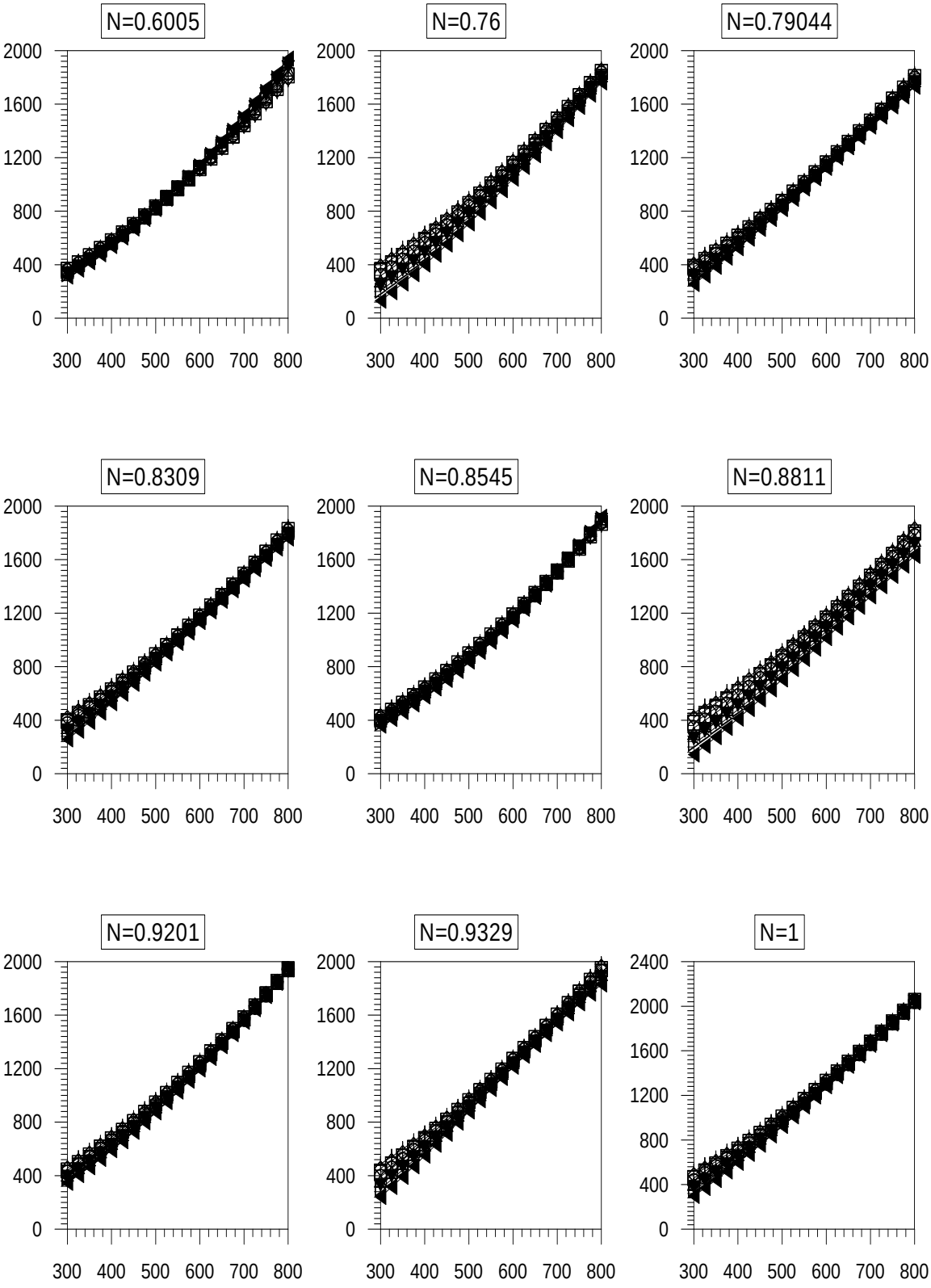


Graph 23: BWR Cv

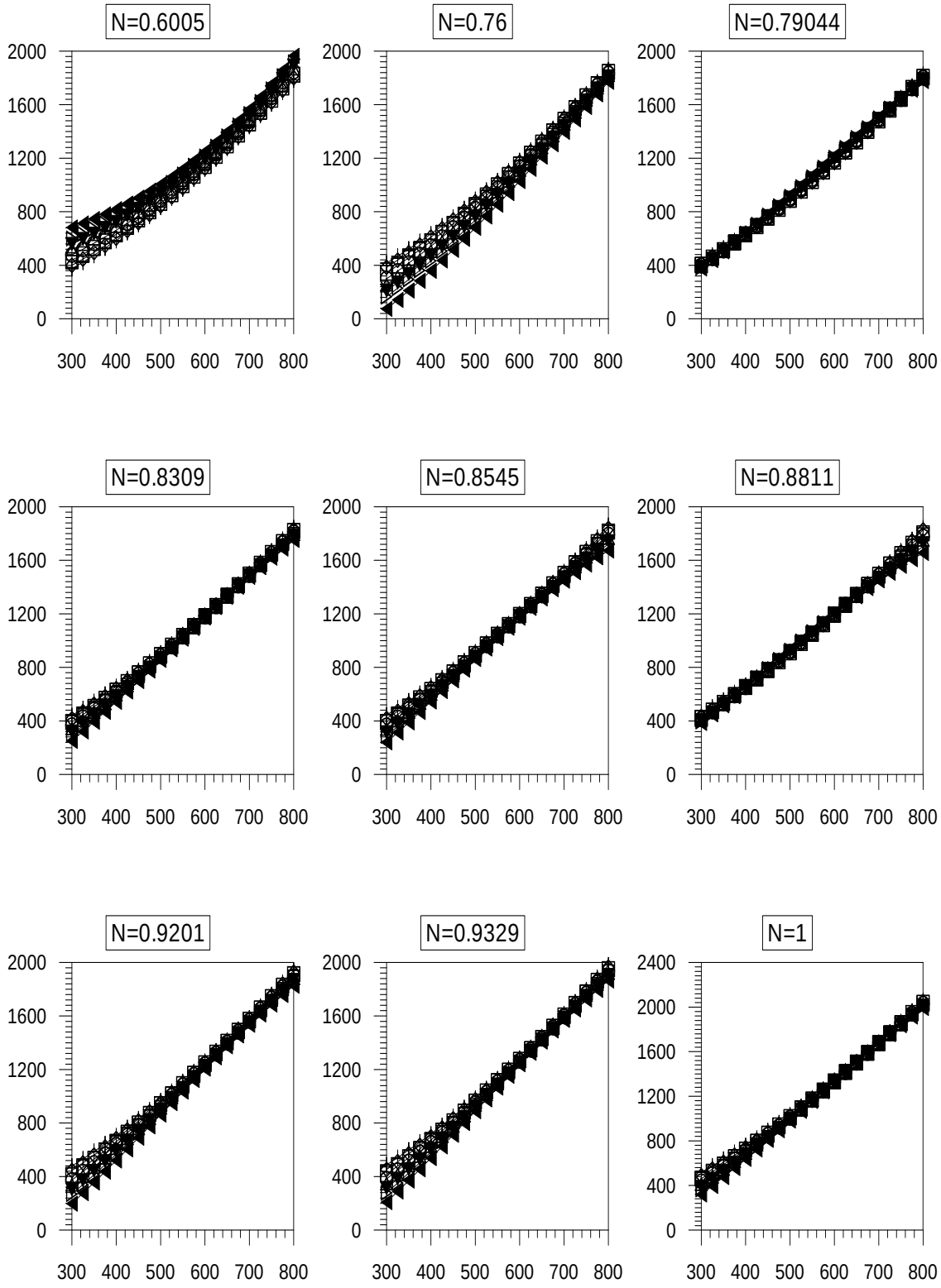


mue at minimum v for van, red, and bwr starting from 50 bar to 400 bar

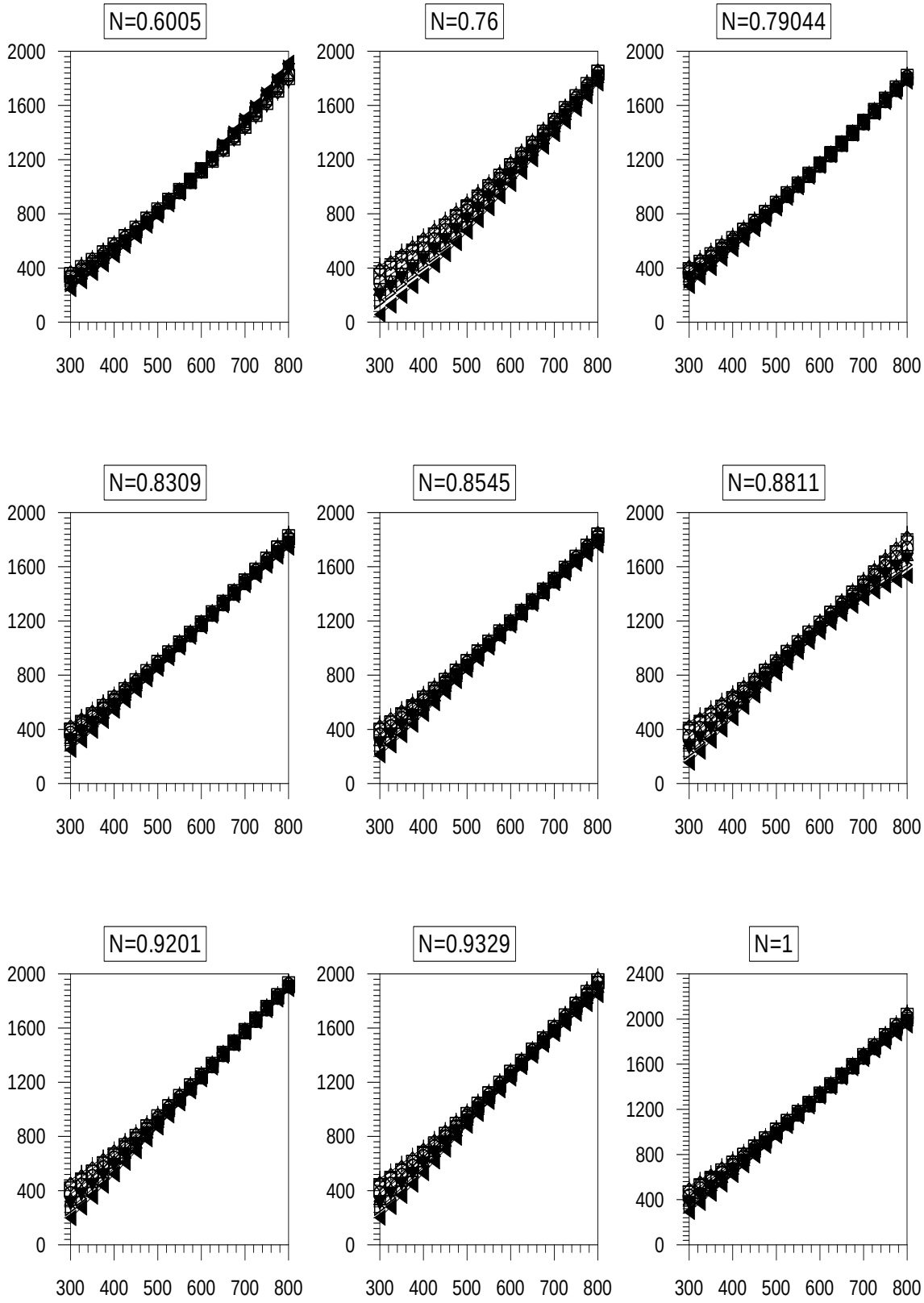
Graph 24



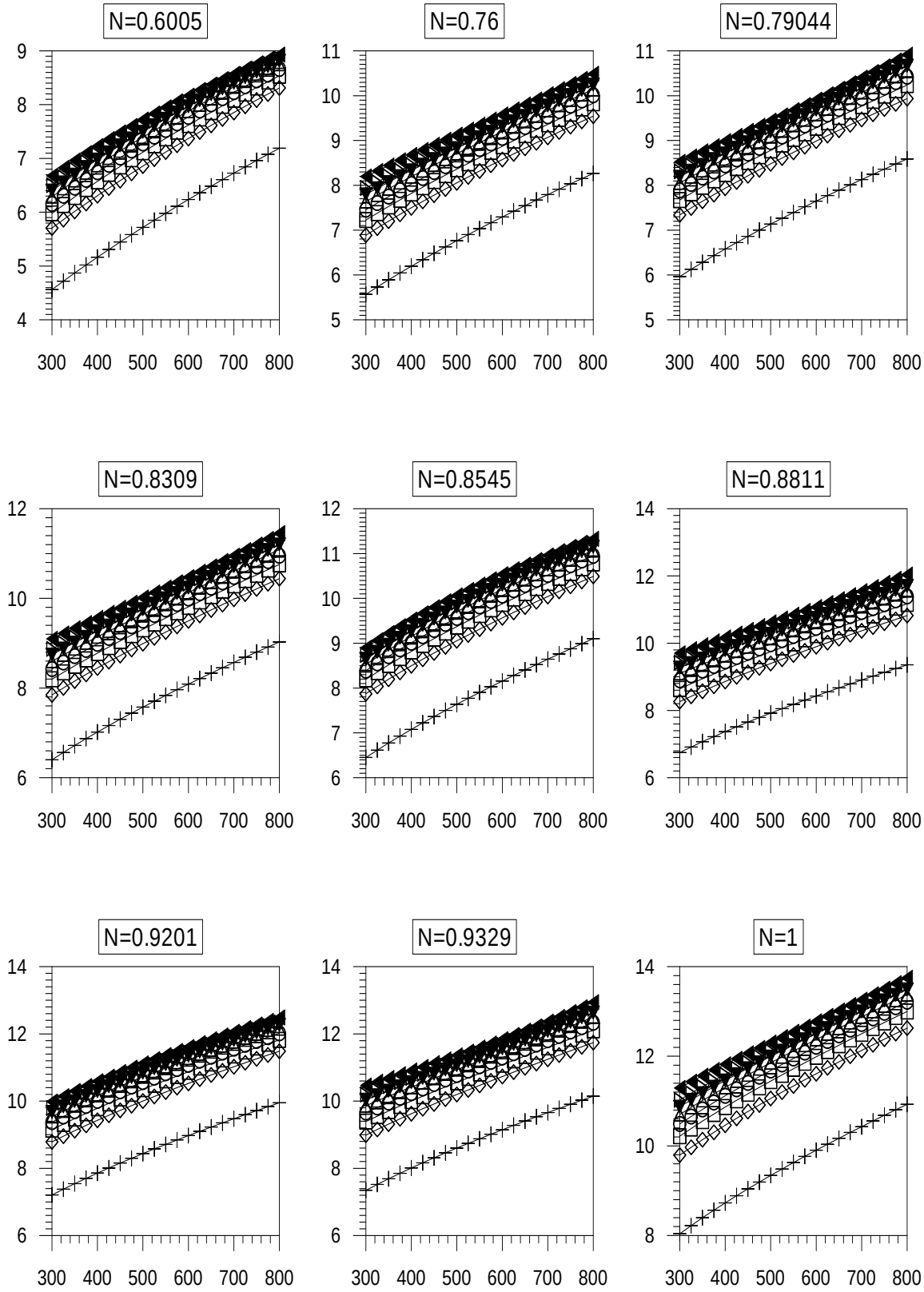
Graph 25: Van Der Waal Enthalpy



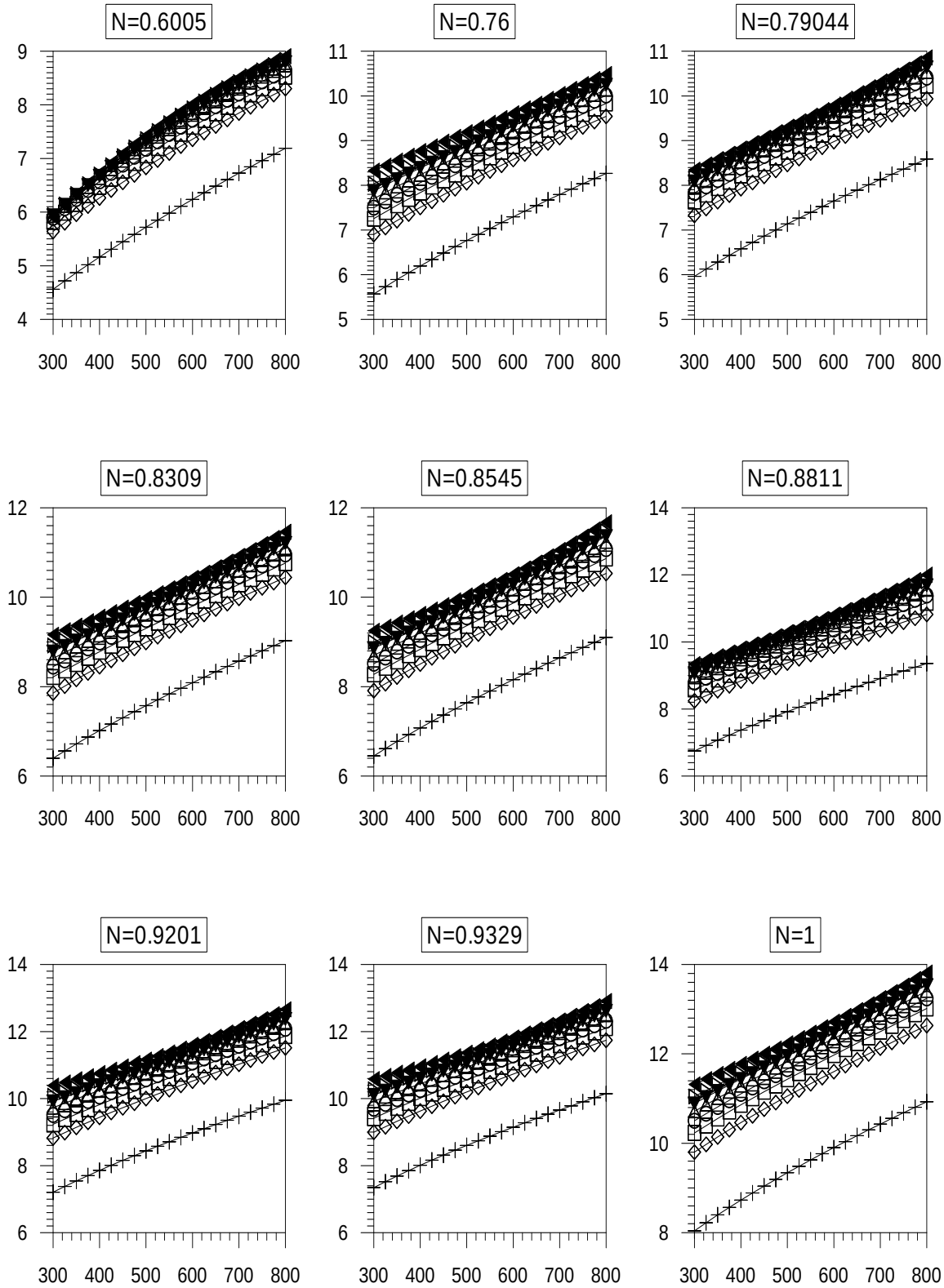
Graph 26: Redlish Enthalpy



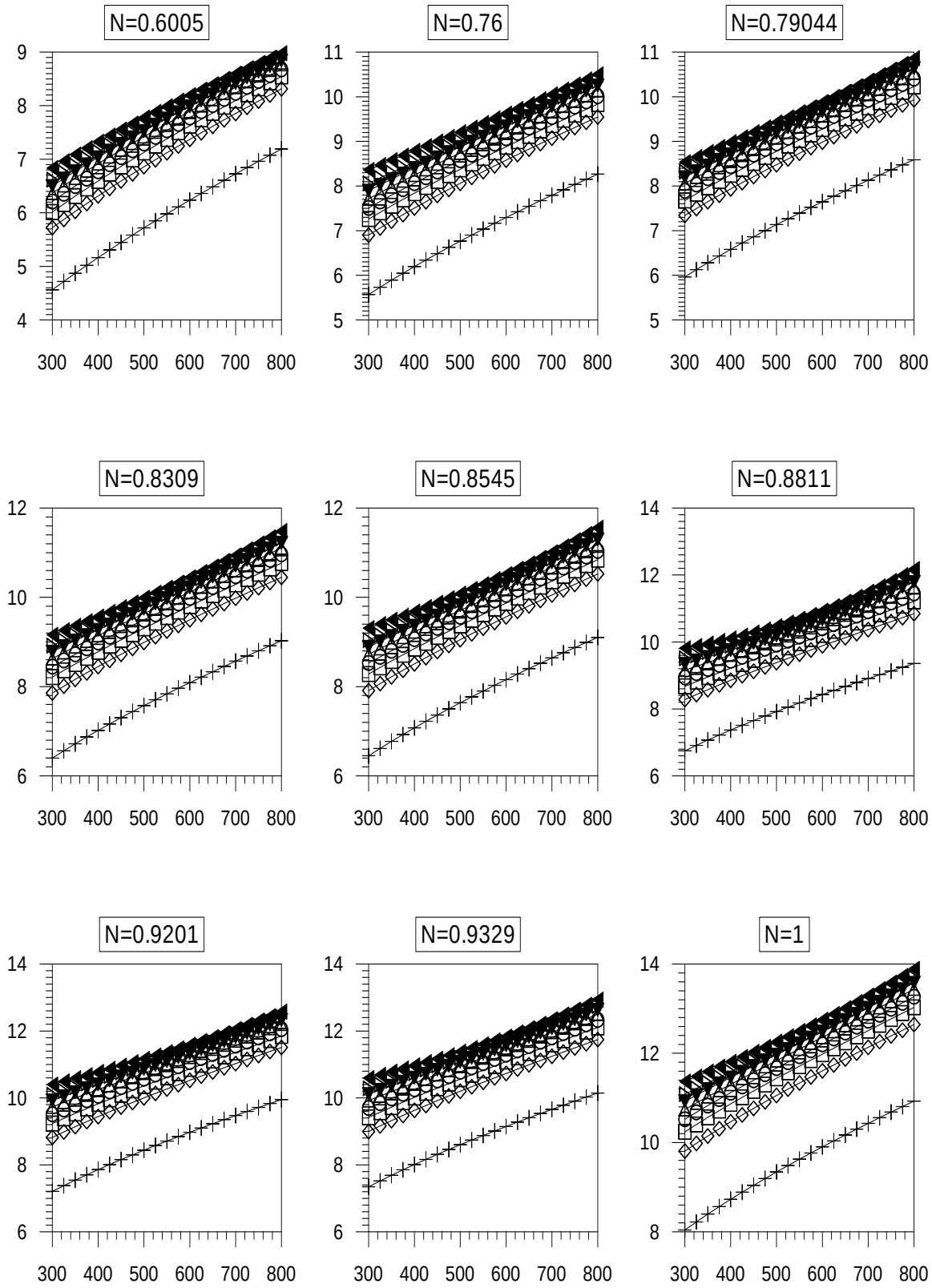
Graph 27: BWR Enthalpy



Graph 28: Van Der Waal Entropy



Graph 29: Redlish Entropy



Graph 30: BWR Entropy

Refrences

[I] Schley. P. and Jaeschke M., Ideal-Gas thermodynamic properties for natural gas applications, international Journal of thermo physics. Vol.16 No 6. 1995.
[II] El Gizawy.