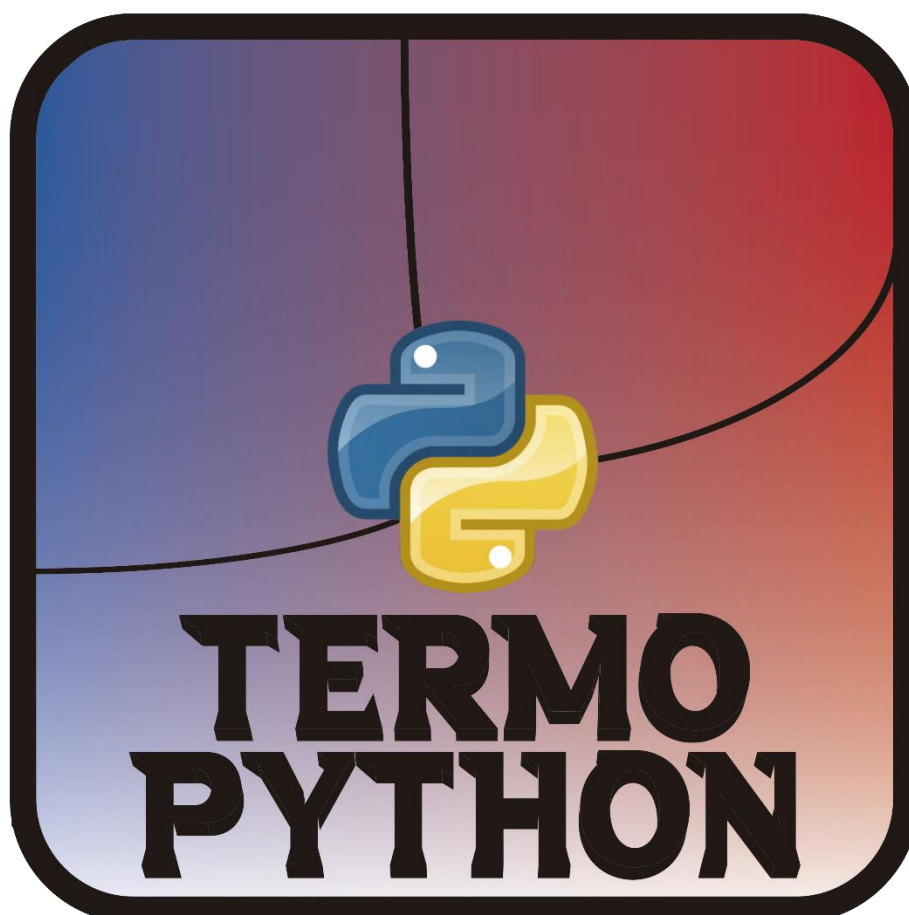


Manual – TermoPython



Desenvolvido por:

Ana Luisa Carvalho Mendonça

Danton de Godoy Antonio

Leonardo Izaias Rodrigues

Maria Carolina Barbosa Silveira

Raissa Alves Morganti Paula

Nº USP: 10872180

Nº USP: 11801348

Nº USP: 11801373

Nº USP: 10781971

Nº USP: 11801400

ana.lucarmendo@usp.br

dan74@usp.br

leoizaias@usp.br

mcarolbs@usp.br

raissa.amp@usp.br

Disciplina:

Computação Científica em Python (LOM3260)

Docente:

Luiz Tadeu Fernandes Eleno

1. Introdução

O programa TermoPython é uma ferramenta para o estudo de Termodinâmica Química Aplicada com foco em Equilíbrio de Fases Líquido-Vapor (ELV). O código implementado tem como objetivo prever, a partir da entrada de pontos experimentais pelo usuário, os valores de pressão de equilíbrio de fases à altas pressões utilizando o método de Peng-Robinson para substâncias puras e o método de Peng-Robinson em conjunto com Regras de Mistura de van der Waals para predição dos valores de pressão e fração molar de vapor do ELV de misturas binárias.

2. Metodologia

2.1 Abordagem Phi – Phi ($\phi - \phi$)

A abordagem Phi – Phi ($\phi - \phi$) é indicada para a modelagem do ELV (Equilíbrio Líquido – Vapor) de sistemas à altas pressões, tanto de substâncias puras quanto de misturas binárias. Nessa abordagem, determina-se a fugacidade (f) de cada um dos componentes nas fases líquida e vapor em função do coeficiente de fugacidade (ϕ) a fim de prever seu comportamento e, considerando a igualdade das fugacidades de cada fase para todos os componentes, fazer a modelagem do equilíbrio.

O cálculo da fugacidade do componente i na fase líquida em função do coeficiente de fugacidade (ϕ_i), fração molar de i na fase líquida (x_i) e pressão de equilíbrio (P), dá-se pela equação (1):

$$f_i^L = \hat{\phi}_i^L * x_i * P \quad (1)$$

Já o cálculo da fugacidade do componente i na fase vapor em função do coeficiente de fugacidade, fração molar de i na fase vapor (y_i) e pressão de equilíbrio (P), dá-se pela equação (2):

$$f_i^V = \hat{\phi}_i^V * y_i * P \quad (2)$$

As equações (1) e (2) aplicadas para substâncias puras, ou seja, um único componente, assumem a forma indicada nas equações (3) e (4), uma vez que para um único componente a fração molar de cada uma das fases será sempre unitária.

$$f^L = \hat{\phi}^L * P \quad (3)$$

$$f^V = \hat{\phi}^V * P \quad (4)$$

Onde os coeficientes de fugacidade (ϕ) são definidos através de uma Equação de Estado Cúbica. Neste trabalho foi utilizada a Equação de Peng-Robinson para misturas.

Os valores de frações molares, pressão e temperatura para a condição de Equilíbrio Líquido – Vapor são otimizados dada a definição em (5):

$$f_i^V = f_i^L \quad (5)$$

2.2 Equação de Peng-Robinson

A Equação de Estado Cúbica de Peng-Robinson (6) é um modelo que tem como objetivo relacionar as propriedades termodinâmicas a fim de determinar o estado termodinâmico. Esta equação, em conjunto com as Regras de Mistura e equações dos Coeficientes de Fugacidade em equilíbrio, permitem o estudo do equilíbrio líquido - vapor tanto para substâncias puras quanto para misturas à altas pressões.

$$P = \frac{R * T}{v - b} - \frac{a_c * \alpha(T)}{v * (v + b) + b * (v - b)} \quad (6)$$

2.3 Regras de mistura

As Regras de Mistura, utilizadas em conjunto com a equação de estado, definem os parâmetros de ajuste necessários levando em consideração a interação binária dos componentes e suas frações molares, as regras de mistura utilizadas neste trabalho são as equações de Van der Waals. O ajuste do parâmetro k_{ij} de interação binária, que mede as forças de atração e repulsão entre as moléculas, deve ser feito e os resultados obtidos para o valor determinado devem ser avaliados. Além do k_{ij} , são determinados os parâmetros a_i e b_i para substâncias puras e a_{ij} , a^L , a^V , b^L , b^V . Os parâmetros das regras de mistura são calculados a partir das seguintes equações:

$$a_i = \frac{0,45724 * R^2 * T c_i^2 * \alpha(T)_i}{P c_i} \quad (7)$$

$$b_i = \frac{0,07780 * T c_i}{P c_i} \quad (8)$$

$$a_{i,j} = (1 - k_{i,j}) \sqrt{a_i a_j} \quad (9)$$

$$a^L = \sum_i^c \sum_j^c x_i x_j a_{ij} \quad (10)$$

$$a^V = \sum_i^c \sum_j^c y_i y_j a_{ij} \quad (11)$$

$$b^L = \sum_i^c x_i b_i \quad (12)$$

$$b^V = \sum_i^c y_i b_i \quad (13)$$

2.4 Método Bolha P – Misturas Binárias

O estudo do ELV (Equilíbrio Líquido – Vapor) de uma mistura binária, de acordo com a Regra de Fases de Gibbs, é dado a partir do conhecimento de uma das propriedades constantes P ou T e uma das frações molares x_i ou y_i . O Método Bolha P, utilizado neste trabalho possui como dados a temperatura (T) constante e os valores de fração molar da fase líquida (x_i) e tem como objetivo calcular a pressão (P) e a fração molar da fase vapor (y_i) para cada um dos pontos experimentais. Determinando-se todas as variáveis pode-se prever o comportamento da pressão de equilíbrio a temperatura constante em função das frações molares, as curvas de equilíbrio Bolha P, pressão em função da fração molar dos componentes na fase líquida e Orvalho P, pressão em função da fração molar dos componentes na fase vapor.

2.5 Fator de Compressibilidade

A Equação de Estado Cúbica de Peng-Robinson pode ser reescrita em função do fator de compressibilidade Z (15), aplicada para cada uma das fases.

$$Z^3 - (1 - B)Z^2 + (A - 2B - 3B^2)Z - AB + B^2 + B^3 = 0 \quad (14)$$

Onde:

$$Z = \frac{P * v}{R * T} \quad (15)$$

$$A = \frac{a * P}{R^2 * T^2} \quad (16)$$

$$B^L = \frac{b^L * P}{R * T} \quad (17)$$

A equação cúbica (14) aplicada para substâncias puras é resolvida numericamente uma única vez, para a condição de equilíbrio, são encontradas três raízes. A maior raiz encontrada equivale ao fator de compressibilidade da fase vapor (Z^V), já na fase líquida (Z^L) é equivalente à menor raiz encontrada. Já para misturas, a equação cúbica (14) é resolvida numericamente tanto para a fase líquida quanto para a fase vapor, são encontradas três raízes em cada uma das fases. Na fase vapor a maior raiz encontrada equivale ao fator de

compressibilidade da fase vapor (Z^V), já na fase líquida (Z^L) é equivalente à menor raiz encontrada. O método numérico para o cálculo das raízes é descrito a seguir:

$$Q = \frac{COEF_1^2 - 3 * COEF_2}{9} \quad (18)$$

$$R = \frac{2 * COEF_1^3 - 9 * COEF_1 * COEF_2 + 27 * COEF_3}{54} \quad (19)$$

Onde:

$$COEF_1 = -(1 - B) \quad (20)$$

$$COEF_2 = A - 2 * B - 3 * B^2 \quad (21)$$

$$COEF_3 = -A * B + B^2 + B^3 \quad (22)$$

Definidos Q e R, calcula-se:

$$Q^3 - R^2 \quad (23)$$

Se esse valor é negativo, a equação possui apenas uma raiz real (x_1), calculada pela equação (24):

$$S = \left(\sqrt{R^2 - Q^3} + |R| \right)^{\frac{1}{3}} \quad (24)$$

$$\begin{aligned} &+1 \text{ se } R \text{ é positivo} \\ sgn(R) \text{ é } &-1 \text{ se } R \text{ é negativo} \\ &0 \text{ se } R \text{ é zero} \end{aligned} \quad (25)$$

$$x_1 = -sgn(R) \left[S + \frac{Q}{S} \right] - \frac{COEF_1}{3} \quad (26)$$

Já se o valor da diferença calculada na equação (21) é positivo, são calculadas 3 raízes reais (x_1 , x_2 e x_3):

$$\theta = \arccos\left(\frac{R}{\sqrt{Q^3}}\right) \quad (27)$$

$$x_1 = -2\sqrt{Q} \cos\left(\frac{\theta}{3}\right) - \frac{COEF_1}{3} \quad (28)$$

$$x_2 = -2\sqrt{Q} \cos\left(\frac{\theta + 2\pi}{3}\right) - \frac{COEF_1}{3} \quad (30)$$

$$x_3 = -2\sqrt{Q} \cos\left(\frac{\theta + 4\pi}{3}\right) - \frac{COEF_1}{3} \quad (31)$$

Determinadas as raízes da equação cúbica e os fatores de compressibilidade de acordo com os critérios para cada uma das fases, pode-se calcular os coeficientes de fugacidade e as fugacidades de cada uma das fases para a substância pura ou para os dois componentes da mistura.

2.6 Coeficientes de fugacidade e fugacidades

A igualdade das fugacidades de cada fase para cada um dos componentes, como estabelecido anteriormente, é o critério utilizado para a modelagem do ponto de bolha P, o cálculo dessas fugacidades é função dos coeficientes de fugacidade, das frações molares e da pressão de equilíbrio. Os coeficientes de fugacidade para cada fase da substância pura são dados por:

$$\ln \phi^L = (z^L - 1) - \ln(z^L - B) + \frac{A}{2\sqrt{2}B} \ln \left(\frac{z^L + (1 - \sqrt{2})B}{z^L + (1 + \sqrt{2})B} \right) \quad (32)$$

$$\ln \phi^V = (z^V - 1) - \ln(z^V - B) + \frac{A}{2\sqrt{2}B} \ln \left(\frac{z^V + (1 - \sqrt{2})B}{z^V + (1 + \sqrt{2})B} \right) \quad (33)$$

Já os coeficientes para cada componente i da mistura é dado por:

$$\begin{aligned} \ln \widehat{\phi}_i^L &= \frac{b_i}{b^L} (Z^L - 1) - \ln(Z^L - B^L) \\ &+ \frac{A^L}{2\sqrt{2}B^L} \left(\frac{2 \sum_k^C x_k a_{k,i}}{a^L} - \frac{b_i}{b^L} \right) \ln \left(\frac{Z^L + (1 - \sqrt{2})B^L}{Z^L + (1 + \sqrt{2})B^L} \right) \end{aligned} \quad (34)$$

$$\begin{aligned} \ln \widehat{\phi}_i^V &= \frac{b_i}{b^V} (Z^V - 1) - \ln(Z^V - B^V) \\ &+ \frac{A^V}{2\sqrt{2}B^V} \left(\frac{2 \sum_k^C y_k a_{k,i}}{a^V} - \frac{b_i}{b^V} \right) \ln \left(\frac{Z^V + (1 - \sqrt{2})B^V}{Z^V + (1 + \sqrt{2})B^V} \right) \end{aligned} \quad (35)$$

3. O Código e seu funcionamento

3.1 Equilíbrio Líquido – Vapor de Substâncias Puras

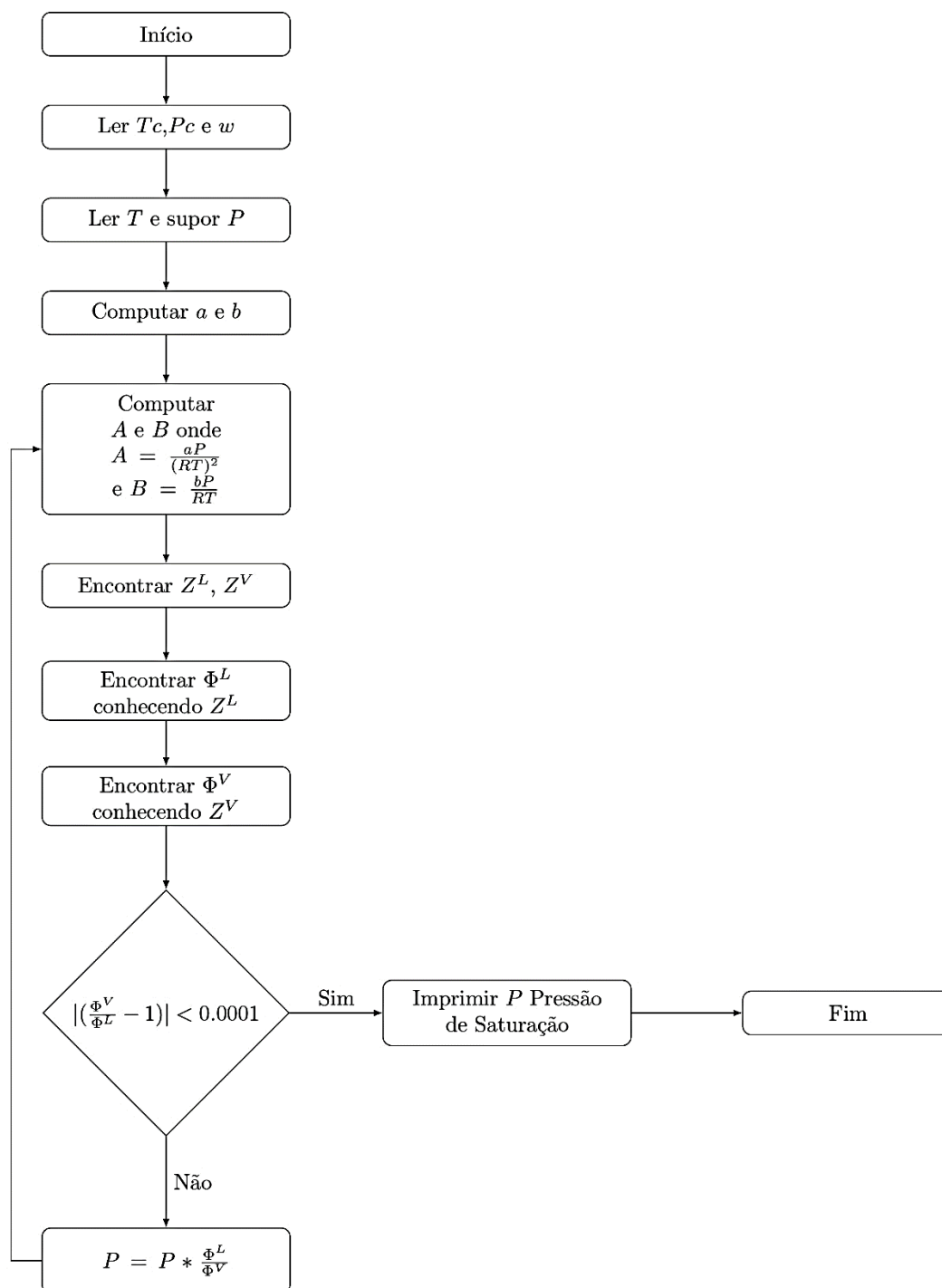
3.1.1 Algoritmo

A modelagem o ELV de uma substância pura a partir de uma equação cúbica é realizada de maneira sequencial, a sequência do cálculo das propriedades e sua eventual otimização é indicada na Figura 1.

- i. Entrada de dados: Inicialmente são computados os dados específicos da substância como temperatura e pressão crítica, temperatura e pressão do ponto triplo e fator acêntrico, em seguida, são inseridos os valores experimentais de temperatura e pressão.

- ii. Após a entrada de dados, são calculados os parâmetros “a”, “b”, “A” e “B” da equação cúbica para calcular o fator de compressibilidade.
- iii. Definidas todas as constantes e parâmetros, os fatores de compressibilidade para cada uma das fases são calculados numericamente pela resolução da equação cúbica, isto é, as raízes da equação são encontradas e avaliadas para definir os fatores de compressibilidade.
- iv. Os valores de fator de compressibilidade são utilizados no cálculo dos coeficientes de fugacidade para cada uma das fases.
- v. Calculados os coeficientes de fugacidade, a condição de igualdade é verificada, considerando uma determinada tolerância, enquanto as propriedades não atendem à condição especificada dentro da tolerância, o código entra em recursão, otimizando o valor de pressão experimental e recalculando todos os parâmetros.
- vi. Otimizada a pressão para atender ao critério de igualdade de fugacidade dentro da tolerância especificada, o código retorna o valor da pressão calculada que atendeu ao critério.

Figura 1: Fluxograma do algoritmo para a modelagem do ELV de uma substância pura pelo método de Peng-Robinson



Fonte: Notas de aula do Professor Pedro Felipe Arce Castillo, DEQUI EEL – USP (Adaptado)

3.1.2 O módulo *func_substpuras*

O módulo em questão define as funções utilizadas no cálculo de parâmetros e constantes para a resolução da equação cúbica e determinação dos coeficientes de fugacidade, assim como as funções de otimização do método.

3.1.2.1 Função *temp_reduzida* (*t*, *tc*)

Função para calcular a temperatura reduzida, parâmetro que relaciona a temperatura crítica e temperatura do ponto em questão.

```
# Cálculo da razão entre as temperaturas do ponto e crítica da substância para encontrar a temperatura reduzida.  
tr = t/tc  
return tr
```

3.1.2.2 Função *parametro_m* (*w*)

Função para calcular o parâmetro *m* de auxílio para o cálculo da função de correção *alpha*.

```
# Função para cálculo do parâmetro m do algoritmo de Peng Robinson.  
m = 0.37464 + 1.54226*w - 0.26992*w*w  
return m
```

3.1.2.3 Função *func_alpha* (*m*, *tr*)

Função do parâmetro *alpha* de correção do termo de atração.

```
# Função para cálculo do parâmetro alpha do algoritmo de Peng Robinson.  
alpha = (1 + m*(1-np.sqrt(tr)))**2  
return alpha
```

3.1.2.4 Função *constante_a* (*r*, *tc*, *pc*, *alpha*)

Cálculo da constante “a” que corrige o parâmetro de atração.

```
# Função para cálculo da constante de correção "a".  
a = 0.45724*((r*r*tc*tc)/pc)*alpha  
return a
```

3.1.2.5 Função *constante_b* (*r*, *tc*, *pc*)

Cálculo da constante “b” que corrige o volume das moléculas.

```
# Função para cálculo da constante de correção "b".  
b = 0.0778*r*tc/pc  
return b
```

3.1.2.6 Função *constante_A* (*a*, *p*, *r*, *t*)

Constante “A” de ajuste do termo de atração de volume para fator de compressibilidade na equação cúbica.

```
# Cálculo do fator de correção da atração na equação do fator de compressibilidade.
A = (a*p)/((r*t)**2)
return A
```

3.1.2.7 Função *constante_B* (b, p, r, t)

Constante “B” de ajuste do termo de volume das moléculas de volume para fator de compressibilidade na equação cúbica.

```
# Cálculo do fator de correção de repulsão na equação do fator de compressibilidade.
B = (b*p)/(r*t)
return B
```

3.1.2.8 Função *coeficientes_aux* (A, B)

Cálculo dos coeficientes de auxílio para a determinação do número de raízes da equação cúbica.

```
# Aplicação das equações dos coeficientes iniciais, do método de Peng Robinson, para o cálculo numérico das raízes.
cf1 = -(1 - B)
cf2 = A - 2*B - 3*(B*B)
cf3 = -A*B + B*B + B**3
return cf1, cf2, cf3
```

3.1.2.9 Função *func_Q* ($cf1, cf2$)

Parâmetro de auxílio para definir se a equação cúbica possui uma ou três raízes reais.

```
# Função para cálculo do parâmetro "Q" para definir se a equação cúbica possui uma ou três raízes reais.
Q = (cf1**2 - 3*cf2)/9
return Q
```

3.1.2.10 Função *func_R* ($cf1, cf2, cf3$)

Parâmetro de auxílio para definir se a equação cúbica possui uma ou três raízes reais.

```
# Função para cálculo de um do parâmetro "R" para definir se a equação cúbica possui uma ou três raízes reais.
R = (2*cf1**3 - 9*cf1*cf2 + 27*cf3)/54
return R
```

3.1.2.11 Função *numero_raizes* (Q, R)

Cálculo da constante que relaciona os parâmetros “Q” e “R” para definir se a equação cúbica possui uma ou três raízes reais.

```
# Função que relaciona os parâmetros "Q" e "R" definindo se a equação cúbica possui uma ou três raízes reais.
fa = Q**3 - R**2
return fa
```

3.1.2.12 Função *calcula_fatorcompress* (*fa*, *R*, *Q*, *cf1*, *cf2*, *cf3*)

Calcula as raízes da equação cúbica e retorna os fatores de compressibilidade, raiz mínima para o líquido e máxima para o vapor. O cálculo do “Z” depende inicialmente do valor de “fa” calculado, o código testa se o valor é menor ou maior que zero e dependendo do resultado calcula uma ou três raízes, em seguida, com as raízes calculadas são definidos os valores de Zl e Zv usando critérios de máximo e mínimo.

```
# Verificação do número de raízes da equação cúbica.
# Condição em que a raiz é única.
if fa < 0:
    # Condições que definem o fator do sinal da raiz.
    if R < 0:
        sgnR = -1

    elif R > 0:
        sgnR = 1

    else:
        sgnR = 0

    S = (np.sqrt(R*R - Q**3) + abs(R))**(1/3)
    # Expressão que calcula a raiz.
    r1 = -sgnR*(S + (Q/S)) - cf1/3
    # Os fatores de compressibilidade serão iguais a raiz única.
    Zl = r1
    Zv = r1
# Condição em que a raiz não é única.
else:
    theta = np.arccos(R/np.sqrt(Q**3))
    # Cálculo das 3 raízes.
    x1 = (-2*np.sqrt(Q)*np.cos(theta/3)) - (cf1/3)
    x2 = (-2*np.sqrt(Q)*np.cos((theta + 2*np.pi)/3)) - (cf1/3)
    x3 = (-2*np.sqrt(Q)*np.cos((theta + 4*np.pi)/3)) - (cf1/3)
    # Matriz de resultados.
    X = [x1,x2,x3]
    Zl = min(X)
    Zv = max(X)

return Zl, Zv
```

3.1.2.13 Função *coef_fugacidade* (*Zl*, *Zv*, *A*, *B*)

Cálculo dos coeficientes de fugacidade para as fases líquida e vapor, inicialmente os logaritmos dos coeficientes e em seguida utiliza a função exponencial para determinar os valores de cL e cV.

```
# Cálculo dos logaritmos dos coeficientes de fugacidade: Líquido e vapor.
ln_cL = (Zl-1)-(np.log(Zl-B))+((A/(2*np.sqrt(2)*B))*(np.log((Zl+((1-np.sqrt(2))*B))/(Zl+((1+np.sqrt(2))*B))))))
ln_cV = (Zv-1)-(np.log(Zv-B))+((A/(2*np.sqrt(2)*B))*(np.log((Zv+((1-np.sqrt(2))*B))/(Zv+((1+np.sqrt(2))*B))))))
# Cálculo dos coeficientes a partir dos logaritmos anteriores.
cL = np.exp(ln_cL)
cV = np.exp(ln_cV)
```

3.1.3 O módulo *linearReg*

O módulo em questão define uma função para um algoritmo de regressão linear utilizando o método dos mínimos quadrados, função que será utilizada para supor pontos experimentais aceitáveis para a aplicação e otimização da curva de equilíbrio líquido – vapor de substâncias puras, utilizando as fórmulas definidas na introdução teórica.

3.1.3.1 Função *linearR* (*M1*, *M2*)

Obtém uma equação linear que supõe chutes para a pressão experimental aceitáveis para o algoritmo.

```
# Registra o número de valores em M1.
n = len(M1)

sa1=0
sa2=0
sb1=0
sb2=0

for n in range(len(M1)):
    # Cálculo da soma dos quadrados.
    a1 = M1[n]*M1[n]
    sa1 += a1
    # Soma dos elementos de M1
    a2 = M1[n]
    sa2 += a2
    # Soma dos produtos de cada elemento de "M1" e "M2".
    b1 = M1[n]*M2[n]
    sb1 += b1
    # A soma de elementos de "M2".
    b2 = M2[n]
    sb2 += b2
# Parametros para o calculo dos coeficientes.
Aline = sa1 - ((sa2*sa2)/n)
Bline = sb1 - (sa2*sb2)/n
# Coeficiente angular da reta.
af = Bline/Aline
# Coeficiente linear da reta.
bf = (sb2/n) - ((sa2/n)*af)

return af, bf
```

3.1.4 O módulo *entrada_dados* para Substâncias Puras

A função `ler_dados_xlsx (arquivo, i)` do módulo em questão utiliza a biblioteca *Pandas* para a leitura do banco de dados em Excel que contém os dados constantes de cada uma das substâncias, essa função é utilizada no *input* de dados caso o usuário selecione a opção de utilizar o *input* automático dos valores das constantes.

```
# Cria um dataframe do arquivo colocado na função
dados = pd.read_excel(arquivo)

#Extrai do arquivo os dados necessários para aplicar no algoritmo de substâncias simples.
subst = dados["Substância"][i]
ppt = dados["Pressão Ponto Triplo (MPa)"][i]
tpt = dados["Temperatura Ponto Triplo (K) "][i]
pc = dados["Pressão Ponto Crítico (MPa)"][i]
tc = dados["Temperatura Ponto Crítico (K)"][i]
w = dados["Fator acêntrico"][i]

# Retorna os dados em um dicionário para que eles possam ser chamados individualmente.
return {'subs':subst, 'ppt': ppt, 'tpt':tpt, 'pc':pc, 'tc':tc, 'w': w}
```

3.2 Equilíbrio Líquido – Vapor de Misturas Binárias

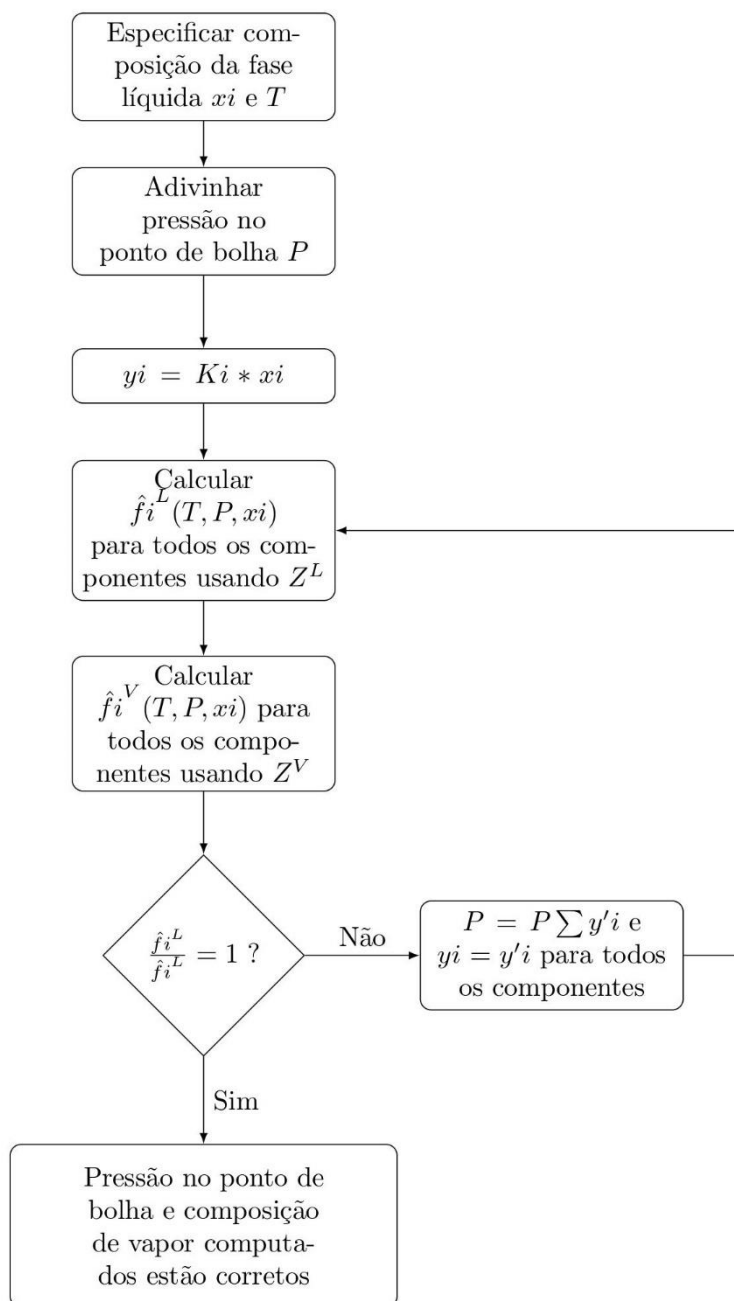
3.2.1 Algoritmo

A modelagem o ELV de uma mistura binária a partir de uma equação cúbica é realizada de maneira sequencial, a sequência do cálculo das propriedades e sua eventual otimização é indicada na Figura 2.

- i. Entrada de dados: Inicialmente são computados os dados específicos de cada um dos componentes do sistema como temperatura e pressão crítica e fator acêntrico, a temperatura do sistema e o parâmetro de interação binária e por fim são inseridos os valores experimentais de pressão, fração molar na fase líquida e fração molar na fase vapor.
- ii. Após a entrada de dados, são calculados os parâmetros constantes da equação cúbica para cada uma das substâncias puras.
- iii. Os parâmetros das substâncias puras calculados são utilizados no cálculo dos parâmetros da equação cúbica para a mistura, esses parâmetros são calculados em função das frações molar da fase líquida para a fase líquida e em função das frações molares da fase vapor para a fase vapor.
- iv. Definidas todas as constantes e parâmetros, os fatores de compressibilidade para cada uma das fases são calculados numericamente pela resolução da equação cúbica, isto é, as raízes da equação são encontradas e avaliadas para definir os fatores de compressibilidade.

- v. Os valores de fator de compressibilidade são utilizados no cálculo dos coeficientes de fugacidade de cada um dos componentes para cada uma das fases.
- vi. Os valores de coeficiente de fugacidade são utilizados no cálculo das fugacidades de cada um dos componentes para cada uma das fases.
- vii. Calculadas as fugacidades, a condição de igualdade é verificada, considerando uma determinada tolerância, enquanto as propriedades não atendem à condição especificada dentro da tolerância, o código entra em recursão, otimizando o valor de pressão experimental e o valor de fração molar da fase valor experimental, recalculando todos os parâmetros da mistura.
- viii. Otimizada a pressão e a fração molar da fase vapor para atender ao critério de igualdade de fugacidade dentro da tolerância especificada, o código retorna os valores da pressão e fração molar calculadas que atenderam aos critérios.

Figura 2: Fluxograma do algoritmo para a modelagem do ELV de uma mistura binário pelo método de Peng-Robinson



Fonte: Prausnitz, Lichtenthaler e Azevedo

3.2.2 O módulo *func_misturas*

O módulo em questão define as funções utilizadas no cálculo de parâmetros e constantes para a resolução da equação cúbica e determinação dos coeficientes de fugacidade, assim como as funções de otimização do método, utilizando as fórmulas definidas na introdução teórica.

3.2.2.1 A função *parametro_aj* ($a1, a2, k = 0.0000001, i = 0$)

Calcula a constante da mistura para correção do parâmetro de atração.

```
j= 1-i # Índice "j" complementar a "i" para localizar os termos na matriz "Ma"
Ma = np.array([a1,a2]) # Gera a matriz com as constantes de correção 1 e 2.
return (1 - k)*np.sqrt(Ma[i]*Ma[j]) # Equação da constante da mistura para correção do parâmetro de atração.
```

3.2.2.2 A função *parametro_atract* (*Matr, Mx, My*)

Calcula os parâmetros de atração do líquido, “al”, e do vapor, “av”.

```
alii = 0
alij = 0

avii = 0
avij = 0

for i in range(0,2):
    j = 1 - i # Índice "j" complementar a "i" para localizar os termos na matriz "Ma"
    # Fatores para o cálculo do parâmetro de atração da fase líquida.
    axii = Matr[i,i]*Mx[i]*Mx[i]
    axij = Matr[i,j]*Mx[i]*Mx[j]
    # Fatores para o cálculo do parâmetro de atração da fase vapor.
    ayii = Matr[i,i]*My[i]*My[i]
    ayij = Matr[i,j]*My[i]*My[j]
    # Somatório dos valores dos fatores da fase líquida.
    alii += axii
    alij += axij
    # Somatório dos valores dos fatores da fase líquida.
    avii += ayii
    avij += ayij
# Calcula os parâmetros de atração a partir dos somatórios obtidos.
atr1 = alii + alij
atr2 = avii + avij
return {'al':atr1, 'av':atr2}
```

3.2.2.3 A função *parametro_repulse* (*Mrep, Mx, My*)

Calcula os parâmetros de repulsão para o líquido, “bl”, e o vapor, “bv”.

```
bl = 0
bv = 0
#
for i in range(0,2):
    # Fatores para o cálculo do parâmetro de repulsão da fase vapor.
    bli = Mx[i]*Mrep[i]
    bvi = My[i]*Mrep[i]
    # Somatório dos valores dos fatores da fase líquida e vapor.
    bl += bli
    bv += bvi

return {'bl':bl, 'bv':bv}
```


3.2.2.4 A função *fatorcompress* (*MA*, *MB*)

Calcula as raízes da equação cúbica e retorna os fatores de compressibilidade, raiz mínima para o líquido e máxima para o vapor. O cálculo do “Z” depende inicialmente do valor de “fa” calculado, o código testa se o valor é menor ou maior que zero e dependendo do resultado calcula uma ou três raízes, em seguida, com as raízes calculadas são definidos os valores de Zl e Zv usando critérios de máximo e mínimo, o cálculo das raízes para uma mistura binária é realizado duas vezes, são encontradas raízes para a fase líquida e raízes para a fase vapor separadamente utilizando os parâmetros definidos anteriormente para cada fase.

Cálculo dos parâmetros iniciais e da raiz única:

```
for i in range(0,2):
    # Aplicação das equações dos coeficientes iniciais, do método de Peng Robinson, para o cálculo numérico das raízes.
    cf1 = -(1 - MB[i])
    cf2 = MA[i] - 2*MB[i] - 3*(MB[i]*MB[i])
    cf3 = (-MA[i]*MB[i]) + (MB[i]*MB[i]) + (MB[i]**3)
    # Função para cálculo do parâmetro "Q" para definir se a equação cúbica possui uma ou três raízes reais.
    Q = (cf1**2 - 3*cf2)/9
    # Função para cálculo de um do parâmetro "R" para definir se a equação cúbica possui uma ou três raízes reais.
    R = ((2*cf1**3) - (9*cf1*cf2) + (27*cf3))/54
    # Função que relaciona os parâmetros "Q" e "R" definindo se a equação cúbica possui uma ou três raízes reais.
    fa = (Q**3) - (R**2)
    # Verificação do número de raízes da equação cúbica.
    # Condição em que a raiz é única.
    if fa < 0:
        # Condições que definem o fator do sinal da raiz.
        if R < 0:
            sgnR = -1

        elif R > 0:
            sgnR = 1

        else:
            sgnR = 0

        S = (np.sqrt((R*R) - (Q**3)) + abs(R))**(1/3)
        # Expressão que calcula a raiz.
        r1 = -sgnR*(S + (Q/S)) - cf1/3
        # Atribui os valores calculados dos fatores de compressibilidade da fase líquida e gasosa para as variáveis Zl e Zv.
        if i == 0:
            Zl = r1
        if i == 1:
            Zv = r1
```

Cálculo de três raízes e saída de valores:

```
# Condição em que a raiz não é única.
else:
    theta = np.arccos(R/np.sqrt(Q**3))
    # Cálculo das 3 raízes.
    x1 = ((-2*np.sqrt(Q)*np.cos(theta/3)) - (cf1/3))
    x2 = ((-2*np.sqrt(Q)*np.cos((theta + 2*np.pi)/3)) - (cf1/3))
    x3 = ((-2*np.sqrt(Q)*np.cos((theta + 4*np.pi)/3)) - (cf1/3))
    # Matriz de resultados.
    X = [x1,x2,x3]
    # Atribui o valor mínimo calculado dos fatores de compressibilidade da fase líquida e gasosa para as variáveis Zl e Zv.
    if i == 0:
        Zl = min(X)
    if i == 1:
        Zv = max(X)

return {'L':Zl, 'v':Zv}
```

3.2.2.5 A função *coef_fugacidade* (*Zl*, *Zv*, *Mx*, *My*, *al*, *av*, *bl*, *bv*, *Ma*, *Mbi*, *MA*, *MB*, *i=0*)

A função em questão utiliza praticamente todos os parâmetros calculados no método até então para encontrar os valores de c_L e c_V para cada um dos componentes, retorna uma matriz com esses valores calculados. A fórmula utilizada para a determinação desses coeficientes foi, por possuir diversas etapas, foi dividida em partes menores a fim de facilitar sua visualização, tanto dos programadores quanto do usuário, em seguida as partes são reunidas para o cálculo do logaritmo dos coeficientes seguido de sua exponencial, assim como no módulo para substâncias puras visto anteriormente.

```
# Cria arrays para armazenar os valores dos logaritmos e dos coeficientes de fugacidade.
Mln_l = np.array([0,0], dtype = float)
Mln_v = np.array([0,0], dtype = float)
Mphi_l = np.array([0,0], dtype = float)
Mphi_v = np.array([0,0], dtype = float)

for i in range(0,2):
    # Cálculo do logaritmos dos coeficientes de fugacidade da fase líquida.
    cla = Mbi[i]/bl*(Zl-1)
    clb = np.log(Zl-MB[0])
    clc1 = MA[0]/(2*(np.sqrt(2))*MB[0])
    clc2 = ((2*((Mx[0]*Ma[0,i])+(Mx[1]*Ma[1,i])))/al)-(Mbi[i]/bl)
    clc3_1 = Zl - (0.41421356237309515*MB[0])
    clc3_2 = Zl + (2.414213562373095*MB[0])
    clc3 = np.log(clc3_1/clc3_2)
    clc = clc1*clc2*clc3
    # Valores são salvos na matriz Mln_l a partir da soma dos componentes a cima.
    Mln_l[i] = cla - clb + clc
    # Cálculo do logaritmos dos coeficientes de fugacidade da fase vapor.
    cva = Mbi[i]/bv*(Zv-1)
    cvb = np.log(Zv-MB[1])
    cvc1 = MA[1]/(2*(np.sqrt(2))*MB[1])
    cvc2 = ((2*((My[0]*Ma[0,i])+(My[1]*Ma[1,i])))/av)-(Mbi[i]/bv)
    cvc3_1 = Zv - (0.41421356237309515*MB[1])
    cvc3_2 = Zv + (2.414213562373095*MB[1])
    cvc3 = np.log(cvc3_1/cvc3_2)
    cvc = cvc1*cvc2*cvc3
    # Valores são salvos na matriz Mln_v a partir da soma dos componentes a cima.
    Mln_v[i] = cva - cvb + cvc
    # Matriz Mphi_l e Mphi_v salvam os valores dos coeficientes de fugacidade líquido e vapor.
    Mphi_l[i] = np.exp(Mln_l[i])
    Mphi_v[i] = np.exp(Mln_v[i])

return {'l':Mphi_l,'v':Mphi_v}
```

3.2.2.6 A função *fugacidade* (M_{phil} , M_{phiv} , M_x , M_y , p)

Calcula as fugacidades de cada fase para cada componente da mistura a partir da relação dos valores de coeficiente de fugacidade, frações molares e pressão.

```
# Cria arrays para armazenar os valores da fugacidade de cada componente da fase líquida e vapor.
Mfl = np.array([0,0], dtype = float)
Mfv = np.array([0,0], dtype = float)

for i in range(0,2):
    # Computa e armazena os valores da fugacidade de cada componente da fase líquida e vapor.
    Mfl[i] = Mphil[i]*Mx[i]*p
    Mfv[i] = Mphiv[i]*My[i]*p

return{'l': Mfl, 'v':Mfv}
```

3.2.2.7 A função *y_otimizado* (*My*, *Mfl*, *Mfv*, *i* = 0)

Função que otimiza o valor de “y” a fim de atender as condições da modelagem pelo método de Peng-Robinson.

```
MyI = np.array([0,0],dtype = float)
# Loop que otimiza o valor de y.
for i in range(0,2):

    MyI[i] = My[i]*(Mfl[i]/Mfv[i])

return MyI
```

3.2.3 O módulo *entrada_dados* para Misturas Binárias

A função *ler_dados_mistura_xlsx* (*arquivo*, *i*) do módulo em questão utiliza a biblioteca *Pandas* para a leitura do banco de dados em Excel que contém os dados constantes de cada um dos sistemas binários apresentados como opção no *input* de dados caso o usuário selecione a opção de utilizar o *input* automático dos valores das constantes.

```
# Cria um dataframe do arquivo colocado na função.
dados_mistura = pd.read_excel(arquivo)

# Retira as informações essenciais para o algoritmo de misturas desse arquivo.
pc1 = dados_mistura["Pressão Crítica do Componente 1 (atm)"][i]
pc2 = dados_mistura["Pressão Crítica do Componente 2 (atm)"][i]
tc1 = dados_mistura["Temperatura Crítica do Componente 1 (K)"][i]
tc2 = dados_mistura["Temperatura Crítica do Componente 2 (K)"][i]
w1 = dados_mistura["Fator Acêntrico do Componente 1"][i]
w2 = dados_mistura["Fator Acêntrico do Componente 2"][i]

# Retorna os dados em um dicionário para que eles possam ser chamados individualmente.
return {"pc1": pc1, "pc2": pc2, "tc1": tc1, "tc2": tc2, "w1": w1, "w2": w2}
```

3.3 O arquivo *MainTermoPython*: Interface com o usuário e aplicação do algoritmo de otimização

O *print* inicial traz informações gerais sobre o programa e suas funções, seguido de breves instruções, em seguida pede ao usuário que faça a primeira escolha, se irá trabalhar com uma

Substância Pura ou Mistura Binária. Todas as escolhas apresentadas ao usuário contêm operadores que testam a validade da resposta e retornam a pergunta caso seja uma resposta inválida.

```
# Define um operador para analisar a resposta dada.
valido = 0

print('Bem vindo ao TermoPython.\n\n --> Esse software foi feito para otimizar valores experimentais de pressão em sistemas de equilíbrios de fase')
# Enquanto a resposta dada não for válida, o software requisita uma nova escolha e orienta novamente o usuário.
while valido == 0:
    # pede ao usuário que escolha uma das aplicações do software e, de acordo com a opção escolhida, exibe o próximo input.
    algoritmo = input('Escolha o tipo de sistema que será usado (-substância pura/sp- ou -mistura binária/mb-): ')
    valido = 1
```

Caso a opção de Substâncias Puras seja selecionada, é apresentada a opção para selecionar uma substância do banco de dados ou inserir as constantes da substância manualmente. Caso a opção selecionada seja a inserção de dados manual, estão definidos os seguintes *inputs*:

```
if algoritmo == 'substância pura' or algoritmo == 'sp':
    valido = 1 # Aceita a resposta do usuário como válida.

    resposta = 0 # Cria outro operador para analisar as respostas do usuário.

    while resposta == 0: # Enquanto a resposta não for aceitável, requisita o input novamente.
        # Requisita ao usuário o método de input de dados.
        command = input('Deseja inserir os dados da substância manualmente?(sim/s ou não/n): ')
        # Se o usuário optar por inserir os dados manualmente, requisita, dado a dado, o necessário para o algoritmo.
        if command == 'sim' or command == 's':
            tpt = float(input('Temperatura de ponto triplo(K): '))
            tc = float(input('Temperatura crítica(K): '))
            pc = float(input('Pressão crítica(MPa): '))
            w = float(input('Fator acêntrico:'))
            r = float(input('Constante universal dos gases(J/mol.K): '))
            tol = float(input('Tolerância para a pressão otimizada: '))
            resposta = 1
```

Já se a opção selecionada seja escolher uma substância do banco de dados, é fornecido o *dataFrame* das substâncias e os requisitos para a seleção:

```
# Caso o usuário não escolha a entrada manual de dados, exibe uma lista de substância.
# A partir da substância escolhida, o software retira os valores de um arquivo .xlsx para completar os dados.
if command == 'não' or command == 'n':

    dados = pd.read_excel('Dados_Subst_Puras.xlsx')
    substancias = dados["Substância"]
    print(substancias) # Exibe a lista de substâncias.
    i = float(input('Escolha uma substância de acordo com a lista acima (número): '))
    print(dados.loc[i]) # Exibe os dados que vão ser usados para a execução do algoritmo.
    # Atribui valores as variáveis, de acordo com a substância escolhida.
    tpt = ed.ler_dados_xlsx('Dados_Subst_Puras.xlsx',i)['tpt']
    tc = ed.ler_dados_xlsx('Dados_Subst_Puras.xlsx',i)['tc']
    pc = ed.ler_dados_xlsx('Dados_Subst_Puras.xlsx',i)['pc']
    w = ed.ler_dados_xlsx('Dados_Subst_Puras.xlsx',i)['w']
    r = 8.3145
    tol = 0.000001
    resposta = 1

if resposta == 0: # Caso a escolha esteja fora do oferecido pelo software, assume esse escolha como inválida.
    print('\nA opção inserida é inválida, escolha sua resposta de acordo com as escolhas dadas (sim/s ou não/n): ')
    resposta = 0
```

Após a definição de dados iniciais, é perguntado o tipo de aplicação desejada, podendo ser o cálculo de um único ponto ou a geração do gráfico com a curva de equilíbrio da substância definida anteriormente. Feita a escolha da aplicação, são solicitados os *inputs* necessários para a entrada no algoritmo.

```
# Cria outro operador para analisar e validar a resposta do usuário.
marcador = 0
# Enquanto a escolha do usuário não for válida, o software requisita uma nova escolha e orienta novamente o usuário.
while marcador == 0:
    escolha = input('\nO software será usado para o cálculo de um único ponto de pressão, ou de uma curva de equilíbrio(-ponto/p- ou -curva)')
    if escolha == 'ponto' or escolha == 'p':
        # Valida a resposta do usuário.
        marcador = 1
        # Pede os pontos de temperatura e pressão experimental
        t = float(input('Temperatura(K): '))
        p = float(input('Pressão experimental(Mpa): '))
        pexp=p

    if escolha == 'curva' or escolha == 'c':
        # Valida a resposta do usuário.
        marcador = 1
        # Orienta o usuário e requisita as sequências de dados que gerarão o gráfico da curva de equilíbrio de pressão.
        print('Observação: Devem ser fornecidos os mesmos números de pontos para pressão e temperatura.\n')
        t = input('Insira uma sequência de pontos para a temperatura(T1,T2,...,Tn): ')
        p = input('Insira uma sequência de pontos para a pressão experimental(P1,P2,...,Pn): ')
        # Organiza os dados em forma de listas para aplicá-los nos algoritmos.
        Tstr = t.split(',')
        T = list(map(float,Tstr))
        Pstr = p.split(',')
        P = list(map(float,Pstr))

# Assume como inválida a resposta e orienta o usuário novamente das escolhas possíveis.
if marcador == 0:
    print('\nEscolha inválida. As respostas devem ser \n"ponto"/\n"p" ou \n"curva"/\n"c".')
```

Caso a opção selecionada seja a geração do gráfico, são solicitados ainda *inputs* para os limites dos dados, precedidos por uma recomendação de valores para os limites em questão. Além disso, são definidos parâmetros iniciais e a função de linearização é aplicada para encontrar uma sequência de pontos para geração do gráfico baseada nos pontos experimentais fornecidos pelo usuário.

```
# Se o usuário optar pela curva de equilíbrio, é definido a equação linear para a suposição das pressões e os limites para a construção do gráfico.
if escolha == 'curva' or escolha == 'c':
    af, bf = lr.linearR(T,P)
    print('\nAviso: se os valores a seguir forem valores muito baixos, o algoritmo pode não convergir nos pontos próximos ao extremos.\n Rec')
    imin = int(input('Distância do ponto triplo de temperatura: '))
    imax = int(input('Distância do ponto crítico de temperatura: '))
    Imin = int(tpt) + imin
    Imax = int(tc) - imax

# Se for escolhida a opção ponto, é limitado o loop a uma única iteração.
if escolha == 'ponto' or escolha == 'p':
    Imin = 0
    Imax = 1

# Define os dados básicos para o algoritmo e as listas que armazenarão os valores para o gráfico.
r = 8.3145
tol = 0.000001
Tx = []
Py = []
# Aplica o algoritmo de acordo com as opções escolhidas, para vários pontos(curva) , ou para um único ponto.
for i in range(Imin,Imax):

    # Aplica a linearização para chutar valores de pressão em função da temperatura.
    if escolha == 'curva' or escolha == 'c':
        t = i
        p = af*t - bf

    # Se o chute for negativo, o chute é aproximado para um valor próximo de zero.
    if p < 0:
        p = 0.000002
```

Assim que todos os parâmetros iniciais estão definidos as funções do método são aplicadas para o cálculo dos valores da primeira iteração para a realização da otimização.

```
# Chama a função para o cálculo da temperatura reduzida.
tr = fsp.temp_reduzida(t,tc)

# Chama a função para o cálculo do parâmetro de auxílio m.
m = fsp.parametro_m(w)

# Chama a função para o cálculo do parâmetro de correção.
alpha = fsp.func_alpha(m,tr)

# Chama a função para o cálculo das constantes de correção dos parâmetros de atração (a) e repulsão (b).
a = fsp.constante_a(r,tc,pc,alpha)
b = fsp.constante_b(r,tc,pc)

# Chama a função para o cálculo dos parâmetros auxiliares da equação cúbica.
A = fsp.constante_A(a,p,r,t)
B = fsp.constante_B(b,p,r,t)

# Chama a função para o cálculo dos coeficientes cf1, cf2 e cf3 para o cálculo de R e Q.
cf1, cf2, cf3 = fsp.coeficientes_aux(A,B)

# Chama as funções para o cálculo dos parâmetros Q e R da equação cúbica.
Q = fsp.func_Q(cf1,cf2)
R = fsp.func_R(cf1,cf2,cf3)

# Chama a função que calcula o fator de análise para atribuir as raízes da equação cúbica.
fa = fsp.numero_raizes(Q,R)

# Chama a função que computa os fatores de compressibilidade do líquido(Zl) e do vapor(Zv).
Zl, Zv = fsp.calcula_fatorcompress(fa,R,Q,cf1,cf2,cf3)

# Chama a função que calcula os coeficientes de fugacidade do líquido(cl) e do vapor(cv).
cL, cV = fsp.coef_fugacidade(Zl,Zv,A,B)
```

Em seguida, os valores de cada ponto são testados, otimizados e armazenados em matrizes para a posterior geração do gráfico.

```
# Inicia um loop que otimiza o valor da pressão até que os coeficientes de fugacidade do líquido e do vapor sejam numericamente iguais.
while abs((cV/cL) - 1) > tol:
    # Recalcula os fatores de correção A e B.
    A = fsp.constante_A(a,p,r,t)
    B = fsp.constante_B(b,p,r,t)

    # Recalcula os coeficientes auxiliares da equação cúbica.
    cf1, cf2, cf3 = fsp.coeficientes_aux(A,B)

    # Recalcula os parâmetros Q e R da equação cúbica.
    Q = fsp.func_Q(cf1,cf2)
    R = fsp.func_R(cf1,cf2,cf3)

    # Computa novamente o fator de análise.
    fa = fsp.numero_raizes(Q,R)

    # Calcula novamente os fatores de compressibilidade do líquido e do vapor.
    Zl, Zv = fsp.calcula_fatorcompress(fa,R,Q,cf1,cf2,cf3)
    # Recalcula os coeficientes de fugacidade do líquido e do vapor.
    cL, cV = fsp.coef_fugacidade(Zl,Zv,A,B)

    # Otimiza o valor da pressão a partir da razão entre os coeficientes de fugacidade do líquido e do vapor.
    razao = cL/cV
    p = p*razao

# Armazena, caso seja escolhido a opção da curva, os valores das temperaturas experimentais e das pressões otimizadas para a construção do
Tx.append(t)
Py.append(p)
```

A seleção de otimização para um único ponto gera um arquivo contendo tanto os dados iniciais fornecidos quanto o valor otimizado e salva no diretório em que se encontra o programa.

```
# Se a opção escolhida foi "ponto", gera um arquivo no formato .txt para exibir os resultados e os dados usados no algoritmo.
if escolha == 'ponto' or escolha == 'p':
    print(f'A pressão de equilíbrio para essa temperatura é {p:.4f} MPa')

# Pede ao usuário um nome para o arquivo de resultados que vai ser gerado.
nomeResultados = input('escolha o nome do arquivo de resultados: ')
with open(nomeResultados, 'w') as arq:

    # Transforma o arquivo .xlsx com os dados das substâncias em um dataframe.
    dados = pd.read_excel('Dados_Subst_Puras.xlsx')

    # As linhas a seguir configuram o arquivo de resultados que será gerado.
    arq.write(' Dados experimentais e Resultados '.center(60, '#')+'\n')
    arq.write('\n ----- Dados experimentais ----- \n')
    arq.write(f'Pressão Crítica: {pc} MPa\n')
    arq.write(f'Temperatura Crítica: {tc} K\n')
    arq.write(f'Fator Acêntrico: {w}\n')
    arq.write(f'Temperatura da substância: {t} K\n')
    arq.write(f'Pressão Experimental: {pexp} MPa\n')
    arq.write('\n ----- Resultado ----- \n')
    arq.write(f'Pressão Calculada: {p:.4f} MPa\n')
    arq.write('#'*60+'\n')

# Avisa ao usuário que o arquivo de resultados foi gerado.
print('Um arquivo com o nome escolhido foi gerado no diretório em que se encontra o TermoPython')
```

Caso a escolha tenha sido a geração de curva, é configurado um gráfico e salvo em formato .png no diretório em que se encontra o programa.

```
# Caso a opção escolhida tenha sido "curva", configura o gráfico da curva de pressão.
if escolha == 'curva' or escolha == 'c':
    plt.figure()
    plt.grid()
    plt.title('Pressão de equilíbrio de fase x Temperatura')
    plt.plot(Tx,Py)
    plt.xlabel('Temperatura (K)')
    plt.ylabel('Pressão de equilíbrio(MPa)')
    plt.savefig('gráfico')
    plt.show()

# Avisa ao usuário que o gráfico gerado foi baixado e onde foi baixado.
print('\n O gráfico foi baixado em formato .png no diretório em que se encontra o TermoPython.')
```

Outra possível utilização do programa TermoPython é a aplicação do método para Misturas Binárias, caso essa opção seja selecionada inicialmente pelo usuário, da mesma forma que a aplicação de substâncias puras, é possível fazer inserir manualmente os dados do sistema ou selecionar um sistema do banco de dados do programa. Caso a opção seja inserir dados manualmente é exibida a sequência de *inputs* dos dados necessários para a aplicação do método.


```

if algoritmo == 'mistura binária' or algoritmo == 'mb':
    # Atualiza a escolha do sistema como válida.
    valido = 1

    # Cria um operador para analisar a escolha da forma de input de dados.
    resposta2 = 0

    # Enquanto a resposta for inválida, pede uma nova resposta ao usuário.
    while resposta2 == 0:
        command = input('Deseja inserir os dados da substância manualmente?(sim/s ou não/n): ')

        # Se o usuário escolher por inserir os dados manualmente, requisita a ele dado por dado.
        if command == 'sim' or command == 's':
            k = 1.13e-4
            t = float(input('Temperatura do sistema(K): '))
            r = 0.08205
            tc1 = float(input('Temperatura crítica do 1º componente(K): '))
            tc2 = float(input('Temperatura crítica do 2º componente(K): '))
            pc1 = float(input('Pressão crítica do 1º componente(atm): '))
            pc2 = float(input('Pressão crítica do 2º componente(atm): '))
            w1 = float(input('fator acêntrico do 1º componente: '))
            w2 = float(input('fator acêntrico do 2º componente: '))
            # Atualiza a resposta como válida.
            resposta2 = 1

        # Caso o usuário use a entrada a partir da base de dados, os valores serão atribuídos às variáveis a partir de um arquivo no formato .xlsx.
        if command == 'não' or command == 'n':
            # Transforma o arquivo .xlsx em um dataframe.
            dados = pd.read_excel('Dados_Misturas.xlsx')

```

A opção de escolha de um sistema a partir do *dataFrame* é realizada da mesma forma que para substâncias puras, é exibida uma lista de sistemas e o usuário fornece como *input* o número equivalente ao sistema desejado.

```

# Exibe a lista de sistemas disponíveis para a entrada automática dos dados característicos do sistema.
sistema = dados["Sistema"]
print(sistema)
i = float(input('Selecione um sistema de acordo com a lista acima (número): '))
print(f'{dados.loc[i]}\n\n')

# A partir da escolha do usuário, atribui os valores das variáveis por meio do arquivo .xlsx.
t = float(input('Temperatura do sistema(K): '))
k = 1.13e-4
r = 0.08205
tc1 = ed.ler_dados_mistura_xlsx('Dados_Misturas.xlsx', i)['tc1']
tc2 = ed.ler_dados_mistura_xlsx('Dados_Misturas.xlsx', i)['tc2']
pc1 = ed.ler_dados_mistura_xlsx('Dados_Misturas.xlsx', i)['pc1']
pc2 = ed.ler_dados_mistura_xlsx('Dados_Misturas.xlsx', i)['pc2']
w1 = ed.ler_dados_mistura_xlsx('Dados_Misturas.xlsx', i)['w1']
w2 = ed.ler_dados_mistura_xlsx('Dados_Misturas.xlsx', i)['w2']

# Assume a escolha do usuário como válida.
resposta2 = 1

# Caso a resposta continue inválida orienta o usuário novamente sobre o input.
if resposta2 == 0:
    print('\nA opção inserida é inválida, escolha sua resposta de acordo com as escolhas dadas (sim/s ou não/n).')

```

A aplicação de misturas binárias pode ser realizada para a otimização de um único ponto ou para a geração de um gráfico contendo as curvas de Bolha P e Orvalho P, essa opção é dada para o usuário e de acordo com a escolha são solicitados os valores de pontos experimentais necessários para cada aplicação. Há ainda um marcador de retorno para eventuais respostas inválidas.


```
# Enquanto a escolha n for de acordo com o proposto, pede novamente uma resposta ao usuário.
while marcador2 == 0:
    escolha = input('\nQual cálculo será efetuado, de um único ponto, ou de uma curva de equilíbrio(-ponto/p- ou -curva/c-): ')
    if escolha == 'ponto' or escolha == 'p':
        # Atualiza a resposta como válida.
        marcador2 = 1

        # Pede os pontos experimentais para calcular a pressão otimizada.
        p = float(input('Pressão experimental(atm): '))
        pexp = p
        x1 = float(input('Fração molar da fase líquida: '))
        y1 = float(input('Fração molar da fase vapor: '))

    if escolha == 'curva' or escolha == 'c':
        # Atualiza a resposta como válida.
        marcador2 = 1

        # Pede a sequência de dados experimentais para aplicar ao algoritmo.
        print('Observação: Devem ser fornecidos a mesma quantidade de pontos para pressão e para cada fração molar.\n')
        p = input('Insira uma sequência de pontos para a pressão experimental(P1,P2,...,Pn): ')
        x = input('Insira uma sequência de pontos para a fração molar da fase líquida(X1,X2,...,Xn): ')
        y = input('Insira uma sequência de pontos para a fração molar da fase vapor(Y1,Y2,...,Yn): ')

        # Separa os dados em forma de listas para serem processados no algoritmo.
        Pstr = p.split(',')
        P = list(map(float,Pstr))
        Xstr = x.split(',')
        X1 = list(map(float,Xstr))
        Ystr = y.split(',')
        Y1 = list(map(float,Ystr))

    # Se a escolha do usuário for inválida, ele é orientado sobre a forma correta de escolha e o input é requisitado novamente.
    if marcador2 == 0:
        print('\nEscolha inválida. As respostas devem ser \"ponto\"/\"p\" ou \"curva\"/\"c\".')

```

Caso seja solicitada a otimização de um único ponto, o método é aplicado utilizando as funções tanto do módulo de substâncias puras quanto de misturas binárias.

```
# Aplica o algoritmo para a escolha "ponto".
if escolha == 'ponto' or escolha == 'p':

    # Chama a função para o cálculo da temperatura reduzida dos dois componentes do sistema.
    tr1 = fsp.temp_reduzida(t,tc1)
    tr2 = fsp.temp_reduzida(t, tc2)

    # Chama a função para o cálculo dos parâmetros de auxílio m dos dois componentes do sistema.
    m1 = fsp.parametro_m(w1)
    m2 = fsp.parametro_m(w2)

    # Chama a função para o cálculo dos parâmetros de correção de cada componente do sistema.
    alpha_1 = fsp.func_alpha(m1,tr1)
    alpha_2 = fsp.func_alpha(m2,tr2)

    # Chama a função para o cálculo das constantes de correção do parâmetro de atração de cada componente.
    a1 = fsp.constante_a(r,tc1,pc1,alpha_1)
    a2 = fsp.constante_a(r,tc2,pc2,alpha_2)

    # Chama a função para o cálculo das constantes de correção do parâmetro de repulsão de cada componente.
    b1 = fsp.constante_b(r,tc1,pc1)
    b2 = fsp.constante_b(r,tc2,pc2)

    # Cálculo das frações molares complementares da fase líquida(x2) e da fase vapor(y2), a partir das frações fornecidas pelo usuário.
    x2 = 1 - x1
    y2 = 1 - y1

    #Arrays que armazenam os valores das frações molares de cada fase do sistema.
    Mx = np.array([x1,x2], dtype = float)
    My = np.array([y1,y2], dtype = float)

    # Array que armazena a temperatura reduzida de cada componente da mistura.
    Mtr = np.array([tr1,tr2], dtype = float)

```

```
# Array que armazena o valor dos parâmetros de correção de cada componente.
Malpha = np.array([alpha_1,alpha_2], dtype = float)

# Array que armazena os valores dos parâmetros auxiliares dos componentes do sistema.
Mm = np.array([m1,m2], dtype = float)

# Arrays que armazenam as constantes de correção dos fatores de atração(Mai) e de repulsão(Mbi).
Mai = np.array([a1,a2], dtype = float)
Mbi = np.array([b1,b2], dtype = float)

# Array de zeros 2 x 2 que vai armazenar os parâmetros aii e aij, que são usados para o cálculo do parâmetro de atração do líquido e do vapor.
Ma = np.array([[0,0],
               [0,0]],dtype = float)

# loop que realiza os cálculos dos parâmetros para os dois componentes.
for i in range(0,2):
    j = 1 - i

    # Cálculo, respectivamente, do parâmetro aii e do parâmetro aij.
    Ma[i][i] = Mai[i]
    Ma[i][j] = fm.parametro_aij(a1,a2,k)

# Chama a função que calcula os parâmetros de atração da fase líquida(al) e da fase vapor(av).
al = fm.parametro_atract(Ma,Mx,My)['al']
av = fm.parametro_atract(Ma,Mx,My)['av']

# Chama a função que calcula os parâmetros de repulsão da fase líquida(bl) e da fase vapor(bv).
bl = fm.parametro_repulse(Mbi,Mx,My)['bl']
bv = fm.parametro_repulse(Mbi,Mx,My)['bv']

# Chama a função que realiza o cálculo da constante A de auxílio do líquido(Al) e do vapor(Av) da equação cúbica.
Al = fsp.constante_A(al,p,r,t)
Av = fsp.constante_A(av,p,r,t)
```

```
# Chama a função que realiza o cálculo da constante B de auxílio do líquido(Bl) e do vapor(Bv) da equação cúbica.
Bl = fsp.constante_B(bl,p,r,t)
Bv = fsp.constante_B(bv,p,r,t)

# Arrays que armazenam as constantes de auxílio A e B da equação cúbica.
MA = np.array([Al,Av])
MB = np.array([Bl,Bv])

# Chama a função que computa, a partir das raízes da equação cúbica.
Zl = fm.fatorcompress(MA,MB)['l']
Zv = fm.fatorcompress(MA,MB)['v']

# Criação de arrays de zeros para armazenar os valores dos logaritmos dos coeficientes de fugacidade do líquido e do vapor e de seus respectivos fatores de compressibilidade.
Mln_l = np.array([0,0], dtype = float)
Mln_v = np.array([0,0], dtype = float)
Mphi_l = np.array([0,0], dtype = float)
Mphi_v = np.array([0,0], dtype = float)

# Chama a função que calcula os coeficientes de fugacidade do líquido e do vapor.
Mphi_l = fm.coef_fugacidade(Zl,Zv,Mx,My,al,av,bl,bv,MA,Mbi,MA,MB)['l']
Mphi_v = fm.coef_fugacidade(Zl,Zv,Mx,My,al,av,bl,bv,MA,Mbi,MA,MB)['v']

# Chama a função que calcula a fugacidade do líquido e do vapor a partir dos coeficientes de fugacidade.
Mfl = fm.fugacidade(Mphi_l,Mphi_v,Mx,My,p)['l']
Mfv = fm.fugacidade(Mphi_l,Mphi_v,Mx,My,p)['v']

# Chama a função que otimiza o valor das frações molares do vapor.
MyI = fm.y_otimizado(My,Mfl,Mfv)

# Laço recursivo que repete o algoritmo até que os valores da pressão e da fração molar do vapor estejam totalmente otimizados.
while abs((Mfl[i]/Mfv[i])-1) > 1e-14:

    # Otimiza o valor da pressão.
    p = p*np.sum(MyI)
```

```
# Troca os valores antigos da fração molar do vapor(y) de cada componente, pelos valores otimizados a cada iteração do while.
for i in range(0,2):
    My[i] = MyI[i]

# Refaz o algoritmo desde os cálculos dos parâmetros de atração e de repulsão para cada iteração do laço.
al = fm.parametro_atract(Ma,Mx,My)['al']
av = fm.parametro_atract(Ma,Mx,My)['av']

bl = fm.parametro_repulse(Mbi,Mx,My)['bl']
bv = fm.parametro_repulse(Mbi,Mx,My)['bv']

Al = fsp.constante_A(al,p,r,t)
Av = fsp.constante_A(av,p,r,t)

Bl = fsp.constante_B(bl,p,r,t)
Bv = fsp.constante_B(bv,p,r,t)

MA = np.array([Al,Av])
MB = np.array([Bl,Bv])

Zl = fm.fatorcompress(MA,MB)['L']
Zv = fm.fatorcompress(MA,MB)['v']

Mln_l = np.array([0,0], dtype = float)
Mln_v = np.array([0,0], dtype = float)
Mphi_l = np.array([0,0], dtype = float)
Mphi_v = np.array([0,0], dtype = float)

Mphi_l = fm.coef_fugacidade(Zl,Zv,Mx,My,al,av,bl,bv,Ma,Mbi,MA,MB)['L']
Mphi_v = fm.coef_fugacidade(Zl,Zv,Mx,My,al,av,bl,bv,Ma,Mbi,MA,MB)['v']
```

Após a otimização dos valores de pressão e fração molar é gerado um arquivo em formato de relatório, salvo no diretório em que se encontra o programa, contendo todos os dados constantes, experimentais e otimizados.

```
Mfl = fm.fugacidade(Mphi_l,Mphi_v,Mx,My,p)['L']
Mfv = fm.fugacidade(Mphi_l,Mphi_v,Mx,My,p)['v']

MyI = fm.y_otimizado(My,Mfl,Mfv)

# Exibe a pressão totalmente otimizada obtida ao fim da recursão.
print(f'\nA pressão de equilíbrio para essa temperatura é {p:.4f} atm')

# Requisita ao usuário um nome para o arquivo .txt que exibirá os resultados e os dados usados no algoritmo.
nomeResultados = input('escolha o nome do arquivo de resultados: ')
with open(nomeResultados, 'w') as arq:

    # Constrói os dados usados e os resultados obtidos dentro do arquivo .txt.
    arq.write('Dados experimentais e Resultados'.center(60, '#')+'\n')
    arq.write('\n ----- Dados experimentais ----- \n')
    arq.write(f'Pressão Crítica do Componente 1: {pc1} atm\n')
    arq.write(f'Pressão Crítica do Componente 2: {pc2} atm \n')
    arq.write(f'Temperatura Crítica do Componente 1: {tc1} K\n')
    arq.write(f'Temperatura Crítica do Componente 2: {tc2} K\n')
    arq.write(f'Fator Acêntrico do Componente 1: {w1}\n')
    arq.write(f'Fator Acêntrico do Componente 2: {w2}\n')
    arq.write(f'Temperatura do Sistema: {t} K\n')
    arq.write(f'Fração Molar Experimental da Fase Líquida: {x1}\n')
    arq.write(f'Fração Molar Experimental da Fase Vapor: {y1}\n')
    arq.write(f'Pressão Experimental: {pexp} atm\n')
    arq.write('\n ----- Resultado ----- \n')
    arq.write(f'Pressão Calculada: {p:.4f} atm\n')
    arq.write(f'Fração Calculada da Fase Vapor: {MyI[0]:.4f}\n')
    arq.write('#'*60+'\n')

# Avisa o usuário que o arquivo foi gerado e onde ele foi gerado.
print(f'O arquivo {nomeResultados} foi gerado no diretório em que se encontra o TermoPython')
```

A seleção da opção curva aplica também as funções definidas nos módulos tanto de substâncias puras quanto de misturas binárias, para uma sequência de pontos.

```
# Se opção escolhida for "curva", realiza o algoritmo para a sequência de pontos fornecidas pelo usuário.
if escolha == 'curva' or escolha == 'c':

    # Realiza o algoritmo para cada ponto fornecido pelo usuário.
    for i in range(len(P)):
        x1 = X1[i]
        y1 = Y1[i]
        p = P[i]

        # Chama a função para o cálculo da temperatura reduzida dos dois componentes do sistema.
        tr1 = fsp.temp_reduzida(t,tc1)
        tr2 = fsp.temp_reduzida(t, tc2)

        # Chama a função para o cálculo dos parâmetros de auxílio m dos dois componentes do sistema.
        m1 = fsp.parametro_m(w1)
        m2 = fsp.parametro_m(w2)

        # Chama a função para o cálculo dos parâmetros de correção de cada componente do sistema.
        alpha_1 = fsp.func_alpha(m1,tr1)
        alpha_2 = fsp.func_alpha(m2,tr2)

        # Chama a função para o cálculo das constantes de correção do parâmetro de atração de cada componente.
        a1 = fsp.constante_a(r,tc1,pc1,alpha_1)
        a2 = fsp.constante_a(r,tc2,pc2,alpha_2)

        # Chama a função para o cálculo das constantes de correção do parâmetro de repulsão de cada componente.
        b1 = fsp.constante_b(r,tc1,pc1)
        b2 = fsp.constante_b(r,tc2,pc2)

        # Cálculo das frações molares complementares da fase líquida(x2) e da fase vapor(y2), a partir das frações fornecidas pelo usuário.
        x2 = 1 - x1
        y2 = 1 - y1
```

```
#Arrays que armazenam os valores das frações molares de cada fase do sistema.
Mx = np.array([x1,x2], dtype = float)
My = np.array([y1,y2], dtype = float)

# Array que armazena a temperatura reduzida de cada componente da mistura.
Mtr = np.array([tr1,tr2], dtype = float)

# Array que armazena o valor dos parâmetros de correção de cada componente.
Malpha = np.array([alpha_1,alpha_2], dtype = float)

# Array que armazena os valores dos parâmetros auxiliares dos componentes do sistema.
Mm = np.array([m1,m2], dtype = float)

# Arrays que armazenam as constantes de correção dos fatores de atração(Mai) e de repulsão(Mbi).
Mai = np.array([a1,a2], dtype = float)
Mbi = np.array([b1,b2], dtype = float)

# Array de zeros 2 x 2 que vai armazenar os parâmetros aii e aij, que são usados para o cálculo do parâmetro de atração do líquido e d
Ma = np.array([[0,0],
               [0,0]],dtype = float)

# laço que realiza os cálculos dos parâmetros aii e aij para os dois componentes.
for i in range(0,2):
    j = 1 - i

    Ma[i][i] = Mai[i]
    Ma[i][j] = fm.parametro_aij(a1,a2,k)

# Chama a função que calcula os parâmetros de atração da fase líquida(al) e da fase vapor(av).
al = fm.parametro_atract(Ma,Mx,My)['al']
av = fm.parametro_atract(Ma,Mx,My)['av']
```

```
# Chama a função para o cálculo das constantes de correção do parâmetro de repulsão de cada componente.
bl = fm.parametro_repulse(Mbi,Mx,My)['bl']
bv = fm.parametro_repulse(Mbi,Mx,My)['bv']

# Chama a função que realiza o cálculo da constante A de auxílio do líquido(Al) e do vapor(Av) da equação cúbica.
Al = fsp.constante_A(al,p,r,t)
Av = fsp.constante_A(av,p,r,t)

# Chama a função que realiza o cálculo da constante B de auxílio do líquido(Bl) e do vapor(Bv) da equação cúbica.
Bl = fsp.constante_B(bl,p,r,t)
Bv = fsp.constante_B(bv,p,r,t)

# Arrays que armazenam as constantes de auxílio A e B da equação cúbica.
MA = np.array([Al,Av])
MB = np.array([Bl,Bv])

# Chama a função que computa, a partir das raízes da equação cúbica.
Zl = fm.fatorcompress(MA,MB)['L']
Zv = fm.fatorcompress(MA,MB)['v']

# Criação de arrays de zeros para armazenar os valores dos logaritmos dos coeficientes de fugacidade do líquido e do vapor e de seus
Mln_l = np.array([0,0], dtype = float)
Mln_v = np.array([0,0], dtype = float)
Mphi_l = np.array([0,0], dtype = float)
Mphi_v = np.array([0,0], dtype = float)

# Chama a função que calcula os coeficientes de fugacidade do líquido e do vapor.
Mphi_l = fm.coef_fugacidade(Zl,Zv,Mx,My,al,av,bl,bv,Ma,Mbi,MA,MB)['L']
Mphi_v = fm.coef_fugacidade(Zl,Zv,Mx,My,al,av,bl,bv,Ma,Mbi,MA,MB)['v']

# Chama a função que calcula a fugacidadde do líquido e do vapor a partir dos coeficientes de fugacidade.
Mfl = fm.fugacidade(Mphi_l,Mphi_v,Mx,My,p)['L']
Mfv = fm.fugacidade(Mphi_l,Mphi_v,Mx,My,p)['v']
```

```
# Chama a função que otimiza o valor das frações molares do vapor.
MyI = fm.y_otimizado(My,Mfl,Mfv)

# Laço recursivo que repete o algoritmo até que os valores da pressão e da fração molar do vapor estejam totalmente otimizados.
while abs((Mfl[i]/Mfv[i])-1) > 1e-14:

    # Otimiza o ponto de pressão
    p = p*np.sum(MyI)

    # Troca os valores antigos da fração molar do vapor(y) de cada componente, pelos valores otimizados a cada iteração do while.
    for i in range(0,2):
        My[i] = MyI[i]

    # Refaz o algoritmo desde os cálculos dos parâmetros de atração e de repulsão para cada iteração do laço.
    al = fm.parametro_atract(Ma,Mx,My)['al']
    av = fm.parametro_atract(Ma,Mx,My)['av']

    bl = fm.parametro_repulse(Mbi,Mx,My)['bl']
    bv = fm.parametro_repulse(Mbi,Mx,My)['bv']

    Al = fsp.constante_A(al,p,r,t)
    Av = fsp.constante_A(av,p,r,t)

    Bl = fsp.constante_B(bl,p,r,t)
    Bv = fsp.constante_B(bv,p,r,t)

    MA = np.array([Al,Av])
    MB = np.array([Bl,Bv])

    Zl = fm.fatorcompress(MA,MB)['L']
    Zv = fm.fatorcompress(MA,MB)['v']
```

```
Mln_l = np.array([0,0], dtype = float)
Mln_v = np.array([0,0], dtype = float)
Mphi_l = np.array([0,0], dtype = float)
Mphi_v = np.array([0,0], dtype = float)

Mphi_l = fm.coef_fugacidade(Zl,Zv,Mx,My,al,av,bl,bv,Ma,Mbi,MA,MB)['L']
Mphi_v = fm.coef_fugacidade(Zl,Zv,Mx,My,al,av,bl,bv,Ma,Mbi,MA,MB)['v']

Mfl = fm.fugacidade(Mphi_l,Mphi_v,Mx,My,p)['L']
Mfv = fm.fugacidade(Mphi_l,Mphi_v,Mx,My,p)['v']

MyI = fm.y_otimizado(My,Mfl,Mfv)

Py.append(p)
```

Após a otimização dos valores, o gráfico é configurado e gerado, sendo salvo em formato *.png* no diretório em que se encontra o programa.

```
# Avisa o usuário sobre o local em que será baixado o código.
print('\n O gráfico foi baixado em formato .png no diretório em que se encontra o TermoPython.')

# Configura o gráfico com as curvas de P em função de x1 e P em função de y1 e com os pontos experimentais de pressão respectivos para ca
plt.figure()
plt.plot(X1,Py,label='Peq. x Fração molar do líquido',color='b')
plt.plot(Y1,Py,label='Peq. x Fração molar do vapor',color='r')
plt.plot(X1,P,'o',label='Pex. x Fração molar do líquido',color='m')
plt.plot(Y1,P,'o',label='Pex. x Fração molar do vapor',color='k')
plt.title('Gráfico de comparação das pressões de equilíbrio(Peq.) e experimental(Pex.)\n')
plt.grid()
plt.xlabel('Pressão (atm)')
plt.ylabel('Fração molar')
plt.legend(fontsize='small')
plt.savefig('gráfico')
plt.show()

# Se a opção escolhida pelo usuário não for aceita, e requisitada uma nova escolha.
if valido == 0:
    print('\n Resposta inválida. Selecciona o uso dentre as opções dadas.')
```

4. Bibliotecas Registradas

As bibliotecas registradas utilizadas no código são: *Numpy*, *Pandas* e *Matplotlib*

5. Referência Bibliográfica

PRAUSNITZ, John M; LICHTENTHALER, Rudiger N; AZEVEDO, Edmund Gomes de. **Molecular Thermodynamics of Fluid-Phase Equilibria**. 3. ed. New Jersey: Prentice Hall Ptr, 1999. 886 p.