# Process Model Documentation

Matthew Cheung BEng (Hons) AMIChemE

November 19, 2020

**Abstract**

Chemical engineering has been defined by topuniversities.com as "a multi-disciplinary branch of engineering that combines natural and experimental sciences (such as chemistry and physics), along with life sciences (such as biology, microbiology and biochemistry) plus mathematics and economics to design, develop, produce, transform, transport, operate and manage the industrial processes that turn raw materials into valuable products."[1] This document covers the theory for each subroutine presented in the accompanying software written in the C programming language.

# Contents

# List of Figures

# List of Tables



# List of Equations

# Introduction

Whether it be from energy production to producing resources for society, we constantly see the products of chemical engineers all around us in modern day life, the demand for which can only rise with each day due to an ever-increasing world population. To be able to meet this increasing demand, process engineers need to be able to perform numerous calculations to engineer solutions that meet society's demand for resources. This program is aimed at process engineers to aid in validating solutions from standard process simulation software (e.g. Aspen HYSYS, ChemCAD etc.).

# 1 Program Structure

## 1.1 File Structure

The C programming language is a procedural language which means that it will systematically read the source code from one line to the next (i.e. it is methodical). Across C programming tutorials, a program is typically laid out as follows:

1. Header files and local variable definitions.
2. System entry point.
3. Additional functions.

Within this program, each file can be separated into performing one of two functions. These are menus and calculation subroutines. The menu functions are self explanatory and are used to navigate through the different subroutines available with the program. The calculation subroutines are used to guide the user through collecting data and running calculations. As such, the header files for the calculation subroutines are available to be used throughout the program.

Thus, for any calculation subroutine, the structure is as follows:

1. Header file declarations from the standard C library.
2. Custom header file declarations local to the program.
3. Variable definitions.
4. Loading of database (where appropriate).
5. Variable collection.
6. Single calculation steps.
7. Iterative calculations (where appropriate).
8. Display on user console.
9. Write to disk.

10. Display/Write switch.

11. System entry point/Pseudo-main function.

## 1.2 Contentious Issues

- Graphics are not utilised to display the generated datasets.

  This program is currently being designed using header files found within the C11 standard. '`Graphics.h`' is a header file found exclusively within the Turbo C DOS compiler. This would therefore mean that this program would only run on windows machines. This is against the authors belief that this program should be able to run on any machine. That being said, this feature will be worked on when the program is ported to C++.

- Equations have been separated into one operation per line.

  This has been done to reduce the workload on the compiler and hence the time required to compile the entire program.

- Numerical data is written to a .txt file rather than a .csv file.

  The primary tool being used to externally validate the results of this program is Microsoft Excel. Currently, Excel is able to open a .csv file natively which can cause complications if the user begins adding formulas to the .csv file. A .txt file was chosen since this forces the user to import the file into a .xls, and related formats, through the in-built data features (Data > Get External Data > From Text).

## 1.3 system.c and system.h

It was found during software development that certain lines were being repeated throughout the program. As this program grows in size, this approach is not does not allow for easy maintenance. As a direct result, a `system.c` and `system.h` file are included which include common subroutines. The functions contained within `system.c` can be accessed by any function by simply calling `system.h` where the functions are required.

### API

```
double inputDouble(int allowZero, int allowNeg, char VariableName[], char Units[])
```

This subroutine is used to collect character input and output the collected characters to a double variable. This subroutine is also able to force the input of a non-zero value and/or a positive value.

```
double timer(struct timespec start, struct timespec end)
```

This subroutine is used to calculate the time taken by the calculation step of a subroutine.

```
double pcterror(double input, double data)
```

This subroutine is used to calculate the percentage error that a calculated value has from the empirical value.

```
int Continue(int ControlVariable)
```

This subroutine is used to ask the user whether or not they would like to exit the subroutine that this subroutine is called to.

# 2 Thermophysical Properties

Chemical engineering involves the design of unit operations which either require reliable empirical data or means of property estimation to design processes whilst reducing the uncertainty in the conducted calculations. This chapter explores the variables often required when it comes to process design and common practices for estimating properties which are not readily available.

## 2.1 Pure Component Constants

### 2.1.1 Critical Properties

### 2.1.2 Acentric Factor

The acentric factor is used to mathematically describe how spherical a real molecule is. [3] provide three different methods for its calculation:

1. From the reduced vapour pressure.

$$\omega = -\log \hat{P}_r|_{T_r=0.7} - 1.000$$

   Where $\hat{P}_r = \hat{P}/P_c$ is the reduced vapour pressure at a reduced temperature of 0.7.

2. From the critical pressure and temperature and the normal boiling point.

$$\omega = \frac{3}{7}\frac{\Theta}{1-\Theta}\log P_c - 1$$

   Where $\Theta = T_b/T_C$ with $T_b$ being the normal boiling point and the critical pressure given in atmospheres.

3. With the Lee-Kesler vapour pressure relations.

$$\omega = \frac{\alpha}{\beta}$$

$$\alpha = -\ln P_C - 5.92714 + \frac{6.096480}{\Theta} + 1.28862\ln\Theta - 0.169347\Theta^6$$

$$\beta = 15.2518 - \frac{15.6875}{\Theta} - 13.4721\ln\Theta + 0.43577\Theta^6$$

   Where:

   - $\Theta = T_b/T_C$ with $T_b$ being the normal boiling point.
   - $P_C$ is the critical pressure is given in atmospheres.

#### API

The implementation for all functions stated below can be found in `Thermophysical Properties/01-Pure Component Constants/02AcentricFactor.c`. At the same file path, the header file containing the function declarations can be found in `02AcentricFactor.h`

`void AcFactorVariable(int mode, double *Pc, double *Tc, double *TBoil, double *ANTA, double *ANTB, double *ANTC)`

This subroutine is used to collect the data required to calculate the acentric factor through either the Antoine equation or the Lee-Kesler vapour pressure relations.

`double AntoineEquation(double ANTA, double ANTB, double ANTC, double T)`

This subroutine is used to calculate the vapour pressure with the Antoine equation in mmHg.

`double AcFactorPHatCalculation(double ANTA, double ANTB, double ANTC, double Tc, double Pc)`

This subroutine is used to calculate the acentric factor through the Antoine equation in mmHg.

`double AcFactorCritCalc(double Pc, double Tc, double TBoil)`

This subroutine is used to calculate the acentric factor using the critical pressure and temperature and the normal boiling point.

`double AcFactorLKCalc(double Pc, double Tc, double TBoil)`

This subroutine is used to calculate the acentric factor through the Lee-Kesler vapour pressure relations.

`void AcFactorDisplay(int mode, double Pc, double Tc, double TBoil, double ANTA, double ANTB, double ANTC, double omegaANT, double omegaCRI, double omegaLK)`

This subroutine is used to display the inputted parameters and calculated acentric factor on the user console.

`void AcFactorWrite(int mode, double Pc, double Tc, double TBoil, double ANTA, double ANTB, double ANTC, double omegaANT, double omegaCRI, double omegaLK)`

This subroutine is used to write the inputted parameters and calculated acentric factor to a .txt file.

`void AcFactorSwitch(int mode1, int mode2, double Pc, double Tc, double TBoil, double ANTA, double ANTB, double ANTC, double omegaANT, double omegaCRI, double omegaLK)`

This subroutine is used to ask the user whether or not they would like to write the inputted parameters and calculated acentric factor to a .txt file.

`double AcentricFactor(void)`

This subroutine is used to guide the user through the data collection and calculation of the acentric factor.

## 2.2 Equations of State

### 2.2.1 Virial Equation of State

The virial equation of state is given by the following expansion after being derived from pertubation theory in Statistical mechanics.

#### Derivation[4]

On a Pressure-Volume diagram for a gas/vapour, it is found that the molar volume decreases as system pressure increases as temperature is held constant. The following expansion can then

be made:

$$PV = a + bP + cP^2$$

Let $b \equiv aB'$, $c \equiv aC'$, etc. Substituting and rearranging the above will then yield:

$$PV = a(1 + B'P + C'P^2 + D'P^3 + ...)$$

In the limit of zero pressure, empirical evidence finds that

$$\lim_{P \to 0} PV = a = f(T)$$

Since we also know that this quantity varies with temperature, we can deduce the following relationship:

$$(PV) \propto T$$

Otherwise known as the ideal gas law:

$$\lim_{P \to 0} PV = a \equiv RT$$

The auxilery thermodynamic property called the compressibility factor is now defined as:

$$Z \equiv \frac{PV}{RT} = \frac{V}{V^{ig}} \tag{2.2.1}$$

The latter part of equation 2.2.1 is put examined closer in section 3.4.1, but for now, let's get back to deriving the Virial equation of state. Recognising that equation 2.2.1 with the evidence that $a = RT$, we can determine the following equation in terms of the compressibility factor:

$$\begin{aligned} Z &= 1 + B'P + C'P^2 + D'P^3 + ... \\ Z &= 1 + \frac{B}{V} + \frac{C}{V^2} + \frac{D}{V^3} + ... \end{aligned} \tag{2.2.2}$$

It now becomes trivial to derive the following equation which can be used to calculate the system pressure given the molar volume and temperature of an isotherm:

$$P = \frac{RT}{V} + \frac{RTB}{V^2} + \frac{RTC}{V^3} + ... \tag{2.2.3}$$

Where the parameters $B$, $C$,... are called the second, third, ... virial coefficients and are functions of temperature only for a pure fluid. When going from experimental data, the second virial coefficient is defined as:

$$B = \lim_{1/V \to 0} \left( \frac{\delta Z}{\delta 1/V} \right)_T \tag{2.2.4}$$

This provides the following observations:

- As the density tends to 0, the truncated virial equations become identical.
- This equation can appropriately model true fluid behaviour at low density values (i.e. for

gases).

- The equation predicts that the compressibility factor is a linear function of pressure along an isotherm.

**Virial coefficient estimation**

To estimate the second virial coefficient, [3] and [4] recommend using the following equations for non-polar molecules:

$$\begin{aligned} \frac{BP_c}{RT_c} &= B^{(0)} + \omega B^{(1)} \\ B^{(0)} &= 0.083 - \frac{0.422}{T_r^{1.6}} \\ B^{(1)} &= 0.139 - \frac{0.172}{T_r^{4.2}} \end{aligned} \tag{2.2.5}$$

Additionally, if the molecule under examination is polar, the above principle equation becomes[3]:

$$\begin{aligned} \frac{BP_c}{RT_c} &= B^{(0)} + \omega B^{(1)} + B^{(2)} \\ B^{(2)} &= \frac{a}{T_r^6} - \frac{b}{T_r^8} \end{aligned} \tag{2.2.6}$$

Where $a$ and $b$ are constants specific to members of a homologous series:

Table 2.2.1: Constants for $B^{(2)}$

| Compound class | $a$ | $b$ |
|---|---|---|
| Ketones, aldehydes, nitriles | $-2.112 \times 10^{-4}\mu_r - 3.877 \times 10^{-21}$ | 0 |
| Ethers, $NH_3$, $H_2S$, HCN, esters | $\mu_r^8$ | 0 |
| Mercaptans | 0 | 0 |
| Monoalkyhalides | $2.076 \times 10^{-11}\mu_r^4 - 7.048 \times 10^{-21}\mu_r^8$ | 0 |
| Alcohols | 0.0878 | $0.04 - 0.06$ |
| Phenol | $-0.0136$ | 0 |

The reduced dipole moment defined as:

$$\mu_r = \frac{10^5 \mu^2 P_c}{T_c^2} \tag{2.2.7}$$

Where:

- $\mu$ = Dipole moment, debyes.
- $P_c$ = Critical pressure, bar.
- $T_c$ = Critical temperature, K.

To estimate the third virial coefficient, [4] recommends using the following equation:

$$\begin{aligned}\frac{CP_c^2}{R^2T_c^2} &= C^{(0)} + \omega C^{(1)} \\ C^{(0)} &= 0.01407 + \frac{0.02432}{T_r} - \frac{0.00313}{T_r^{10.5}} \\ C^{(1)} &= -0.02676 + \frac{0.05539}{T_r^{2.7}} - \frac{0.00242}{T_r^{10.5}}\end{aligned} \tag{2.2.8}$$

[4] also recommends two methods of calculating the compressibility factor, both of which utilise the virial coefficients correlated by Pitzer.

The first of which utilises the following definition of the compressibility factor, a ratio used to measure the "deviation of the real-gas molar volume from its ideal gas state":

$$Z = \frac{PV}{RT} = Z^0 + \omega Z^1 \tag{2.2.9}$$

The above correlation is based on data from the noble gases: argon, krypton and xenon. Comparing the above to 2.2.5 reveals the following relationships for estimating the quantities $Z^0$ and $Z^1$:

$$\begin{aligned}Z &= 1 + B^0\frac{P_r}{T_r} + \omega B^1\frac{P_r}{T_r} \\ Z^0 &= 1 + B^0\frac{P_r}{T_r} \\ Z^1 &= B^1\frac{P_r}{T_r}\end{aligned} \tag{2.2.10}$$

The source reports that the above is valid at low to moderate pressures when the compressibility factor is linear in pressure providing good applicability into standard chemical processes. It was also reported that that the above correlation is most accurate for nonpolar species and least accurate for highly polar and associating molecules (molecules where not just hydrogen bonding is present).

The second method that [4] recommends uses the following equation for converging a value through successive substitution of $Z$:

$$\begin{aligned}Z &= 1 + \hat{B}\frac{P_r}{T_rZ} + \hat{C}\left(\frac{P_r}{T_rZ}\right)^2 \\ \hat{B} &= \frac{BP_c}{RT_c} \\ \hat{C} &= \frac{CP_c^2}{R^2T_c^2}\end{aligned} \tag{2.2.11}$$

It was reported that $Z$ can be found by inserted an initial value of $Z = 1$ on the right hand side and successively converging a solution such that the following iteration scheme is true:

$$Z_{i+1} = 1 + \hat{B}\frac{P_r}{T_rZ_i} + \hat{C}\left(\frac{P_r}{T_rZ_i}\right)^2$$

### API

The implementations for all functions declared in this API section can be found in `Thermophysical Properties/02 State Property Relations/02VirialEOS.c`. Due to the shear number of function declarations made, the declarations for the functions directly involving the Virial equation of state can be found in `02VirialEOS.h` and any functions directly related to the compressibility factor can be found in `02Compressibility.h`. These header files can be found in the same file path.

```
void VirialEOSVariable(int polar, double *Pc, double *Tc, double *Vc, double *acFactor,
double *a, double *b)
```

This subroutine is used to collect the data required to calculate an isotherm for the Virial equation of state.

```
double reducedProperty(double critical, double process)
```

This subroutine is used to calculate the reduced state variable.

```
double VirialEOSB0Calc(double Tr)
```

This subroutine is used to calculate $B^(0)$ required for calculating the second virial coefficient.

```
double VirialEOSB1Calc(double Tr)
```

This subroutine is used to calculate $B^(1)$ required for calculating the second virial coefficient.

```
double VirialEOSB2Calc(double a, double b, double Tr)
```

This subroutine is used to calculate $B^(2)$ required for calculating the second virial coefficient for a polar molecule.

```
double VirialEOSBHat(double B0, double B1, double B2, double accFactor)
```

This subroutine is used to calculate $\hat{B}$ required for calculating the second virial coefficient.

```
double VirialEOSBCalc(double BHat, double Pc, double Tc)
```

This subroutine is used to calculate the second virial coefficient.

```
double VirialEOSC0Calc(double Tr)
```

This subroutine is used to calculate $C^(0)$ required for calculating the third virial coefficient.

```
double VirialEOSC1Calc(double Tr)
```

This subroutine is used to calculate $C^(1)$ required for calculating the third virial coefficient.

```
double VirialEOSCHat(double C0, double C1, double accFactor)
```

This subroutine is used to calculate $\hat{C}$ required for calculating the third virial coefficient.

```
double VirialEOSCCalc(double CHat, double Pc, double Tc)
```

This subroutine is used to calculate the third virial coefficient.

```
double VirialEOSCalc(double T, double V, double B, double C)
```

This subroutine is used to calculate the Pressure from the virial equation of state.

```
double VirialEOSCompCalc(double P, double T, double B, double C)
```

This subroutine is used to calculate the compressibility factor using the state definition from 'EOSIsotherm' data and virial coefficients.

```
EOSIsotherm VirialEOSIsotherm(double Pc, double Tc, double Vc, double T, double omega,
double *B, double *C)
```

This subroutine is used to generate the isotherm related to the virial equation for a non-polar molecule.

```
EOSIsotherm VirialEOSIsothermPolar(double Pc, double Tc, double Vc, double T, double omega,
double a, double b, double *B, double *C)
```

This subroutine is used to generate the isotherm related to the virial equation for a polar molecule.

```
ZFactor VirialEOSCompIsotherm(double Pc, double Tc, double T, double omega, double *B,
double *C)
```

This subroutine is used to calculate the isotherm for a Compressibility-factor graph for a non-polar molecule.

```
ZFactor VirialEOSCompIsothermPolar(double Pc, double Tc, double T, double omega, double
a, double b, double *B, double *C)
```

This subroutine is used to calculate the isotherm for a Compressibiltiy-factor graph for a polar molecule.

```
void VirialEOSDisplay(int polar, double Pc, double Tc, double Vc, double T, double omega,
double a, double b, EOSIsotherm data, double B, double C)
```

This subroutine is used to display the calculated isotherm related to the virial equation for any molecule on the user console.

```
void VirialEOSCompDisplay(int polar, double Pc, double Tc, double Vc, double T, double
omega, double a, double b, ZFactor data, double B, double C)
```

This subroutine is used to display the generated Compressibiltiy factor isotherm.

```
void VirialEOSWrite(int polar, double Pc, double Tc, double Vc, double T, double omega,
double a, double b, EOSIsotherm data, double B, double C)
```

This subroutine is used to write the generated isotherm to a .txt file.

```
void VirialEOSCompWrite(int polar, double Pc, double Tc, double Vc, double T, double omega,
double a, double b, ZFactor data, double B, double C)
```

This subroutine is used to write the compressibility factor isotherm to a .txt file.

```
void VirialEOSSwitch(int mode1, int mode2, int polar, double Pc, double Tc, double Vc,
double T, double omega, double a, double b, EOSIsotherm dataV, ZFactor dataZ, double B,
double C)
```

This subroutine is used to ask the user whether they would like to write the generated isotherm to a .txt file.

```
void VirialEOS(void)
```

This subroutine is used to guide the user through collecting the data and generating the isotherms associated with the Virial equation of state.

### 2.2.2 Cubic Equations of State

The above derivation showed an important relationship that formed the basis of the following cubic equations of state.

$$PV_m = RT \tag{2.2.12}$$

Although this relationship can also be derived from physical chemistry, it is important to recognise the shortcomings of the above equation. The derivation for the virial equation of state shows that this relationship is only true at low pressures. Additional research finds that the

ideal gas assumption assumes that all molecules present are point particles with no mass and negligible physical volume in addition to there being no intermolecular forces. With many real substances, this is not the case. In fact, we know that molecules of the same species can interact with each other (e.g. H-bonds, dimerisation etc.). So how do we account for such? Well, J.D. van der Waals found in 1873 his famous cubic equation of state. However, going from first principles:

$$Pv = RT$$

Attractive and repulsive forces between molecules will influence the change in momentum that a molecule will experience when in close proximity of another assuming no chemical reaction. Additionally, all molecules, at the macroscopic scale, possess mass, sizing down to the quantum scale, this must also be true as this observation must come from somewhere. Since the molecule must possess mass, it must also have some volume. This constant must therefore have a value smaller than the observed molar volume. We also know from action potentials that there must be some minimum distance at which the repulsive force of the single species will be exerted on neighbouring species. This will noticeably increase the observed thermodynamic pressure.

$$\Rightarrow (P + (C_F))(V_m - V_{\text{actual}}) = RT$$

Empirical evidence finds that $C_F$ is somehow related to the energy density, per mole of species. Indeed, this is what van der Waals found in his equation which is commonly stated as:

$$\left(P + \frac{a}{V^2}\right)(V_m - b) = RT \tag{2.2.13}$$

The past century of research has revealed that the van der Waals equation of state is wrong with respect that it fails to account for the molecular conformation in addition to molecules that possess the same critical properties as a different species will produce the same isotherm. Since molecules with different arrangements of atoms must have different intermolecular forces at play, later proposed equations of state account for the shortcomings of the van der Waals equation of state. These modifications either provide an update to the constants used or further consider the acentricity of a molecule.

Throughout industry, the most widely used equations of state are the Redlich-Kwong, Soave-Redlich-Kwong and Peng-Robinson equations of state. It should be noted that amongst these equations, the Peng-Robinson equation of state provides the most accurate predictions of thermodynamic properties. [3] provide the following general form for a cubic equation of state which is also mathematically similar to the general form stated in [4]:

$$P = \frac{RT}{V - b} - \frac{a}{V^2 + ubV + wb^2} \tag{2.2.14}$$

The constants to be used in the equations are as follows:

Traditionally, all the stated above equations are sufficient for modelling vapour behaviour. Shortcomings are introduced within the Redlich-Kwong and Soave-Redlich-Kwong equations when a liquid phase, particularly within mixtures, are being modelled. This was of interest to

Table 2.2.2: Constants for the general cubic equations of state

| Equation | u | w | b | a |
|---|---|---|---|---|
| van der Waals (1873) | 0 | 0 | $\frac{RT_c}{8P_c}$ | $\frac{27}{64}\frac{R^2T_c^2}{P_c}$ |
| Redlich-Kwong (1949) | 1 | 0 | $\frac{0.08664RT_c}{P_c}$ | $\frac{0.42748R^2T_c^{2.5}}{P_cT^{1/2}}$ |

Table 2.2.3: Constants for the general cubic equations of state (Continued)

| Equation | u | w | b | a |
|---|---|---|---|---|
| Soave-Redlich-Kwong (1949) | 1 | 0 | $\frac{0.08664RT_c}{P_c}$ | $\frac{0.42748R^2T_c^2}{P_c}[1+f(\omega)(1-T_r^{1/2})]^2$ $f(\omega)=0.48+1.574\omega-0.176\omega^2$ |
| Peng-Robinson (1976) | 2 | -1 | $\frac{0.07780RT_c}{P_c}$ | $\frac{0.45724R^2T_c^2}{P_c}[1+f(\omega)(1-T_r^{1/2})]^2$ $f(\omega)=0.37464+1.54226\omega-0.26992\omega^2$ |

Peneloux (1982) who proposed the following modification to the Soave-Redlich-Kwong equation of state:

$$V \rightarrow V_m + c$$

Where $c = \sum_i x_i c_i$.
For petroleum and oil:

$$c_i \approx 0.40768\frac{RT_{c,i}}{P_{c,i}}(0.29441 - Z_{\mathrm{RA},i})$$

Where the Rackett compressibility factor can be estimated with:

$$Z_{\mathrm{RA},i} \approx 0.29056 - 0.08775\omega_i$$

It is clear to see that an equation of state can quickly become tailored for members of a homologous series. This also implies that efforts should be made to find an equation of state which matches the process stream. Since the above equations of state are only true for a pure species, this introduces some error into a process engineer's sizing calculations. Peneloux's correction allowed for mixing parameters to be introduced into equations of state where high errors ($> 10\%$) are not acceptable.

### API (Ideal Gas Law)

The implementation for all functions related to calculating the Isotherm related to the ideal gas law can be found in `Thermophysical Properties/02 State Property Relations/01Ideal-Gas.c`. The declarations of the following functions can be found in `01IdealGas.h` at the same file path.

```
void IdealEOSVariable(double *T)
```

This subroutine is used to collect the temperature at which the isotherm according to the Ideal Gas law.

```
double IdealEOSCalculation(double V, double T)
```

This subroutine is used to calculate the pressure according to the Ideal Gas law.

`EOSIsotherm IdealEOSIsotherm(double T)`

This subroutine is used to generate the isotherm along which the ideal gas law is true.

`void IdealEOSDisplay(double T, EOSIsotherm data)`

This subroutine is used to display the inputted parameter and generated isotherm on the user console.

`void IdealEOSWrite(double T, EOSIsotherm data)`

This subroutine is used to write the inputted parameter and generated isotherm to a .txt file.

`void IdealEOSWriteSwitch(double T, EOSIsotherm data)`

This subroutine is used to ask the user whether they would like to save the inputted parameter and generated isotherm to a .txt file.

`void IdealEOS(void)`

This subroutine is used to guide the user through generating the isotherm according to the ideal gas law.

---

The implementation for all functions related to inferring state variables through the combined gas law or through the ideal gas equation can be found in `Common Subroutines/IdealGasLaw.c`. The declarations of the following functions can be found in `IdealGasLaw.h` at the same file path. Note that all the subroutines available below are not explicitly available to the user at run time and are accessed when required during calculation of thermodynamic process profiles for example.

`double IdealPressure(double n, double T, double V)`

This subroutine is used to calculate the pure component pressure from rearrangement of the ideal gas law. This function returns the pure component pressure.

`double SpecPressure(double n, double T, double V, double R)`

This subroutine is used to calculate the pure component pressure from rearrangement of the ideal gas law applied to a real gas. This function returns the pure component pressure.

`double IdealTemperature(double n, double P, double V)`

This subroutine is used to calculate the system temperature from rearrangement of the ideal gas law. This function returns the system temperature.

`double SpecTemperature(double n, double P, double V, double R)`

This subroutine is used to calculate the system temperature from rearrangement of the ideal gas law. This function returns the system temperature.

`double IdealVolume(double n, double P, double T)`

This subroutine is used to calculate the system volume from rearrangement of the ideal gas law. This function returns the pure component volume.

`double SpecVolume(double n, double P, double T, double R)`

This subroutine is used to calculate the system volume from rearrangement of the ideal gas law. This function returns the pure component volume.

---

### API (Cubic Equations of State)

The implementation for all functions related to calculating the Isotherm related to a cubic equation of state except functions used to calculate equation specific constants can be found in `Thermophysical Properties/02 State Property Relations/03CubicEOS.c`. The implementation of functions related to calculating component specific constants can be found in `03CubicEOSConstants.c`. The declarations of the following functions can be found in `03CubicEOS.h`. Both of these files can be found at the same file as the first stated file.

```
void CubicEOSVariable(double *Pc, double *Tc, double *omega)
```

This subroutine is used to collect the required component specific data to determine an isotherm for a P-V-T surface.

```
double VdWcalculateA(double Tc, double Pc)
```

This subroutine is used to calculate the repulsive term in the Van der Waals Equation of State.

```
double VdWcalculateB(double Tc, double Pc)
```

This subroutine is used to calculate the term actual molecular volume term in the Van der Waals Equation of State.

```
double RKcalculateA(double Tc, double Pc, double T)
```

This subroutine is used to calculate the repulsive term in the Redlich-Kwong Equation of State.

```
double RKcalculateB(double Tc, double Pc)
```

This subroutine is used to calculate the actual molecular volume term in the Redlich-Kwong Equation of State.

```
double SRKcalculateAcFunc(double omega)
```

This subroutine is used to calculate the acentric factor function for the Soave-Redlich-Kwong Equation of State.

```
double SRKcalculateA(double Tc, double Pc, double T, double omega)
```

This subroutine is used to calculate the repulsive term in the Redlich-Kwong Equation of State (Soave modification).

```
double PRcalculateAcFunc(double omega)
```

This subroutine is used to calculate the acentric factor function for the Peng-Robinson Equation of State.

```
double PRcalculateA(double Tc, double Pc, double T, double omega)
```

This subroutine is used to calculate the repulsive term in the Peng-Robinson Equation of State.

```
double PRcalculateB(double Tc, double Pc)
```

This subroutine is used to calculate the actual molecular volume term in the Peng-Robinson Equation of State.

```
EOSIsotherm CubicEOSIsotherm(double T, double a, double b, double u, double w)
```

This subroutine is used to generate the isotherm according to the general cubic equation of state.

```
void CubicEOSDisplay(int eqn, double Pc, double Tc, double omega, double T, double a, double
b, EOSIsotherm Isotherm)
```

This subroutine is used to display the inputted parameters and generated isotherm on the user console.

`void CubicEOSWrite(int eqn, double Pc, double Tc, double omega, double T, double a, double b, EOSIsotherm Isotherm)`

This subroutine is used to write the inputted parameters and generated isotherm to a .txt file.

`void CubicEOSSwitch(int mode, int eqn, double Pc, double Tc, double omega, double T, double VdWa, double VdWb, double RKa, double RKb, double SRKa, double SRKb, double PRa, double PRb, EOSIsotherm VdWIsotherm, EOSIsotherm RKIsotherm, EOSIsotherm SRKIsotherm, EOSIsotherm PRIsotherm)`

This subroutine is used to ask whether the user would like to display or write the calculated subroutine results either to the user console or a .txt file.

`void CubicEOS(void)`

This subroutine is used to guide the user through the collecting the necessary data and generating the isotherms associated with the cubic equation of state.

### 2.2.3 Benedict-Webb-Rubin Equation of state

## 2.3 Viscosity

The theory for the meaning of Viscosity is described later in this documentation under section 4.2.2. To summarise the correlation made available in the student guide:

$$\mu = ae^{\frac{b}{T}} \tag{2.3.1}$$

$$\mu = \frac{aT^{1.5}}{(b+T)} \tag{2.3.2}$$

The kinematic viscosity is defined as:

$$\upsilon = \frac{\mu}{\rho} \tag{2.3.3}$$

### API

The implementation for all functions used to estimate the viscosity can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 2/ 02f ViscCorr.c`. The declaration of the following functions can be found in `02fViscCorr.h` at the same file path with the exception of '`ViscosityCorrelation(void)`' which can be found in `B48BB_T2.h`.

`void ViscCorrVariable(double *a, double *b, double *T, double *rho)`

This subroutine is used to collect the data required for standard fluid viscosity correlations.

```
double LiquidViscCalculation(double a, double b, double T)
```

This subroutine is used to calculate the liquid viscosity using equation 2.3.1.

```
double VapourViscCalculation(double a, double b, double T)
```

This subroutine is used to calculate the vapour viscosity using equation 2.3.2.

```
double KineticVisc(double mu, double rho)
```

This subroutine is used to calculate the kinematic viscosity using equation 2.3.3.

```
void ViscDisplay(int method, double a, double b, double T, double rho, double mu, double
upsi)
```

This subroutine is used to output the viscosity correlation results to the user console.

```
void ViscWrite(int method, double a, double b, double T, double rho, double mu, double
upsi)
```

This subroutine is used to write the viscosity correlation results to a .txt file.

```
void ViscWriteSwitch(int method, double a, double b, double T, double rho, double mu,
 double upsi)
```

This subroutine is used to ask the user if they would like to save the results of this program to a file.

```
void ViscosityCorrelation(void)
```

This subroutine guides the user through gathering the data and calculation of dynamic and kinematic viscosity.

# 3 Thermodynamics

## 3.1 Thermodynamic Cycles

The volume work is defined as the work done on or by a closed system as the boundary moves. This definition equates the process to the mechanical work required as the volume changes as some external force is applied through a distance $s$:

$$W_V = -\int_{s_1}^{s_2} F(s)ds \tag{3.1.1}$$

Applied to a piston, the above then becomes:

$$W_V = -\int_{z_1}^{z_2} F_{\text{ext}}(z)dz$$

Applying this equation to a quasi-static process, the external force is approximately the same as the internal force:

$$\because V = zA$$

$$dz = \frac{dV}{A}$$

$$W_V = -\int_{v_1}^{v_2} F_{\text{int}}(V)\frac{dV}{A}$$

$$\because P = \frac{F}{A}$$

$$W_V = -\int_{v_1}^{v_2} P(V)dV \tag{3.1.2}$$

An alternative definition can also be made to define the total work done on the system. Within standard texts, this is defined as the shaft work and is the sum of the volume work and flow work. The flow work is defined as the work required to push a fluid into/out of a system. Mathematically, the flow work is defined as:

$$W_f = V(P_2 - P_1) \tag{3.1.3}$$

To compare this against volume work, the shaft work is then the total work required to induce flow into and out of the unit operation and perform the thermodynamic process. Mathematically, the shaft work is defined as:

$$W_S = \int_{P_1}^{P_2} V dP \tag{3.1.4}$$

### 3.1.1 Thermodynamic Processes

**Polytropic Process**

For a polytropic process, the following state relation is followed by the system:

$$\frac{P_i V_i^{\alpha}}{T_i} = \frac{PV^{\alpha}}{T} = \text{ constant} \tag{3.1.5}$$

Where $\alpha$ is the polytropic index.

$$\because P = P_1 V_1^{\alpha} \frac{1}{V^{\alpha}}$$

$$\therefore W_V = -\int_{v_1}^{v_2} \frac{P_1 V_1^{\alpha}}{V^{\alpha}} dV = -P_1 V_1^{\alpha} \int_{v_1}^{v_2} \frac{1}{V^{\alpha}} dV$$

$$W_V = -P_1 V_1^{\alpha} [V^{-\alpha+1}]_{v_1}^{v_2} = -\frac{P_1 V_1^{\alpha}}{1-\alpha}(V_2^{1-\alpha} - V_1^{1-\alpha})$$

$$\because V_1^{\alpha} = V_1^{\alpha-1} V_1$$

$$W_V = -\frac{P_1 V_1}{1-\alpha}(V_2^{1-\alpha} V_1^{\alpha-1} - V_1^{1-\alpha} V_1^{\alpha-1})$$

$$W_V = -\frac{P_1 V_1}{1-\alpha}\left(\frac{V_2^{1-\alpha}}{V_1^{1-\alpha}} - 1\right) = \frac{P_1 V_1}{1-\alpha}\left(1 - \frac{V_1^{\alpha-1}}{V_2^{\alpha-1}}\right)$$

In terms of the pressure ratio, the polytrope is defined as:

$$\left(\frac{V_1}{V_2}\right)^{\alpha} = \frac{P_2}{P_1}$$

$$\therefore \left(\frac{V_1}{V_2}\right)^{\alpha-1} = \left(\frac{P_2}{P_1}\right)^{\frac{\alpha-1}{\alpha}}$$

$$W_V = \frac{P_1 V_1}{1-\alpha}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\alpha-1}{\alpha}}\right]$$

For a compression, the above expression will return a positive value which is inconsistent with sign convention:

$$W_V = -\frac{P_1 V_1}{\alpha-1}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\alpha-1}{\alpha}}\right] \tag{3.1.6}$$

For an ideal gas, the following temperature pressure relation must be true:

$$T_1 P_1^{\frac{1-\alpha}{\alpha}} = T_2 P_2^{\frac{1-\alpha}{\alpha}} = T P^{\frac{1-\alpha}{\alpha}} = \text{Constant}$$

$$\therefore W_V = -\frac{nRT_1}{\alpha-1}\left[\left(\frac{T_2}{T_1}\right) - 1\right] \tag{3.1.7}$$

The shaft work for an polytropic process can be found using equation 3.1.4:

$$W_S = \int_{P_1}^{P_2} V dP$$

An polytrope is defined by:

$$P_1 V_1^{\alpha} = P_2 V_2^{\alpha} = P V^{\alpha} = \text{constant}$$

$$V = \sqrt[\alpha]{\frac{P_1 V_1^{\alpha}}{P}}$$

The shaft work for a polytrope can now be derived as follows:

$$W_S = P_1^{\frac{1}{\alpha}} V_1 \int_{P_1}^{P_2} P^{-\frac{1}{\alpha}} dP$$

$$W_S = P_1^{\frac{1}{\alpha}} V_1 \left[ \frac{1}{-\frac{1}{\alpha}+1} P^{-\frac{1}{\alpha}+1} \right]_{P_1}^{P_2}$$

$$\because -\frac{1}{\alpha} + 1 = \frac{\alpha - 1}{\alpha}$$

$$W_S = P_1^{\frac{1}{\alpha}} V_1 \left[ \frac{\alpha}{\alpha - 1} P^{\frac{\alpha-1}{\alpha}} \right]_{P_1}^{P_2} = \frac{\alpha P_1^{\frac{1}{\alpha}} V_1}{\alpha - 1} \left[ P^{\frac{\alpha-1}{\alpha}} \right]_{P_1}^{P_2}$$

$$W_S = \frac{\alpha P_1^{\frac{1}{\alpha}} V_1}{\alpha - 1} \left[ P_2^{\frac{\alpha-1}{\alpha}} - P_1^{\frac{\alpha-1}{\alpha}} \right]$$

$$\because P_1 = P_1^{\frac{1}{\alpha}} P_1^{\frac{\alpha-1}{\alpha}}$$

$$\Rightarrow W_S = \frac{\alpha}{\alpha - 1} P_1 P_1^{\frac{1-\alpha}{\alpha}} V_1 \left[ P_2^{\frac{\alpha-1}{\alpha}} - P_1^{\frac{\alpha-1}{\alpha}} \right] = \frac{\alpha}{\alpha - 1} P_1 V_1 \left[ P_2^{\frac{\alpha-1}{\alpha}} P_1^{\frac{1-\alpha}{\alpha}} - P_1^{\frac{\alpha-1}{\alpha}} P_1^{\frac{1-\alpha}{\alpha}} \right]$$

$$W_S = \frac{\alpha}{\alpha - 1} P_1 V_1 \left[ \left( \frac{P_2}{P_1} \right)^{\frac{\alpha-1}{\alpha}} - 1 \right]$$

$$W_S = -\frac{\alpha}{\alpha - 1} P_1 V_1 \left[ 1 - \left( \frac{P_2}{P_1} \right)^{\frac{\alpha-1}{\alpha}} \right] \tag{3.1.8}$$

$$W_S = -\frac{\alpha}{\alpha - 1} n R T_1 \left[ 1 - \left( \frac{P_2}{P_1} \right)^{\frac{\alpha-1}{\alpha}} \right] \tag{3.1.9}$$

Therefore, the shaft work is related to the volume work with the equation:

$$W_S = \alpha W_V$$

### API (Polytropic volume work)

The implementation for all function used to calculate the polytropic volume work can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic-Processes)/ Topic 1/ 01aPolytropic.c`. The declarations of the following functions can be found in `01aPolytropic.h` at the same file path with the exception of '`Polytropic(void)`' which can be found in `B48BC_T1.h`.

```
void PolyVariable(int method, double *P1, double *P2, double *V1, double *T1,double *T2,
double *n, double *R, double *alpha)
```

This subroutine is used for collecting data for calculating the volume work for a polytropic process.

```
double PolyVolume(double P1, double P2, double V1, double alpha)
```

Subroutine used to calculate the volume work for a polytropic process utilising the system pressure and volume (equation 3.1.6).

```
double PolyTemperature(double n, double R, double T1, double T2, double alpha)
```

Subroutine used to calculate the volume work for a polytropic process utilising the changes to system temperature (equation 3.1.7).

```
double PolyFinalVolume(double P1, double P2, double V1, double alpha)
```

Subroutine used to determine the final volume from the pressure-volume relationship for a polytropic process.

```
double PolyFinalPressure(double T1, double T2, double P1, double alpha)
```

Subroutine used to determine the final pressure from the pressure-temperature relationship for a polytropic process.

```
double PolyFinalTemperature(double T1, double P1, double P2, double alpha)
```

Subroutine used to determine the final temperature from the pressure-temperature relationship for a polytropic process.

```
T1ThermoProf PolyProfile(int method, double P1, double P2, double V1, double T1, double
T2, double n, double R, double alpha)
```

This subroutine is used to determine the process profile given the input parameters.

```
void PolyProcDisp(double P1, double P2, double V1, double V2, double T1, double T2, double
n, double R, double alpha, T1ThermoProf profile)
```

This subroutine is used to output the collected data and the simulated process profile generated in "PolyProfile(...)" to the user console.

```
void PolyProcWrite(double P1, double P2, double V1, double V2, double T1, double T2, double
n, double R, double alpha, T1ThermoProf profile)
```

This subroutine is used to write the collected data and the simulated process profile generated in "PolyProfile(...)" to a .txt file.

```
void PolyProcSwitch(int mode, double P1, double P2, double V1, double V2, double T1, double
T2, double n, double R, double alpha, T1ThermoProf profile)
```

Subroutine to ask the user if they would like to either display the results on the console or save the results of this program to a file.

```
void Polytropic(void)
```

This subroutine is used to guide the user through collecting the data required to estimate the volume work and plot the thermodynamic profile for a polytropic process.

### API (Polytropic shaft work)

The implementation for all functions used to calculate the polytropic shaft work can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 2/ 02bPolyShaftWork.c`. The declarations of the following functions can be found in `02bPolyShaftWork.h` at the same file path with the exception of '`PolyShaftWork(void)`' which can be found in `B48BC_T2.h`.

```
int PolyShaftVariable(double *P1, double *P2, double *T1, double *n, double *R, double
*alpha)
```

This subroutine is used to collect the data required to calculate shaft work associated with a polytropic process.

```
double IdealShaftCalculation(double n, double R, double T1, double P1, double P2)
```

This subroutine is used to calculate the shaft work associated with an isothermal process (equation 3.1.11) This subroutine returns the shaft work in Watts (W).

```
double PolyShaftCalculation(double n, double R, double T1, double P1, double P2, double
alpha)
```

This subroutine is used to calculate the shaft work associated with a polytropic process. This subroutine returns the shaft work in Watts (W) (equation 3.1.9).

```
void PolyShaftDisplay(double n, double R, double T1, double P1, double P2, double alpha,
double W_S)
```

This subroutine is used to ouput the collected data and calculated shaft work to the user console.

```
void PolyShaftWrite(double n, double R, double T1, double P1, double P2, double alpha,
double W_S)
```

This subroutine is used to write the collected data and calculated shaft work into a .txt file.

```
void PolyShaftWriteSwitch(double n, double R, double T1, double P1, double P2, double alpha,
double W_S)
```

Subroutine to ask the user if they would like to save the results of this program to a file.

```
void PolyShaftWork(void)
```

This subroutine guides the user through the calculations to find the shaft work for an a polytropic process

### Isothermal Process

This is a special case of a polytropic process where $\alpha = 1$. However, doing such within the polytropic volume work equation will result in a non-defined value. This then alludes to needing to derive the isothermal volume work expression from equation 3.1.2.

$$W_V = -\int_{v_1}^{v_2} P(V)dV$$

$$\because P = \frac{nRT}{V}$$

$$\therefore W_V = -nRT \int_{v_1}^{v_2} \frac{1}{V} dV$$

$$W_V = -nRT \ln \frac{V_2}{V_1} \tag{3.1.10}$$

As Boyle's law states that the following property is conserved across different thermodynamic states:

$$P_1 V_1 = P_2 V_2 = PV = \text{constant}$$

$$\frac{P_1}{P_2} = \frac{V_2}{V_1}$$

$$W_V = -nRT \ln \frac{P_1}{P_2} = nRT \ln \frac{P_2}{P_1} \tag{3.1.11}$$

The shaft work for an isothermal process can be found using equation 3.1.4:

$$W_S = \int_{P_1}^{P_2} \frac{nRT}{P} dP$$

$$W_S = nRT \ln \frac{P_2}{P_1}$$

$$\therefore W_S = W_V$$

### API

The implementation for all functions used to calculate the isothermal volume work can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 1/ 01bIsothermal.c`. The declarations of the following functions can be found in `01bIsothermal.h` at the same file path with the exception of '`Isothermal(void)`' which can be found in `B48BC_T1.h`.

```
void IsotVariable(int method, double *P1, double *P2, double *V1, double *V2, double *T,
double *n)
```

This subroutine is used to collect the data to required to calculate the volume work for an isothermal process.

```
double IsotVolume(double n, double T, double V1, double V2)
```

This subroutine used to calculate the volume work associated with an isothermal process from a volume change. This subroutine then returns the volume work associated with the volume change (Equation 3.1.10).

```
double IsotPressure(double n, double T, double P1, double P2)
```

This subroutine used to calculate the volume work associated with an isothermal process from a pressure change. This subroutine then returns the volume work associated with the pressure change (Equation 3.1.11).

```
double IsotFinalPressure(double P1, double V1, double V2)
```

This subroutine is used to calculate the final pressure of a process undergoing an isothermal process.

```
double IsotFinalVolume(double V1, double P1, double P2)
```

This subroutine is used to calculate the final volume of a process undergoing an isothermal process.

```
T1ThermoProf IsotProfile(int method, double n, double T, double P1, double P2, double V1,
double V2)
```

This subroutine is used to estimate the process profile for an isothermal process.

```
void IsotProcDisplay(double P1, double P2, double V1, double V2, double T, double n
, T1ThermoProf profile)
```

This subroutine is used to output the collected data and generated profile to the user console.

```
void IsotProcWrite(double P1, double P2, double V1, double V2, double T, double n
, T1ThermoProf profile)
```

This subroutine is used to write the collected data and generated profile to a .txt file.

```
void IsotProcSwitch(int mode, double P1, double P2, double V1, double V2, double T, double
n, T1ThermoProf profile)
```

This subroutine is used to ask the user if they would like to either display the results on the console or save the results of this program to a file.

```
void Isothermal(void)
```

This subroutine is used to guide the user through collecting the data required to estimate the volume work and plot the thermodynamic profile for an Isothermal process.

### Isobaric Process

This is a special case of a polytropic process where $\alpha = 0$. This is another case where the volume work from the equivalent polytropic process will result in a undefined value. Starting once again from equation 3.1.2:

$$W_V = -\int_{V_1}^{V_2} P(V)dV$$

$$\because P = \text{constant}$$

$$W_V = P[V]_{V_1}^{V_2}$$

$$\therefore W_V = P(V_2 - V_1) \tag{3.1.12}$$

This can be converted to be in terms of temperature using the ideal gas law:

$$\because V = \frac{nRT}{P}$$

$$W_V = P\left(\frac{nRT_2}{P} - \frac{nRT_1}{P}\right)$$

$$W_V = nR(T_2 - T_1)$$

This is inconsistent with a compression process where the volume work should be negative for

a compression process. This results in the equation being multiplied by negative one:

$$W_V = -nR(T_2 - T_1) \tag{3.1.13}$$

### API

The implementation for all function used to calculate the isobaric volume work can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic- Processes)/ Topic 1/ 01cIsobaric.c`. The declarations of the following functions can be found in `01cIsobaric.h` at the same file path with the exception of '`Isobaric(void)`' which can be found in `B48BC_T1.h`.

```
void IsobVariable(int method, double *P, double *V1,double *V2, double *T1, double *T2,
double *n)
```

This subroutine is used for collecting data for calculating the volume work for a isobaric process.

```
double IsobVolume(double P, double V1, double V2)
```

This subroutine is used to calculate the volume work done associated with a change in volume for an isobaric process (equation 3.1.12).

```
double IsobTemperature(double n, double T1, double T2)
```

This subroutine is used to calculate the volume work done associated with a change in temperature for an isobaric process (equation 3.1.13).

```
double IsobFinalTemperature(double V1, double V2, double T1)
```

This subroutine calculates the final system temperature from the combined gas law modified for an isobaric process.

```
T1ThermoProf IsobProfile(int method, double P, double V1, double V2, double T1, double
T2, double n)
```

This subroutine is used to calculate the process profile associated with an isobaric process.

```
void IsobProcDisplay(double P, double V1, double V2, double T1, double T2, double n
, T1ThermoProf profile)
```

This subroutine is used to output the collected data and generated profile from "IsobProfile(...)" to the user console.

```
void IsobProcWrite(double P, double V1, double V2, double T1, double T2, double n
, T1ThermoProf profile)
```

This subroutine is used to write the collected data and generated profile from "IsobProfile(...)" to a .txt file.

```
void IsobProcSwitch(int mode, double P, double V1, double V2, double T1, double T2, double
n, T1ThermoProf profile)
```

This subroutine is used to ask the user if they would like to either display the results on the console or save the results of this program to a file.

```
void Isobaric(void)
```

This subroutine is used to guide the user through collecting the data required to estimate the volume work and plot the thermodynamic profile for an Isobaric process.

### Isochoric Process

This is a special case of a polytropic process where $\alpha \to \infty$. As such, this results in the volume work for this process being zero since there is no changes being made to the volume. To derive the heat required for this process, we can start from the first law:

$$Q + W = U + E_{\text{kin}} + E_{\text{pot}}$$

Assuming that there are negligible changes to the kinetic energy and height above the reference level.

$$dQ + dW = dU$$

$$dQ = dU$$

$$\because c_V = \left(\frac{dU}{dT}\right)_V$$

$$\therefore dQ = n\int_{T_1}^{T_2} c_V dT$$

For a quasi-static change, the heat capacity at constant volume can be treated as a constant:

$$\therefore dQ = nc_V(T_2 - T_1) \tag{3.1.14}$$

$$\because T_i = \frac{P_i v}{R}$$

$$Q = nc_V \frac{v}{R}(P_2 - P_1) \tag{3.1.15}$$

### API

The implementation for all functions used to calculate the heat associated with an isochoric process can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 1/ 01dIsochoric.c`. The declarations of the following functions can be found in `01dIsochoric.h` at the same file path with the exception of '`Isochoric(void)`' which can be found in `B48BC_T1.h`.

```
void IsocVariable(int method, double *P1, double *P2, double *V, double *T1, double *T2,
double *n, double *cv)
```

This subroutine is used for collecting data for calculating the heat for a isochoric process.

```
double IsocPressure(double P1, double P2, double V, double n, double cv)
```

This subroutine is used to calculate the heat associated with a pressure change for an isochoric process (equation 3.1.15).

`double IsocTemperature(double T1, double T2, double n, double cv)`

This subroutine is used to calculate the heat associated with a temperature change for an isochoric process (equation 3.1.14).

`double IsocFinalTemperature(double T1, double P1, double P2)`

This subroutine is used to calculate the final temperature for an isochoric process associated with a given pressure change and initial temperature..

`T1ThermoProf IsocProfile(int method, double P1, double P2, double V, double T1, double T2, double n, double cv)`

This subroutine is used for calculating the process profile for an isochoric process. This subroutine requires "IdealTemperature(...)" from "IdealGasLaw.h" to function as intended.

`void IsocProcDisplay(double P1, double P2, double V, double T1, double T2, double n, double c_v, T1ThermoProf profile)`

This subroutine is used to output the collected data and generated profile from "IsocProfile(...)" to the user console.

`void IsocProcWrite(double P1, double P2, double V, double T1, double T2, double n, double c_v, T1ThermoProf profile)`

This subroutine is used to output the collected data and generated profile from "IsocProfile(...)" to a .txt file.

`void IsocProcSwitch(int mode, double P1, double P2, double V, double T1, double T2, double n, double c_v, T1ThermoProf profile)`

This subroutine is used to ask the user if they would like to either display the results on the console or save the results of this program to a file.

`void Isochoric(void)`

This subroutine is used to guide the user through collecting the data required to estimate the volume work and plot the thermodynamic profile for an Isochoric process.

### Adiabatic Process

This is a special case of a polytropic process where $\alpha = \gamma$. To be able to prove the equivalence of the two, we can prove the relationship between state variables before deriving the volume work expression from the first law assuming negligible changes to kinetic energy and potential energy:

$$dQ + dW = dU$$

An adiabatic process, by definition, exchanges no heat with its surroundings, therefore $dQ = 0$:

$$dW_V = dU$$

$$-PdV = n\int_{T_1}^{T_2} c_V dT$$

$$-\frac{nRT}{V}dV = nc_V dT$$

$$-\frac{R}{c_V}\frac{1}{V}dV = \frac{1}{T}dT$$

$$-\frac{R}{c_V}\ln\frac{V_2}{V_1}=\ln\frac{T_2}{T_1}$$

$$\left(\frac{V_2}{V_1}\right)^{-\frac{R}{c_v}}=\left(\frac{T_2}{T_1}\right)$$

For an ideal gas:

$$R=c_P-c_V$$

$$\therefore\left(\frac{V_2}{V_1}\right)^{\frac{c_v-c_p}{c_v}}=\left(\frac{T_2}{T_1}\right)$$

$$\left(\frac{V_2}{V_1}\right)^{1-\gamma}=\left(\frac{T_2}{T_1}\right)$$

$$\left(\frac{V_1}{V_2}\right)^{\gamma-1}=\left(\frac{T_2}{T_1}\right)$$

The temperature-volume relationship is then:

$$\boxed{T_1V_1^{\gamma-1}=T_2V_2^{\gamma-1}=TV^{\gamma-1}=\text{constant}}$$

Applying the combined gas law allows for the pressure-volume relation to be derived:

$$T=PV$$

$$\boxed{P_1V_1^{\gamma}=P_2V_2^{\gamma}=PV^{\gamma}=\text{constant}}$$

This is equivalent to:

$$P_1^{1/\gamma}V_1=P_2^{1/\gamma}V_2=P^{1/\gamma}V=\text{constant}$$

$$\because V=\frac{T}{P}$$

$$P_1^{1/\gamma}\frac{T_1}{P_1}=P_2^{1/\gamma}\frac{T_2}{P_2}=P^{1/\gamma}\frac{T}{P}=\text{constant}$$

$$\boxed{P_1^{\frac{1-\gamma}{\gamma}}T_1=P_2^{\frac{1-\gamma}{\gamma}}T_2=P^{\frac{1-\gamma}{\gamma}}T=\text{constant}}$$

To begin the derivation for the volume work, we can once again start from equation 3.1.2:

$$W_V=-\int_{v_1}^{v_2}P(V)dV$$

Using the pressure volume relationship shown above:

$$P=\frac{P_1V_1^{\gamma}}{V^{\gamma}}$$

$$W_V=-\int_{v_1}^{v_2}\frac{P_1V_1^{\gamma}}{V^{\gamma}}dV=-P_1V_1^{\gamma}\int_{v_1}^{v_2}\frac{1}{V^{\gamma}}dV$$

$$W_V=-P_1V_1^{\gamma}[V^{-\gamma+1}]_{v_1}^{v_2}=-\frac{P_1V_1^{\gamma}}{1-\gamma}(V_2^{1-\gamma}-V_1^{1-\gamma})$$

$$\because V_1^{\gamma} = V_1^{\gamma-1} V_1$$

$$W_V = -\frac{P_1 V_1}{1-\gamma}(V_2^{1-\gamma} V_1^{\gamma-1} - V_1^{1-\gamma} V_1^{\gamma-1})$$

$$W_V = -\frac{P_1 V_1}{1-\gamma}\left(\frac{V_2^{1-\gamma}}{V_1^{1-\gamma}} - 1\right) = \frac{P_1 V_1}{1-\gamma}\left(1 - \frac{V_1^{\gamma-1}}{V_2^{\gamma-1}}\right)$$

In terms of the pressure ratio, an adiabat has been defined as:

$$P_1 V_1^{\gamma} = P_2 V_2^{\gamma} = PV^{\gamma} = \text{constant}$$

$$\left(\frac{V_1}{V_2}\right)^{\gamma} = \frac{P_2}{P_1}$$

$$\therefore \left(\frac{V_1}{V_2}\right)^{\gamma-1} = \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}$$

$$W_V = \frac{P_1 V_1}{1-\gamma}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

For a compression, the above expression will return a positive value which is inconsistent with sign convention:

$$W_V = -\frac{P_1 V_1}{\gamma-1}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right] \tag{3.1.16}$$

$$\because PV = nRT$$

$$W_V = -\frac{nRT_1}{\gamma-1}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right] \tag{3.1.17}$$

Alternatively, the adiabatic volume work can be written in terms of temperature using:

$$P_1^{\frac{1-\gamma}{\gamma}} T_1 = P_2^{\frac{1-\gamma}{\gamma}} T_2 = P^{\frac{1-\gamma}{\gamma}} T = \text{constant}$$

$$\left(\frac{P_1}{P_2}\right)^{\frac{1-\gamma}{\gamma}} = \frac{T_2}{T_1} \Rightarrow \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}} = \frac{T_2}{T_1}$$

$$W_V = -\frac{nRT_1}{\gamma-1}\left(1 - \frac{T_2}{T_1}\right)$$

$$W_V = \frac{nRT_1}{\gamma-1}\left(\frac{T_2}{T_1} - 1\right)$$

$$W_V = \frac{nR}{\gamma-1}(T_2 - T_1)$$

$$\because \gamma = \frac{c_P}{c_V}$$

$$W_V = \frac{c_V nR}{c_P - c_V}(T_2 - T_1)$$

For an ideal gas:

$$c_P = c_V + R$$
$$W_V = \frac{c_V nR}{R}(T_2 - T_1)$$
$$W_V = nc_V \Delta T \tag{3.1.18}$$

The shaft work for an adiabatic process can be found using equation 3.1.4:

$$W_S = \int_{P_1}^{P_2} V dP$$

An adiabat is defined by:

$$PV^\gamma = P_1 V_1^\gamma = P_2 V_2^\gamma = \text{constant}$$
$$V = \sqrt[\gamma]{\frac{P_1 V_1^\gamma}{P}}$$

The shaft work for an adiabat can now be derived as follows:

$$W_S = P_1^{\frac{1}{\gamma}} V_1 \int_{P_1}^{P_2} P^{-\frac{1}{\gamma}} dP$$

$$W_S = P_1^{\frac{1}{\gamma}} V_1 \left[ \frac{1}{-\frac{1}{\gamma}+1} P^{-\frac{1}{\gamma}+1} \right]_{P_1}^{P_2}$$

$$\because -\frac{1}{\gamma} + 1 = \frac{\gamma - 1}{\gamma}$$

$$W_S = P_1^{\frac{1}{\gamma}} V_1 \left[ \frac{\gamma}{\gamma - 1} P^{\frac{\gamma-1}{\gamma}} \right]_{P_1}^{P_2} = \frac{\gamma P_1^{\frac{1}{\gamma}} V_1}{\gamma - 1} \left[ P^{\frac{\gamma-1}{\gamma}} \right]_{P_1}^{P_2}$$

$$W_S = \frac{\gamma P_1^{\frac{1}{\gamma}} V_1}{\gamma - 1} \left[ P_2^{\frac{\gamma-1}{\gamma}} - P_1^{\frac{\gamma-1}{\gamma}} \right]$$

$$\because P_1 = P_1^{\frac{1}{\gamma}} P_1^{\frac{\gamma-1}{\gamma}}$$

$$\Rightarrow W_S = \frac{\gamma}{\gamma - 1} P_1 P_1^{\frac{1-\gamma}{\gamma}} V_1 \left[ P_2^{\frac{\gamma-1}{\gamma}} - P_1^{\frac{\gamma-1}{\gamma}} \right] = \frac{\gamma}{\gamma - 1} P_1 V_1 \left[ P_2^{\frac{\gamma-1}{\gamma}} P_1^{\frac{1-\gamma}{\gamma}} - P_1^{\frac{\gamma-1}{\gamma}} P_1^{\frac{1-\gamma}{\gamma}} \right]$$

$$W_S = \frac{\gamma}{\gamma - 1} P_1 V_1 \left[ \left( \frac{P_2}{P_1} \right)^{\frac{\gamma-1}{\gamma}} - 1 \right]$$

$$W_S = -\frac{\gamma}{\gamma - 1} P_1 V_1 \left[ 1 - \left( \frac{P_2}{P_1} \right)^{\frac{\gamma-1}{\gamma}} \right] \tag{3.1.19}$$

$$W_S = -\frac{\gamma}{\gamma - 1} nRT_1 \left[ 1 - \left( \frac{P_2}{P_1} \right)^{\frac{\gamma-1}{\gamma}} \right] \tag{3.1.20}$$

Therefore, the shaft work is related to the volume work with the equation:

$$W_S = \gamma W_V$$

In addition, one will also find that the above derivation is nearly identical to the equations derived for a polytropic process. Since $\gamma$ is a temperature specific constant, the net fluid work for this process will be non-zero:

$$W_f = -P_1V_1 + P_2V_2$$

From the adiabat, we can observe that the final volume can be calculated with:

$$V_2 = V_1\left(\frac{P_2}{P_1}\right)^{-\frac{1}{\gamma}}$$

$$W_f = -P_1V_1 + P_2V_1\left(\frac{P_2}{P_1}\right)^{-\frac{1}{\gamma}} = -P_1V_1\left[1 - \left(\frac{P_2}{P_1}\right)^{-\frac{1}{\gamma}+1}\right]$$

$$\therefore W_f = -P_1V_1\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

$$\because W_S = W_{\sum f} + W_V$$

$$W_S = -P_1V_1\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right] + -\frac{P_1V_1}{\gamma-1}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

$$W_S = -\frac{\gamma-1}{\gamma-1}P_1V_1\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right] + -\frac{P_1V_1}{\gamma-1}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

$$W_S = -\frac{1+(\gamma-1)}{\gamma-1}P_1V_1\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right] = -\frac{\gamma}{\gamma-1}P_1V_1\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right] \blacksquare$$

### API

The implementation for all functions used to calculate the work associated with an adiabatic process can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 1/ 01eAdiabatic.c`. The declarations of the following functions can be found in `01eAdiabatic.h` at the same file path with the exception of '`Adiabatic(void)`' which can be found in `B48BC_T1.h`.

```
void AdiaVariable(int method, double *P1, double *P2, double *V1, double *V2, double *T1,
double *n, double *gamma)
```

This subroutine is used for collecting data for calculating the volume work for a adiabatic process.

```
double AdiaVolume(double P1, double V1, double V2, double gamma)
```

This subroutine is used to calculate the adiabatic volume work from the system pressure and volume (equation 3.1.16).

```
double AdiaTemperature(double n, double T1, double P1, double P2, double gamma)
```

This subroutine is used to calculate the adiabatic volume work from the system pressure and temperature (equation 3.1.17).

```
double AdiaFinalPress(double P1, double V1, double V2, double gamma)
```

This subroutine is used to calculate the final pressure using the Pressure-Volume relationship for an adiabat.

```
double AdiaFinalTemp(double T1, double P1, double P2, double gamma)
```

This subroutine is used to calculate the final temperature using the Pressure-Temperature relationship for an adiabat.

```
double AdiaFinalVol(double V1, double P1, double P2, double gamma)
```

This subroutine is used to calculate the final volume using the Pressure-Temperature relationship for an adiabat.

```
T1ThermoProf AdiaProfile(int method, double P1, double P2, double V1, double V2, double
T1, double n, double gamma)
```

This subroutine is used to determine the process profile given the input parameters.

```
void AdiaProcDisplay(double P1, double P2, double V1, double V2, double T1, double T2,
double n, double gamma, T1ThermoProf profile)
```

Subroutine used to output the collected data and generated dataset from AdiaProfile(...) to the user console.

```
void AdiaProcWrite(double P1, double P2, double V1, double V2, double T1, double T2,
 double n, double gamma, T1ThermoProf profile)
```

Subroutine used to write the collected data and generated dataset from AdiaProfile(...) to a .txt file.

```
void AdiaProcSwitch(int mode, double P1, double P2, double V1, double V2, double T1,
 double T2, double n, double gamma, T1ThermoProf profile)
```

Subroutine to ask the user if they would like to either display the results on the console or save the results of this program to a file.

```
void Adiabatic(void)
```

This subroutine is used to guide the user through collecting the data required to estimate the volume work and plot the thermodynamic profile for an Adiabatic process.

### Reciprocating compressor

This unit operation utilises the above thermodynamic processes to compress a gas to a desired pressure through an isothermal, adiabatic or polytropic compression step. To model the intake and discharge steps, this can be done as an isobaric expansion/compression. In practice, this type of compressor can be found when the pressure ratio of the unit is less than 6. Where the pressure ratio is greater than 6 or where there can be no pulsations in the flow, a rotary screw compressor should be used.

As stated in the previous paragraph, this compressor can be modelled with an isobaric expansion, followed by an isothermal/adiabatic/polytropic compression followed by an isobaric compression. For the purposes of this documentation, the polytropic and isothermal compression cases are considered.

[Insert graph]

1. Reversible constant pressure intake.

   The expression for isobaric volume work has been derived previously in section 3.1.1. Thus for this stage, the volume work is:

   $$W_V = -P_1(V_b - V_a)$$

   Where $V_a$ is the clearance volume and $V_b$ is the volume inside the compressor immediately prior to compression. A clearance volume is required within a compressor to avoid the piston hitting the intake/exhaust valves during the discharge step. In other words, this isobaric process is the volume work required to push fluid into the compressor.

2. Isothermal/Polytropic compression.

   The expression for these types of volume work has been derived previously. For this stage, the volume work is then:

   - $W_V = nRT \ln \frac{P_2}{P_1}$
   - $W_V = -\frac{P_1 V_1}{\alpha - 1}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\alpha-1}{\alpha}}\right]$

   In practice, an isothermal step is an impossibility due to insufficient cooling to allow the process to be considered isothermal. Additionally, an adiabatic compression is often unachievable given real gas behaviour. For these reasons, many reciprocating compressors operate polytropically since there will be some degree of cooling present.

3. Reversible constant pressure exhaust.

   This process can be described mathematically as before with an isobaric process with the following form:

   $$W_V = -P_1(V_d - V_c)$$

   Where $V_c$ is the compressor volume immediately after compression and $V_d$ is the clearance volume inside the compressor. In other words, this is the work needed to push the fluid out of the compressor.

The total volume work for this compressor can now be evaluated as:

$$W_V = -\oint_a^d P dV = W_V^{ab} + W_V^{bc} + W_V^{cd}$$

$$\therefore W_V = P_1 V_a - P_1 V_b + W_V^{bc} + P_2 V_c - P_2 V_d$$

### API

The implementation for all functions used to generate the process profile for a reciprocating compressor can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 2/ 02aCompressor.c`. The declarations of the following functions can be found in `02aCompressor.h` at the same file path with the exception of '`Compressor(void)`' which can be found in `B48BC_T2.h`.

```
void CompressorVariable(int method, double *P1, double *P2, double *Vc, double *V1, double *T1, double *T2, double *n, double *R, double *alpha)
```

This subroutine is used to collect the data required to perform the calculations for a reciprocating compressor undergoing an isothermal or polytropic process.

```
T2CompProfile CompressorProfile(int method, double P1, double P2, double Vc, double *V1, double *V2, double T1, double n, double R, double alpha)
```

This subroutine is used to generate the profile for one sweep of the compressor. For this subroutine to operate, it requires functions contained within "IdealGasLaw.h", "Isobaric.h", "Isothermal.h" and "Polytropic.h" (Equations 3.1.11, 3.1.6 and 3.1.9) to function. Behaviour is dependent on the value of 'int method'.

```
void CompresDisplay(double P1, double P2, double Vc, double V1, double V2, double T1, double T2, double n, double R, double alpha, T2CompProfile profile)
```

This subroutine is used to output the collected data and generated process simulation to the user console.

```
void CompresWrite(double P1, double P2, double Vc, double V1, double V2, double T1, double T2, double n, double R, double alpha, T2CompProfile profile)
```

This subroutine is used to write the collected data and generated process simulation to a .txt file.

```
void CompresSwitch(int mode, double P1, double P2, double Vc, double V1, double V2, double T1, double T2, double n, double R, double alpha, T2CompProfile profile)
```

Subroutine to ask the user if they would like to either display the results on the console or save the results of this program to a file.

```
void Compressor(void)
```

This subroutine is used to guide the user through the calculations to design a reciprocating processor using a polytropic or isothermal process.

### Multistage arrangements

As stated in the previous section, a single-stage reciprocating compressor is not appropriate for use with pressure ratios greater than 6. Hypothetically speaking, lets suppose that a pressure ratio greater than 6 is required. How can we justify the cost of an equivalent rotary screw compressor if a reciprocating compressor is unsuitable? The solution is to use multiple reciprocating compressors. The question of whether to use the rotating screw compressor or the reciprocating compressor is now a question of which solution is more economical since it is trivial to note that the capital cost of the section will increase in line with the number of compression stages. This also raises the question of how one can minimise the total required shaft work whilst delivering the specified pressure ratio.

To begin deriving the expression that covers the multiple compression stages, that are currently purely hypothetical, we can start from the definition of shaft work (Equation 3.1.4). Empirical evidence also suggests that complete intercooling between compression stages can reduce capital costs due to the compression occurring at a lower temperature. This allows for the longevity of the compressor to be considered by implicitly factoring in the increased costs associated with operating at a higher temperature. In addition, intercooling will allow the over-

arching process to assume quasi-isothermal behaviour which will return the minimal amount of shaft work by definition. Intercooling is the heat exchange operation that will cool the outlet from one compression stage back to the inlet temperature ready for the next compression stage. Since another problem of using a reciprocating compressor is flow pulsations, a receiver can be placed at the section outlet to dampen pulsations.

Practically speaking, such an arrangement will allow us to approximate the equivalent isothermal compression through carrying out a series of adiabatic compressions. Practically, this process will typically behave polytropically operating between the isothermal and adiabatic shaft work. There are further advantages to be found through the incorporation of intercooling as the thermal-decomposition temperature and flammable gases can now be accounted for by maintaining the system temperatures below the thermal decomposition temperature or the auto-ignition temperature. They do however have the disadvantage when accounting for clearance volume meaning that some mass will inherently remain within the compressor. This can be quantified into a ratio by using the volumetric efficiency. This is defined as:

$$\eta_{\text{vol.}} = \frac{V_{\text{induced}}}{V_{\text{system}}} \tag{3.1.21}$$

Causes of volumetric inefficiencies mainly occur due to the inlet valve remaining shut during an intake stroke. Furthermore, high discharge pressures may result in the inlet valve remaining shut during operation due to the pressure inside the compressor being too high to allow the inlet valve to open. This is in fact another reason for why multi-stage arrangements are adopted throughout industry since an optimised multi-stage compression should maximise the total gas throughput whilst maintaining an acceptable volumetric efficiency and minimal shaft work.

To begin deriving the equations to mathematically describe this behaviour, we will start from looking at the shaft work for a two stage compression. We will then expand on this derivation to find the general case that would apply to multistage compression.

It has been derived earlier (Equation 3.1.20) that the shaft work for an adiabatic process is:

$$W_S = -\frac{\gamma \dot{n} R T_1}{\gamma - 1}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

Thus for a two-stage compression between $P_1$ and $P_3$ involving an ideal gas, the shaft work can be calculated as:

$$W_{S1} = -\frac{\gamma \dot{n} R T_1}{\gamma - 1}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right]$$
$$W_{S2} = -\frac{\gamma \dot{n} R T_2}{\gamma - 1}\left[1 - \left(\frac{P_3}{P_2}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

Thus the total shaft work for this two-stage compression will be:

$$W_{\sum S} = -\frac{\gamma \dot{n} R T_1}{\gamma - 1}\left[1 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}\right] + -\frac{\gamma \dot{n} R T_2}{\gamma - 1}\left[1 - \left(\frac{P_3}{P_2}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

Due to the temperature dependence of the heat capacity ratio and how the system temperature will change through the compression, a more accurate estimation of the shaft work can be made assuming a quasi-static change and taking the average ratio between the two system pressures. As stated before, the minimal work scenario for this compression will be the isothermal case which can be approximated by allowing for complete intercooling between compression stages. Mathematically, this can be described for this process as:

$$T_1 = T_2 = T$$

$$W_{\sum S} = -\frac{\gamma \dot{n} R T}{\gamma - 1}\left[2 - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}} - \left(\frac{P_3}{P_2}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

Although the design engineer is free to choose the value of $P_2$, it would be sensible to find the value that minimises the overall shaft work. The above equation then implies that the minimal shaft work must occur at the inflection point where the following must be true:

$$\frac{\delta W_S(P_2)}{\delta P_2} = 0$$

$$\frac{dW_S}{dP_2} = (0) - \frac{\gamma \dot{n} R T_1}{\gamma - 1}\frac{\gamma - 1}{\gamma}\left(\frac{P_2}{P_1}\right)^{-\frac{1}{\gamma}}\left(\frac{1}{P_1}\right) + \frac{\gamma \dot{n} R T_1}{\gamma - 1}\frac{\gamma - 1}{\gamma}\left(\frac{P_3}{P_2}\right)^{-\frac{1}{\gamma}}\left(\frac{P_3}{P_2^2}\right) = 0$$

$$\Rightarrow \left(\frac{P_3}{P_2}\right)^{-\frac{1}{\gamma}}\left(\frac{P_3}{P_2^2}\right) = \left(\frac{P_2}{P_1}\right)^{-\frac{1}{\gamma}}\left(\frac{1}{P_1}\right)$$

$$\left(\frac{P_3}{P_2}\right)^{-\frac{1}{\gamma}}\left(\frac{P_3}{P_2}\right) = \left(\frac{P_2}{P_1}\right)^{-\frac{1}{\gamma}}\left(\frac{P_2}{P_1}\right) \Rightarrow \left(\frac{P_3}{P_2}\right)^{\frac{\gamma-1}{\gamma}} = \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}}$$

$$\therefore \left(\frac{P_3}{P_2}\right) = \left(\frac{P_2}{P_1}\right)$$

This means that the the shaft work is minimal/is optimal when the pressure ratio across the two stages is the same.

Although not shown within this documentation, we can expand the above derivation to N-stages. This finds that the total shaft work for a multi-stage compression is:

$$W_{\sum S} = -\frac{\gamma \dot{n} R T}{\gamma - 1}\left[N - \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma}} - \left(\frac{P_3}{P_2}\right)^{\frac{\gamma-1}{\gamma}} + ...\right]$$

Differentiating to find the inflection point for intermediary pressures, shows that a minimal amount of shaft work is obtained when the pressure ratios across all stages are the same. Thus for an individual stage, the following statement can be made:

$$r_{P,i} = r_P^{1/N}$$

Where $r_{P,i}$ is the pressure ratio for a single stage within a multistage compression and $r_P$ is the pressure ratio for the entire multi-stage compression. Substituting this into the total shaft

work for a multistage compression finds that:

$$W_{\sum S} = -\frac{\gamma \dot{n} R T}{\gamma - 1}\left[N - N(r_P)^{\frac{1}{N}\frac{\gamma-1}{\gamma}}\right]$$

Therefore, for a N-stage compression process, the total shaft work across all stages can be found with the following expression:

$$W_{\sum S} = -\frac{\gamma \dot{n} R T N}{\gamma - 1}\left[1 - \left(\frac{P_{\text{out}}}{P_{\text{in}}}\right)^{\frac{1}{N}\frac{\gamma-1}{\gamma}}\right] \tag{3.1.22}$$

### API (Multi-stage Compression)

The implementation for all functions used to generate the process profile for a multistage reciprocating compressor can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 3/ 03bMultistageCompressor.c`. The declarations of the following functions can be found in `03bMultistageCompressor.h` at the same file path with the exception of '`MultistageCompressor(void)`' which can be found in `B48BC_T3.h`.

```
void MSCompVariable(double *P1, double *P2, double *Vc, double *T1, double *n, int *N,
double *gamma)
```

This subroutine is used to collect the required data to calculate a multistage gas compression profile accounting for clearance volume.

```
double calculatePressureRatio(double P1, double P2)
```

This subroutine is used to calculate the pressure ratio between the starting pressure and desired final pressure.

```
int stageIntake(int elements, int stage, int stages)
```

This subroutine is used to calculate the location of the start of the compressor intake stroke within the declared struct at a specified stage.

```
int stageProcess(int elements, int stage, int stages)
```

This subroutine is used to calculate the location of the start of the adiabatic portion of the compression process within the declared struct at a specified stage.

```
int ProcessStages(int elements, int stages)
```

This subroutine is used to calculate the number of rows required for the adiabatic portion of the compression process.

```
int stageDischarge(int elements, int stage, int stages)
```

This subroutine is used to calculate the location of the start of the discharge stroke of the compression process as the gas is expelled from one compressor at a specified stage.

```
T3CompProfile MSCompProfile(double P1, double P2, double Vc, double T1, double n, int N,
double gamma)
```

This subroutine is used to calculate the profile for a multistage gas compression process using equations 3.1.12, 3.1.17 and 3.1.9.

```
void MSCompDisplay(double P1, double P2, double Vc, double V1, double V2, double T1, double
T2, double n, double N, double gamma, T3CompProfile profile)
```

This subroutine is used to display the generated data table for a multistage gas compression process.

```
void MSCompWrite(double P1, double P2, double Vc, double V1, double V2, double T1, double T2, double n, double N, double gamma, T3CompProfile profile)
```

This subroutine is used to write the generated data to a .txt file.

```
void MSCompSwitch(int mode, double P1, double P2, double Vc, double V1, double V2, double T1, double T2, double n, double N, double gamma, T3CompProfile profile)
```

This subroutine is used to ask the user whether they would like to either display the results on the console or save the results of this program to a file.

```
void MultistageCompressor(void)
```

This subroutine is used to calculate the work input for a multistage gas compressor using the pressure-temperature statement of an adiabatic process assuming complete intercooling between stages.

### API (Multi-stage Shaft Work Estimation)

The implementation for all functions used to estimate the shaft work for a multistage reciprocating compressor can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 3/ 03cMultistageEstimation.c`. The declarations of the following functions can be found in `03cMultistageEstimation.h` at the same file path with the exception of '`MultistageShaftWork(void)`' which can be found in `B48BC_T3.h`.

```
void MSShaftWorkVariable(double *P1, double *P2, double *T1, double *mol, int *N, double *gamma)
```

This subroutine is used to collect the required data to estimate the total shaft work required by a multistage gas compression.

```
double MSShaftWorkEquation(double P1, double P2, double T1, double mol, double gamma, double N)
```

This subroutine is used to calculate the shaft work contribution attributed to a pressure ratio assuming complete intercooling between stages using equation 3.1.22.

```
double MSShaftWorkCalculation(double P1, double P2, double T1, double mol, double gamma, double N)
```

This subroutine is used to split the given pressure ratio into smaller segments to improve the accuracy of equation 3.1.22.

```
void MSShaftWorkDisplay(double P1, double P2, double T1, double mol, double gamma, double N, double shaftwork)
```

This subroutine is used to display the results of this program on the user console.

```
void MSShaftWorkWrite(double P1, double P2, double T1, double mol, double gamma, double N, double shaftwork)
```

This subroutine is used to write this subroutine's collected data and calculated results to a .txt file.

```
void MSShaftWorkWriteSwitch(double P1, double P2, double T1, double mol, double gamma, double N, double shaftwork)
```

This subroutine is used to check whether the user would like to write the results of this subroutine to a .txt file or not.

```
void MultistageShaftWork(void)
```

This subroutine is used to estimate the total shaft work for a multistage gas compressor using the pressure-temperature statement of an adiabatic process assuming complete intercooling between stages.

### 3.1.2 Thermodynamic Cycles

A thermodynamic cycle is defined as a sequence of processes in which the working fluid undergoes a series of thermodynamic changes but must return to its original state after the sequence finishes prior to it restarting. This definition implies that the working fluid must exchange heat and work with its surroundings. Thus it allows heat to be converted to work, and vice versa, without becoming a perpetual motion machine of the second order.

#### The Carnot Cycle

A Carnot cycle is defined as the following sequence of processes which produces no net work or net heat. This can be achieved with the following sequence of processes [4]:

1. The system at the cold reservoir temperature undergoes a reversible adiabatic compression from $T_C$ to $T_H$. This is equivalent to pumping the working fluid through $P_1 \rightarrow P_2$
    - This only has reversible work since the energy content remains constant.
2. The system then undergoes an isothermal expansion. This is equivalent to boiling the working fluid resulting in the pressure drop $P_2 \rightarrow P_3$.
    - This only has reversible heat for $Q_H$ attributed to keeping the system temperature constant.
3. The system at the hot reservoir temperature undergoes a reversible adiabatic expansion from $T_H$ to $T_C$. This is equivalent to sending the working fluid through a turbine resulting in the pressure drop $P_3 \rightarrow P_4$.
    - This only has reversible work since the energy content remains constant.
4. The system at the cold reservoir temperature undergoes a reversible isothermal compression from $T_C$ to $T_H$. This is equivalent to condensing the working fluid from $P_4 \rightarrow P_1$ (a pressure increase).
    - This only has reversible heat for $Q_C$ attributed to keeping the system temperature constant.

Supposing that we did want to use this cycle to produce energy, we would require $|w_{12}| < |w_{34}|$. That is, the system would need to lose more work to the turbine than the amount of work required by the pump. This cycle would then also imply that the heat provided to the working fluid in the boiler generates work in the turbine. This would mean that this cycle becomes a

heat engine since heat is being converted into work. The proof for this can be found by using the first law applied to a closed system, which is just a special case of equation 3.2.2:

$$0 = (Q + W_s) - (\Delta H + \Delta E_{\text{kin}} + \Delta E_{\text{pot}})$$

For a thermodynamic cycle, there is, by definition, no net change to the state variables. Hence:

$$w_{\text{net}} = -q_{\text{net}}$$

The Kelvin-Planck machine will be discussed later within this documentation, but the matter of being able to turn heat into useful work appears to apparently violate this statement. However, we can do this process in reality through the Rankine cycle (assuming that the pressure that the pump raises the working fluid to is the saturation pressure at that temperature) which begs the question of how this cycle, in reality, does not become a perpetual motion machine of the second order. The answer to such a question is that the cycle operates between a hot and cold reservoir to produce useful work.

However, as practically useless the Carnot cycle is in reality, it does provide a useful insight in the terms of maximum thermodynamic efficiency. This will be later defined as the ratio between the amount of useful work that the cycle can produce to the amount of heat entering the cycle. To begin with quantifying these variables, we can define the net work and net heat as follows:

The net work can be found, assuming that there is no net fluid work, in the cycle. It has been found earlier that this now equates the volume work to the shaft work.

$$w_{\text{net}} = w_{12} + w_{23} + w_{34} + w_{41}$$

Similarly, the net heat can be found using:

$$q_{\text{net}} = q_{12} + q_{23} + q_{34} + q_{41}$$

Assuming that all sources of energy are now known, this proves the first law description of this cycle:

$$w_{\text{net}} = -q_{\text{net}} \blacksquare$$

To mathematically describe this cycle, we can use the earlier derived thermodynamic processes to begin quantifying this cycle:

- Processes $1 \to 2$ and $3 \to 4$ are isothermal:

  This means that the work required for the two processes are found using equation 3.1.11:

$$w_{12} = RT_H \ln \frac{P_2}{P_1} = -q_H$$

$$w_{34} = RT_C \ln \frac{P_4}{P_3} = -q_C$$

- Processes $2 \to 3$ and $4 \to 1$ are adiabatic:

This means that the work required for the two processes are found using equation 3.1.20:

$$w_{23} = -\frac{\gamma R T_2}{\gamma - 1}\left[1 - \left(\frac{P_3}{P_2}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

$$w_{41} = -\frac{\gamma R T_4}{\gamma - 1}\left[1 - \left(\frac{P_1}{P_4}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

By definition, there is no heat inputted/removed from these processes.

The first law statement of the Carnot cycle can now be described using the following equation:

$$w_{\text{net}} = -q_{\text{net}}$$

$$RT_H \ln\frac{P_2}{P_1} - \frac{\gamma R T_2}{\gamma - 1}\left[1 - \left(\frac{P_3}{P_2}\right)^{\frac{\gamma-1}{\gamma}}\right] + RT_C \ln\frac{P_4}{P_3} - \frac{\gamma R T_4}{\gamma - 1}\left[1 - \left(\frac{P_1}{P_4}\right)^{\frac{\gamma-1}{\gamma}}\right] = -(q_{12} + q_{34})$$

For cyclic behaviour to be found, both adiabatic processes must therefore be equal and opposite. To begin to find when this would be true, we start by equating the two terms:

$$\frac{\gamma R T_H}{\gamma - 1}\left[1 - \left(\frac{P_3}{P_2}\right)^{\frac{\gamma-1}{\gamma}}\right] = \frac{\gamma R T_C}{\gamma - 1}\left[1 - \left(\frac{P_1}{P_4}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

Simplifying finds that:

$$\frac{\gamma T_H}{\gamma - 1}\left[1 - \left(\frac{P_3}{P_2}\right)^{\frac{\gamma-1}{\gamma}}\right] = \frac{\gamma T_C}{\gamma - 1}\left[1 - \left(\frac{P_1}{P_4}\right)^{\frac{\gamma-1}{\gamma}}\right]$$

Rearranging for the pressure ratios for both adiabats finds that:

$$\frac{P_3}{P_2} = \left[1 - \frac{\frac{\gamma_C T_C}{\gamma_C - 1}}{\frac{\gamma_H T_H}{\gamma_H - 1}}\left(1 - \left(\frac{P_1}{P_4}\right)^{\frac{\gamma_C - 1}{\gamma_C}}\right)\right]^{\frac{\gamma_H}{\gamma_H - 1}}$$

$$\frac{P_1}{P_4} = \left[1 - \frac{\frac{\gamma_H T_H}{\gamma_H - 1}}{\frac{\gamma_C T_C}{\gamma_C - 1}}\left(1 - \left(\frac{P_3}{P_2}\right)^{\frac{\gamma_H - 1}{\gamma_H}}\right)\right]^{\frac{\gamma_C}{\gamma_C - 1}}$$

It is then trivial to note that these pressure ratios must be solved either simultaneously or iteratively. One could also determine the pressure ratio by substituting one into the other. For clarity, this is not done within this documentation. Alternatively, one can make a Carnot cycle by creating two isotherms at the reservoir temperatures using an equation of state and then mapping all possible adiabats. This method would then show every possible Carnot cycle within the given data range. Additionally, the above system of equations show that a Carnot cycle in reality is very hard to fabricate in reality due to the co-dependency of the two pressure ratios. Matters are further complicated by remembering that since the heat capacities are functions of temperature and pressure, the heat capacity ratios are also dependent on the state variables.

So assuming that both adiabats are equal and opposite simplifies the first law statement to:

$$RT_H \ln \frac{P_2}{P_1} + RT_C \ln \frac{P_4}{P_3} = -(q_{12} + q_{34})$$

### API

The implementation for all functions used to simulate the processes that constitute a Carnot cycle can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 4/ 04cCarnotCycle.c`. The declarations of the following functions can be found in `04cCarnotCycle.h` at the same file path with the exception of '`CarnotCycle(void)`' which can be found in `B48BC_T4.h`.

```
void CarnotVariable(double *P1, double *P2, double *P3, double *P4, double *THot, double *TCold, double *n, double *gamma1, double *gamma2)
```

This subroutine is used to collect the data required to construct a Carnot cycle.

```
T4CarnotProfile CarnotProfileCalc(double P1, double P2, double P3, double P4, double THot, double TCold, double n, double gamma1, double gamma2, double *worknet, double *qhot, double *qcold)
```

This subroutine is used to calculate the process path that a fluid undertakes when performing the Carnot cycle using equations 3.1.17 and 3.1.11.

```
void CarnotDisplay(double P1, double P2, double P3, double P4, double THot, double TCold, double n, double gamma1, double gamma2, T4CarnotProfile profile, double worknet, double qhot, double qcold)
```

This subroutine is used to display the inputted parameters, calculated variables and generated profile on the user console. It also calculates the thermal efficiency of from the process descriptors and the equivalent reversible process.

```
void CarnotWrite(double P1, double P2, double P3, double P4, double THot, double TCold, double n, double gamma1, double gamma2, T4CarnotProfile profile, double worknet, double qhot, double qcold)
```

This subroutine is used to write the inputted parameters, calculated variables and generated profile to a .txt file. It also calculates the thermal efficiency of from the process descriptors and the equivalent reversible process.

```
void CarnotSwitch(int mode, double P1, double P2, double P3, double P4, double THot, double TCold, double n, double gamma1, double gamma2, T4CarnotProfile profile, double worknet, double qhot, double qcold)
```

This subroutine is used to ask the user whether they would like to either display the results on the console or save the results of this program to a file.

```
void CarnotCycle(void)
```

This subroutine is used to run a simulation of a carnot cycle and evaluates and quantifies the irreversibility of the process by using the functions given with "ThermalEfficiency(void)" and evaluating the Clausius inequality.

## 3.2 The Laws of Thermodynamics

### 3.2.1 The Zeroth Law

### 3.2.2 The First Law

This law of thermodynamics states that energy must always be conserved. Mathematically, this is stated as:

$$Q + W_V = \Delta U + \Delta E_{\text{kin}} + \Delta E_{\text{pot}} \tag{3.2.1}$$

This tells us that the energy content of a stream is made up of its:

- Microscopic internal energy, $U_i$ (kJ).
- Macroscopic kinetic energy, $\Delta E_{\text{kin, i}}$ (kJ).
- Macroscopic potential energy, $\Delta E_{\text{pot, i}}$ (kJ).

Rewriting explicitly in terms of mass:

$$\dot{m}dq + \dot{m}dw_v = \dot{m}du_i + \dot{m}de_{\text{kin, i}} + \dot{m}de_{\text{pot, i}}$$

This equation tells us that the change in energy stored in the fluid between the system inlet and outlet is directly proportional to the net energy exchanged between the inlet and outlet.

#### Applications to open systems

Carrying on from above, the terms can be expanded with the following substitutions:

$$w_v = w_s - w_f$$

$$de_{\text{kin}} = \frac{1}{2}u^2$$

$$de_{\text{pot}} = gz$$

$$\dot{m}q + \dot{m}w_s - w_f = \dot{m}du_i + \dot{m}\frac{1}{2}u_i^2 + \dot{m}gz_i$$

$$\because w_f = -P_1v + P_2v$$

$$q + w_s - (-P_1v + P_2v) = du_i + \frac{1}{2}u_i^2 + gz_i$$

$$q + w_s + P_1v - P_2v = du_i + \frac{1}{2}u_i^2 + gz_i$$

This can now be expanded into the form:

$$q + w_s + P_1v - P_2v = (e_{u,2} - e_{u,1}) + \left(\frac{1}{2}u_2^2 - \frac{1}{2}u_1^2\right) + (gz_2 - gz_1)$$

$$q + w_s = ((e_{u,2} + P_2v) - (e_{u,1} + P_1v)) + \left(\frac{1}{2}u_2^2 - \frac{1}{2}u_1^2\right) + (gz_2 - gz_1)$$

$$\because h = e_u + Pv$$

$$q + w_s = (h_2 - h_1) + \left(\frac{1}{2}u_2^2 - \frac{1}{2}u_1^2\right) + (gz_2 - gz_1)$$

This therefore implies that utilising enthalpy over internal energy allows for the flow work to be accounted for. A process is defined to be at steady state when the following is true:

$$0 = (q + w_s) - (\Delta h + \Delta e_{\text{kin}} + \Delta e_{\text{pot}})$$

Thus when the left hand side is non-zero, then the process is defined as unsteady. By considering mass within this statement of first law:

$$\begin{aligned} 0 &= \dot{m}(q + w_s) - \dot{m}(\Delta h + \Delta e_{\text{kin}} + \Delta e_{\text{pot}}) \\ 0 &= (Q + W_s) - (\Delta H + \Delta E_{\text{kin}} + \Delta E_{\text{pot}}) \end{aligned} \tag{3.2.2}$$

Looking at the above derivation, it is clear that the above equation is only true for one input and one output. This is clearly not reflective of a process in reality and so the inlets and outlets can be generalised as follows neglecting changes to kinetic and potential energy:

$$\sum_i^{\text{in}} \dot{m}_i h_i + \sum \dot{Q} + \sum \dot{W} = \sum_i^{\text{out}} \dot{m}_i h_i$$

It is now important to note that an appropriate reference temperature must be selected such that it reflective of the input and output streams. Typically, this would be the lowest temperature stream which would mean that there is one less value to calculate.

### API

The implementation for all functions used to determine the state of an open-system process can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 2/ 02cOpenFirstLaw.c`. The declarations of the following functions can be found in `02cOpenFirstLaw.h` at the same file path with the exception of '`OpenFirstLaw(void)`' which can be found in `B48BC_T2.h`.

```
void OpenFirstLawProcessVariable(double *q, double *w_s)
```

This subroutine collects values associated with the unit operation under examination.

```
T2StateEnergy OpenFirstLawFluidVariable(int ins)
```

This subroutine collects values associated with the fluid streams at a particular time state.

```
double OpenFirstLawCalculation(double q, double w_s, T2StateEnergy state1, T2StateEnergy
state2)
```

This subroutine is used to calculate the state of the open system using equation 3.2.2.

```
void OpenInitialValue(T2StateEnergy state, double *u, double *z)
```

This subroutine is used to calculate the values of velocity and height from the first law applied to an open system.

```
void OpenFirstLawDisplay(T2StateEnergy state1,T2StateEnergy state2, double q, double w_s,
double sysstate)
```

This subroutine is used to output the input variables and evaluated system state to the user console.

```
void OpenFirstLawWrite(T2StateEnergy state1,T2StateEnergy state2, double q, double w_s,
double sysstate)
```

This subroutine is used to write the input variables and evaluated system state to a .txt file

```
void OpenFirstLawWriteSwitch(T2StateEnergy state1,T2StateEnergy state2, double q, double
w_s, double sysstate)
```

This subroutine is used to ask the user if they would like to save the results of this program to a file.

```
void OpenFirstLaw(void)
```

This subroutine guides the user through the calculations to first the overall energy content for an open systems and notifies the user of whether the process is operating at steady- or unsteady-state.

### 3.2.3 The Second Law

To begin expanding on why we need a second law of thermodynamics, we observe with the first law that energy must be conserved between any two points of a process. However, the first law insufficiently describes the direction in which heat can be naturally transferred. This infers that the first law shows the reversibility of a system and so predicts the impossibility of extracting useful work from no overall energy input (A perpetual motion machine of the first order). However, we also know that when we compress a gas, for example, that the system does not magically go back to the input state. It can then be said that first law then fails to account for system irreversibilities or spontaneous processes or even the direction with which a process can proceed. This therefore infers that there must be some heirarchy within the energy domain which is indicative of the quality. The main culprit for this has been found to be down to heat.

This question of the equivalence of work and heat has been examined with the Kelvin-Planck machine. To summarise this machine, it questions how we know that heat will be transferred from a high to a low temperature. Alternatively, how do we know that the internal energy of a system will transfer energy to the surroundings which are at a lower temperature? The purpose of this subsection is to predict the direction with which heat transfer will occur and formulate the criteria for whether a process is indeed spontaneous and irreversible, reversible or impossible.

To begin with analysing a process to determine whether it is, or isn't, a perpetual motion machine of the second order we need to find a way to represent the heat and work flows. This is possible through drawing a heat engine. For a typical heat engine, a cycle (represented as a circle) has heat flows from/into the hot/cold reservoirs respectively. The cycle then has an additional arrow pointing away from the cycle to represent the net work being removed from the system. Note that this description is reversed in the case of a reverse heat engine (e.g. a heat pump/refrigerator). It is important to note that the depicted reservoirs are infinite in size and so the heat flows do not affect their temperature. To summarise the remainder of this section, we shall first explore the thermal efficiency with respect to a heat engine and consider the Kelvin-Planck statement of the second law to determine the limits of a cycle's reversibility.

We will then explore the coefficient of performance and its relation to the thermal efficiency for a heat pump by also considering the Clausius statement of the second law. It will then be discussed how the two statements are equivalent statements of the second law.

**The second law applied to a heat engine**

From the heat engine representation, we can start to evaluate the system and hence start optimisation. When evaluating a thermodynamic cycle, we often want to optimise the cycle by maximising the net work and minimising the amount of heat we supply. As such, the thermal efficiency of a thermodynamic cycle is defined as:

$$\eta = \frac{|w_{\text{net}}|}{|q_{\text{supplied}}|} = \frac{|w_{\text{net}}|}{|q_{\text{H}}|} \tag{3.2.3}$$

This equation also works economically since it is akin to the process making a return on investment and also is reflective of the operating cost of such an engine. It is clear to see that this equation would also suggest that a heat engine could possibly operate on solely the hot reservoir. As will be found later, this is in violation of the Kelvin-Planck statement of the second law. Additionally, we also know that the net work produced through a thermodynamic cycle is equal and opposite the the net heat from the first law statement of the cycle. Hence:

$$\eta = \frac{|w_{\text{net}}|}{|q_{\text{H}}|} = \frac{|q_{\text{net}}|}{|q_{\text{H}}|} = \frac{|q_{\text{H}}| - |q_{\text{C}}|}{|q_{\text{H}}|}$$

$$\eta = 1 - \frac{|q_{\text{C}}|}{|q_{\text{H}}|}$$

$$\eta_{\text{Carnot}} = 1 - \frac{|RT_C \ln \frac{P_4}{P_3}|}{|RT_H \ln \frac{P_2}{P_1}|}$$

For a Carnot cycle, it is trivial to note that the two pressure ratios should be equivalent. This results in the following equation:

$$\eta_{\text{Carnot}} = 1 - \frac{T_C}{T_H} = \frac{T_H - T_C}{T_H} \tag{3.2.4}$$

This means that the efficiency of a Carnot cycle is also dependent on the reservoir temperatures. We now have two different definitions of the thermal efficiency which begs the question of how to appropriately use the two equations. Since equation 3.2.4 applies to the reservoir temperatures for an ideal gas, this clearly does not reflect reality. But we also know that a Carnot cycle also does not reflect reality so this equation can be used to describe the thermal efficiency of a a thermodynamic cycle's Carnot equivalent. Additionally, irrespective of the cycle, the equation infers that the only way to alter the thermal efficiency would be to alter the reservoir temperatures. Since this is simply not true, the equation must then appropriately describe the thermal efficiency of a Carnot cycle. This then leaves equation 3.2.3 to describe the real process.

In practice, the cold reservoir is limited to the temperature of ambient coolants (about 300 K) and the hot reservoir is limited by the technology used in boiler design (about 800 K). Inputting these values of the hot and cold reservoirs into equation 3.2.4 finds that the maximum thermal efficiency that a process can have is 62.5%. This proves that heat and work are inherently non-interchangable with modern day technology.

---

### API

The implementation for all functions used to calculate the thermal efficiency of a heat engine can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 4/ 04aThermalEfficiency.c`. The declarations of the following functions can be found in `04aThermalEfficiency.h` at the same file path with the exception of '`ThermalEfficiency(void)`' which can be found in `B48BC_T4.h`.

```
void ThermEffVariable(int method, double *wnet, double *qhot, double *qcold, double *Thot,
double *Tcold)
```

This subroutine is used to collect the data required for calculating the thermal efficiency of a thermodynamic cycle and comparing it to its equivalent Carnot cycle.

```
double ThermEffCalc1(double wnet, double qhot)
```

This subroutine is used to calculate the thermal efficiency of a process from the net work and heat supplied to the cycle (Equation 3.2.3).

```
double ThermEffCalc2(double qhot, double qcold)
```

This subroutine is used to calculate the thermal efficiency of a process from the heat supplied and dumped to/from the cycle using the alternative definition of equation 3.2.3.

```
double ThermEffCarnotCalculation(double THot, double TCold)
```

This subroutine is used to calculate the thermal efficiency of a process using the thermal reservoir temperatures (Equation 3.2.4).

```
void ThermEffDisplay(int method, double wnet, double qhot, double qcold, double THot,
 double TCold, double eta, double etac)
```

This subroutine is used to display the inputted parameters and the calculated values on the user console.

```
void ThermEffWrite(int method, double wnet, double qhot, double qcold, double THot,
 double TCold, double eta, double etac)
```

This subroutine is used to write the inputted parameters and the calculated values to a .txt file.

```
void ThermEffWriteSwitch(int method, double wnet, double qhot, double qcold, double THot,
double TCold, double eta, double etac)
```

This subroutine is used to ask the user whether they would like to output the inputted parameters and the calculated values to a .txt file.

```
void ThermalEfficiency(void)
```

This subroutine is used to calculate the thermal efficiency of a heat engine extracting heat from a hot thermal reservoir and dumping heat into a cold thermal reservoir. It also contains

the calculations comparing the given process to its equivalent carnot cycle to determine whether or not the proposed is in violation of the quality argument of the second law of thermodynamics.

---

It should now be noted that the Carnot cycle is intrinsically reversible. That is the isothermal and adiabatic processes are all reversible and so if the cycle was operated as a heat engine all heat and work effects would cancel out. This is also due in part that the equations modelling such processes being symmetrical. Since all the processes contained within the cycle are reversible, the cycle itself must also be reversible. This leads to the question of what makes a cycle irreversible?

Irreversibilities occur due to dissipative effects on the working fluid. This can then be further separated into external (e.g. heat transfer to the surroundings) and internal (e.g. viscosity) irreversibilities. This means that an irreversible process must either release some additional heat or lose some work. This means that the reversible Carnot cycle will produce a minimal amount of net heat and a maximal amount of net work. This then means that the thermal efficiency of an irreversible thermodynamic cycle must be less than that of the equivalent Carnot cycle.

However, what if the thermal efficiency of some real process does exceed the predicted Carnot cycle thermal efficiency? Putting the proposed real cycle in heat engine form and pairing it with the Carnot cycle such that both cycles are operating between the same two reservoirs will find that the heat supplied to the process is equivalent. Reversing the heat flows in the Carnot cycle will then render the hot reservoir useless since there is no net hot heat flow. This means that the combined heat engine is effectively operating from solely the cold reservoir since the cold heat flows are also dependent on net work produced from the process. Looking further into this problem, we find that the net work exchanged with the surroundings exceed zero thus the combined cycle is producing useful work from a single reservoir. It should now be said that the Kelvin-Planck statement of the second law is formally stated as:

"It is impossible to construct a system that will operate cyclically, extract heat from a single reservoir and supply the same amount of net work to the surroundings."

This states that the above proposed machine is impossible and is backed through empirical findings or that the Carnot cycle without the condenser present is impossible. This would then imply that the thermal efficiency of a real irreversible cycle must be smaller than its Carnot cycle counterpart.

**The second law applied to a heat pump**

A heat pump is alternatively called a reverse heat engine. They operate by absorbing heat from a cold reservoir and moving it to a higher temperature reservoir by inputting work into the cycle. This is seen in reality with a refrigerator. In the case of a refrigerator, the cold reservoir is the refrigerator volume and hot reservoir is the surrounding environment. Since we are no longer extracting work, the thermal efficiency is no longer of any use to us. Instead, the performance of such a cycle is now characterised by the amount of work that we input into the system and how much heat we can move from the cold space to the surroundings. Alternatively, a heat pump's (e.g. an air conditioning system) performance is rated by how much heat can

be supplied to the hot reservoir. This is termed the coefficient of performance and is defined as follows:

$$\begin{aligned} \text{COP}_{\text{fridge}} &= \frac{|q_C|}{|w_{\text{net}}|} \\ \text{COP}_{\text{Heat pump}} &= \frac{|q_H|}{|w_{\text{net}}|} \end{aligned} \tag{3.2.5}$$

The relationship that the coefficient of performance has with the thermal efficiency is then:

$$\begin{aligned} \text{COP}_{\text{fridge}} &= \frac{1}{\eta} - 1 = \frac{T_C}{T_H - T_C} \\ \text{COP}_{\text{Heat pump}} &= \frac{1}{\eta} = \frac{T_H}{T_H - T_C} \end{aligned} \tag{3.2.6}$$

Once again, the latter part of the above equations is true for the equivalent reversible process and equation 3.2.5 should be used for the real process. Since we know that a heat pump must have irreversibilities, it should be noted that the natural direction for irreversibilities to occur is excess heat being lost to the cold reservoir.

---

### API

The implementation for all functions used to calculate the Coefficient of Performance for a reverse heat engine can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 4/ 04bCoefficientofPerformance.c`. The declarations of the following functions can be found in `04bCoefficientofPerformance.h` at the same file path with the exception of '`CoefficientofPerformance(void)`' which can be found in `B48BC_T4.h`.

```
void CoPVariable(int method, double *wnet, double *qhot, double *qcold, double *THot,
 double *TCold)
```

This subroutine is used to collect the data required to calculate the Coefficient of Performance for either a refrigerator or a heat pump.

```
double CoPFridge(double qcold, double wnet)
```

This subroutine is used to calculate the coefficient of performance for a refrigerator (Equation 3.2.5)

```
double CoPHeatPump(double qhot, double wnet)
```

This subroutine is used to calculate the coefficient of performance for a heat pump (Equation 3.2.5).

```
double CoPRevFridge(double TCold, double THot)
```

This subroutine is used to calculate the reversible coefficient of performance for a refrigerator (Equation 3.2.6).

```
double CoPRevHeatPump(double TCold, double THot)
```

This subroutine is used to calculate the reversible coefficient of performance for a heat pump (Equation 3.2.6).

```
double etaFridge(double CoP)
```

This subroutine is used to calculate the thermal efficiency of a refrigerator (Equation 3.2.6).

```
double etaHeatPump(double CoP)
```

This subroutine is used to calculate the thermal efficiency of a heat pump (Equation 3.2.6).

```
void CoPDisplay(int method, double wnet, double qhot, double qcold, double THot, double
TCold, double CoP, double CoPRev, double eta, double etac)
```

This subroutine is used to display the inputted parameters and calculated values on the user console.

```
void CoPWrite(int method, double wnet, double qhot, double qcold, double THot, double TCold,
double CoP, double CoPRev, double eta, double etac)
```

This subroutine is used to write the inputted parameters and calculated values to a .txt file.

```
void CoPWriteSwitch(int method, double wnet, double qhot, double qcold, double THot,
 double TCold, double CoP, double CoPRev, double eta, double etac)
```

This subroutine is used to ask the user whether they would like to write the subroutine data to a .txt file.

```
void CoefficientofPerformance(void)
```

This subroutine is used to calculate the coefficient of performance for a refrigerator or a heat pump. This, like with "ThermalEfficiency(void)" calculates the equivalent reversible process and checks whether or not the proposed process is in violation of the quality argument of the second law of thermodynamics.

---

The question of whether a heat pump requires work input was explored by Clausius who upon experimentation stated that:

"It is impossible to construct a system that will operate cyclically, while transferring heat from a cooler to a hotter body without a net quantity of work being supplied to the system by the surroundings."

This statement is consistent with the fact that hot things cool down instead of getting hotter or considering energy from the perspective of energy quality. This also infers that in order to transfer heat against the temperature gradient, an energy penalty must be paid to offset this increase in energy quality. Additionally, constructing the Clausius machine finds that the heat transferred from the cold reservoir to the cycle must be equivalent to the heat supplied to the hot reservoir.

To begin exploring the equivalence of the two statements we can construct the combined heat engine with the Clausius and Kelvin-Planck engines. Since the heat flows are in opposing directions, we find that the heat flows from/to the cold reservoir are equal and opposite. This then means that the combined cycle is operating from a single reservoir which would then make the proposed combined heat engine in violation of the Kelvin-Planck statement of the second.

To summarise the findings of the thermodynamic feasibility with regards to the second law:

Table 3.2.1: Thermodynamic feasibility

| Thermal efficiency | Coefficient of Performance | Feasibility | Reversibility |
|---|---|---|---|
| $\eta = \eta_{\text{Carnot}}$ | $\text{COP} = \text{COP}^{\text{rev}}$ | Feasible | Reversible |
| $\eta < \eta_{\text{Carnot}}$ | $\text{COP} < \text{COP}^{\text{rev}}$ | Feasible | Irreversible |
| $\eta > \eta_{\text{Carnot}}$ | $\text{COP} > \text{COP}^{\text{rev}}$ | Impossible | |

**The Clausius Inequality**

Although considering thermodynamic feasibility from the system viewpoint does allow us to make predictions about the feasibility of a process, we may also want to consider the impact that a process will have on its direct surroundings. To begin finding out how to calculate this impact, we should start by considering ratio of the heat flows within the system.

$$\frac{q_H}{q_C} = \frac{RT_H \ln \frac{P_2}{P_1}}{RT_C \ln \frac{P_4}{P_3}} = -\frac{T_H}{T_C}$$

The negative sign reappears on the right hand side since $q_C$ is negative. From this equation we find that:

$$\frac{q_H}{T_H} + \frac{q_C}{T_C} = 0$$

This equation is counter-intuitive since we know that we are reliant on this quantity being non-zero to be able to extract work from a thermodynamic cycle. This implies that this quantity must be conserved regardless of the irreversibilities of the process. Thus for a reversible process:

$$\sum_i \frac{q_i}{T_i} = 0$$

Over an infinite series:

$$\oint_{\text{rev}} \frac{q_i}{T_i} = 0$$

Since this does not apply to an irreversible process, we can turn to table 3.2.1 for some insight. For an irreversible process, the following must be true:

$$\because \eta < \eta_{\text{Carnot}}$$

$$\frac{|w_{\text{net}}^{\text{irrev.}}|}{|q_{\text{H}}^{\text{irrev.}}|} = \frac{|q_{\text{H}}^{\text{irrev.}}| - |q_{\text{C}}^{\text{irrev.}}|}{|q_{\text{H}}^{\text{irrev.}}|} < \frac{T_H - T_C}{T_H}$$

Rearrangement finds that:

$$\frac{q_{\text{H}}^{\text{irrev.}}}{T_H} + \frac{q_{\text{C}}^{\text{irrev.}}}{T_C} < 0$$

$$\oint_{\text{irrev}} \frac{q_i}{T_i} < 0$$

Once again, we see that the quantity $dq_i/T_i$ is conserved throughout the process. This quantity by definition is known as the entropy. The above derived equations are otherwise known as the

Clausius inequality which is stated throughout literature as:

$$\oint \frac{dQ}{T} \leq 0 \tag{3.2.7}$$

### API

The implementation for all functions used to evaluate the Clausius Inequality can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 4/ 04dClausiusInequality.c`. The declarations of the following functions can be found in `04dClausiusInequality.h` at the same file path with the exception of '`ClausiusInequality(void)`' which can be found in `B48BC_T4.h`.

`T4EntropyDef EntropyVariable(int i, T4EntropyDef data)`

This subroutine is used to get the variables required to calculate the entropy at a given time state.

`T4EntropyDef EntropyCalculation(int i, T4EntropyDef data)`

This subroutine is used to calculate the entropy at a given time state from the data contained within the struct 'data' (Equation 3.2.7).

`double EntropyCalc(double q, double T)`

This subroutine is used to calculate the entropy using the Clausius inequality (Equation 3.2.7). This subroutine is not used within this file and is meant for use when the entropy is needed for calculation elsewhere in the program.

`void EntropyDisplay(int imax, T4EntropyDef data)`

This subroutine is used to display the collected and calculated data on the user console.

`void EntropyWrite(int imax, T4EntropyDef data)`

This subroutine is used to write the collected and calculated data to a .txt file.

`void EntropyWriteSwitch(int imax, T4EntropyDef data)`

This subroutine is used to ask the user if they would like to output the inputted parameters and calculated data to a .txt file.

`void ClausiusInequality(void)`

This subroutine is used to guide the user through evaluating the thermodynamic feasibility of a process through analysis of the conserved heat-temperature property (entropic contributions) throughout a process. After all heat contributions have been inputted into the subroutine, it then notifies the user of whether the given process is in violation of the second law of thermodynamics.

### 3.2.4 The Third Law

## 3.3 The Joule-Thomson effect

This effect is alternatively known as an isenthalpic throttling process and is observed when a gas is expanded from a higher to a lower pressure. For a real gas upon expansion, it has been observed empirically that the temperature will at first rise then fall. To begin deriving

the mathematical description of this effect, we shall first consider the ideal scenario before expanding to the real case. Then after considering the theory for this effect, we then validate that this effect can only be applied to real gases in addition to applying the derived theory to a van der Waals gas using the van der Waals equation of state (Equation 2.2.13).

The ideal scenario to observe this effect would be to consider the ideal gas. Starting from Boyle's law:

$$P_1\dot{V}_1 = P_2\dot{V}_2$$

$$P_2 = P_1\frac{\dot{V}_1}{\dot{V}_2}$$

$$P_1 = P_2\frac{\dot{V}_2}{\dot{V}_1}$$

$$\because \Delta P = P_2 - P_1 = P_1\frac{\dot{V}_1}{\dot{V}_2} - P_2\frac{\dot{V}_2}{\dot{V}_1}$$

$$\Delta P = P_1\dot{V}_1\left(\frac{1}{\dot{V}_2} - \frac{P_2\dot{V}_2}{P_1\dot{V}_1}\frac{1}{\dot{V}_1}\right)$$

$$\because P_1\dot{V}_1 = P_2\dot{V}_2 \Rightarrow \frac{P_2\dot{V}_2}{P_1\dot{V}_1} = 1$$

$$\Delta P = P_1\dot{V}_1\left(\frac{1}{\dot{V}_2} - \frac{1}{\dot{V}_1}\right) = P_1\dot{V}_1\left(\frac{\dot{V}_1 - \dot{V}_2}{\dot{V}_1\dot{V}_2}\right)$$

$$\Delta P = -\frac{P_1}{\dot{V}_2}\Delta\dot{V}$$

This equation provides a relationship between changes in pressure and volume (A proportional relationship). This means that for a large pressure drop, there must be a correspondingly large expansion of gas. This appears to be in violation of Bernoulli's principle (Equation 4.2.5) which also tells us that there must be a large increase in gas velocity with the same pressure drop. To examine whether this is indeed true or not, we can go back to the first law for an open system. This is later stated (Equation 3.2.2) as:

$$(Q + W_s) = (\Delta H + \Delta E_{\text{kin}} + \Delta E_{\text{pot}})$$

The change in kinetic energy is defined as:

$$\Delta E_{\text{kin}} = \frac{1}{2}\dot{m}(u_2^2 - u_1^2)$$

In this form, it is irrelevant to our current purposes, however we can make the following substitution to determine the flow velocity in terms of volumetric flow rate assuming that the pipe diameter across the two time states remains constant:

$$\because u_i = \frac{\dot{V}_i}{A}$$

In terms of the volumetric flow ratio the kinetic energy can be rewritten as:

$$\Delta e_{\text{kin}} = \frac{1}{2}\left(\frac{\dot{V}_2^2}{A^2} - \frac{\dot{V}_1^2}{A^2}\right) = \frac{1}{2}\frac{V_1^2}{A^2}\left[\left(\frac{\dot{V}_2}{\dot{V}_1}\right)^2 - 1\right]$$

For an isothermal process:

$$P_1 V_1 = P_2 V_2$$

$$\frac{V_2}{V_1} = \frac{P_1}{P_2}$$

$$\therefore \Delta e_{\text{kin}} = \frac{1}{2}\frac{V_1^2}{A^2}\left[\left(\frac{P_1}{P_2}\right)^2 - 1\right]$$

$$\because (Q + W_s) = (\Delta H + \Delta E_{\text{kin}} + \Delta E_{\text{pot}})$$

Since there is no heat or work occurring and assuming that there are negligible changes in potential energy:

$$0 = \Delta h + \Delta e_{\text{kin}}$$

$$\therefore \Delta h = -\Delta e_{\text{kin}}$$

$$\because \Delta h = c_P \Delta T$$

$$\Delta T = \frac{\Delta h}{c_P} = \frac{-\Delta e_{\text{kin}}}{c_P}$$

For an ideal gas, this temperature change is typically less that $1K$ and so the process can indeed be assumed to be operating isothermally.

In the real case, we now need to consider the enthalpy as a function of the system temperature and pressure. Fortunately, this relationship is given with the Gibbs-Duhem equation which holds for any state variable. Thus the specific enthalpy is defined as:

$$dh = \left(\frac{\delta h}{\delta T}\right)_P dT + \left(\frac{\delta h}{\delta P}\right)_T dP$$

The temperature dependence of enthalpy, at constant pressure is otherwise known as the heat capacity at constant pressure.

$$c_P = \left(\frac{\delta h}{\delta T}\right)_P$$

[4] gives the pressure dependence of enthalpy as:

$$\left(\frac{\delta h}{\delta P}\right)_T = v - T\left(\frac{\delta v}{\delta T}\right)_P$$

These substitutions then transform the Gibbs-Duhem statement of specific enthalpy to:

$$dh = c_P dT + \left[v - T\left(\frac{\delta v}{\delta T}\right)_P\right] dP$$

The isobaric thermal expansion coefficient is, rather conveniently, defined as:

$$\alpha = \frac{1}{v}\left(\frac{\delta v}{\delta T}\right)_P$$

This, once again, transforms the Gibbs-Duhem statement of enthalpy to be:

$$dh = c_P dT + v\left[1 - \frac{T}{v}\left(\frac{\delta v}{\delta T}\right)_P\right] dP = c_P dT + v\left[1 - T\alpha\right] dP$$

Assuming that the statements made about the Joule-Thomson effect applied to ideal gases are true, we can then make the observation that $dh \approx 0$ which must then apply to any isenthalpic expansion. This allows the Gibbs-Duhem statement to be rearranged to:

$$0 = c_P dT + v\left[1 - T\alpha\right] dP$$

$$c_P dT = -v\left[1 - T\alpha\right] dP$$

$$dT = -\frac{v}{c_P}\left[1 - T\alpha\right] dP$$

The Joule-Thomson coefficient coefficient is defined as:

$$\mu_{J-T} = \left(\frac{dT}{dP}\right)_h = -\frac{v}{c_P}\left[1 - T\alpha\right] \left\{ \frac{\mathrm{K}}{\mathrm{Pa}} \right. \tag{3.3.1}$$

To explain this effect through kinetic theory of real gases we first need to understand that the thermodynamic pressure is simply the frequency of molecular-level collisions that a gas will have on the system boundary. Thus as the pressure falls due to a throttling process, the gas particles must move further apart and the potential energy of the gaseous particles must then increase, as can be observed with Hamiltonian mechanics. Since a throttling process is isenthalpic, there must be no external exchange of energy between the system and its surroundings. Since temperature is proportional to the kinetic energy, this leads to the observation that a gas may either cool or heat up as it expands. From the generalisation that lighter particles must have higher root mean squared velocities than heavier gases. This root mean squared velocity must therefore implicitly affect the inversion temperature. This can be validated by looking at the inversion temperature of lighter gases (such as Hydrogen or Helium) and comparing to heavier gases (such as Nitrogen or Oxygen).

To begin testing the applications of this theory to real life situations, we shall first test the Joule-Thomson coefficient on an ideal gas and then on a van der Waals gas. Since we know that this effect is only true for a real gas, we can expect the Joule-Thomson coefficient to be zero for an ideal gas. It is later discovered that the Joule-Thomson coefficient for a real gas can indeed be zero but only at the inversion temperature. This is the temperature at which there will be no temperature change as the gas expands.

To begin calculating the Joule-Thomson coefficient, we first need to determine the isobaric thermal expansion coefficient. To determine this value, we first need to determine the relationship that the molar volume has with the system pressure. For an ideal gas, the molar volume

is related to pressure with the following equation:

$$v = \frac{RT}{P}$$

The isobaric thermal expansion coefficient can now be defined as:

$$\alpha = \frac{1}{v}\left(\frac{\delta v}{\delta T}\right)_P = \frac{R}{Pv} = \frac{1}{T}$$

$$\therefore [1 - T\alpha] = 0$$

$$\therefore \mu_{J-T} = 0$$

This then suggests that the ideal gas is always at its inversion temperature. Going from empirical evidence, this is simply not true. This then requires more rigorous analysis by looking at a real gas whose thermodynamic properties can be modelled by a cubic equation of state.

The van der Waals equation of state has been stated as:

$$\left(P + \frac{a}{V^2}\right)(V_m - b) = RT$$

Expanding:

$$Pv + \frac{a}{v} - Pb - \frac{ab}{v^2} = RT$$

Implicitly differentiating:

$$P\left(\frac{\delta v}{\delta T}\right)_P - \frac{a}{v^2}\left(\frac{\delta v}{\delta T}\right)_P - (0) - \frac{ab}{v^3}\left(\frac{\delta v}{\delta T}\right)_P = R$$

$$\therefore \left(\frac{\delta v}{\delta T}\right)_P = \frac{R}{P - \frac{a}{v^2} + \frac{2ab}{v^3}}$$

$$\alpha = \frac{1}{v}\left(\frac{\delta v}{\delta T}\right)_P = \frac{R}{Pv - \frac{a}{v} + \frac{2ab}{v^2}}$$

$$\therefore \mu_{J-T} = \frac{v}{c_P}\left[\frac{RT}{Pv - \frac{a}{v} + \frac{2ab}{v^2}} - 1\right]$$

Since a real gas will either cools or heats up as it expands, this implies that there must be some temperature at which neither process will occur. This observation is termed the inversion temperature. For any real gas at its inversion temperature:

$$[T_i\alpha - 1] = 0 \Rightarrow T_i = \frac{1}{\alpha}$$

$$\therefore T_i = \frac{Pv - \frac{a}{v} + \frac{2ab}{v^2}}{R} \tag{3.3.2}$$

To summarise the Joule-Thomson effect, the mathematical description has been derived

using the Gibbs-Duhem expansion of enthalpy to be:

$$\mu_{J-T} = \left(\frac{dT}{dP}\right)_h = -\frac{v}{c_P}\left[1 - T\alpha\right]\left\{\frac{\mathrm{K}}{\mathrm{Pa}}\right. \tag{3.3.3}$$

Where:

- $v$ is the molar volume.
- $c_P$ is the heat capacity at constant pressure.
- $T$ is the temperature at which the throttling process is occurring.
- $\alpha$ is the isobaric thermal expansion coefficient.

The temperature change following the throttling process can be found using the first law applied to open system (Equation 3.2.2). It has been derived that at nearly all temperatures, a real gas will either cool or heat up. The temperature will neither rise nor fall when the gas is at its inversion temperature and so this effect is general to any gas but is component specific. This means that there are three general cases for this effect:

- $[1 - T\alpha] > 0 \therefore \mu_{J-T} = +\text{ve}$

  This indicates that there is a proportional relationship between pressure and temperature. Alternatively, the gas will cool as it expands.

- $[1 - T\alpha] = 0 \therefore \mu_{J-T} = 0$

  This indicates that the system temperature will remain constant throughout the throttling process, otherwise known as the "Joule-Thomson inversion temperature". This can be calculated using $T_{\text{inv}} = 1/\alpha$.

- $[1 - T\alpha] < 0 \therefore \mu_{J-T} = -\text{ve}$

  This indicates that there is a inversely proportional relationship between pressure and temperature. Alternatively, the gas will heat up as it expands.

### API

The implementation for all functions used to calculating the Joule-Thomson coefficient and inversion temperature can be found in `Course Material/ Year 2/ 5. B48BC - Process Engineering B (Thermodynamic Processes)/ Topic 3/ 03aJTEffect.c`. The declarations of the following functions can be found in `03aJTEffect.h` at the same file path with the exception of '`JTEffect(void)`' which can be found in `B48BC_T3.h`.

```
void JTEffectVariable(double *Tc, double *Pc, double *T, double *P, double *v, double *c_p)
```

This subroutine is used to collect the data required to calculate the Van der Waals statement of the Joule-Thomson coefficient and associated variables.

```
double JTCoefficientCalculation(double v, double c_p, double T, double P, double a, double
b)
```

This subroutine is used to calculate the Joule-Thomson coefficient using the Van der Waals equation of state (equation 3.3.3).

```
double JTInvTemperatureCalculation(double P, double v, double a, double b)
```

This subroutine is used to calculate the Joule-Thomson Inversion temperature for a van der Waals gas (equation 3.3.2).

```
void JTEffectDisplay(double Tc, double Pc, double T, double P, double v, double c_p, double
a, double b, double mu_JT, double Tinv)
```

This subroutine is used to display the collect and calculated data within this file.

```
void JTEffectWrite(double Tc, double Pc, double T, double P, double v, double c_p, double
a, double b, double mu_JT, double Tinv)
```

This subroutine is used to write the inputted parameters and calculated data to a .txt file.

```
void JTEffectWriteSwitch(double Tc, double Pc, double T, double P, double v, double c_p,
double a, double b, double mu_JT, double Tinv)
```

This subroutine is used to ask the user whether they would like to save the inputted parameters and calculated data to a .txt file.

```
void JouleThomsonEffect(void)
```

This subroutine is used to calculate the Joule-Thomson coefficient which provides a prediction of gas behaviour when being throttled isenthalpically. This subroutine also calculates the inversion temperature.

## 3.4 Thermodynamic Properties

### 3.4.1 Fugacity

# 4 Fluid Dynamics

## 4.1 Fluid Statics

This section is concerned with studying fluids at steady-state (See section 3.2.2).

### 4.1.1 Fluid Coefficient of Compressibility

Within the student guide this has been defined as:

$$\boxed{c = -\frac{1}{V}\left(\frac{\delta V}{\delta P}\right)_T} \tag{4.1.1}$$

#### The Ideal Gas Law

With the ideal gas law given as:

$$PV = nRT \Rightarrow V = \frac{nRT}{P}$$

The fluid coefficient of compressibility for an ideal gas can now be defined as:

$$\left(\frac{\delta V}{\delta P}\right)_T = \frac{-nRT}{P^2}$$

$$c = -\frac{1}{V}\left(\frac{-nRT}{P^2}\right) = \frac{nRT}{VP^2}\left\{\frac{\mathrm{m}^3}{\mathrm{Pa}}\right. \tag{4.1.2}$$

#### General Cubic Equation of State

#### Analysis

It is worth remembering that the ideal gas assumes that all molecules under analysis are point particles with no volume so this definition is essentially useless since it will always state that the fluid is compressible. However, analysing the units shows that the thermal energy contained within the system (the numerator) should be significant smaller than the mechanical stressors on the system (the denominator) for the fluid to be considered compressible. Looking at the mathematical output of this equation shows that for an ideal gas, numbers tending to 0 are considered compressible. As the value tends to unity, the fluid tends to become incompressible.

This equation has applications in Computational Fluid Dynamics (CFD) to determine which set of the Navier-Stokes equations should be used to model the fluid in question.

#### API

The implementation for all functions used to calculate the fluid coefficient of compressibility can be found in `Course Material/ Year 2/ 3. B48BC - Process Engineering B (Fluid`

`Dynamics)/ Topic 1/ 01aFluComp.c`. The declarations of the following functions can be found in `01aFluComp.h` at the same file path with the exception of '`CoefficientofCompressibility(void)`' which can be found in `B48BB_T1.h`.

```
void FluCompVariable(double *P, double *V, double *n, double *T)
```

This subroutine is used to collect the data to required calculate the fluid coefficient of compressibility.

```
double FluCompCalculation(double P, double V, double n, double T)
```

This subroutine is used to calculate the fluid compressibility factor from the ideal gas law (Equation 4.1.2).

```
void FluCompDisplay(double P, double V, double n, double T, double c)
```

This subroutine is used to output the inputted variables and program results to the user console.

```
void FluCompWrite(double P, double V, double n, double T, double c)
```

This subroutine is used to write the results of this program to a .txt file.

```
void FluCompWriteSwitch(double P, double V, double n, double T, double c)
```

This subroutine is used to ask the user if they would like to save the results of this program to a file.

```
void CoefficientofCompressibility(void)
```

This subroutine guides the user through gathering the data and calculation of the fluid coefficient of compressibility derived from the ideal gas law.

### 4.1.2 Hydrostatic Pressure Theorems

Assuming that the fluid is incompressible, mass has the units:

$$m = \rho V$$

For a cylinder:

$$V = A.\Delta h$$

**Pascal's law** is defined as:

$$\boxed{P = \frac{F}{A}\left\{\frac{\mathrm{N}}{\mathrm{m}^2} = \frac{\mathrm{kg}}{\mathrm{m.s}}\right.} \tag{4.1.3}$$

#### Derivation of the Hydrostatic Pressure Theorems

The student guide provides the following derivations:

#### Hydrostatic Pressure Theorem for a Vertical Fluid Element

For a plug of fluid travelling vertically, the forces can be balanced vertically:
[Insert figure]

$$\sum F = P_2.A - P_1.A - \rho A(h_2 - h_1)g = 0$$

Dividing the above by $A$:

$$\frac{F}{A} = P = P_2 - P_1 = \rho g(h_2 - h_1)$$

Taking the lower height as the reference height allows for the hydrostatic pressure gradient to be defined as:

$$P_2 - P_1 = \Delta P = \rho g h \tag{4.1.4}$$

### Hydrostatic Pressure Theorem for a Horizontal Fluid Element

[Insert figure]
Resolving forces for a plug of fluid travelling horizontally results in:

$$\begin{aligned} \leftrightarrow &: \sum F = P_2 A - P_1 A = 0 \\ \updownarrow &: F = mg \end{aligned} \tag{4.1.5}$$

$$P_2 = P_1$$

Therefore, for a fluid at rest the hydrostatic pressure is equal at any horizontal point.

### Analysis

This equation shows that there is zero hydrostatic pressure at the free fluid surface of an incompressible fluid and that hydrostatic pressure increases as you descend from the the free fluid surface to the bottom of the container. It can be thought that this equation is analogous to the 'weight'(sic) of fluid above a fixed point in a volume. Something not explicitly shown in this equation is the principle of hydrostatic equilibrium which is "that the pressure at any point in a fluid at rest is just due to the weight of the overlying fluid"[5]. In other words, the hydrostatic pressure at any point in the fluid is the same at any point directly horizontal to said point.

### API

The implementation for all functions used to calculate the vertical hydrostatic pressure for a stationary fluid can be found in `Course Material/ Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 1/ 01bFluidVHyd.c`. The declarations of the following functions can be found in `01bFluidVHyd.h` at the same file path with the exception of '`FluidVerticalHydrostaticPressure(void)`' which can be found in `B48BB_T1.h`.

```
void FluidVHydVariable(double *rho, double *h)
```

Subroutine used for collecting the data required for calculating the fluid hydrostatic pressure.

```
double FluidVHydCalculation(double rho, double h)
```

Subroutine used to calculate the fluid hydrostatic pressure using equation 4.1.4. The value of fluid hydrostatic pressure is returned to the calling function.

```
void FluidVHydDisplay(double rho, double h, double P)
```

This subroutine is used to write the results of this program to the user console.

```
void FluidVHydWrite(double rho, double h, double P)
```

This subroutine is used to write the results of this program to a .txt file.

```
void FluidVHydWriteSwitch(double rho, double h, double P)
```

This subroutine is used to ask the user if they would like to save the results of this program to a file.

```
void FluidVerticalHydrostaticPressure(void)
```

This subroutine guides the user through gathering the data and calculation of the hydrostatic pressure gradient for a stationary fluid.

### 4.1.3 Manometers

Manometers are devices used to measure the pressure that a fluid exerts on a pipe's walls by utilising the hydrostatic pressure theorems. Although not widely used in industry due to their size making them impractical in industrial applications, they can be used for experimental purposes where size is not an issue. Pressure is typically measured in industry with a Bourdon tube.

[Insert figure]

Taking a datum line just above the manometer bend results in the following force balances:

$$A : P_A = P_1 + \rho_1 g h_1$$
$$B : P_B = P_2 + \rho_2 g h_2$$

Since the horizontal hydrostatic pressure theorem states that:

$$P_A = P_B$$

It can then be inferred that:

$$P_1 + \rho_1 g h_1 = P_2 + \rho_2 g h_2$$
$$P_2 - P_1 = \Delta P = g(\rho_1 h_1 - \rho_2 h_2) \tag{4.1.6}$$

Equation 4.1.6 allows two things to be found since:

- The process fluid and manometer fluid densities are known.
- The pressure of the surroundings, $P_2$, is known.
- Either the height of the process fluid in the manometer or the pressure of the process fluid is known.

Thus, the remaining variables that can be calculated are either the pressure of fluid inside the pipe at the point of measurement, $P_1$, or the height of manometer fluid, $h_2$. It is therefore trivial to say that when calculating one, the other must be specified.

During a live process, it is good to know the pressure at some length of pipe to ensure that fluid is flowing through the pipe and is in agreement with the estimated frictional pressure losses

(See section 4.3.1). Equation 4.1.6 can then be rearranged as follows for the fluid pressure inside the pipe.

$$P_1 = P_2 + g(\rho_2 h_2 - \rho_1 h_1) \tag{4.1.7}$$

Another area of research would be in the simulation of these results, in which case the height of manometer fluid would also be known. This result can also be used to size a manometer by finding the pressure constraints that a manometer can measure.

$$\begin{aligned} h_2 &= \frac{\rho_1 g h_1 - \Delta P}{\rho_2 g} \\ &= \frac{\rho_1 h_1}{\rho_2} - \frac{\Delta P}{\rho_2 g} \end{aligned} \tag{4.1.8}$$

It may sometimes be the case that the manometer arm is inclined such that the following diagram is representative of the case: [Insert figure] Some basic trigonometry finds that:

$$h_2 = L \sin\theta$$

which can then be substituted in either equation 4.1.7 or 4.1.8.

### API

The implementation for all functions used to calculate the pressure of a process fluid or estimate the height of fluid in a manometer can be found in `Course Material/ Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 1/ 01cMano.c`. The declarations of the following functions can be found in `01cMano.h` at the same file path with the exception of '`Manometer(void)`' which can be found in `B48BB_T1.h`.

```
void ManoMeasVariable(double *P2, double *rho1, double *h1, double *rho2, double *h2)
```

This subroutine is used to collect the data required for completing manometer measurement calculations. The stated units state the units that the data is collected in prior to being converted for calculation.

```
void ManoEstiVariable(double *P1, double *P2, double *rho1, double *rho2, double *h1)
```

This subroutine is used to collect the data required for completing calculations to estimate the rise of manometer fluid. The stated units state the units that the data is collected in prior to being converted for calculation.

```
double ManoMeasCalculation(double P2, double rho1, double h1, double rho2, double h2)
```

This subroutine is used for calculating the process fluid pressure given some atmospheric conditions using equation 4.1.7. After calculation, the function returns the value of P1 or the fluid pressure.

```
double ManoEstiCalculation(double P1, double P2, double rho1, double h1, double rho2)
```

This subroutine is used for estimating the height of manometer fluid given some process fluid pressure in an attached pipe using equation 4.1.8.

```
void ManoMeasDisplay(double P1, double P2, double rho1, double h1, double rho2, double
h2)
```

This subroutine is used to write the results from manometer measurement calculations to

the user console.

`void ManoMeasWrite(double P1, double P2, double rho1, double h1, double rho2, double h2)`

This subroutine is used to output the results from the manometer estimation calculations to the user console.

`void ManoMeasWriteSwitch(double P1, double P2, double rho1, double h1, double rho2, double h2)`

This subroutine is used to ask the user if they would like to save the results of this measurement program to a file.

`void ManoEstiDisplay(double P1, double P2, double rho1, double h1, double rho2, double h2)`

This subroutine is used to write the results from the manometer estimation calculations to a .txt file.

`void ManoEstiWrite(double P1, double P2, double rho1, double h1, double rho2, double h2)`

This subroutine is used to write the results from the manometer estimation calculations to a .txt file.

`void ManoEstiWriteSwitch(double P1, double P2, double rho1, double h1, double rho2, double h2)`

This subroutine is used to ask the user if they would like to save the results of the estimation program to a file.

`void Manometer(void)`

This subroutine guides the user through gathering the data and calculation of the pressure that a fluid exerts of a system wall through either a vertical or inclined manometer.

### 4.1.4 Surface tension and Wettability

Within a fluid, there are two forces [6] that can be thought to hold a fluid together:

- **Cohesive forces** hold the liquid molecules together. These forces balance such that there is **no net effect** in any direction.
- **Adhesive forces** show the interaction of forces liquid and gas molecules at the interface.

The resultant of these two forces mean that the liquid molecules at the surface will seek a net inward attraction towards the bulk of the liquid. This provides the following observations of liquid behaviour:

- The surface will contract and seek to adopt a minimum area.
- The surface will behave like a stretched elastic sheet. This in turn gives liquid a property called **surface tension**.

Surface tension causes many fluids to act the way that we see them today, such as water droplets adopting a spherical shape. This is due to the inward cohesive forces pulling the free liquid surface inward to adopt a minimum surface area. Surface tension will also support the weight of a light paper-clip as the unbalanced cohesive force exerted on the paper-clip will cause

the paper-clip to be supported by the cohesive forces in the liquid. An overview of methods that can be used to calculate the adhesive forces has been given in [7].

The surface tension is defined as [6]:

$$\sigma = \frac{F}{L}\left\{\frac{\mathrm{N}}{\mathrm{m}}\right. \tag{4.1.9}$$

Where:

- $F$ = Force (N).
- $L$ = Characteristic length (m).

This is the unidirectional force in any given direction. This then infers that prior to an object breaking a liquid's surface, there will be some maximum force that the surface can resist before the object enters the liquid. I.e.

$$F = \sigma L$$

This is demonstrated in the example below which is taken directly from the student guide:

**Example**

**Question**: Consider a droplet of water falling through the air. Calculate the internal pressure as the drop as it rises to counteract the surface contraction arising due to surface tension.

**Answer:** The net inward forces arise due to surface tension acting over the perimeter of the drop.

$$F_{\mathrm{IN}} = 2\pi r\sigma$$

This must be balanced by the pressure exerted outwards by the fluid acting on the surface area of the droplet, hence:

$$F_{\mathrm{OUT}} = \pi r^2 P$$

At mechanical equilibrium:

$$F_{\mathrm{IN}} = F_{\mathrm{OUT}}$$

$$\Rightarrow 2\pi r\sigma = \pi r^2 P$$

$$\therefore P = \frac{2\sigma}{r} \tag{4.1.10}$$

**API**

The implementation for all functions used to calculate the bubble pressure of a spherical droplet can be found in `Course Material/ Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 1/ 01eBubPres.c`. The declarations of the following functions can be found in `01eBubPres.h` at the same file path with the exception of '`BubblePressure (void)`' which can be found in `B48BB_T1.h`.

`void BubPresVariable(double *sigma, double *r)`

This subroutine is used for collecting the data required for calculating the pressure required to form a bubble of set radius.

`double BubPresCalculation(double sigma, double r)`

This subroutine is used to calculate the bubble pressure using equation 4.1.10. After calculation, this subroutine returns the pressure required to form the given bubble.

`void BubPresDisplay(double sigma, double r, double P)`

This subroutine is used to output the results from bubble pressure calculation to the user console.

`void BubPresWrite(double sigma, double r, double P)`

This subroutine is used to write the results from bubble pressure calculation to a .txt file.

`void BubPresWriteSwitch(double sigma, double r, double P)`

Subroutine to ask the user if they would like to save the results of this program to a file.

`void BubblePressure(void)`

This subroutine guides the user through gathering the data and calculation of bubble pressure using the fluid surface tension.

### 4.1.5 Contact angle and Wettability

Young's equation [8] allows for the interfacial tensions between the three states to exhibit a physical observation known as the contact angle.:

[Insert figure]

$$\sigma_{\text{sg}} = \sigma_{\text{sl}} + \sigma_{\text{lg}} \cos\theta_c \tag{4.1.11}$$

Where:

- $\sigma_{\text{sg}}$ = Surface free energy of the solid [9]
- $\sigma_{\text{sl}}$ = Interfacial tension between liquid and solid [10].
- $\sigma_{\text{lg}}$ = Surface tension of the liquid (See above).
- $\phi$ = Contact angle.

The contact angle is representative of the strength of adhesive forces between the states present. Although the surface tension falls out of scope of this course, a starting point for how one may go about predicting the contact angle has been shown in the references above. Young's equation shows that the adhesive force between the liquid and solid and cohesive forces within the liquid phase control the equilibrium contact angle.

### 4.1.6 du Nouy Ring Method

The du Nouy ring method uses equation 4.1.9 to calculate the surface tension by finding the force required to break the liquid surface from below. This gives the following design equation:

$$\sigma = \frac{(C_F)F}{2L\cos\theta_c} \tag{4.1.12}$$

Table 4.1.1: Contact angle and wettability

| Contact angle | Wettability | Adhesive forces | Cohesive forces |
|---|---|---|---|
| $\theta_c = 0$ | Perfect wetting | Very strong | Weak |
| $0 < \theta_c < 90$ | High wettability | Strong | Strong |
| $90 < \theta_c < 180$ | High non-wettability | Weak | Strong |
| $\theta_c = 180$ | Perfect non-wetting | Very weak | Strong |

Where:

- $C_F$ = Correction factor.
- $L$ = mean circumference of the ring (m).
- $\theta_c$ = Contact angle (degrees or radians).

There is an additional 2 placed on the denominator as the surface tension will 'hold onto' both the top and bottom of the ring causing the surface tension to affect 2 surfaces through the contact angle.

### API

The implementation for all functions used to calculate the Surface tension using the du Nouy ring method can be found in `Course Material/ Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 1/ 01dSurfTens.c`. The declarations of the following functions can be found in `01dSurfTens.h` at the same file path with the exception of '`SurfaceTension (void)`' which can be found in `B48BB_T1.h`.

```
void wettabilityfacts(double cang)
```

This subroutine is used to provide dialogue on fluid characteristics based on the contact angle of the fluid with a solid, horizontal surface.

```
void duNouyVariable(double *F, double *L, double *C_F, double *cang)
```

This subroutine is used for collecting the data required for calculating the surface tension using the du Nouy ring method (Equation 4.1.12). This functions calls "`wettabilityfacts(...)`" to provide facts about the fluid wettability during data entry.

```
double duNouyCalculation(double F, double L, double C_F, double cang)
```

This subroutine is used to calculate the surface tension given the following variables. The function, after calculation steps, returns the value of sigma or surface tension in N/m.

```
void duNouyDisplay(double F, double L, double C_F, double cang, double sigma)
```

This subroutine is used to output the results of calculating the surface tension through the du Nouy Ring method to the user console.

```
void duNouyWrite(double F, double L, double C_F, double cang, double sigma)
```

This subroutine is used to write the results of calculating the surface tension through the du Nouy Ring method to a .txt file.

```
void duNouyWriteSwitch(double F, double L, double C_F, double cang, double sigma)
```

This subroutine is used to ask the user if they would like to save the results of this program to a file.

```
void SurfaceTension(void)
```

This subroutine guides the user through gathering the data and calculation of the force required to break a fluid's surface through the Du Nouy ring method.

### 4.1.7 Pendent Drop Method

Another method used to calculate the surface tension of a liquid is the pendent drop method. This method involves forming a droplet and inferring the surface tension from the size of the droplet just before the droplet breaks. This method can be argued to be advantageous over the du Nouy Ring method as it requires less set-up in addition to errors being accrued from measuring the droplet diameter rather than errors in the ring circumference.

### 4.1.8 Capillarity

Another phenomenon arising from surface tension is capillarity. This is when the free liquid level is raised (or lowered, in the case of a non-wetting fluid) to a certain height. The terminology used to describe this behaviour is:

- Capillary pressure, $P_C$, shows the pressure required to push liquid into the tube.
- Capillary rise, $h$, is the height to which the fluid is raised into the capillary tube.

[Insert figure]

As detailed in equation 4.1.9, the force exerted partially upwards by surface tension is:

$$F = 2\pi r\sigma$$

Since this force acts through the contact angle, the total force upwards must be:

$$F(\uparrow) = 2\pi r\sigma\cos\theta$$

Similarly the force downwards would be the weight:

$$F(\downarrow) = mg = \rho V_c g = \rho(\pi r^2 h)g$$

Since this system must eventually reach mechanical equilibrium:

$$\pi r^2 h\rho g = 2\pi r\sigma\cos\theta_c$$

$$h = \frac{2\sigma\cos\theta_c}{\rho g r} = \frac{4\sigma\cos\theta_c}{\rho g d} \tag{4.1.13}$$

This can be used to calculate the capillary pressure by substituting into equation **??**. Hence:

$$P_c = \rho g h \tag{4.1.14}$$

This can otherwise be the difference between the air pressure and the water pressure on either side of the meniscus. It is important to note that as you descend to the free water level, the pressure will once again be set to 0. Since the wettability of the fluid is a key parameter, there are three possible situations:

1. $0 < \theta_c < 90$: Fluid is wetting so adhesive forces are strong and good affinity between the fluid and the capillary tube. Capillary rise will be observed with a lower meniscus.

2. $\theta_c = 0$: Fluid is neither wetting or non-wetting. Neither capillary rise nor fall will be observed.

3. $90 < \theta_c < 180$: Fluid is non-wetting so adhesive forces are weak, thus there is a poor affinity between the fluid and the capillary tube. Capillary fall will be observed with an upper meniscus forming.

In case 3, we observe an upper meniscus thus allowing for a negative capillary pressure. This introduces the concept of "entry pressure" which must be exceeded before mercury can enter the tube.

### API

The implementation for all functions used to estimate capillarity effects can be found in `Course Material/ Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 1 / 01fCapp.c`. The declarations of the following functions can be found in `01fCapp.h` at the same file path with the exception of '`Capillarity(void)`' which can be found in `B48BB_T1.h`.

```
void CappVariable(double *sigma, double *cang, double *rho, double *d)
```

This subroutine is used to collect the data required to determine the capillary rise and pressure.

```
double CappCalculateHeight(double sigma, double cang, double rho, double d)
```

This subroutine is used to calculate the capillary rise using equation 4.1.13. This subroutine, after calculation, returns the capillary rise (m) to the calling function.

```
double CappCalculatePressure(double sigma, double cang, double d)
```

This subroutine is used to calculate the capillary pressure using equation 4.1.14. This subroutine, after calculation, returns the capillary pressure (Pa) to the calling function.

```
void CappDisplay(double sigma, double cang, double d, double h, double Pc)
```

This subroutine is used to output the results of capillary rise and pressure from capillarity calculations to the user console.

```
void CappWrite(double sigma, double cang, double d, double h, double Pc)
```

This subroutine is used to output the results of capillary rise and pressure from capillarity calculations to a .txt file.

```
void CappWriteSwitch(double sigma, double cang, double d, double h, double Pc)
```

This subroutine is used to ask the user if they would like to save the results of this program to a file.

```
void Cappilarity(void)
```

This subroutine guides the user through gathering the data and calculation of capillarity effects from surface tension data.

## 4.2 Incompressible fluid dynamics

This chapter is concerned with the study of how we can simulate the movement of fluids within a computer and make reliable predictions of fluid behaviour from both theoretical and empirical observations. The first principles for such is the first law and Bernoulli's principle. Although another principle that forms the basis of the following calculations can also be hydrostatic pressure theorems (See section 4.1.2), these predict negligible pressure losses across a horizontal element from the outset but closer examination of the theory finds that the equations only hold true when the fluid is at rest. This then has implications on a moving fluid element such that these equations mostly do no longer apply.

### 4.2.1 Principles

#### Steady flow energy equation

The energy content of a fluid is defined as the sum of its internal energy, kinetic energy and potential energy. This can be summarised as follows:

$$e_{i,u} + \frac{1}{2}u_i^2 + Z_i g = \text{constant} \tag{4.2.1}$$

This equation has the limitation that it does not account for the force required to push it through the pipework. The flow work can now be defined as:

$$\text{Flow Work} = (P_1 A_1)\left(\frac{1}{\rho_1 A_1}\right) = \frac{P_1}{\rho_1}\left\{\frac{\text{kN}}{\text{m}^2}.\text{m}^2\frac{\text{m}^3}{\text{kg}}\frac{1}{\text{m}^2} = \frac{\text{kN.m}}{\text{kg}}\right.$$

This allows equation 4.2.1 to be rewritten as:

$$e_{i,u} + \frac{P_i}{\rho_i} + \frac{1}{2}u_i^2 + Z_i g = \text{constant}$$

The fluid enthalpy is defined as:

$$h_i = e_{i,u} + \frac{P_i}{\rho_i}$$

Substituting in the fluid enthalpy and allowing for heat and work to occur, the equation becomes a statement of the first law:

$$h_{1,i} + \frac{1}{2}u_1^2 + Z_1 g + (q_{12}) = h_{2,i} + \frac{1}{2}u_2^2 + Z_2 g + (-w_{12}) \tag{4.2.2}$$

$$q_{12} + w_{12} = (h_{2,i} - h_{1,i}) + \frac{1}{2}(u_2^2 - u_1^2) + g(Z_2 - Z_1) \tag{4.2.3}$$

The above equation does however have a disadvantage that it does limit itself in terms of requiring no physical parameters from the process explicitly.

### API

The implementation for all functions used to evaluate steady-state behaviour using the first law of thermodynamics can be found in `Course Material/ Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 2/ 02bEnerCon.c`. The declarations of the following functions can be found in `02bEnerCon.h` at the same file path with the exception of '`EnergyConservation(void)`' which can be found in `B48BB_T2.h`.

```
void EnerConVariable(double *h1, double *h2, double *u1, double *u2, double *z1, double
*z2, double *q, double *w)
```

This subroutine is used for collecting the data required to validate the steady-flow energy equation.

```
double EnthalpyConversion(double u, double P, double rho)
```

This subroutine is used to convert between internal energy and enthalpy. After calculation, the subroutine returns the value of enthalpy to the calling function.

```
double EnerConFluidCalculation(double h, double u, double z)
```

This subroutine used to calculate the fluid intensive portion at a process time state of equation 4.2.3. The subroutine returns the energy content of the fluid after some manipulation of the required arguments.

```
double EnerConProcessCalculation(double q, double w)
```

This subroutine is used to determine the total energy input into the open system using equation 4.2.3. The subroutine, after calculation, returns to sum of the heat and work inputted into the system.

```
void EnerConDisplay(double h1, double h2, double u1, double u2, double z1, double z2,
 double q, double w, double state1, double state2, double process, double check)
```

This subroutine is used to output the results of this program to the user console.

```
void EnerConWrite(double h1, double h2, double u1, double u2, double z1, double z2,
 double q, double w, double state1, double state2, double process, double check)
```

This subroutine is used to save the results of this program to a .txt file.

```
void EnerConWriteSwitch(double h1, double h2, double u1, double u2, double z1, double z2,
double q, double w, double state1, double state2, double process, double check)
```

Subroutine to ask the user if they would like to save the results of this program to a file.

```
void EnergyConservation(void)
```

This subroutine guides the user through gathering the data and determining whether the system is at steady-state or not.

---

However, equation 4.2.3 is under the assumption that the flow rate is constant with respect to time. Since the only units for time are included within the velocity, this leads us to conclude that there must be some relationship between the two velocities that account for physical

parameters. Mathematically, the mass flowrate at any given point is:

$$\dot{m}_i = \rho_i A_i u_i \left\{ \frac{\text{kg}}{\text{m}^3} \text{m}^2 \frac{\text{m}}{\text{s}} = \frac{\text{kg}}{\text{s}} \right.$$

Comparing the mass flowrates at state 1 and state 2 leads to the following observation:

$$\dot{m}_1 = \dot{m}_2$$

$$\Rightarrow \rho_1 A_1 u_1 = \rho_2 A_2 u_2$$

This can now be rearranged to infer the value of state 2's velocity from the initial state's velocity.

$$u_2 = \frac{\rho_1 A_1 u_1}{\rho_2 A_2} \tag{4.2.4}$$

As will be explored shortly, a simplification can be made by assuming isothermal fluid flow which would cause the fluid densities in both the numerator and denominator to cancel out.

### API

The implementation for all functions used to estimate the final fluid velocity given a change in pipe diameter can be found in `Course Material/ Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 2/ 02aMassCon.c`. The declarations of the following functions can be found in `02aMassCon.h` at the same file path with the exception of '`MassConservation(void)`' which can be found in `B48BB_T2.h`.

```
void MassConVariable(double *rho1, double *rho2, double *d1, double *d2, double *u1)
```

This subroutine is used to collect the data required to determine the average fluid velocity, volumetric flowrate and mass flowrate.

```
double FinalVelocityCalculation(double u1, double d1, double d2)
```

This subroutine is used to calculate the final fluid velocity at some point past the initial measurement using equation 4.2.4. After calculation, this function returns the value of u2 to the calling function.

```
double VolumetricFlowCalculation(double u, double d)
```

This subroutine is used to calculate the volumetric flow rate from the fluid velocity and pipe diameter. This calculated value is then returned to the calling function.

```
double MassFlowCalculation(double rho, double d, double u)
```

This subroutine is used to calculate the mass flow rate from the given arguments. This calculated value is then returned to the calling function.

```
void MassConDisplay(double rho1, double rho2, double d1, double d2, double u1, double u2,
double q1, double q2, double m1, double m2)
```

This subroutine is used to output the calculation results to the user console.

```
void MassConWrite(double rho1, double rho2, double d1, double d2, double u1, double u2,
double q1, double q2, double m1, double m2)
```

This subroutine is used to output the calculation results to a .txt file.

`void MassConWriteSwitch(double rho1, double rho2, double d1, double d2, double u1, double u2, double q1, double q2, double m1, double m2)`

This subroutine is used to ask the user if they would like to save the results of this program to a file.

`void MassConservation(void)`

This subroutine guides the user through gathering the data and calculation of volumetric and mass flow rates at a process endstate.

### Bernoulli's Equation

Starting from equation 4.2.1 modified for the flow work:

$$e_{1,u} + \frac{P_1}{\rho_1} + \frac{1}{2}u_1^2 + Z_1 g = e_{2,u} + \frac{P_2}{\rho_2} + \frac{1}{2}u_2^2 + Z_2 g$$

Assuming that the temperature remains constant allows the internal energy on either side of the equation to cancel. Similarly, density will also become constant since correlations calculate this value from state pressure and temperature with temperature being a major contribution to its value.

$$\frac{P_1}{\rho} + \frac{1}{2}u_1^2 + Z_1 g = \frac{P_2}{\rho} + \frac{1}{2}u_2^2 + Z_2 g \tag{4.2.5}$$

Explicitly, this equation has been derived for an adiabatic process and there must therefore be no net heat or work being done on or by the fluid. Since we also know that there will be some pressure losses between the two points due to friction, it is customary to add a frictional loss term:

$$\frac{P_1}{\rho} + \frac{1}{2}u_1^2 + Z_1 g = \frac{P_2}{\rho} + \frac{1}{2}u_2^2 + Z_2 g + E_f$$

Since the frictional energy loss is a hard concept to fully grasp, we can convert Bernoulli's equation into heads. In order, these are the static, dynamic and hydrostatic heads which can be found by dividing equation 4.2.5 by $g$ to get the following equation:

$$\frac{P_1}{\rho g} + \frac{1}{2g}u_1^2 + Z_1 = \frac{P_2}{\rho g} + \frac{1}{2g}u_2^2 + Z_2 + h_f$$

This equation can be used further to calculate the final pressure of a flowing fluid. As evidenced through the water hammer effect, the equation can also be utilised to estimate pressure losses through vertical pipe sections:

$$P_1 + \frac{\rho}{2}u_1^2 + \rho g Z_1 = P_2 + \frac{\rho}{2}u_2^2 + \rho g Z_2 + \Delta P_f$$

$$P_2 = \left(P_1 + \frac{\rho}{2}u_1^2 + \rho g Z_1\right) - \left(\frac{\rho}{2}u_2^2 + \rho g Z_2 + \Delta P_f\right)$$

A further simplification can be made by substituting the value of $u_2$ from equation 4.2.4:

$$P_2 = \left(P_1 + \frac{\rho}{2}u_1^2 + \rho g Z_1\right) - \left(\frac{\rho}{2}\left[\frac{A_1 u_1}{A_2}\right]^2 + \rho g Z_2 + \Delta P_f\right)$$

### API

The implementation for all functions used to predict the final fluid pressure through Bernoulli's equation can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 2/ 02cBernEqn.c`. The declarations of the following functions can be found in `02cBernEqn.h` at the same file path with the exception of '`BernoulliEquation(void)`' which can be found in `B48BB_T2.h`.

```
void BernEqnVariable(double *P1, double *rho, double *u1, double *u2, double *Z1, double
*Z2, double *hf)
```

This subroutine is used for data collection to calculate the end pressure through Bernoulli's equation. The water hammer effect is observed is u2 is set to 0 m/s.

```
double StaticHeadCalculation(double P, double rho)
```

This subroutine is used to calculate the static head of a fluid (m).

```
double DynamicHeadCalculation(double u)
```

This subroutine is used to calculate the dynamic head of a fluid (m).

```
double BernEqnCalculation(double stathead, double dynhead, double Z)
```

This subroutine is used to calculate the total head contribution of a fluid at a process time state (m) defined in equation 4.2.5.

```
void BernEqnDisplay(double P1, double P2, double rho, double u1, double u2, double z1,
double z2, double hf)
```

This subroutine is used to output the collected data and final result to the user console.

```
void BernEqnWrite(double P1, double P2, double rho, double u1, double u2, double z1, double
z2, double hf)
```

This subroutine is used to output the collected data and final result to a .txt file.

```
void BernEqnWriteSwitch(double P1, double P2, double rho, double u1, double u2, double
z1, double z2, double hf)
```

Subroutine to ask the user if they would like to save the results of this program to a file.

```
void BernoulliEquation(void)
```

This subroutine guides the user through gathering the data and calculation of the fluid pressure at a process time state through Bernoulli's equation.

### 4.2.2 Newton's Law of Viscosity

This law encompasses the resistive forces that a fluid will exert on an external force to allow the element to remain in mechanical equilibrium. It inherently relies on the assumption that at the quantum level, a fluid flows in distinct layers which flow over each other. This movement of fluid layers creates a viscous drag on the layer moving faster than the other.

To examine this theory, lets analyse two parallel layers of fluid where the one atop is moving incrementally faster than the layer beneath it. Since the top layer is moving faster, there will be a greater 'displacement' of that layer thus it will be displaced further along the general volume

than the layer beneath from some reference point. Mathematically, this displacement can be described as layer 1 being at a distance $y$ from the solid surface and layer 2 being a distance $y + \delta y$ from the same solid surface. Similarly, layer 1 will be travelling at a velocity $v_x$ with the second layer travelling at a velocity $v_x + \delta v_x$.

[Insert graphic]

As $\delta y \to 0$, layer 1 will exert a viscous drag on the layer atop which will hold back the affected layer from moving forward. Since molecular diffusion will also occur between the layers we can say that this is acting solely in the $y$ direction and make the following statement:

$$\text{Rate of Mass Diffusion} = k\frac{A}{\delta y}$$

Where $k$ is the constant of proportionality and $A$ is the area of the layer being examined. This rate can also be determined from the velocities of the travelling layers:

$$\text{Rate of Mass Diffusion}_{2\to1} = kA\frac{v_x + \delta v_x}{\delta y}$$
$$\text{Rate of Mass Diffusion}_{1\to2} = kA\frac{v_x}{\delta y}$$

This results in the overall rate for mass diffusion being:

$$\text{Rate of Mass Diffusion} = kA\frac{\delta v_x}{\delta y}$$

This can be argued to be an applied statement of Newton's second law of motion ($F = ma$) from the viewpoint of rate of change of momentum. This means that the above is equivalent to:

$$F = kA\frac{\delta v_x}{\delta y}$$

The shear stress is defined as:

$$\tau = \frac{F}{A}$$

The following simplification can then be made:

$$\tau = k\frac{\delta v_x}{\delta y}$$

Taking a limit to reduce the distance between fluid layers results in the following:

$$\tau = \lim_{\delta y \to 0} k\frac{\delta v_x}{\delta y} = k\frac{dv_x}{dy}$$

Assuming that shear stress is proportional to the shear rate (this assumption is the basic characteristic of a Newtonian fluid), this now allows us to equate the constant of proportionality to the dynamic viscosity (N.s/m$^2$). Similarly, the shear rate is defined as the change of velocity with respect to the distance between the layers of the examined fluid:

$$\frac{dv_x}{dy} = \dot{\gamma} = \text{shear rate}\left\{\frac{1}{\text{s}}\right.$$

This simplifies the shear stress being:

$$\tau = \mu\dot{\gamma}$$

The general observations can then be deduced from the above:

- Shear stress is the force per unit area required to displace some volume of fluid.
- Shear rate is the rate at which shear stress is being applied.
- Viscosity is the force needed to 'stir' a given fluid at a particular rate. Where the work required is the force multiplied over the displacement. Thus more work is required to stir a highly viscous fluid compared to a less viscous fluid.

**Shear stress variation in a horizontal circular pipe**

It was found earlier, in section 4.1.2, that the hydrostatic pressure theorem across a horizontal element is:

$$\leftrightarrow : \sum F = P_2 A - P_1 A = 0$$
$$\updownarrow : F = mg$$

For the horizontal contribution, an additional pressure loss term can be added to account for frictional losses due to internal and external forces. This additional term is encapsulated by the radial shear stress acting through the pipe cross-sectional area and total pipe length.

$$F_x = P_1(\pi r^2) = P_2(\pi r^2) + \tau_r(2\pi r)L$$

$$\therefore F_x = P_1(\pi r^2) - P_2(\pi r^2) - \tau_r(2\pi r)L$$

$$(\pi r^2)\Delta P = \tau_r(2\pi r)L$$

$$\Delta P = \frac{\tau_r(2\pi r)L}{\pi r^2} = \frac{2\tau_r L}{r} = \frac{4\tau_w L}{d}$$

Assuming that the pressure loss on a horizontal axis is solely affected by the shearing experienced by the travelling fluid:

$$\Delta P = \frac{2\tau_r L}{r}$$
$$\Rightarrow \frac{2\tau_r L}{r} = \frac{4\tau_w L}{d}$$
$$\tau_r = \frac{2\tau_w r}{d} \tag{4.2.6}$$

Remembering from 4.2.2 that:

$$\tau_r = -\mu\frac{dv}{dr}$$
$$-\mu\frac{dv}{dr} = \frac{2\tau_w r}{d}$$
$$\frac{dv}{dr} = -\frac{2\tau_w r}{\mu d}$$

$$\because \tau_w = \frac{d\Delta P}{4L}$$

$$\frac{dv}{dr} = -\frac{2r}{\mu d}\frac{d\Delta P}{4L} = \left(\frac{\Delta P}{L}\right)\frac{r}{2\mu} = \left(\frac{\Delta P}{L}\right)\frac{d}{4\mu}$$

This can be substituted into Newton's law of viscosity and applying a limit to the pipe wall.

$$\lim_{\tau_r \to \tau_w} \tau_r = \tau_w = \left(\frac{\Delta P}{L}\right)\frac{d}{4}$$

This then implies that:

$$\tau_r = \left(\frac{\Delta P}{L}\right)\frac{r}{2}$$

### 4.2.3 Reynolds Number

Experimental tests on fluid flow found that for a flowing fluid in a single phase, there are two distinct regimes that a fluid can take. These are laminar flow and turbulent flow. In order to quantify the regimes and make a judgment, the dimensionless quantity of the Reynolds number can be calculated which is simply the ratio of inertial fluid momentum forces to viscous forces:

$$\mathrm{Re} = \frac{\rho u d}{\mu} \tag{4.2.7}$$

The two regimes can now be classified as:

- Laminar flow (Re $< 2000$)
  - No macroscopic mixing perpendicular to flow direction.
  - Small amounts of radial microscopic diffusion
  - Individual fluid layers are able to slide smoothly past each other whilst remaining a separate identity through the pipe. This results in "glassy" flow.
  - A maximum fluid velocity can be found to occur along the pipe centreline and no velocity at the pipe wall due to the no-slip condition.
  - The velocity profile is parabolic.
- Turbulent flow (Re $< 3000$)
  - A buffer layer and laminar sub-layer encase a turbulent core.
  - Turbulent eddies, in the core, cause macroscopic mixing. This overwhelms the effect of microscopic mixing which is still present.
  - The turbulent core still possesses the bulk fluid velocity.

It is trivial to note that there is a discrepancy between the two flow regimes. This is known as the transition regime where the fluid possesses features of both laminar and turbulent flow and can switch between the regimes.

### API

The implementation for all functions used to calculate the Reynolds number can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 2/ 02dReyNo.c`. The declarations of the following functions can be found in `02dReyNo.h` at the same file path with the exception of '`ReynoldsNumber(void)`' which can be found in `B48BB_T2.h`.

`void ReyNoVariable(double *rho, double *u, double *d, double *mu)`

This subroutine is used to collect the data required to calculate the Reynolds number.

`double ReyNoCalculation(double rho, double u, double d, double mu)`

This subroutine is used to calculate the Reynolds number using equation 4.2.7.

`void ReyNoDisplay(double rho, double u, double d, double mu, double ReyNum)`

This subroutine is used to output the collected data and calculated Reynolds number to the user console.

`void ReyNoWrite(double rho, double u, double d, double mu, double ReyNum)`

This subroutine is used to output the collected data and calculated Reynolds number to a .txt file.

`void ReyNoWriteSwitch(double rho, double u, double d, double mu, double ReyNum)`

This subroutine is used to ask the user if they would like to save the results of this program to a file.

`void ReynoldsNumber(void)`

This subroutine guides the user through gathering the data and calculation of Reynold's number for a fluid

---

In the case of non-circular pipes, it would be inappropriate to utilise the above equation. In which case, it is beneficial to find the hydraulic diameter which is essentially the diameter of the equivalent circle to the non-circular pipe. To calculate this, the hydraulic diameter is given as:

$$d_H = \frac{4 \times A_F}{P_W} \tag{4.2.8}$$

Where $A_F$ is the flow area and $P_W$ is the wetted perimeter. It should be noted that this equation is inherently only able to calculate the hydraulic diameter for a single continuous phase and is able to deal with partial filling of a form.

### API

The implementation for all functions used to calculate the hydraulic diameter can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 3/ 03cHydrDiam.c`. The declarations of the following functions can be found in `03cHydrDiam.h` at the same file path with the exception of '`HydraulicDiameter(void)`' which can be found in `B48BB_T3.h`.

`void HydrDiamVariable(double *A_F, double *P_W)`

This subroutine is used for collecting the data required to calculate the hydraulic diameter of a non-cylindrical pipe.

```
double HydrDiamCalculation(double A_F, double P_W)
```

This subroutine is used to calculate the hydraulic diameter of a non-cylindrical pipe using equation 4.2.8.

```
void HydrDiamDisplay(double A_F, double P_W, double d_H)
```

This subroutine is used to display the collected data and the calculated hydraulic diameter to the user console.

```
void HydrDiamWrite(double A_F, double P_W, double d_H)
```

This subroutine is used to write the collected data and the calculated hydraulic diameter to a .txt file.

```
void HydrDiamWriteSwitch(double A_F, double P_W, double d_H)
```

Subroutine to ask the user if they would like to save the results of this program to a file.

```
void HydraulicDiameter(void)
```

This subroutine guides the user through gathering the data and calculation of hydraulic diameter for a non-circular pipe.

### 4.2.4 Velocity distribution for a fluid flowing through a cylindrical pipe

#### Laminar flow velocity distribution

Newton's law of viscosity (Sec. 4.2.2) states that:

$$\tau = \mu \frac{dv_x}{dy}$$

Taking a limit to the pipe radius:

$$\lim_{y \to r} \tau = \mu \frac{dv_x}{dr}$$

Since the velocity is greatest at the centreline due to this being the area where shear stresses are at a minimum and zero at the wall due to the no-slip condition, this allows us to infer that the velocity will decrease as the point radius increases. This then implies that Newton's law of viscosity can be modified to account for this observation:

$$\tau_r = -mu \frac{dv_x}{dr}$$

It was shown earlier that:

$$\Delta P = \frac{4 \tau_w L}{d}$$

Assuming that there is no radial variation of shear stress:

$$\lim_{\tau_w \to \tau_r} \Delta P = \frac{4 \tau_r L}{d}$$

$$\Delta P = -\frac{4 \left( mu \frac{dv_x}{dr} \right) L}{d}$$

$$\therefore \frac{dv_x}{dr} = -\left(\frac{\Delta P}{L}\right)\frac{d}{4\mu} = -\left(\frac{\Delta P}{L}\right)\frac{r}{2\mu}$$

$$v_x = -\left(\frac{\Delta P}{L}\right)\frac{d}{4\mu} = -\left(\frac{\Delta P}{L}\right)\frac{1}{2\mu}\int r dr$$

$$v_x = -\left(\frac{\Delta P}{L}\right)\frac{1}{4\mu}r^2 + C$$

At the wall, due to the no-slip condition, $v_x = 0$:

$$\therefore C = \left(\frac{\Delta P}{L}\right)\frac{1}{4\mu}r^2$$

Since this is occurring at the wall, it makes sense to change the pipe radius to the pipe diameter:

$$C = \left(\frac{\Delta P}{L}\right)\frac{1}{4\mu}\left(\frac{d}{2}\right)^2$$

$$\therefore v_x = -\left(\frac{\Delta P}{L}\right)\frac{1}{4\mu}r^2 + \left(\frac{\Delta P}{L}\right)\frac{1}{16\mu}d^2$$

$$v_x = \left(\frac{\Delta P}{L}\right)\frac{1}{16\mu}d^2 - \left(\frac{\Delta P}{L}\right)\frac{4}{16\mu}r^2$$

$$v_x = \left(\frac{\Delta P}{L}\right)\frac{d^2}{16\mu}\left[1 - \frac{4r^2}{d^2}\right]$$

$$\therefore v_x = \left(\frac{\Delta P}{L}\right)\frac{d^2}{16\mu}\left[1 - \left(\frac{2r}{d}\right)^2\right] \tag{4.2.9}$$

A maximum velocity can be found by substituting $r = 0|_{v_x = v_{\max}}$:

$$\therefore v_{\max} = \left(\frac{\Delta P}{L}\right)\frac{d^2}{16\mu}$$

This now allows us to calculate the general velocity profile to be:

$$\frac{v_x}{v_{\max}} = \left[1 - \left(\frac{2r}{d}\right)^2\right] \tag{4.2.10}$$

To find the location of average fluid velocity ($u$), we can use the volumetric flow rate ($Q$) definition:

$$dQ = (2\pi r)v dr$$

Expanding on the definition of the point velocity (Equation 4.2.9):

$$\therefore Q = 2\pi\left(\frac{\Delta P}{L}\right)\frac{d^2}{16\mu}\int_0^{d/2}\left[1 - \left(\frac{2r}{d}\right)^2\right] r dr$$

$$Q = \left(\frac{\Delta P}{L}\right)\frac{\pi d^2}{8\mu}\int_0^{d/2}\left[r - \frac{4r^3}{d^2}\right] dr$$

$$Q = \left(\frac{\Delta P}{L}\right)\frac{\pi d^2}{8\mu}\left[\frac{r^2}{2} - \frac{4r^4}{4d^2}\right]_0^{d/2} = \left(\frac{\Delta P}{L}\right)\frac{\pi d^2}{8\mu}\left[\frac{r^2}{2} - \frac{r^4}{d^2}\right]_0^{d/2}$$

$$Q = \left(\frac{\Delta P}{L}\right)\frac{\pi d^2}{8\mu}\left[\frac{d^2}{8} - \frac{d^4}{16d^2}\right] = \left(\frac{\Delta P}{L}\right)\frac{\pi d^2}{8\mu}\left[\frac{d^2}{8} - \frac{d^2}{16}\right]$$

$$Q = \left(\frac{\Delta P}{L}\right)\frac{\pi d^2}{8\mu}\frac{d^2}{16}$$

$$Q = \left(\frac{\Delta P}{L}\right)\frac{\pi d^4}{128\mu} \tag{4.2.11}$$

$$\because Q = u\left(\frac{\pi d^2}{4}\right)$$

$$u = \left(\frac{\Delta P}{L}\right)\frac{d^2}{32\mu}$$

The maximum fluid velocity was earlier derived as:

$$v_{\max} = \left(\frac{\Delta P}{L}\right)\frac{d^2}{16\mu}$$

$$\therefore \frac{u}{v_{\max}} = \frac{16}{32} = 0.5$$

### API

The implementation for all functions used to generate the velocity profile for a fluid with laminar flow properties can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 2/ 02g1LamVelPro.c`. The declarations of the following functions can be found in `02g1LamVelPro.h` at the same file path.

```
void LamVelProVariable(double *dP, double *L, double *d, double *mu)
```

This subroutine is used to collect the data required for calculating the velocity profile of a fluid flowing with laminar flow characteristics.

```
double LamVelCalculation(double dP, double L, double d, double mu, double r)
```

This subroutine is used to calculate the point velocity at some radius in a cylindrical pipe segment using equation 4.2.9.

```
double LamVelGeneralCalculation(double r, double d)
```

This subroutine is used to calculate the general velocity profile from the point radius and fixed pipe diameter using equation 4.2.10. This subroutine returns the value of $v/v_{max}$ to the calling function.

```
LamVelProf LamVelProfCalculation(double dP, double L, double d, double mu, int *rows)
```

This subroutine is used to calculate the fully developed velocity profile for a fluid flowing with laminar properties. This subroutine returns the generated array to the calling function.

```
void LamVelProDisplay(double dP, double L, double d, double mu, int rows, LamVelProf
 profile)
```

This subroutine is used to write the collected data and generated velocity profile to the user console.

```
void LamVelProWrite(double dP, double L, double d, double mu, int rows, LamVelProf
```

```
profile)
```

This subroutine is used to write the collected data and generated velocity profile to a .txt file.

```
void LamVelProSwitch(int mode, double dP, double L, double d, double mu, int rows,
 LamVelProf profile)
```

Subroutine to ask the user if they would like to either display the results on the console or save the results of this program to a file.

```
void LaminarVelPro(void)
```

This is the main subroutine controlling the behaviour of the subroutines listed above.

### Turbulent flow velocity distribution

To determine the velocity profile for a turbulent flow, analysis is complicated due to eddies causing stronger radial mixing. This was generalised through Prandtl's one-seventh law who found that these effects resulted in a flatter velocity profile in comparison to the laminar profile (Eqn. 4.2.10):

$$\frac{v}{v_{\max}} = \left(\frac{y}{R}\right)^{1/7} \tag{4.2.12}$$

Where $y = R - r$, $R$ is the fixed pipe radius and $r$ is the point radius. As stated before:

$$dQ = 2\pi r v dr$$

In terms of Prandtl's one-seventh law:

$$dQ = -2\pi(R-y)v dy$$

$$\because v = v_{\max}\left(\frac{y}{R}\right)^{1/7}$$

$$\therefore dQ = -2\pi(R-y)\left[v_{\max}\left(\frac{y}{R}\right)^{1/7}\right]dy$$

$$dQ = -2\pi v_{\max}\int_R^0 (R-y)\left(\frac{y}{R}\right)^{1/7} dy = -2\pi v_{\max}\int_R^0 y^{1/7}R^{6/7} - \frac{y^{8/7}}{R^{1/7}}dy$$

$$Q = -2\pi v_{\max}\left[\frac{7}{8}y^{8/7}R^{6/7} - \frac{7}{15R^{1/7}}y^{15/7}\right]_R^0$$

$$Q = -2\pi v_{\max}\left[\frac{7}{8}R^{8/7}R^{6/7} - \frac{7}{15R^{1/7}}R^{15/7}\right]$$

$$Q = -2\pi v_{\max}R^2\left[\frac{7}{8} - \frac{7}{15}\right] = -2\pi v_{\max}R^2\left[\frac{98}{120}\right]$$

$$Q = \frac{49}{60}\pi v_{\max}R^2$$

$$\therefore u = \frac{49}{60}v_{\max}$$

Enough information has now been given to calculate the location of average velocity for turbulent

flow.

$$\frac{u}{v} = 1 = \frac{49}{60}\left(\frac{(R-r)}{R}\right)^{-1/7}$$

$$1 = \frac{60}{49}\left(\frac{(R-r)}{R}\right)^{1/7}$$

$$1 = \left(\frac{60}{49}\right)^{7}\frac{(R-r)}{R}$$

$$\frac{(R-r)}{R} = 1 - \left(\frac{60}{49}\right)^{7}$$

$$\therefore r = R\left[1 - \left(\frac{60}{49}\right)^{7}\right]$$

### API

The implementation for all functions used to generate the velocity profile for a fluid with turbulent flow properties can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 2/ 02g2TurVelPro.c`. The declarations of the following functions can be found in `02g2TurVelPro.h` at the same file path.

```
void TurVelProVariable(double *umax, double *d)
```

This subroutine is used for the data collection of the turbulent flow velocity profile using Prandtl's one-seventh law.

```
double TurVelCalculation(double vmax, double r, double d, double *gen)
```

This subroutine is used to determine the point velocity by utilising the relationship between the point velocity and maximum velocity given by Prandtl's one-seventh law (Equation 4.2.12).

```
TurVelProf TurVelProCalculation(double vmax, double d, int *rows)
```

This subroutine is used to generate the data for the velocity profile for a turbulent fluid that obeys Prandtl's one-seventh law. This subroutine returns the generated array to the calling function.

```
void TurVelProDisplay(double umax, double d, int rows, TurVelProf profile)
```

This subroutine is used to output the results of this program to the user console.

```
void TurVelProWrite(double umax, double d, int rows, TurVelProf profile)
```

This subroutine is used to output the results of this program to a .txt file.

```
void TurVelProSwitch(int mode, double umax, double d, int rows, TurVelProf profile)
```

Subroutine to ask the user if they would like to either display the results on the console or save the results of this program to a file.

```
void TurbulentVelPro(void)
```

This pseudomain subroutine is used to control the behaviour of subroutines shown above.

## 4.3 Frictional pressure loss

### 4.3.1 General equation

Starting from equation 4.2.11 and assuming that all pressure losses arise from frictional pressure losses:

$$Q = \left(\frac{\Delta P_f}{L}\right)\frac{\pi d^4}{128\mu}$$

$$\Delta P_f = \frac{128QL\mu}{\pi d^4}$$

$$\because Q = u\left(\frac{\pi d^2}{4}\right)$$

$$\therefore \Delta P_f = \frac{128u\pi d^2 L\mu}{4\pi d^4} = \frac{32u\mu L}{d^2} \tag{4.3.1}$$

The above is otherwise known as the Hagen-Poiseuille equation and allows us to calculate the fluid pressure loss directly from the pipe geometry and fluid properties for a fluid flowing with the laminar regime. The equation also tells us that the absolute roughness of the pipe is not a factor of the frictional pressure losses.

---

**API**

The implementation for all functions used to calculate the Reynolds number can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 2/ 02hHagPois.c`. The declarations of the following functions can be found in `02hHagPois.h` at the same file path with the exception of '`HagenPoiseuille(void)`' which can be found in `B48BB_T2.h`.

```
void HagPoisVariable(double *u, double *mu, double *L, double *d)
```

This subroutine is used to collect the data required to calculate the pressure loss of a fluid flowing with laminar properties through the Hagen-Poiseuille equation.

```
double HagPoisCalculation(double u, double mu, double L, double d)
```

This subroutine is used to calculate the pressure loss of a fluid flowing with laminar properties using equation 4.3.1.

```
void HagPoisDisplay(double u, double mu, double L, double d, double dP)
```

This subroutine is used to output the data collected and calculated pressure loss to the user console.

```
void HagPoisWrite(double u, double mu, double L, double d, double dP)
```

This subroutine is used to write the data collected and calculated pressure loss to a .txt file.

```
void HagPoisWriteSwitch(double u, double mu, double L, double d, double dP)
```

This subroutine is used to ask the user if they would like to save the results of this program to a file.

```
void HagenPoiseuille(void)
```

This subroutine guides the user through gathering the data and calculation of the pressure losses incurred by a fluid flowing with fully developed laminar flow properties.

---

It was shown earlier in section 4.2.2 that the frictional pressure losses associated with a moving fluid has the following relation with the shear stress at the wall:

$$\Delta P = \frac{4\tau_w L}{d}$$

Comparing the above to equation 4.3.1, we find that the shear stress at the wall is equivalent to:

$$\tau_w = \frac{8u\mu}{d}$$

At this point, it would be unreasonable to assume that the same statement will also apply to turbulent flow. However, as an initial approximation, the pressure loss must be related to at least the point velocity. The research area of turbulent viscosity may allow this statement to apply, however viscosity refers to the shearing between fluid layers so it cannot be this as in turbulent flow these layers are not so well defined. However, we can assume that pressure losses must have some relation to the moment that the fluid will exert on the pipe wall as it travels due to Reynolds number. So multiplying through the pressure loss expression by unity allows us to make the following statement.

$$\because 1 = \frac{(\rho u^2)/2}{(\rho u^2)/2}$$

$$\Delta P = 8\left(\frac{\tau_w}{\rho u^2}\right)\left(\frac{L}{d}\right)\frac{\rho u^2}{2}$$

The first bracketed term on the right hand side is otherwise known as the friction factor which Stanton and Pannell have found to be a function of the Reynolds number and relative pipe roughness.

$$\Delta P = 8\phi\left(\frac{L}{d}\right)\frac{\rho u^2}{2} \tag{4.3.2}$$

It may be required such that the head loss is required by calculations. In terms of head loss, equation 4.3.2 can be transformed to the following expression:

$$h_f = 8\phi\left(\frac{L}{d}\right)\frac{u^2}{2g}$$

### API

The implementation for all functions used to calculate the fluid frictional pressure loss can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics) / Topic 3/ 03bGenPressureLoss.c`. The declarations of the following functions can be found in `03bGenPressureLoss.h` at the same file path with the exception of '`GeneralPressureLoss(void)`' which can be found in `B48BB_T3.h`.

```
void PressLossVariable(double *rho, double *u, double *d, double *mu, double *L, double
```

*vareps)

This subroutine is used to collect the data required for estimating the pressure loss of any incompressible fluid generally flowing within the range of 0 ¡ Re ¡ 10,000,000.

`double phiCalculation(double rho, double u, double d, double mu, double vareps)`

This subroutine is used to calculate the friction factor dependent on the Reynolds number and absolute roughness of the pipe using the equations listed later in this section. After finding the friction factor, the value is returned to the calling function.

`double LossCalculation(double phi, double L, double d, double rho, double u)`

This subroutine is used to estimate the frictional pressure loss using equation 4.3.2. This subroutine then returns the pressure loss to the calling function.

`void PressLossDisplay(double rho, double u, double d, double mu, double L, double vareps, double phi, double dP)`

Subroutine used to write the collected data and calculated values for pressure loss and friction factor to the user console.

`void PressLossWrite(double rho, double u, double d, double mu, double L, double vareps, double phi, double dP)`

Subroutine used to write the collected data and calculated values for pressure loss and friction factor to a .txt file.

`void PressLossWriteSwitch(double rho, double u, double d, double mu, double L, double vareps, double phi, double dP)`

Subroutine to ask the user if they would like to save the results of this program to a file.

`void GeneralPressureLoss(void)`

This subroutine guides the user through gathering the data and calculation of pressure losses for a moving fluid between Reynold's number of 0 ¡ Re ¡= 10,000,000.

### The friction factor, $\phi$

Across literature there have been two reported friction factors. These are the Fanning and Moody friction factors. In relation to $\phi$:

- Fanning friction factor: $f = 2\phi$
- Moody friction factor: $f' = 8\phi$

Within simulation, it is often preferable to be able to utilise correlations to determine a near accurate value for the friction factor rather than refer to graphs for data values. Fortunately, these have been tabulated throughout literature and have been found to be as follows:

- Laminar flow (Re $<$ 2000).

This means that the Hagen-Poiseuille equation applies (Eqn. 4.3.1. This means that we can extract the value of the friction factor from such. Therefore, equating the pressure loss equation with the Hagen-Poiseuille equation results in the following:

$$8\phi \left(\frac{L}{d}\right) \frac{\rho u^2}{2} = \frac{32 u \mu L}{d^2}$$

$$\phi \frac{\rho u}{2} = \frac{4\mu}{d}$$
$$\therefore \phi = \frac{8\mu}{\rho u d} = \frac{8}{\mathrm{Re}}$$

- Turbulent flow ($\mathrm{Re} \geq 3000$).
  - $2500 < \mathrm{Re} < 100{,}000$: $\phi = 0.0396\mathrm{Re}^{-0.25}$
  - $2500 < \mathrm{Re} < 10{,}000{,}000$: $\phi^{-0.5} = 2.5 \ln\left(\mathrm{Re}\phi^{0.5}\right) + 0.3$

For turbulent flow, it has been found that the friction factor is dependent on the Reynolds number and relative roughness. To account for this additional contribution, Blasius has developed two correlations to describe this behaviour shown above.

- Turbulent flow ($\mathrm{Re} > 3000$) and dependence on relative roughness.

$$\phi^{-0.5} = -2.5 \ln\left(0.27\frac{\varepsilon}{d} + \frac{0.885}{\mathrm{Re}\phi^{0.5}}\right)$$

In this region, the laminar sub-layer is thick enough to envelope the absolute surface roughness such that the pipe becomes hydraulically smooth. This means that rough pipe wall elements protrude through the laminar sub-layer is superimposed on fluid skin friction causing additional drag.

- Turbulent flow ($\mathrm{Re} > 3000$) and independence of relative roughness.

In this region, the laminar sublayer is not thick enough to fully envelope the pipe surface roughness causing additional frictional losses.

$$\phi^{-0.5} = 3.2 - 2.5 \ln \frac{\varepsilon}{d}$$

---

### API

The implementation for all functions used to calculate the friction factor can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 3/ 03aFrictFactor.c`. The declarations of the following functions can be found in `03aFrictFactor.h` at the same file path. Note that all the subroutines available below are not explicitly available to the user at run time and are accessed through calculating the pressure losses through the general frictional pressure loss equation (Equation 4.3.2). The friction factors for turbulent flow are listed in the subroutine declarations in the same order as above.

```
double Laminar(double rho, double u, double d, double mu)
```

This subroutine is used to calculate the friction factor associated with a fluid flowing with laminar flow properties.

```
double Turbulent1(double rho, double u, double d, double mu)
```

This subroutine is used to calculate the friction factor associated with the pressure loss of a fluid with a turbulent flow profile. This equation is valid between $2500 < \mathrm{Re} < 100{,}000$.

`double Turbulent2(double rho, double u, double d, double mu)`

This subroutine is used to calculate the friction factor associated with the pressure loss of a fluid with a turbulent flow profile through a brute force iteration scheme. This equation is valid when $2500 < \text{Re} < 10{,}000{,}000$.

`double Turbulent3(double rho, double u, double d, double mu, double vareps)`

This subroutine is used to calculate the friction factor associated with the pressure loss of a fluid with a turbulent flow profile through a brute force iteration scheme. This equation is valid when $\text{Re} > 3000$ and the friction factor is dependent on relative roughness.

`double Turbulent4(double d, double vareps)`

This subroutine is used to calculate the friction factor associated with the pressure loss of a fluid with a turbulent flow profile. This equation is valid when $\text{Re} > 3000$ and the Friction Factor is independent of surface roughness.

---

It should be noted that in the transitionary region ($2000 \leq \text{Re} < 3000$) no reasonable pressure drop values can be obtained and is thus heavily reliant on the type of fluid and observed flow regime. In practise, when the friction factor does fall into one of the above regions, it is customary to calculate all possible values and utilise the highest value to give a worst case scenario pressure loss.

Typical absolute roughness values are shown in the table below:

Table 4.3.1: Typical absolute roughness values

| Pipe material | Absolute roughness, $\varepsilon$ (mm) |
|---|---|
| Drawn tube(e.g. Copper, Aluminium, PVC) | 0.0015 |
| Commercial steel pipe | 0.046 |
| Cast iron pipe | 0.26 |
| Galvanised iron | 0.15 |
| Concrete | 0.3 - 3 |

### 4.3.2 Pressure losses through pipe fittings

Within a pipe, pressure losses can occur from either skin frictional losses (arising from fluid viscosity) or form frictional losses (arising from pipe geometry). As stated earlier, skin frictional losses are dependent on fluid viscosity and thus arise from continual shearing of fluid as it moves. Form frictional losses arise when the flow path diverges or converges too quickly causing fluid to separate from the walls and resulting in turbulent eddies. This will therefore have consequences on internal fluid shearing forces as they adapt to different forms. Within pipes, form frictional losses is the dominant mechanism of energy loss in pipeline components.

To account for pressure losses through pipe fittings, there are two dominant methods that can be used. These are:

- Equivalent length method which equates pipe fittings to an equivalent length of straight pipe.
- Lost velocity heads method which equates pipe fittings to a resistance coefficient, $K$.

### Equivalent length method

The principle equation utilised within this method is similar to equation 4.3.2 with the length ($L$) term being replaced with the equivalent length ($L_e$):

$$\Delta P = 8\phi \left(\frac{L_e}{d}\right) \frac{\rho u^2}{2} \tag{4.3.3}$$

The parameters associated with pipe fittings is shown in the table below:

Table 4.3.2: Equivalent length constants

| Fitting | Number of pipe diameters, $L_e/d$ |
|---|---|
| Standard 45° Elbow | 15 |
| 90° elbow standard radius | 30 - 40 |
| 90° square elbow | 75 |
| Entry from T-piece | 60 |
| Entry into T-piece | 90 |
| Sudden reduction (Tank outlet) | 25 |
| Sudden expansion (Tank inlet) | 50 |
| Unions and Couplings | 2 |
| Globe valve (fully open) | 450 |
| Gate valve (100% open) | 7.5 |
| Gate valve (75% open) | 40 |
| Gate valve (50% open) | 200 |
| Gate valve (25% open) | 800 |
| Ball valve (100% open) | 18 |
| Plug valve (Open) | 18 |

### API

The implementation for all functions used to calculate the fluid frictional pressure loss through pipe fittings using the equivalent length method can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 3/ 03dEquivLeng.c`. The declarations of the following functions can be found in `03dEquivLeng.h` at the same file path.

```
EquivLenFits EquivLengData(EquivLenFits input)
```

This subroutine is used to write the equivalent length database constants to the struct where data is being collected to.

```
EquivLenFits EquivLengVariable(EquivLenFits data, double *rho, double *u, double *d,
 double *mu, double *vareps, double *phi)
```

This subroutine is used to collect the necessary data to calculate the pressure and head losses through pipe fittings through the equivalent length method.

```
double EquivLengCalculateL_e(double input, double d)
```

This subroutine is used to calculate the equivalent length from the database constant and internal pipe diameter.

```
double EquivLengCalculateLoss(int count, double phi, double L_e, double rho, double u,
double d)
```

This subroutine is used to calculate the pressure loss through a specific pipe fitting. This value is then multiplied by the multiple of the specific fitting.

```
EquivLenFits EquivLengFinalTable(EquivLenFits data, double rho, double u, double d, double phi)
```

This subroutine is used to generate the calculation table for estimating the pressure and head losses through the Equivalent length method.

```
void EquivLengDisplay(EquivLenFits table, double rho, double u, double d, double mu, double vareps, double phi, double totalP, double totalh)
```

This subroutine is used to present the collected data and calculated parameters on the console.

```
void EquivLengWrite(EquivLenFits table, double rho, double u, double d, double mu, double vareps, double phi, double totalP, double totalh)
```

This subroutine is used to write the collected data and calculated parameters to a .txt file.

```
void EquivLengWriteSwitch(EquivLenFits table, double rho, double u, double d, double mu, double vareps, double phi, double totalP, double totalh)
```

This subroutine is used to ask the user if they would like to write the results to a file.

```
void EquivalentLength(void)
```

This subroutine is used to guide the user through collecting the data and performing the calculation of pressure and head losses through the Equivalent Length method.

### Excess head methods

This method is also referred to as the lost velocity heads methods. As stated earlier, this method is reliant on the calculation of a resistance coefficient. Within this method, there are three ways of calculating this coefficient, aptly named the 1K, 2K and 3K methods.

- 1K Method

The principle equation for this method is given by:

$$h_f = k\frac{u^2}{2g} \tag{4.3.4}$$

The constants used to calculate the resistance coefficient has been provided as:

Table 4.3.3: 1K Resistance coefficient constants

| Fitting | Resistance coefficient, $k$ |
|---|---|
| Standard 45° Elbow | 0.35 |
| 90° elbow standard radius | 0.6 - 0.8 |
| 90° square elbow | 1.5 |
| Entry from T-piece | 1.2 |
| Entry into T-piece | 1.8 |
| Standard 45° Elbow | 0.35 |
| 90° elbow standard radius | 0.6 - 0.8 |
| 90° square elbow | 1.5 |
| Entry from T-piece | 1.2 |
| Entry into T-piece | 1.8 |

Table 4.3.4: 1K Resistance coefficient constants (Continued)

| Fitting | Resistance coefficient, $k$ |
|---|---|
| Sudden reduction (Tank outlet) | 0.5 |
| Sudden expansion (Tank inlet) | 1,0 |
| Unions and Couplings | 0.04 |
| Globe valve (fully open) | 6 |
| Gate valve (100% open) | 0.15 |
| Gate valve (75% open) | 1 |
| Gate valve (50% open) | 4 |
| Gate valve (25% open) | 16 |
| Ball valve (100% open) | 0.4 |
| Plug valve (Open) | 0.4 |

### API

The implementation for all functions used to calculate the fluid frictional pressure loss through pipe fittings using the 1K method can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 3/ 03dOneK.c`. The declarations of the following functions can be found in `03dOneK.h` at the same file path.

```
OneKFittings OneKData(OneKFittings input)
```

This subroutine is used to input K-values into the table being used by the "OneK(void)"

```
OneKFittings OneKVariable(OneKFittings table, double *u)
```

This subroutine is used to collect the fluid velocity and counts for each fitting available.

```
double OneKCalculation(int count, double data, double u)
```

This subroutine is used to calculate the head loss associated with one fitting using equation 4.3.4. The subroutine returns the total head loss for the singular fitting, multiplied by the count.

```
OneKFittings OneKFinalTable(OneKFittings data, double u)
```

This subroutine is used to perform the calculations required for the excess head/ 1K method of pressure losses through pipe fittings.

```
void OneKDisplay(OneKFittings table, double u, double total)
```

This subroutine is used to display the collected data and calculated table on the console.

```
void OneKWrite(OneKFittings table, double u, double total)
```

This subroutine is used to write the collected data and calculated table to a .txt file.

```
void OneKWriteSwitch(OneKFittings table, double u, double total)
```

This subroutine is used to check whether the user would like to write the generated results to a file.

```
void OneK(void)
```

This subroutine is used to guide the user through calculating the head loss associated with standard pipe fittings with the 1K method.

- 2K Method [11]

The principle equation for this method is given by:

$$K = \frac{K_1}{\text{Re}} + K_\infty \left(1 + \frac{1}{D}\right)$$
$$h_f = K \frac{u^2}{2g} \tag{4.3.5}$$

Where:

- $K_1$ is the resistance coefficient at Re = 1.
- $K_\infty$ is the resistance coefficient at Re = $\infty$.
- $D$ is the internal pipe diameter in inches.

The fitting-specific constants used to calculate the resistant coefficient are as follows:

Table 4.3.5: 2K Resistance coefficient constants

| Fitting | Type | $k_1$ | $k_\infty$ |
|---|---|---|---|
| 90° Curved elbow | Threaded, Standard radius (R/D = 1) | 800 | 0.4 |
| | Flanged/ Welded, Standard radius (R/D = 1) | 800 | 0.25 |
| | All types, Long radius (R/D = 1.5) | 800 | 0.2 |
| 90° Elbow mitered R/D = 1.5 | 1 Weld (90° Angle) | 1000 | 1.15 |
| | 2 Welds (90° Angle) | 800 | 0.35 |
| | 3 Welds (90° Angle) | 800 | 0.3 |
| | 4 Welds (90° Angle) | 800 | 0.27 |
| | 5 Welds (90° Angle) | 800 | 0.25 |
| 45° Elbow | All types, Standard radius (R/D = 1) | 500 | 0.2 |
| | All types, Long radius (R/D = 1.5) | 500 | 0.15 |
| 45° Elbow mitered | 1 Weld (45° Angle) | 500 | 0.25 |
| | 2 Welds (45° Angle) | 500 | 0.15 |
| 180° | Screwed, Standard radius (R/D = 1) | 1000 | 0.6 |
| | Screwed, Long radius | 1000 | 0.35 |
| | All types, Long radius (R/D = 1.5) | 1000 | 0.3 |
| Tee, used as elbow | Screwed, Standard radius (R/D = 1) | 500 | 0.7 |
| | Screwed, Long radius | 800 | 0.4 |
| | Flanged/ Welded, Standard radius (R/D = 1) | 800 | 0.8 |
| | Stub-in type branch | 1000 | 1 |
| Tee, run-through | Screwed | 200 | 0.1 |
| | Flanged/ Welded | 150 | 0.05 |
| | Stub-in type branch | 100 | 0 |
| Valves (Gate/ Ball/ Plug) | Full line size, $\beta = 1$ | 300 | 0.1 |
| | Reduced trim, $\beta = 0.9$ | 500 | 0.15 |
| | Reduced trim, $\beta = 0.8$ | 1000 | 0.25 |
| Valves | Globe, Standard | 1500 | 4 |
| | Globe, Angle | 1000 | 2 |
| | Diaphragm, dam type | 1000 | 2 |
| | Butterfly | 800 | 0.25 |
| Check valves | Lift | 2000 | 10 |
| | Swing | 1500 | 1.5 |
| | Tilting-disk | 1000 | 0.5 |

### API

The implementation for all functions used to calculate the fluid frictional pressure loss through pipe fittings using the 2K method can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 3/ 03dTwoK.c`. The declarations of the following functions can be found in `03dTwoK.h` at the same file path.

```
TwoKFittings TwoKData(TwoKFittings input)
```

This subroutine is used to write the 2K database into the struct where data is to be entered.

```
TwoKFittings TwoKVariable(TwoKFittings table, double *rho, double *u, double *d, double *mu, double *Impd)
```

This subroutine is used to collect the required data to run calculate the head and pressure losses through the 2K method.

```
double TwoKCalculateK(double Re, double d, double k1, double kinf)
```

This subroutine is used to calculate the resistance coefficient using equation 4.3.5.

```
double TwoKCalculateHead(int count, double K, double u)
```

This subroutine is used to calculate the head loss associated with a specific fitting using equation 4.3.5.

```
double TwoKCalculatePLoss(double h, double rho)
```

This subroutine is used to calculate the pressure loss associated with a specific fitting.

```
TwoKFittings TwoKFinalTable(TwoKFittings data, double rho, double u, double d, double mu, double Impd, double *Re)
```

This subroutine is used to perform the calculations for the head and pressure losses through the 2K method.

```
void TwoKDisplay(TwoKFittings data, double rho, double u, double d, double mu, double Re, double TotalP, double TotalH)
```

This subroutine is used to display the collected data and generated data table on the user console.

```
void TwoKWrite(TwoKFittings data, double rho, double u, double d, double mu, double Re, double TotalP, double TotalH)
```

This subroutine is used to write the collect data and calculated parameters to a .txt file.

```
void TwoKWriteSwitch(TwoKFittings data, double rho, double u, double d, double mu, double Re, double TotalP, double TotalH)
```

This subroutine is used to ask the user if they would like to write the generated dataset and calculated parameters to a file.

```
void TwoK(void)
```

This subroutine us used to guide the computer through gathering the required data and performing calculations for the 2K head loss method.

- 3K Method [12]

The principle equation for this method is given by:

$$K = \frac{K_1}{\mathrm{Re}} + K_\infty \left(1 + \frac{K_d}{D_n^{0.3}}\right)$$
$$h_f = K\frac{u^2}{2g} \tag{4.3.6}$$

Where:

- $K_1$ is the resistance coefficient at Re = 1.
- $K_\infty$ is the resistance coefficient at Re = $\infty$.
- $K_d$ is the corrective resistance coefficient for the nominal pipe diameter.
- $D_n$ is the nominal pipe diameter in inches or the diameter nominal size in millimetres.

The fitting-specific constants used to calculate the resistant coefficient are shown in the tables below:

Table 4.3.6: 3K Resistance coefficient constants

| Fitting | Type | $k_1$ | $k_\infty$ | $k_d$ (inches$^{0.3}$) | $k_d$ (mm$^{0.3}$) |
|---|---|---|---|---|---|
| 90° Elbow, threaded | Standard radius (R/D = 1) | 800 | 0.14 | 4 | 10.6 |
| | Long radius (R/D = 1.5) | 800 | 0.071 | 4.2 | 11.1 |
| 90° Elbow, Flanged or Welded | Standard radius (R/D = 1) | 800 | 0.091 | 4 | 10.6 |
| | Long radius (R/D = 2) | 800 | 0.056 | 3.9 | 10.3 |
| | Long radius (R/D = 4) | 800 | 0.066 | 3.9 | 10.3 |
| | Long radius (R/D = 6) | 800 | 0.075 | 4.2 | 11.1 |
| 90° Elbow, Mitered | 1 Weld (90°) | 1000 | 0.27 | 4 | 10.6 |
| | 2 Welds (45°) | 800 | 0.068 | 4.1 | 10.8 |
| | 3 Welds (30°) | 800 | 0.075 | 4.2 | 11.1 |
| 45° Elbow, Threaded | Standard radius (R/D = 1) | 500 | 0.071 | 4.2 | 11.1 |
| | Long radius (R/D = 1.5) | 500 | 0.052 | 4 | 10.6 |
| 45° Elbow mitered | 1 Weld (45°) | 500 | 0.086 | 4 | 10.6 |
| | 2 Welds (45°) | 500 | 0.052 | 4 | 10.6 |
| 180° | Threaded, close-return (R/D = 1) | 1000 | 0.23 | 4 | 10.6 |
| | Flanged (R/D = 1) | 1000 | 0.12 | 4 | 10.6 |
| | All types (R/D = 1.5) | 1000 | 0.1 | 4 | 10.6 |

Table 4.3.7: 3K Resistance coefficient constants (Continued)

| Fitting | Type | $k_1$ | $k_\infty$ | $k_d$ (inches$^{0.3}$) | $k_d$ (mm$^{0.3}$) |
|---|---|---|---|---|---|
| Tee, used as elbow | Threaded (R/D = 1) | 500 | 0.274 | 4 | 10.6 |
| | Threaded, Long radius (R/D = 1.5) | 800 | 0.14 | 4 | 10.6 |
| | Flanged (R/D = 1) | 800 | 0.28 | 4 | 10.6 |
| | Stub-in type branch | 1000 | 0.34 | 4 | 10.6 |
| Tee, run-through | Threaded (R/D = 1) | 200 | 0.091 | 4 | 10.6 |
| | Flanged (R/D = 1) | 150 | 0.05 | 4 | 10.6 |
| | Stub-in type branch | 100 | 0 | 0 | 0 |
| Angle valve | 45°, full line size, $\beta = 1$ | 950 | 0.25 | 4 | 10.6 |
| | 90°, full line size, $\beta = 1$ | 1000 | 0.69 | 4 | 10.6 |
| Globe valve | Standard, $\beta = 1$ | 1500 | 1.7 | 3.6 | 9.5 |
| Plug valve | Branch flow | 500 | 0.41 | 4 | 10.6 |
| | Straight through | 300 | 0.084 | 3.9 | 10.3 |
| Plug Valve | Three-way (Flow-through) | 300 | 0.14 | 4 | 10.6 |
| Gate valve | Standard, $\beta = 1$ | 300 | 0.037 | 3.9 | 10.3 |
| Ball valve | Standard, $\beta = 1$ | 300 | 0.017 | 3.5 | 9.2 |
| Diaphragm valve | Dam type | 1000 | 0.69 | 4.9 | 12.9 |
| Swing check valve | $V_{min} = 35\ [\rho(lb_m/ft^3)]^{-1/2}$ | 1500 | 0.46 | 4 | 10.6 |
| Lift check valve | $V_{min} = 40\ [\rho(lb_m/ft^3)]^{-1/2}$ | 2000 | 2.85 | 3.8 | 10 |

## API

The implementation for all functions used to calculate the fluid frictional pressure loss through pipe fittings using the 3K method can be found in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 3/ 03dThreeK.c`. The declarations of the following functions can be found in `03dThreeK.h` at the same file path.

```
ThreeKFittings ThreeKData(ThreeKFittings input)
```

This subroutine is used to load the 3K database into the struct stated above.

```
ThreeKFittings ThreeKVariable(ThreeKFittings table, double *DN, double *rho, double *u, double *d, double *mu)
```

This subroutine is used to collect the variables required to run the calculations.

```
double ThreeKCalculateK(double Re, double DN, int k1, double kinf, double kd)
```

This subroutine is used to calculate the Resistant Coefficient using equation 4.3.6.

```
double ThreeKCalculateHead(double count, double K, double u)
```

This subroutine is used to calculate the head loss associated with a singular fitting using equation 4.3.6. This calculated value is then multiplied by the count.

```
double ThreeKCalculatePLoss(double h, double rho)
```

This subroutine is used to calculate the pressure loss associated with the head loss attributed to a single fitting.

```
ThreeKFittings ThreeKFinalTable(ThreeKFittings data, double rho, double u, double d, double mu, double DN, double *Re)
```

This subroutine is used to amalgamate the data into a single variable.

```
void ThreeKDisplay(ThreeKFittings data, double rho, double u, double d, double mu, double Re, double DN, double TotalH, double TotalP)
```

This subroutine is used to display the contents of the generated data table in addition to all inputted variables.

```
void ThreeKWrite(ThreeKFittings data, double rho, double u, double d, double mu, double Re, double DN, double TotalH, double TotalP)
```

This subroutine is used to write the collected data and generated dataset to a .txt file.

```
void ThreeKWriteSwitch(ThreeKFittings data, double rho, double u, double d, double mu, double Re, double DN, double TotalH, double TotalP)
```

This subroutine is used to check if the user would like to write the generated dataset and collected parameters to a file.

```
void ThreeK(void)
```

This subroutine is used to guide the user through collecting the necessary data and calculating and displaying head losses associated with the 3K method.

## 4.4 Fluid measurement

When measuring fluids in a process, it is desirable to be able to measure the pressure and mass flow rate of the stream being analysed. This section covers the following devices:

- Pitot static tube.
- Orifice plate meter.
- Venturi meter.
- Linear flow meters.

### 4.4.1 Pitot static tube

This measurement device utilises a static impact connection connected to a manometer to infer the velocity of an imaginary fluid element travelling towards the impact connection using Bernoulli's equation (Eqn 4.2.5). To ensure the validity of results, the pitot tube must be fitted perpendicular to the pipe wall and thus is inherently reliant on the flow direction (i.e. this device is unable to accurately measure the flow rate of a stream not able to travel directly onto its impact connection). This device can also be optimised such that the device occupies a minimal amount of space.

The equation that the results from using a pitot static tube starts its derivation from Bernoulli's equation:

$$\frac{P_1}{\rho} + \frac{1}{2}u_1^2 + Z_1 g = \frac{P_2}{\rho} + \frac{1}{2}u_2^2 + Z_2 g + g h_f$$

Assuming that the pipe section is horizontal and that the velocity at the impact connection is $u_2 = 0$ m/s:

$$\frac{P_1}{\rho} + \frac{1}{2}u_1^2 = \frac{P_2}{\rho} + h_f$$

Over a short horizontal length, the head loss due to friction can be considered to be negligible thus meaning that it can be ignored from the consequent derivation. In addition, if the impact connection lies on the pipe centreline, there will be, by definition (See 4.2.4), a minimal amount of skin frictional losses in addition to there being a local minimum of shear stress (See 4.2.2). This results in the following statement:

$$\frac{P_1}{\rho} + \frac{1}{2}u_1^2 = \frac{P_2}{\rho}$$

Rearranging for the initial velocity:

$$\frac{1}{2}u_1^2 = \frac{P_2 - P_1}{\rho}$$

$$u_1 = \sqrt{\frac{2(P_2 - P_1)}{\rho}} \tag{4.4.1}$$

In practise, a pitot static tube may have a thin capillary tube-like entry where the entry fluid hits a diaphragm membrane in contact with a static pressure (e.g. a reference pressure) connection. It can be argued that at this point it may be appropriate to account for frictional losses. If this is the case, it results in the above equation becoming:

$$u_1 = \sqrt{\frac{2(P_2 - P_1)}{\rho} + g h_f}$$

### API

The implementation for all functions used to calculate the fluid velocity from a pitot static tube measurement in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 3/ 03ePitot.c`. The declarations of the following functions can be found in `03ePitot.h` at the same file path with the exception of '`PitotStaticTube(void)`' which can be found in `B48BB_T3.h`.

```
void PitotVariable(double *P2, double *rho1, double *rho2, double *h1, double *h2, double
*d)
```

This subroutine is used to collect the necessary data to predict the data associated with a pitot static tube.

```
void PitotCalculation(double P2, double rho1, double rho2, double h1, double h2, double
d, double *P1, double *v, double *Q)
```

This subroutine is used to predict the volumetric flow rate of a fluid travelling with a pitot static tube. This function also returns the values of the initial fluid pressure, point velocity and volumetric flow rate to the calling function.

```
void PitotDisplay(double P1, double P2, double rho1, double rho2, double h1, double h2,
double d, double v, double Q)
```

This subroutine is used to output the collected data and calculated values to the user console.

```
void PitotWrite(double P1, double P2, double rho1, double rho2, double h1, double h2, double
d, double v, double Q)
```

This subroutine is used to output the collected data and calculated values to a .txt file.

```
void PitotWriteSwitch(double P1, double P2, double rho1, double rho2, double h1, double
h2, double d, double v, double Q)
```

Subroutine to ask the user if they would like to save the results of this program to a file.

```
void PitotStaticTube(void)
```

This subroutine guides the user through gathering the data and calculation of pressure of a flowing fluid using a pitot static tube.

### 4.4.2 Orifice Plate meter

This measurement device utilises a sudden reduction in flow cross-sectional area in order to measure the differential pressure at two points. Once again, this device utilises Bernoulli's equation (Eqn 4.2.5) to produce a result.

The mechanism for which this works is that as the fluid contracts to match the reduced area, the fluid pressure will reduce as the fluid travels faster. The derivation for this device starts from Bernoulli's equation:

$$\frac{P_1}{\rho g} + \frac{1}{2g}u_1^2 + Z_1 = \frac{P_2}{\rho g} + \frac{1}{2g}u_2^2 + Z_2 + h_f$$

Assuming that the device is horizontal, the equation becomes:

$$\frac{P_1}{\rho} + \frac{1}{2}u_1^2 = \frac{P_2}{\rho} + \frac{1}{2}u_2^2 + gh_f$$

$$\frac{P_1}{\rho} - \frac{P_2}{\rho} - gh_f = \frac{1}{2}u_2^2 - \frac{1}{2}u_1^2$$

$$\frac{2}{\rho}(P_1 - P_2) - 2gh_f = u_2^2 - u_1^2$$

The mass conservation principle shown in equation 4.2.4 relates the final velocity $u_2$ to $u_1$ by:

$$u_2 = \frac{A_1 u_1}{A_2}$$

$$\frac{2}{\rho}(P_1 - P_2) - 2gh_f = u_1^2\left(\frac{A_1}{A_2}\right)^2 - u_1^2 = u_1^2\left[\left(\frac{A_1}{A_2}\right)^2 - 1\right]$$

At this point, it is important to remember that since the pressure loss is determined from the pressure being measured at two points, the frictional pressure losses are indirectly measured.

This means that the head loss due to friction can be ignored. Rearranging for the initial fluid velocity:

$$u_1^2 = \frac{2(P_1 - P_2)}{\rho\left[\left(\frac{A_1}{A_2}\right)^2 - 1\right]}$$

$$u_1 = \sqrt{\frac{2(P_1 - P_2)}{\rho\left[\left(\frac{A_1}{A_2}\right)^2 - 1\right]}}$$

This equation can be used to infer the volumetric flowrate using the definition:

$$\because Q = uA$$

$$Q = A_1\sqrt{\frac{2(P_1 - P_2)}{\rho\left[\left(\frac{A_1}{A_2}\right)^2 - 1\right]}}$$

In an ideal world, $A_2$ is not easily measured due to this being the area of the vena contracta (The minimum flow area that the flow temporarily adopts after a sudden reduction). However, an area that is easy to measure is the hole inside the Orifice plate meter. Since the area of the vena contracta must be smaller than area of the hole, a modification can be made to the above equation:

$$\text{Let: } A_2 = C_C A_o$$

$$Q = A_1\sqrt{\frac{2(P_1 - P_2)}{\rho\left[\left(\frac{A_1}{C_C A_o}\right)^2 - 1\right]}}$$

Since this is also not very pleasant to deal with, the correction factor can be brought outside the square root and this value is typically stated by the device manufacturer:

$$Q = C_d A_1\sqrt{\frac{2(P_1 - P_2)}{\rho\left[\left(\frac{A_1}{A_o}\right)^2 - 1\right]}} \tag{4.4.2}$$

Multiplying through by the fluid density will therefore result in the mass flow rate.

$$\dot{m} = C_d A_1\sqrt{\frac{2\rho(P_1 - P_2)}{\left[\left(\frac{A_1}{A_o}\right)^2 - 1\right]}}$$

### 4.4.3 Venturi meter

This measurement device is similar in working principle to the orifice plate meter (shown previously). Although more costly, it can be engineered to make significant reductions to pressure losses over an orifice plate meter and so the frictional pressure losses can most probably be ignored.

Due to the similar working principle, the derivation for this device is much the same as the

orifice plate meter:

$$Q = C_d A_1 \sqrt{\frac{2(P_1 - P_2)}{\rho\left[\left(\frac{A_1}{A_2}\right)^2 - 1\right]}} \tag{4.4.3}$$

Similarly, the mass flow rate can then be calculated with the following equation:

$$\dot{m} = C_d A_1 \sqrt{\frac{2\rho(P_1 - P_2)}{\left[\left(\frac{A_1}{A_2}\right)^2 - 1\right]}}$$

Here $A_2$ is the cross-sectional area of the venturi meter at the thinnest point, its neck.

### API

The implementation for all functions used to calculate the fluid velocity from an orifice plate /venturi meter measurement in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 3/ 03fOrifice.c`. The declarations of the following functions can be found in `03fOrifice.h` at the same file path with the exception of '`OrificePlateMeter(void)`' which can be found in `B48BB_T3.h`.

```
void OrificeVariable(double *C_d, double *d1, double *d2, double *rho, double *P1, double *P2, double *h_f)
```

This subroutine is used to collect the data required to predict the mass flowrate through an orifice plate.

```
void OrificeCalculation(double C_d, double d1, double d2, double rho, double P1, double P2, double h_f, double *u, double *Q, double *m)
```

This subroutine is used to calculate the mass flow rate, volumetric flow rate and average fluid velocity as it flows through an orifice plate meter through Bernoulli's principle.

```
void OrificeDisplay(double P1, double P2, double rho, double d1, double d2, double C_d, double h_f, double u, double Q, double m)
```

This subroutine is used to output the collected data and the calculated values to the user console.

```
void OrificeWrite(double P1, double P2, double rho, double d1, double d2, double C_d, double h_f, double u, double Q, double m)
```

This subroutine is used to write the collected data and the calculated values to a .txt file.

```
void OrificeWriteSwitch(double P1, double P2, double rho, double d1, double d2, double C_d, double h_f, double u, double Q, double m)
```

Subroutine to ask the user if they would like to save the results of this program to a file.

```
void OrificePlateMeter(void)
```

This subroutine guides the user through gathering the data and calculation of the mass flowrate of a flowing fluid through an orifice plate/venturi meter.

### 4.4.4 Linear flow meters

These types of measurement devices linearly correlate the flow rate with the value of some indicator. Perhaps the most common of these devices is a rotameter but other devices worth

mentioning are a turbine flow meter (correlates rotation speed with flow rate)or a positive displacement meter (correlates the time required to fill a fixed volume to a flowrate).

**Rotameter**

This measurement device relies on a flotation device inferring the flow rate of the stream flowing through the device. It goes without saying that at high flow rates, the float will be raised higher than at a lower flow rate as the force pushing the float upwards balances with the floats weight. In reality, these devices tend to push a fluid upwards, against gravity, in a tube that is tapered from the top to the bottom. It is important to note that these devices have a constant pressure drop which is balanced with a variable area. The following equation, for mass flow rate through a rotameter, can be determined in much the same steps as before using Bernoulli's equation.

$$\dot{m} = C_d A_1 \sqrt{\frac{2\rho\Delta P}{\rho\left[\left(\frac{A_1}{A_2}\right)^2 - 1\right]}} \tag{4.4.4}$$

Where $A_1$ is the cross-sectional area of the tube at the point indicated by the float, $A_2$ is the annular area between the float and the tube at the point and $\Delta P$ is the constant pressure drop across the float. It can be found from a balancing forces vertically that the pressure drop of the float is:

$$\Delta P = \frac{V_f(\rho_f - \rho)g}{A_f}$$

Where $V_f$ is the volume of the float, $\rho_f$ is the density of the float and $A_f$ is the maximum cross-sectional area of the float. This results in the following expression for mass flow rate:

$$\dot{m} = C_d A_1 \sqrt{\frac{2\rho(V_f(\rho_f - \rho)g)}{\rho A_f\left[\left(\frac{A_1}{A_2}\right)^2 - 1\right]}}$$

In terms of the rotameter specific and fluid specific component and writing in terms of the volumetric flow rate:

$$Q = C_d A_1 \sqrt{\frac{2V_f g}{\rho A_f\left[\left(\frac{A_1}{A_2}\right)^2 - 1\right]}}\sqrt{\frac{(\rho_f - \rho)}{\rho}}$$

From the manufacturer, this equation is now simplified to:

$$Q = k(h)h\sqrt{\frac{(\rho_f - \rho)}{\rho}}$$

Due to these devices being reliant on the $k(h)$ function, these devices are often calibrated against water at standard conditions. This means that the reported value can often not be representative of the actual flowrate. To account for this, the following steps can be undertaken:

$$Q_{\text{calc}} = k(h)h\sqrt{\frac{(\rho_f - \rho_w)}{\rho_w}}$$

However, since the fluid typically travelling through the rotameter is not going to be water, to change the reference point we start from:

$$Q_{\text{act}} = k(h)h\sqrt{\frac{(\rho_f - \rho)}{\rho}}$$

Therefore, the relationship between these two statements is then:

$$Q_{\text{act}} = k(h)h\sqrt{\frac{(\rho_f - \rho)}{\rho_f - \rho_w}}Q_{\text{calc}}$$

### API

The implementation for all functions used to calculate the liquid mass flow rate from a Rotameter measurement in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 3/ 03gRotameter.c`. The declarations of the following functions can be found in `03gRotameter.h` at the same file path with the exception of '`Rotameter(void)`' which can be found in `B48BB_T3.h`.

```
void RotameterVariable(double *C_d, double *V_f, double *rho_f, double *rho, double *A_f,
double *are1, double *are2)
```

This subroutine is used to collect data such that the mass flow rate can be predicted from a theoretical rotameter.

```
void RotameterCalculation(double C_d, double V_f, double rho_f, double rho, double A_f,
double are1, double are2, double *dP, double *m, double *Q, double *u)
```

This subroutine is used to calculate the pressure drop, mass flow rate, volumetric flow rate and fluid velocity through a rotameter.

```
void RotameterDisplay(double rho, double V_f, double rho_f, double A_f, double are1,
 double are2, double C_d, double dP, double m, double Q, double u)
```

This subroutine is used to output the collected data and calculated variables to the user console.

```
void RotameterWrite(double rho, double V_f, double rho_f, double A_f, double are1, double
are2, double C_d, double dP, double m, double Q, double u)
```

This subroutine is used to output the collected data and calculated variables to a .txt file.

```
void RotameterWriteSwitch(double rho, double V_f, double rho_f, double A_f, double are1,
double are2, double C_d, double dP, double m, double Q, double u)
```

This subroutine is used to ask the user if they would like to save the results of this program to a file.

```
void Rotameter(void)
```

This subroutine guides the user through gathering the data and calculation of the mass flowrate of a flowing fluid through a rotameter.

### Turbine flow meter

This device is appropriate for measuring the flow rate of both liquids and gases from the rotational speed of the turbine. To minimise errors, magnetic couplings are used to measure the

speed to ensure that there is no mechanical connection between the rotor and sense. This fact allows this device to be suitable for corrosive or toxic fluids assuming that the device has been designed to account for this behaviour.

Unfortunately, this rotor is easily damaged by solids thus suspending them from applications involving fluid suspensions. Additionally, from to form frictional losses, a turbine flow meter are sensitive to location and are therefore require about 10 pipe diameters of straight pipe prior and 5 diameters of straight pipe downstream in order to achieve their headline accuracy. Additionally, these devices are unsuitable for viscous fluids since the shear force on the turbine will begin to disrupt the relationship between the volumetric flow speed of rotation. An additional disadvantage of these devices is that although they are expensive to buy, they are often convenient to use due to the lack of calculations required after making a measurement.

## 4.5 Pump sizing

Pumps are an essential part of any process to ensure that fluid can be moved between vessels being used to perform unit operations. Selection of a pump is dependent on a fluid's physical properties, volumetric flow rate, the state of the fluid being either a gas or liquid in addition to the saturation properties of the fluid being pumped. Since mechanical work is being done, the motor efficiency is also a factor in selection of a pump. Although many pumps differ physically, they are often attributed to an increase in fluid velocity resulting in an increase to fluid pressure aside from physically compressing the fluid into a smaller volume.

Pumps work by pushing fluid into a chamber and as a result of fluid being trapped and squeezed within the chamber, the pressure is raised due to volume work being done on the fluid adiabatically. Technically, at the inlet, the liquid pressure is raised and is thus called the pump suction. Similarly, the pump outlet is termed the pump discharge. This then infers that pumps work against a pressure gradient (since a fluid moves from an area of high pressure to low pressure) which means that a pump must inherently be able to overcome the high pressure (termed backpressure) fluid located at the outlet.

Across different processes, there are a variety of pumps used which falls into one of two types of pumps. These are positive displacement (where a fluid is compressed as it travels through the pump casing) and rotodynamic pumps (where a fluid's velocity is raised through performing volume work resulting in an increase to fluid pressure).

### 4.5.1 Positive displacement pumps

A positive displacement pump, or a metering/dosing pump, works by initially drawing a fluid into a compression chamber. Then, the fluid is trapped and squeezed from a movable element forcing itself against the body of trapped fluid. When the fluid pressure is able to overcome the backpressure, the fluid is then discharged at the outlet. These types of pumps either use reciprocating action (e.g. a reciprocating compressor) or rotary action (e.g. a sliding vane pump/vacuum pump) to achieve this increase in pressure thus allowing them to deliver a fixed volumetric flow rate at a known motor speed which can also be used to control them. Of this type of pump, there are three types of positive displacement pump available to the process designer:

- Piston pump.
  - This comprises of a stationary chamber where a moving piston reciprocates from one end to another.
  - Backflow is prevented with placement of check valves placed at pump suction and discharge.
- Gear pump.
  - This comprises of a master and slave gear to move fluid through the casing. The master gear is controlled by the connected motor which consequently also drives the slave gear.
  - Backflow is prevented by rotating the master gear against the flow direction (typically clockwise).
- Sliding vane pump.
  - This comprises of an acentric element filled with elements that create distinct subsections within the pump casing. Due to the mechanical element being acentric, volume is decreased as the fluid moves through the casing before being discharged.
  - Backflow is prevented by placing check valves at the pump suction and discharge.

In general, positive displacement pumps are capable of generating high pressures thus making them suitable for vacuum pump operation. In addition to this, positive displacement pumps are characterised by their ability to deliver a fixed flow against a variable back pressure. However, using these pumps will making the flow rate 'pulse' and so work better in parallel with other pumps when a constant flow rate is required or by fitting a pulsation damper to smooth out the pulsations. In practice, a pressure relief valve should be fitted at the discharge side to avoid damage to the pump and downstream pipework.

### 4.5.2 Rotodynamic pumps

Rotodynamic pumps work by inducing fluid flow through a impeller spinning at high speed. When a fluid element is induced into the pump casing, it gains kinetic energy as it is spun centrifugally across the impeller face, guided by vanes. As the element exits the pump, it passes through a diffuser which reduces the fluid velocity head and experiences gain to its static head (as per equation 4.2.5). These types of pumps are very common within industry and are typically used to transport low viscosity liquids and are typically supplied with a motor. In this category lies a centrifugal pump which can work either radially (perpendicular to flow direction at suction) or axially (Parallel to flow direction at suction). Additionally, the pump can also operate partly radial and axial which is termed a mixed flow pump. Impeller design is thus specific to the type of centrifugal pump being designed.

### 4.5.3 Pump sizing calculations

As stated previously, a pump needs to generate enough head to overcome the backpressure at the discharge. This separates pump calculations into a suction or discharge side. Equation 4.2.5 tells us that static and hydrostatic heads will assist in fluid flow to the pump and frictional head losses will hinder flow into the pump. To begin deriving the equations used to design a pump, we can start from Bernoulli's equation (Eqn 4.2.5). As the fluid flows from the suction-side vessel, Bernoulli's equation becomes:

$$\frac{P_1}{\rho g} + \frac{1}{2g}u_1^2 + Z_1 = \frac{P_2}{\rho g} + \frac{1}{2g}u_2^2 + Z_2 + h_f$$

$$\therefore \frac{P_1}{\rho g} + \frac{1}{2g}u_1^2 + (Z_1 - Z_2) - h_f = \frac{P_2}{\rho g} + \frac{1}{2g}u_2^2$$

The equation used to calculate the suction head is:

$$h_s = \frac{P_s}{\rho g} + h_{s1} + h_{s2} - h_{fs} \tag{4.5.1}$$

Where $P_s$ is the pressure in the suction-side vessel, $\rho$ is the fluid density, $h_{s1}$ is the liquid level in the suction-side vessel, $h_{s2}$ is the vertical fluid elevation above the pump inlet and $h_{fs}$ is the frictional head losses on the suction side given by the following equation:

$$h_f = 8\phi\left(\frac{L}{d}\right)\frac{u^2}{2g}$$

This provides an equivalence between $(Z_1 - Z_2)$ and $(h_{s1} + h_{s2})$.

Similarly, the total discharge head can be derived from Bernoulli's equation to be:

$$\frac{P_4}{\rho g} + \frac{1}{2g}u_1^2 + (Z_4 - Z_3) + h_f = \frac{P_3}{\rho g} + \frac{1}{2g}u_2^2$$

The discharge head can be calculated with the following equation:

$$h_d = \frac{P_d}{\rho g} + h_{d1} + h_{d2} + h_{fd} \tag{4.5.2}$$

Where $P_d$ is the pressure in the discharge-side vessel, $h_{d1}$ is the liquid level in the discharge-side vessel, $h_{d2}$ is the vertical fluid elevation above the pump inlet and $h_{fd}$ is the frictional head losses on the discharge side.

The pump head is then the difference between the discharge head and suction head:

$$\Delta h_P = h_d - h_s \tag{4.5.3}$$

In terms of Bernoulli's equation:

$$\Delta h_P = \left[\frac{P_4}{\rho g} + \frac{1}{2g}u_1^2 + (Z_4 - Z_3) + h_{fd}\right] - \left[\frac{P_1}{\rho g} + \frac{1}{2g}u_1^2 + (Z_1 - Z_2) - h_{fs}\right]$$

Assuming that the fluid velocity remains constant, this form of the pump head can be changed to:

$$\left[\frac{P_1}{\rho g} + Z_1 - h_{fs}\right] + \Delta h_P = \left[\frac{P_4}{\rho g} + Z_4 + h_{fd}\right]$$

This can then be transformed to the pump pressure increase being equated to:

$$\Delta P_P = \rho g \Delta h_P \tag{4.5.4}$$

Assuming that the fluid pressure is being raised adiabatically, the minimum hydraulic power required to drive the fluid through the pump is given by:

$$\dot{W}_h = \rho g \Delta h_P Q \tag{4.5.5}$$

Allowing for pump inefficiencies:

$$\dot{W}_h = \frac{\rho g \Delta h_P Q}{\eta} \tag{4.5.6}$$

Where $\eta$ is a value between zero and one.

Technically, the pump will cavitate when the suction head drops below the vapour pressure head due to cavitation being characterised by the generation of vapour bubbles. Typically, cavitation will occur around the impeller vanes as liquid will rush into voids and generate shock waves as vapour bubbles collapse. This can cause excessive vibration, rattling in the pump casing, reductions to the pump efficiency and physical damage to the pump impeller or casing. The available NPSH is defined mathematically as:

$$\text{Available NPSH} = \frac{P_s - \hat{P}}{\rho g} + h_{s1} + h_{s2} - h_{fs} \tag{4.5.7}$$

### API

The implementation for all functions used to size a centrifugal pump in `Course Material/Year 2/ 3. B48BC - Process Engineering B (Fluid Dynamics)/ Topic 4/ 04bPumpSizing.c`. The declarations of the following functions can be found in `04bPumpSizing.h` at the same file path with the exception of '`PumpSizing(void)`' which can be found in `B48BB_T4.h`.

```
void PumpVariable(double *Q, double *rho, double *Psat, double *NPSHr, double *eta)
```

This subroutine is used to collect the variables that do not belong to the struct.

```
head PumpHeadVariable(int type, head var)
```

This subroutine is used to collect the data required to calculate either the suction or discharge head.

```
double HeadCalculation(head var, double rho)
```

This subroutine is used to calculate the head on either side of the pump.

```
double NPSHCalculation(head var, double Psat, double rho)
```

This subroutine is used to calculate the NPSH available.

`double PumpHeadCalculation(double hs, double hd)`

This subroutine is used to calculate the pump head.

`double PumpPressureCalculation(double rho, double hp)`

This subroutine is used to calculate the pressure 'drop' across the pump.

`double PumpPower(double dP_p, double Q, double eta)`

This subroutine is used to calculate the pump power requirement.

`void PumpDisplay(head suction, head discharge, double Q, double rho, double Psat, double NPSHr, double NPSHa, double eta, double phead, double ppressure, double ppower)`

This subroutine is used to display the collected data and calculated parameters on the user console.

`void PumpWrite(head suction, head discharge, double Q, double rho, double Psat, double NPSHr, double NPSHa, double eta, double phead, double ppressure, double ppower)`

This subroutine is used to output the generated dataset and collected parameters to a .txt file.

`void PumpWriteSwitch(head suction, head discharge, double Q, double rho, double Psat, double NPSHr, double NPSHa, double eta, double phead, double ppressure, double ppower)`

This subroutine is used to ask the user if they would like to write the generated dataset and collected parameters to a .txt file.

`void PumpSizing(void)`

This subroutine guides the user through gathering the data and sizing of a centrifugal pump.

### General guidance

In general, the student guide for this section provides the following guidance when designing a pump:

- Cavitation within a pump should be avoided. This is when the fluid vaporises due to the pressure inside the pump casing matching the saturation pressure of the fluid travelling through the casing.
  - Maintain a net positive suction head at the pump inlet.
  - Provide a safety factor above the cited Net Positive Suction Head (NPSH) value.
  - All designs should ensure that the available NPSH exceeds the state required NPSH value.
  - No throttling valves at the suction side. This can causes turbulence within the casing which can cause noisy pump operation.
- When there is insufficient NPSH, the following steps are recommended:
  - Operate the suction-side vessel at a higher pressure or a lower temperature.
  - Increase the vertical separation between the liquid level in the suction-side tank and the pump inlet.
  - Minimise the number of bends and fittings used in the suction pipework.
  - Increase the pipe diameter used at the pump suction.

In addition to this, it is common practise to:

- Place a check valve at the discharge side to prevent backflow through the pump when not in operation.
- Two pumps are fitted in parallel with the additional pump being on standby in case of the pump failing.
- Install a smaller impeller within a larger pump will allow for the plant throughput to grow.
- Fit a pump with a variable speed drive to allow greater control over flow rate. This has also been done historically by placing a valve on the discharge side.

# References


[1] Top Universities, *Chemical engineering degrees*, 2020. [Online]. Available: https://www.topuniversities.com/courses/engineering-chemical/guide.

[2] madnight, *Github language stats*, 2020. [Online]. Available: https://madnight.github.io/githut/#/pull_requests/2020/3.

[3] B. E. Poling, *The properties of gases and liquids / Bruce E. Poling, John M. Prausnitz, John P. O'Connell.* eng, 5th ed. New York, N.Y.: McGraw-Hill, 2001, ISBN: 9780070116825.

[4] J. M. J. M. Smith, *Introduction to chemical engineering thermodynamics / J.M. Smith, H.C. Van Ness, M.M. Abbott, M.T. Swihart.* eng, 8th ed. New York, N.Y.: McGraw-Hill, 2018, ISBN: 9781259921896.

[5] University of Arizona, *Hydrostatic pressure principle*, 2020. [Online]. Available: http://web.sahra.arizona.edu/education2/fosssww/PCL/Hydrostatic.pdf.

[6] Lumen Learning, *Cohesion and adhesion in liquids: Surface tension and capillary action*, 2020. [Online]. Available: https://courses.lumenlearning.com/physics/chapter/11-8-cohesion-and-adhesion-in-liquids-surface-tension-and-capillary-action/.

[7] Kruss Scientific, *Interfacial tension*, Oct. 2020. [Online]. Available: https://www.kruss-scientific.com/services/education-theory/glossary/interfacial-tension/.

[8] ——, *Young's equation*, Oct. 2020. [Online]. Available: https://www.kruss-scientific.com/services/education-theory/glossary/youngs-equation/.

[9] ——, *Surface free energy (sfe), surface energy*, Oct. 2020. [Online]. Available: https://www.kruss-scientific.com/services/education-theory/glossary/surface-free-energy/.

[10] T Dreher, C Lemarchand, L Soulard, E Bourasseau, P Malfreyt, and N Pineau, "Calculation of a solid/liquid surface tension: A methodological study," eng, *The Journal of chemical physics*, vol. 148, no. 3, pp. 034 702–034 702, 2018, ISSN: 1089-7690.

[11] neutrium.net, *Pressure loss from fittings - 2k method*, 2012. [Online]. Available: https://neutrium.net/fluid-flow/pressure-loss-from-fittings-2k-method/.

[12] ——, *Pressure loss from fittings - 3k method*, 2012. [Online]. Available: https://neutrium.net/fluid-flow/pressure-loss-from-fittings-3k-method/.