import math  
import scipy.optimize  
from math import log as ln  
import numpy as np  
import os  
import matplotlib.pyplot as plt  
  
# уравнение Рашфорда – Райса  
def f(x, z, K):  
 res = 0  
 for i in range(len(z)):  
 res += z[i] \* (K[i] - 1) / (1 + x \* (K[i] - 1))  
 return res  
  
# Избавление от компл. корней кубического ур-я  
def no\_compl(array):  
 ans = []  
 for found\_root in array:  
 if found\_root == found\_root.real:  
 ans.append(found\_root.real)  
 return ans  
  
# Метод биссекции  
def bisection(a, b, eps, z, K):  
 iter\_b = 0  
 root = None  
 while abs(f(b, z, K) - f(a, z, K)) > eps:  
 mid = (a + b) / 2  
 if f(mid, z, K) == 0 or abs(f(mid, z, K)) < eps:  
 root = mid  
 break  
 elif f(a, z, K) \* f(mid, z, K) < 0:  
 b = mid  
 else:  
 a = mid  
 iter\_b+=1  
 if root is None:  
 print('Корень не найден')  
  
 return root  
  
def SRK\_method(z, w, P\_c, T\_c,P,T, c\_ij):  
 *'''Объявим все нужные массивы и создадим их нужного размера'''* N = len(z)  
 K = []; K\_1 = []; K\_0 = []  
 alpha = []; m = []; y = []; x = []  
 B\_v = 0.0; A\_v = 0.0; B\_l = 0.0; A\_l = 0.0  
 F\_v = []; F\_l = []; check = [];  
 T\_r = []; P\_r = [];  
 A\_p = []; B\_p = [];  
 eps = 0.00001;  
 A\_v = 0; B\_v = 0; A\_l = 0; B\_l = 0;  
 k1 = []; k2 = []; k3 = [];  
  
 '''Вычислим значение коэф-ов m, alpha, K\_0'''  
 for i in range(N):  
 T\_r.append(T / T\_c[i])  
 P\_r.append(P / P\_c[i])  
 m.append(0.48 + 1.574 \* w[i] - 0.176 \* w[i] \*\* 2)  
 alpha.append((1 + m[i] \* (1 - math.sqrt(T\_r[i]))) \*\* 2)  
 K.append((P\_c[i] / P) \* math.exp(5.37 \* (1 + w[i]) \* (1 - (T\_c[i] / T)))) # Начальное значение K по корреляции Вильсона  
  
 '''Далее проведем одну итерацию вручную, чтобы получить 2 значения K1 и K0'''  
 # Решим уравнение Рашфорда – Райса  
 V = bisection(1 / (1 - max(K)) + 0.0001, 1 / (1 - min(K)) - 0.0001, 0.0001, z, K)  
  
 # Вычислим y\_i и x\_i и сразу же проверим условие их сумм = 1  
 sum\_x = 0; sum\_y = 0; sum\_z = 0  
 for i in range(N):  
 y.append(z[i] \* K[i] / (V \* (K[i] - 1) + 1))  
 x.append(y[i] / K[i])  
 sum\_x += x[i]; sum\_y += y[i]; sum\_z += z[i]  
  
 # Вычислим коэффициенты A\_p, B\_p и сразу же B\_v и B\_l  
 for i in range(N):  
 A\_p.append(0.42747 \* alpha[i] \* P\_r[i] / T\_r[i] \*\* 2)  
 B\_p.append(0.08664 \* P\_r[i] / T\_r[i])  
 B\_v += y[i] \* B\_p[i]  
 B\_l += x[i] \* B\_p[i]  
  
 for i in range(N):  
 for j in range(N):  
 A\_v += y[i] \* y[j] \* (1-c\_ij[i][j]) \* math.sqrt(A\_p[i] \* A\_p[j])  
 A\_l += x[i] \* x[j] \* (1-c\_ij[i][j]) \* math.sqrt(A\_p[i] \* A\_p[j])  
  
 # Ищем решение кубического уравнения относительно Z\_v и Z\_l  
  
 pol\_K\_y = [1, -1, 0, 0]  
 pol\_K\_y[2] = A\_v - B\_v - B\_v \*\* 2  
 pol\_K\_y[3] = -A\_v \* B\_v  
 z\_fact\_y = np.roots(pol\_K\_y)  
 z\_fact\_y = no\_compl(z\_fact\_y)  
 Z\_v\_max = round(max(z\_fact\_y), 6) # Корень ур-я Z\_v  
 pol\_K\_x = [1, -1, 0, 0]  
 pol\_K\_x[2] = A\_l - B\_l - B\_l \*\* 2  
 pol\_K\_x[3] = -A\_l \* B\_l  
 z\_fact\_x = np.roots(pol\_K\_x)  
 z\_fact\_x = no\_compl(z\_fact\_x)  
 Z\_l\_min = round(min(z\_fact\_x), 6) # Корень ур-я Z\_l  
  
 K\_0 = K.copy() # Начальное значение, вычисленное по корреляции Вильсона  
  
 for i in range(N):  
 k1.append(((Z\_v\_max - 1) \* B\_p[i] / B\_v))  
 k2.append((ln(Z\_v\_max - B\_v)))  
 k3.append(((A\_v / B\_v) \* (2 \* math.sqrt(A\_p[i] / A\_v) - B\_p[i] / B\_v) \* ln((Z\_v\_max + B\_v) / Z\_v\_max)))  
 F\_v.append(math.exp(k1[i] - k2[i] - k3[i]))  
  
 k1[i] = ((Z\_l\_min - 1) \* B\_p[i] / B\_l)  
 k2[i] = (ln(Z\_l\_min - B\_l))  
 k3[i] = ((A\_l / B\_l) \* (2 \* math.sqrt(A\_p[i] / A\_l) - B\_p[i] / B\_l) \* ln((Z\_l\_min + B\_l) / Z\_l\_min))  
 F\_l.append(math.exp(k1[i] - k2[i] - k3[i]))  
  
 K\_1.append(F\_l[i] / F\_v[i])  
 check.append(abs(K\_1[i] - K\_0[i]) / abs(K\_0[i]))  
 '''На этом шаге мы получили значения K\_1 и K\_0, с которых мы далее начнем проверять условие сходимости '''  
 # print(K\_0)  
 # print(K\_1)  
 iter = 0  
 '''Непосредственно сам расчетный цикл'''  
 while max(check) >= eps: # пока максимальная ошибка > eps  
 B\_v = 0; B\_l = 0; A\_v = 0; A\_l = 0  
 '''K\_1 - текущее значение, K\_0 - предыдущее значение'''  
 # Решим уравнение Рашфорда – Райса  
 V = bisection(1 / (1 - max(K\_1)) + 0.00001, 1 / (1 - min(K\_1)) - 0.00001, 0.0001, z, K\_1)  
  
 # Вычислим y\_i и x\_i и сразу же проверим условие их сумм = 1  
 sum\_x = 0; sum\_y = 0; sum\_z = 0  
 for i in range(N):  
 y[i] = z[i] \* K\_1[i] / (V \* (K\_1[i] - 1) + 1)  
 x[i] = y[i] / K\_1[i] # z[i]/(V \* (K[i] - 1) + 1)  
 sum\_x += x[i]; sum\_y += y[i]; sum\_z += z[i]  
 # print("Суммы x, y", sum\_x, sum\_y)  
 # Вычислим коэффициенты A\_p, B\_p и сразу же B\_v и B\_l  
 for i in range(N):  
 A\_p[i] = 0.42747 \* alpha[i] \* P\_r[i] / T\_r[i] \*\* 2  
 B\_p[i] = 0.08664 \* P\_r[i] / T\_r[i]  
 B\_v += y[i] \* B\_p[i]  
 B\_l += x[i] \* B\_p[i]  
  
 for i in range(N):  
 for j in range(N):  
 A\_v += y[i] \* y[j] \* math.sqrt(A\_p[i] \* A\_p[j])  
 A\_l += x[i] \* x[j] \* math.sqrt(A\_p[i] \* A\_p[j])  
  
 # Ищем решение кубического уравнения относительно Z\_v и Z\_l  
 pol\_K\_y = [1, -1, 0, 0]  
 pol\_K\_y[2] = A\_v - B\_v - B\_v \*\* 2  
 pol\_K\_y[3] = -A\_v \* B\_v  
 z\_fact\_y = np.roots(pol\_K\_y)  
 z\_fact\_y = no\_compl(z\_fact\_y)  
 Z\_v\_max = round(max(z\_fact\_y), 6) # Корень ур-я Z\_v  
  
 pol\_K\_x = [1, -1, 0, 0]  
 pol\_K\_x[2] = A\_l - B\_l - B\_l \*\* 2  
 pol\_K\_x[3] = -A\_l \* B\_l  
 z\_fact\_x = np.roots(pol\_K\_x)  
 z\_fact\_x = no\_compl(z\_fact\_x)  
 Z\_l\_min = round(min(z\_fact\_x), 6) # Корень ур-я Z\_l  
  
 # Тут я буду пересчитывать K -> мне нужно запомнить предыдущий шаг -> K\_0\_i = K\_1\_i  
 K\_0 = K\_1.copy()  
 for i in range(N):  
 # Нахождение F\_v  
 k1[i] = ((Z\_v\_max - 1) \* B\_p[i] / B\_v)  
 k2[i] = (ln(Z\_v\_max - B\_v))  
 k3[i] = ((A\_v / B\_v) \* (2 \* math.sqrt(A\_p[i] / A\_v) - B\_p[i] / B\_v) \* ln((Z\_v\_max + B\_v) / Z\_v\_max))  
 F\_v[i] = math.exp(k1[i] - k2[i] - k3[i])  
 # Нахождение F\_l  
 k1[i] = ((Z\_l\_min - 1) \* B\_p[i] / B\_l)  
 k2[i] = (ln(Z\_l\_min - B\_l))  
 k3[i] = ((A\_l / B\_l) \* (2 \* math.sqrt(A\_p[i] / A\_l) - B\_p[i] / B\_l) \* ln((Z\_l\_min + B\_l) / Z\_l\_min))  
 F\_l[i] = math.exp(k1[i] - k2[i] - k3[i])  
 # Нахождение K  
 K\_1[i] = F\_l[i] / F\_v[i]  
 check[i] = abs(K\_1[i] - K\_0[i]) / abs(K\_0[i]) # Норма  
 iter += 1  
  
 '''Проверка некоторых выражений'''  
 #print("Iter:", iter)  
 #print("Summ\_z: ", sum\_z, "Summ\_y: ", sum\_y, "Summ\_x: ", sum\_x)  
 root\_r = 0  
 for i in range(N):  
 root\_r += z[i] \* (K\_1[i] - 1) / (1 + V \* (K\_1[i] - 1))  
 #print("Подставим найденное V в уравнение Рашфорда – Райса:", root\_r)  
 L = (sum\_z - sum\_y \* V) / sum\_x  
 #L = 1 - V  
 #print("L: ",L , " -> следовательно жидкая фаза существует!")  
  
  
 return L, K\_1  
  
def find\_root\_3 (A\_v, A\_l, B\_v, B\_l):  
 pol\_K\_y = [1, -1, 0, 0]  
 pol\_K\_y[2] = A\_v - B\_v - B\_v \*\* 2  
 pol\_K\_y[3] = -A\_v \* B\_v  
 z\_fact\_y = np.roots(pol\_K\_y)  
 z\_fact\_y = no\_compl(z\_fact\_y)  
 Z\_v\_max = round(max(z\_fact\_y), 6) # Корень ур-я Z\_v  
 pol\_K\_x = [1, -1, 0, 0]  
 pol\_K\_x[2] = A\_l - B\_l - B\_l \*\* 2  
 pol\_K\_x[3] = -A\_l \* B\_l  
 z\_fact\_x = np.roots(pol\_K\_x)  
 z\_fact\_x = no\_compl(z\_fact\_x)  
 Z\_l\_min = round(min(z\_fact\_x), 6) # Корень ур-я Z\_l  
  
 return Z\_v\_max, Z\_l\_min  
  
# Свой range с не целыми шагами  
def frange(x, y, step):  
 while x < y:  
 yield x  
 x += step  
  
def graph(z, w, P\_c, T\_c, P, T, c\_ij):  
 fig = plt.figure()  
 plt.xlabel(u'Давление [МПа]', fontsize=12)  
 plt.ylabel(u'Процент жидкой фазы [%]', fontsize=12)  
 plt.title(u'Зависимость процента жидкой фазы от давления L(P)', fontsize=12)  
  
  
 result\_SRK = []  
 result\_brusilovski = []  
 P\_arr = []  
 for P in frange(3, 8, 1):  
 tmp = SRK\_method(z, w, P\_c, T\_c, P, T, c\_ij)  
 #print("СРК","P=", P, "L =", tmp[0])  
 tmp1 = brusilovski\_method(z, w, P\_c, T\_c, P, T, c\_ij)  
 #print("Б","P=", P, "L =", tmp1[0])  
 result\_SRK.append(tmp[0] \* 100)  
 result\_brusilovski.append(tmp1[0] \* 100)  
 P\_arr.append(P)  
 del tmp  
 del tmp1  
  
 plt.plot(P\_arr, result\_SRK, label=u'СРК Метод')  
 plt.plot(P\_arr,result\_brusilovski, label = u'Метод Брусиловского')  
 fig.legend(loc='center right')  
 plt.savefig('L(P)')  
 plt.grid(True, color='black')  
 plt.show()  
  
def brusilovski\_method(z, w, P\_c, T\_c, P, T, c\_ij):  
 K = []; x = []; y = []  
 Bi = []; Ci = []; Di = []  
 alpha = []; beta = []; sigma = []; delta = []  
 a\_m\_y = 0; b\_m\_y = 0; c\_m\_y = 0; d\_m\_y = 0  
 a\_m\_x = 0; b\_m\_x = 0; c\_m\_x = 0; d\_m\_x = 0  
 k1 = []; k2 = []; k3 = []; k4 = []  
 F\_v = []; F\_l = []  
 K\_0 = []; K\_1 = []  
 check = []  
 eps = 0.00001  
  
  
 R = 8.314462 # Дж/(моль·K) если Дж/(кг К), то R\*1000/94.288  
 N = len(z)  
 '''Определим параметры Z\_c, Omega\_c, Psi'''  
 Z\_c = [0.33294, 0.31274, 0.31508, 0.30663, 0.31232, 0.0, 0.0, 0.0, 0.34626, 0.31933, 0.30418]  
 omega\_c = [0.75630, 0.77698, 0.76974, 0.78017, 0.76921, 0.75001, 0.75001, 0.75001, 0.75001, 0.75282, 0.78524]  
 psi = [0.37447, 0.49550, 0.53248, 0.63875, 0.57594, 0.0, 0.0, 0.0, 0.37182, 0.74212, 0.38203]  
  
 # Заполнение недостающих элементов массивов Z\_c и Psi  
 num = 0  
 for i in Z\_c:  
 if i == 0:  
 Z\_c[num] = 0.3357 - 0.0294\*w[num]  
 num += 1  
 num = 0  
 for i in psi:  
 if i == 0:  
 if w[num] < 0.4489:  
 psi[num] = 1.050 + 0.105 \* w[num] + 0.482 \* w[num]\*\*2  
 else:  
 psi[num] = 0.429 + 1.004 \* w[num] + 1.561 \* w[num] \*\* 2  
 num += 1  
 del num  
  
 '''Вычислим коэф-ы альфа, бета, сигма, дельта'''  
  
 for i in range(N):  
 alpha.append(omega\_c[i]\*\*3)  
 beta.append(Z\_c[i] + omega\_c[i]-1)  
 sigma.append(-Z\_c[i] + omega\_c[i] \* (0.5 + (omega\_c[i] - 0.75) \*\* 0.5))  
 delta.append(-Z\_c[i] + omega\_c[i] \* (0.5 - (omega\_c[i] - 0.75) \*\* 0.5))  
  
 '''Далее вычислим коэффициенты a, b, c, d для каждого компонента связи'''  
 a = []; b = []; c = []; d = []  
 for i in range(N):  
 a\_c = alpha[i] \* (R\*\*2) \* (T\_c[i]\*\*2)/P\_c[i]  
 al\_T\_w = (1 + psi[i] \* (1 - (T/T\_c[i]) \*\* 0.5)) \*\* 2  
 a.append(a\_c \* al\_T\_w)  
 b.append(beta[i] \* R \* T\_c[i]/P\_c[i])  
 c.append(sigma[i] \* R \* T\_c[i]/P\_c[i])  
 d.append(delta[i] \* R \* T\_c[i]/P\_c[i])  
 # print(a)  
 # print(b)  
 # print(c)  
 # print(d)  
  
 '''Расчитаем начальное приближение K'''  
 for i in range(N):  
 K.append((P\_c[i] / P) \* math.exp(5.37 \* (1 + w[i]) \* (1 - (T\_c[i] / T)))) # Нач. знач. K по корреляции Вильсона  
 # print(K)  
  
 '''Проведем одну итерацию вручную'''  
 # Решим уравнение Рашфорда – Райса  
 V = bisection(1 / (1 - max(K)) + 0.0001, 1 / (1 - min(K)) - 0.0001, 0.0001, z, K)  
 #print("V: ", V)  
 # Вычислим y\_i и x\_i и сразу же проверим условие их сумм = 1  
 sum\_x = 0; sum\_y = 0; sum\_z = 0  
 for i in range(N):  
 y.append(z[i] \* K[i] / (V \* (K[i] - 1) + 1))  
 x.append(y[i] / K[i])  
 sum\_x += x[i]; sum\_y += y[i]; sum\_z += z[i]  
 for i in range(N):  
 if y[i] == 0:  
 y[i] = y[i] + eps  
 if x[i] == 0:  
 x[i] = x[i] + eps  
 #print(y)  
 #print(x)  
 #print(sum\_x, sum\_y)  
 '''Вычислим все коэф-ы a\_m\_x, a\_m\_y, b\_m\_x, b\_m\_y, c\_m\_x, c\_m\_y, d\_m\_x, d\_m\_y'''  
 for i in range(N):  
 c\_m\_y += y[i] \* c[i]  
 c\_m\_x += x[i] \* c[i]  
 d\_m\_y += y[i] \* d[i]  
 d\_m\_x += x[i] \* d[i]  
 for j in range(N):  
 a\_m\_y += (y[i] \* y[j] \* (1 - c\_ij[i][j]) \* math.sqrt(a[i] \* a[j]))  
 b\_m\_y += (y[i] \* y[j] \* (0.5 \* (b[i] + b[j])))  
 a\_m\_x += (x[i] \* x[j] \* (1 - c\_ij[i][j]) \* math.sqrt(a[i] \* a[j]))  
 b\_m\_x += (x[i] \* x[j] \* (0.5 \* (b[i] + b[j])))  
  
  
 # Обозначим Am\_x, Bm\_x, Cm\_x, Dm\_x, Bi, Ci, Di  
 Am\_y = a\_m\_y \* P / (R\*R \* T\*T); Am\_x = a\_m\_x \* P / (R\*R \* T\*T)  
 Bm\_y = b\_m\_y \* P / (R \* T); Bm\_x = b\_m\_x \* P / (R \* T)  
 Cm\_y = c\_m\_y \* P / (R \* T); Cm\_x = c\_m\_x \* P / (R \* T)  
 Dm\_y = d\_m\_y \* P / (R \* T); Dm\_x = d\_m\_x \* P / (R \* T)  
 for i in range(N):  
 Bi.append(b[i]\*P/(R\*T))  
 Ci.append(c[i]\*P/(R\*T))  
 Di.append(d[i]\*P/(R\*T))  
  
 # Ищем решение кубического уравнения относительно Z\_v и Z\_l  
 pol\_K\_y = [1, 0, 0, 0]  
 pol\_K\_y[1] = Cm\_y + Dm\_y - Bm\_y - 1  
 pol\_K\_y[2] = Am\_y - Bm\_y \* Cm\_y + Cm\_y \* Dm\_y - Bm\_y \* Dm\_y - Dm\_y - Cm\_y  
 pol\_K\_y[3] = -(Bm\_y \* Cm\_y \* Dm\_y + Cm\_y \* Dm\_y + Am\_y \* Bm\_y)  
 z\_fact\_y = np.roots(pol\_K\_y)  
 z\_fact\_y = no\_compl(z\_fact\_y)  
 z\_v\_max = round(max(z\_fact\_y), 6) # Корень ур-я Z\_v  
  
 pol\_K\_x = [1, 0, 0, 0]  
 pol\_K\_x[1] = Cm\_x + Dm\_x - Bm\_x - 1  
 pol\_K\_x[2] = Am\_x - Bm\_x \* Cm\_x + Cm\_x \* Dm\_x - Bm\_x \* Dm\_x - Dm\_x - Cm\_x  
 pol\_K\_x[3] = -(Bm\_x \* Cm\_x \* Dm\_x + Cm\_x \* Dm\_x + Am\_x \* Bm\_x)  
 z\_fact\_x = np.roots(pol\_K\_x)  
 z\_fact\_x = no\_compl(z\_fact\_x)  
 z\_l\_min = round(min(z\_fact\_x), 6) # Корень ур-я Z\_l  
  
 K\_0 = K.copy() # Начальное значение, вычисленное по корреляции Вильсона  
  
 for i in range(N):  
 sum5 = 0  
 for j in range(N):  
 sum5 += y[j]\*(1 - c\_ij[i][j]) \* math.sqrt(a[i] \* a[j])  
 #print(y[i]\*P )  
 k1.append(ln(y[i] \* P ))  
 k2.append(ln(z\_v\_max - Bm\_y))  
 k3.append( (Am\_y/(Cm\_y - Dm\_y)) \* (2 \* sum5 /a\_m\_y - (c[i]-d[i])/(c\_m\_y - d\_m\_y)) \* ln( (z\_v\_max + Cm\_y) / (z\_v\_max + Dm\_y)) )  
 k4.append(Bi[i]/(z\_v\_max - Bm\_y) - Am\_y/(Cm\_y-Dm\_y) \* (Ci[i]/(z\_v\_max + Cm\_y) - Di[i]/(z\_v\_max + Dm\_y) ))  
 F\_v.append(math.exp(k1[i]-k2[i]-k3[i]+k4[i]))  
  
 sum5 = 0  
 for j in range(N):  
 sum5 += x[j] \* (1 - c\_ij[i][j]) \* math.sqrt(a[i] \* a[j])  
 k1[i] = (ln(x[i]\*P))  
 k2[i] = (ln(z\_l\_min - Bm\_x))  
 k3[i] = ((Am\_x/(Cm\_x - Dm\_x))\*(2 \* sum5 / a\_m\_x - (c[i]-d[i])/(c\_m\_x - d\_m\_x))\*ln((z\_l\_min + Cm\_x)/(z\_l\_min + Dm\_x)))  
 k4[i] = (Bi[i]/(z\_l\_min - Bm\_x) - Am\_x/(Cm\_x-Dm\_x) \* (Ci[i]/(z\_l\_min + Cm\_x) - Di[i]/(z\_l\_min + Dm\_x) ) )  
 F\_l.append(math.exp(k1[i]-k2[i]-k3[i]+k4[i]))  
  
 K\_1.append(K\_0[i] \* F\_l[i] / F\_v[i])  
 check.append(abs(K\_1[i] - K\_0[i]) / abs(K\_0[i]))  
  
 '''На этом шаге мы получили значения K\_1 и K\_0, с которых мы далее начнем проверять условие сходимости '''  
 #print(K\_1)  
 #print(K\_0)  
  
 iter = 1  
 '''Непосредственно сам расчетный цикл'''  
  
 while max(check) >= eps: # пока максимальная ошибка > eps  
 a\_m\_x = 0; a\_m\_y = 0; b\_m\_x = 0; b\_m\_y = 0; c\_m\_x = 0; c\_m\_y = 0; d\_m\_x = 0; d\_m\_y = 0  
 # Решим уравнение Рашфорда – Райса  
 V = bisection(1 / (1 - max(K\_1)) + 0.0001, 1 / (1 - min(K\_1)) - 0.0001, 0.0001, z, K\_1)  
 #print("a: ", 1 / (1 - max(K\_1)) + 0.0001)  
 #print("b: ", 1 / (1 - min(K\_1)) + 0.0001)  
 #print("V: ", V)  
 # Вычислим y\_i и x\_i и сразу же проверим условие их сумм = 1  
 sum\_x = 0; sum\_y = 0; sum\_z = 0  
 for i in range(N):  
 y[i] = (z[i] \* K\_1[i] / (V \* (K\_1[i] - 1) + 1))  
 x[i] = (y[i] / K\_1[i])  
 sum\_x += x[i]; sum\_y += y[i]; sum\_z += z[i]  
 for i in range(N):  
 if y[i] == 0:  
 y[i] = y[i] + eps  
 if x[i] == 0:  
 x[i] = x[i] + eps  
 #print(y)  
 #print(x)  
 #print(sum\_x, sum\_y)  
 #Вычислим все коэф-ы a\_m\_x, a\_m\_y, b\_m\_x, b\_m\_y, c\_m\_x, c\_m\_y, d\_m\_x, d\_m\_y  
  
 for i in range(N):  
 c\_m\_y += y[i] \* c[i]  
 c\_m\_x += x[i] \* c[i]  
 d\_m\_y += y[i] \* d[i]  
 d\_m\_x += x[i] \* d[i]  
 for j in range(N):  
 a\_m\_y += (y[i] \* y[j] \* (1 - c\_ij[i][j]) \* math.sqrt(a[i] \* a[j]))  
 b\_m\_y += (y[i] \* y[j] \* (0.5 \* (b[i] + b[j])))  
 a\_m\_x += (x[i] \* x[j] \* (1 - c\_ij[i][j]) \* math.sqrt(a[i] \* a[j]))  
 b\_m\_x += (x[i] \* x[j] \* (0.5 \* (b[i] + b[j])))  
  
 # Обозначим Am\_x, Bm\_x, Cm\_x, Dm\_x, Bi, Ci, Di  
 Am\_y = a\_m\_y \* P / (R \* R \* T \* T); Am\_x = a\_m\_x \* P / (R \* R \* T \* T)  
 Bm\_y = b\_m\_y \* P / (R \* T); Bm\_x = b\_m\_x \* P / (R \* T)  
 Cm\_y = c\_m\_y \* P / (R \* T); Cm\_x = c\_m\_x \* P / (R \* T)  
 Dm\_y = d\_m\_y \* P / (R \* T); Dm\_x = d\_m\_x \* P / (R \* T)  
 for i in range(N):  
 Bi[i] = (b[i] \* P / (R \* T))  
 Ci[i] = (c[i] \* P / (R \* T))  
 Di[i] = (d[i] \* P / (R \* T))  
  
 # Ищем решение кубического уравнения относительно Z\_v и Z\_l  
 pol\_K\_y = [1, 0, 0, 0]  
 pol\_K\_y[1] = Cm\_y + Dm\_y - Bm\_y - 1  
 pol\_K\_y[2] = Am\_y - Bm\_y \* Cm\_y + Cm\_y \* Dm\_y - Bm\_y \* Dm\_y - Dm\_y - Cm\_y  
 pol\_K\_y[3] = -(Bm\_y \* Cm\_y \* Dm\_y + Cm\_y \* Dm\_y + Am\_y \* Bm\_y)  
 z\_fact\_y = np.roots(pol\_K\_y)  
 z\_fact\_y = no\_compl(z\_fact\_y)  
 z\_v\_max = round(max(z\_fact\_y), 6) # Корень ур-я Z\_v  
  
 pol\_K\_x = [1, 0, 0, 0]  
 pol\_K\_x[1] = Cm\_x + Dm\_x - Bm\_x - 1  
 pol\_K\_x[2] = Am\_x - Bm\_x \* Cm\_x + Cm\_x \* Dm\_x - Bm\_x \* Dm\_x - Dm\_x - Cm\_x  
 pol\_K\_x[3] = -(Bm\_x \* Cm\_x \* Dm\_x + Cm\_x \* Dm\_x + Am\_x \* Bm\_x)  
 z\_fact\_x = np.roots(pol\_K\_x)  
 z\_fact\_x = no\_compl(z\_fact\_x)  
 z\_l\_min = round(min(z\_fact\_x), 6) # Корень ур-я Z\_l  
  
  
 K\_0 = K\_1.copy() # Далее буду считать следующее значение K -> нужно зафиксировать предыдущее  
  
 for i in range(N):  
 sum5 = 0  
 for j in range(N):  
 sum5 += y[j] \* (1 - c\_ij[i][j]) \* math.sqrt(a[i] \* a[j])  
 k1[i] = (ln(y[i] \* P))  
 k2[i] = (ln(z\_v\_max - Bm\_y))  
 k3[i] = ((Am\_y / (Cm\_y - Dm\_y)) \* (2 \* sum5 / a\_m\_y - (c[i] - d[i]) / (c\_m\_y - d\_m\_y)) \* ln((z\_v\_max + Cm\_y) / (z\_v\_max + Dm\_y)))  
 k4[i] = (Bi[i] / (z\_v\_max - Bm\_y) - Am\_y / (Cm\_y - Dm\_y) \* (Ci[i] / (z\_v\_max + Cm\_y) - Di[i] / (z\_v\_max + Dm\_y)))  
 F\_v[i] = (math.exp(k1[i] - k2[i] - k3[i] + k4[i]))  
  
 sum5 = 0  
 for j in range(N):  
 sum5 += x[j] \* (1 - c\_ij[i][j]) \* math.sqrt(a[i] \* a[j])  
 k1[i] = (ln(x[i] \* P))  
 k2[i] = (ln(z\_l\_min - Bm\_x))  
 k3[i] = ((Am\_x / (Cm\_x - Dm\_x)) \* (2 \* sum5 / a\_m\_x - (c[i] - d[i]) / (c\_m\_x - d\_m\_x)) \* ln(  
 (z\_l\_min + Cm\_x) / (z\_l\_min + Dm\_x)))  
 k4[i] = (Bi[i] / (z\_l\_min - Bm\_x) - Am\_x / (Cm\_x - Dm\_x) \* (  
 Ci[i] / (z\_l\_min + Cm\_x) - Di[i] / (z\_l\_min + Dm\_x)))  
 F\_l[i] = (math.exp(k1[i] - k2[i] - k3[i] + k4[i]))  
  
 K\_1[i] = (K\_0[i] \* F\_l[i] / F\_v[i])  
 #check[i] = abs(K\_1[i] - K\_0[i]) / abs(K\_0[i])  
 check[i] = abs(F\_l[i] / F\_v[i] - 1)  
 iter+=1  
  
 '''Проверка некоторых выражений'''  
 # print("Iter:", iter)  
 # print("Summ\_z: ", sum\_z, "Summ\_y: ", sum\_y, "Summ\_x: ", sum\_x)  
 root\_r = 0  
 for i in range(N):  
 root\_r += z[i] \* (K\_1[i] - 1) / (1 + V \* (K\_1[i] - 1))  
 #print("Подставим найденное V в уравнение Рашфорда – Райса:", root\_r)  
 L = (sum\_z - sum\_y \* V) / sum\_x  
 #L = 1 - V  
 #print("L: ", L, " -> следовательно жидкая фаза существует!")  
  
 return L, K\_1  
  
  
# Главная функция  
def main():  
 # Зададим все начальные данные, а именно: массовые доли z, ...  
 # z - массовые доли N - компонентов в первоначальной газовой смеси. Порядок такой, какой был дан в таблице  
 z = [0.80324, 0.06024, 0.03066, 0.00001, 0.00001, 0.04381, 0.03108, 0.02431, 0.00478, 0.00188, 0.00001]  
 # w - ацентрический фактор  
 w = [0.01142, 0.0995, 0.1521, 0.184, 0.201, 0.251, 0.349, 0.4884, 0.0372, 0.22394, 0.1005]  
 # P\_ci - критическое значение давление  
 P\_c = [4.5992, 4.8722, 4.2512, 3.629, 3.796, 3.370, 2.736, 2.103, 3.3958, 7.3773, 9.00]  
 # T\_ci - критическое давление температуры  
 T\_c = [190.56, 305.32, 369.89, 407.81, 425.13, 469.6, 540.13, 617.7, 126.19, 304.13, 373.10]  
 # Для дальнейших расчетов определим значение P  
 # P = float(input("Рабочее давление: ")) # randint(3, 7) # Давление, смеси в конце  
 # Значение T берем из таблицы  
 # T = float(input("Рабочее давление: ")) + 273.0 # Температура, до которой охладилась смесь  
  
 c\_ij = [[0.000, 0.005, 0.010, 0.010, 0.030, 0.030, 0.035, 0.045, 0.025, 0.105, 0.07],  
 [0.005, 0.000, 0.005, 0.010, 0.010, 0.020, 0.020, 0.02, 0.010, 0.13, 0.085],  
 [0.010, 0.005, 0.000, 0.000, 0.020, 0.005, 0.005, 0.005, 0.090, 0.125, 0.08],  
 [0.010, 0.010, 0.000, 0.000, 0.005, 0.005, 0.005, 0.005, 0.095, 0.115, 0.075],  
 [0.030, 0.010, 0.020, 0.005, 0.000, 0.000, 0.000, 0.000, 0.100, 0.115, 0.07],  
 [0.030, 0.020, 0.005, 0.005, 0.000, 0.000, 0.000, 0.000, 0.110, 0.115, 0.07],  
 [0.035, 0.020, 0.005, 0.005, 0.000, 0.000, 0.000, 0.000, 0.115, 0.115, 0.06],  
 [0.045, 0.020, 0.005, 0.005, 0.000, 0.000, 0.000, 0.000, 0.125, 0.115, 0.055],  
 [0.025, 0.010, 0.090, 0.095, 0.100, 0.110, 0.115, 0.125, 0, 000, 0.000, 0.13],  
 [0.105, 0.13, 0.125, 0.115, 0.115, 0.115, 0.115, 0.115, 0.000, 0.000, 0.135],  
 [0.07, 0.085, 0.08, 0.075, 0.07, 0.070, 0.060, 0.055, 0.130, 0.135, 0.000]]  
  
 '''Тут необходимо ввести нужные значения P и T'''  
 P = 5.0#float(input("Рабочее давление, МПа: "))  
 T = 5 + 273#float(input("Рабочая температура, C: ")) + 273  
 SRK\_method(z, w, P\_c, T\_c, P, T, c\_ij) # Вызов СРК метода  
 brusilovski\_method(z, w, P\_c, T\_c, P, T, c\_ij)  
 graph(z, w, P\_c, T\_c, P, T, c\_ij)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()