高等工程热力学编程部分作业

何陬　3123101186

采用 Python 语言进行计算程序编写，将 PR 方程相关计算封装在 PR 类中，程序默认要求输入两种流体性质，对于同种物质，输入相同参数。对于不同的问题在 if ___name___ == "___main___" 部分进行编辑以进行计算。计算思路如下：

（1）导入流体的相关性质如临界温度 T_{c}、临界压力 p_{c}、偏心因子 ω 和摩尔质量 M，以及理想气体常数 $R = 8.314462618\mathrm{J}/(\mathrm{mol}\cdot\mathrm{K})$；

（2）给定温度 T 计算参数 a、b 和 a'，其中 a' 根据书中式 (3.110) 给出的混合法则对 T 进行求导, 求导结果如下：

$$a' = x_1^2 a_1' + x_1 x_2 (1 - k_{ij})(a_1' \sqrt{\frac{a_2}{a_1}} + a_2' \sqrt{\frac{a_1}{a_2}}) + x_2^2 a_2'$$

（3）给定温度 T 和压力 p 计算参数 A 和 B；

（4）计算压缩因子 Z 的多项式系数 C_2, C_1, C_0；

（5）使用牛顿法分别计算压缩因子 Z 的液相值 Z_l 和气相值 Z_g；

（6）计算比体积 v 的液相值 v_l 和气相值 v_g；

（7）绘制比体积 v 与温度 T 的关系曲线，并标注饱和温度 T_{sat}；

（8）计算焓的余函数 h^{r} 和熵的余函数 s^{r}；

（9）计算热容 c_p 的积分和热容除以温度 $\frac{c_p}{T}$ 的积分；

（10）计算焓 h 和熵 s；

（11）计算气相的逸度系数 $\hat{\phi}$ 和逸度 $\hat{f}$；

（12）绘制气相的逸度系数 $\hat{\phi}$ 和逸度 $\hat{f}$ 与温度 T 的关系曲线。

计算所用的 PR 类程序如下：

```python
import numpy as np
import matplotlib.pyplot as plt
import os

# 使用 Times New Roman 作为 matplotlib 全局字体
plt.rcParams["font.family"] = "serif"
plt.rcParams["font.serif"] = ["Times New Roman"]
plt.rcParams["mathtext.fontset"] = "stix"
plt.rcParams["font.size"] = 14 # 增大全局字体
```

```
plt.rcParams["axes.linewidth"] = 1.5 # 增粗坐标轴


class PR:
    def __init__(self, Tc1, pc1, omega1, M1, x1, Tc2, pc2, omega2, M2, kij, ps0
        ):
        self.Tc1 = Tc1 # K
        self.pc1 = pc1 * 1e6 # Pa, 输入MPa
        self.omega1 = omega1 # 无量纲
        self.M1 = M1 / 1e3 # kg/mol, 输入g/mol
        self.x1 = x1 # 组分1的摩尔分数

        self.Tc2 = Tc2 # K
        self.pc2 = pc2 * 1e6 # Pa, 输入MPa
        self.omega2 = omega2 # 无量纲
        self.M2 = M2 / 1e3 # kg/mol, 输入g/mol
        self.x2 = 1 - x1 # 组分2的摩尔分数

        self.ps0 = ps0 # MPa

        self.kij = kij # 无量纲

    R = 8.314462618 # J/(mol*K)

    # 计算a和b
    def params(self, T):
        kappa1 = 0.37464 + 1.54226 * self.omega1 - 0.26992 * self.omega1**2
        kappa2 = 0.37464 + 1.54226 * self.omega2 - 0.26992 * self.omega2**2
        Tr1 = T / self.Tc1
        Tr2 = T / self.Tc2
        alpha1 = (1 + kappa1 * (1 - Tr1**0.5)) ** 2
        alpha2 = (1 + kappa2 * (1 - Tr2**0.5)) ** 2
        a1 = 0.45724 * self.R**2 * self.Tc1**2 / self.pc1 * alpha1
        a2 = 0.45724 * self.R**2 * self.Tc2**2 / self.pc2 * alpha2
        da1 = (
            -0.45724
            * self.R**2
            * self.Tc1**2
            / self.pc1
            * kappa1
```

```
            * (1 + kappa1 * (1 - Tr1**0.5))
            * (Tr1**-0.5)
            / self.Tc1
        )
        da2 = (
            -0.45724
            * self.R**2
            * self.Tc2**2
            / self.pc2
            * kappa2
            * (1 + kappa2 * (1 - Tr2**0.5))
            * (Tr2**-0.5)
            / self.Tc2
        )
        b1 = 0.07780 * self.R * self.Tc1 / self.pc1
        b2 = 0.07780 * self.R * self.Tc2 / self.pc2

        a = (
            self.x1**2 * a1
            + self.x2**2 * a2
            + 2 * self.x1 * self.x2 * (a1 * a2) ** 0.5 * (1 - self.kij)
        )
        b = self.x1 * b1 + self.x2 * b2
        da = (
            self.x1**2 * da1
            + self.x2**2 * da2
            + self.x1
            * self.x2
            * (1 - self.kij)
            * ((a2 / a1) ** 0.5 * da1 + (a1 / a2) ** 0.5 * da2)
        )
        return a1, a2, a, b1, b2, b, da

    # 计算A和B
    def AB(self, T, p):
        a1, a2, a, b1, b2, b, da = self.params(T)
        A = a * p * 1e6 / (self.R * T) ** 2
        B = b * p * 1e6 / (self.R * T)
        return A, B

```

```
    # 计算C2, C1, C0
    def C(self, T, p):
        A, B = self.AB(T, p)
        C2 = -(1 - B)
        C1 = A - 3 * B**2 - 2 * B
        C0 = -(A * B - B**2 - B**3)
        return C2, C1, C0

    # 计算压缩因子Z
    # 液相
    def Zl(self, T, p):
        C2, C1, C0 = self.C(T, p)
        # 牛顿法求解Z
        Zl = 0.001 # 初始猜测值
        for _ in range(100):
            f = Zl**3 + C2 * Zl**2 + C1 * Zl + C0
            df = 3 * Zl**2 + 2 * C2 * Zl + C1
            Zl_new = Zl - f / df
            if abs(Zl_new - Zl) < 1e-6:
                break
            Zl = Zl_new
        return Zl

    # 气相
    def Zg(self, T, p):
        C2, C1, C0 = self.C(T, p)
        # 牛顿法求解Z
        Zg = 1.0 # 初始猜测值
        for _ in range(100):
            f = Zg**3 + C2 * Zg**2 + C1 * Zg + C0
            df = 3 * Zg**2 + 2 * C2 * Zg + C1
            Zg_new = Zg - f / df
            if abs(Zg_new - Zg) < 1e-6:
                break
            Zg = Zg_new
        return Zg

    # 计算比体积v
    # 液相
    def vl(self, T, p):
```

```
        Zl = self.Zl(T, p)
        vl = Zl * self.R * T / (p * 1e6)
        return vl # m³/mol

    # 气相
    def vg(self, T, p):
        Zg = self.Zg(T, p)
        vg = Zg * self.R * T / (p * 1e6)
        return vg # m³/mol

    def plot_Tv(
        self,
        fluid_name, # 流体名称
        p, # 压力 MPa
        Tsat, # 饱和温度 K
        T_min, # 温度范围最小值 K
        T_max, # 温度范围最大值 K
        nT=220, # 温度点数
    ):
        T_grid = np.linspace(T_min, T_max, nT) # 温度网格
        v_grid = np.empty_like(T_grid) # 比体积网格
        # 计算比体积
        for i, T in enumerate(T_grid):
            if T < Tsat:
                v_grid[i] = self.vl(T, p) / (self.x1 * self.M1 + self.x2 * self.
                    M2)
            elif T > Tsat:
                v_grid[i] = self.vg(T, p) / (self.x1 * self.M1 + self.x2 * self.
                    M2)
            else:
                v_grid[i] = (
                    0.5
                    * (self.vl(T, p) + self.vg(T, p))
                    / (self.x1 * self.M1 + self.x2 * self.M2)
                )
        fig, ax = plt.subplots(1, figsize=(8, 6)) # 创建图像和坐标轴
        # 主曲线
        ax.plot(
            v_grid,
            T_grid,
```

```
            linewidth=2,
            label=fluid_name,
        )
        xmin, xmax = np.nanmin(v_grid), np.nanmax(v_grid)
        # Tsat 虚线
        ax.hlines(Tsat, xmin, xmax, linestyles="--", label=r"$T_{\mathrm{sat}}$"
            )
        # 标注 Tsat
        yt = list(ax.get_yticks())
        # 加入Tsat并排序
        if not any(abs(t - Tsat) < 1e-8 for t in yt):
            yt.append(Tsat)
        yt = np.array(sorted(yt))
        # 生成刻度标签：对 Tsat 使用仅数值标签（两位小数），其它刻度保留数字格式
            （根据范围选择小数位）
        deltaT = T_grid.max() - T_grid.min()
        labels = []
        for t in yt:
            if abs(t - Tsat) < 1e-8 or abs(t - Tsat) < 1e-6 * max(1.0, deltaT):
                labels.append(f"{Tsat:.2f}")
            else:
                # 根据温度范围决定格式，避免过多小数
                if deltaT > 50:
                    labels.append(f"{t:.0f}")
                else:
                    labels.append(f"{t:.2f}")
        ax.set_yticks(yt)
        ax.set_yticklabels(labels)
        # 轴标签
        ax.set_xlabel(r"$v$ (m³/kg)")
        ax.set_ylabel(r"$T$ (K)")
        # 标题
        # ax.set_title(f"{fluid_name} $T$-$v$ at $p$ = {p:.1f} MPa")
        ax.grid(True)
        ax.set_xscale("log") # 使用对数刻度
        ax.legend(loc="upper left", frameon=True, fancybox=True, framealpha=0.9)

        # 固定保存路径为脚本同目录下的 figs 文件夹
        base_dir = os.path.dirname(os.path.abspath(__file__))
        fig_dir = os.path.join(base_dir, "figs")
```

```
        os.makedirs(fig_dir, exist_ok=True)

        # 文件名固定为"流体名称.png"
        filename = f"{fluid_name}.png"
        savepath = os.path.join(fig_dir, filename)

        # 保存图像，固定参数
        fig.savefig(savepath, dpi=600, bbox_inches="tight", transparent=False)
        plt.close(fig)

    # 计算焓的余函数
    # 液相
    def h_res_l(self, T, p):
        a1, a2, a, b1, b2, b, da = self.params(T)
        Zl = self.Zl(T, p)
        vl = self.vl(T, p)
        hr_l = (T * da - a) / (b * np.sqrt(8)) * np.log(
            (vl - 0.414 * b) / (vl + 2.414 * b)
        ) + self.R * T * (1 - Zl)
        return hr_l

    # 气相
    def h_res_g(self, T, p):
        a1, a2, a, b1, b2, b, da = self.params(T)
        Zg = self.Zg(T, p)
        vg = self.vg(T, p)
        hr_g = (T * da - a) / (b * np.sqrt(8)) * np.log(
            (vg - 0.414 * b) / (vg + 2.414 * b)
        ) + self.R * T * (1 - Zg)
        return hr_g

    # 计算熵的余函数
    # 液相
    def s_res_l(self, T, p):
        a1, a2, a, b1, b2, b, da = self.params(T)
        vl = self.vl(T, p)
        sr_l = (
            -self.R * np.log((vl - b) / vl)
            - self.R * np.log(vl / (self.R * T / (p * 1e6)))
            + da / (b * np.sqrt(8)) * np.log((vl - 0.414 * b) / (vl + 2.414 * b)
```

```
            )
        )
        return sr_l

    # 气相
    def s_res_g(self, T, p):
        a1, a2, a, b1, b2, b, da = self.params(T)
        vg = self.vg(T, p)
        sr_g = (
            -self.R * np.log((vg - b) / vg)
            - self.R * np.log(vg / (self.R * T / (p * 1e6)))
            + da / (b * np.sqrt(8)) * np.log((vg - 0.414 * b) / (vg + 2.414 * b)
                )
        )
        return sr_g

    # 计算c_p积分
    def cp(self, T, A, B, C, D):
        cp = (
            A * (T - 273.15)
            + B / 2 * (T**2 - 273.15**2)
            + C / 3 * (T**3 - 273.15**3)
            + D / 4 * (T**4 - 273.15**4)
        )
        return cp

    # 计算c_p/T积分
    def cpT(self, T, A, B, C, D):
        cp = (
            A * np.log(T / 273.15)
            + B * (T - 273.15)
            + C / 2 * (T**2 - 273.15**2)
            + D / 3 * (T**3 - 273.15**3)
        )
        return cp

    # 计算焓和熵
    # 液相
    def h_l(self, T, A, B, C, D, p):
        h_r_ps_0 = self.h_res_l(273.15, self.ps0)
```

```
        cp0 = self.cp(T, A, B, C, D)
        h_res_l = self.h_res_l(T, p)
        hl = (
            200 * 1e3
            + cp0
            + (h_r_ps_0 - h_res_l) / (self.x1 * self.M1 + self.x2 * self.M2)
        ) # J/kg
        return hl

    def s_l(self, T, A, B, C, D, p):
        s_r_ps_0 = self.s_res_l(273.15, self.ps0)
        cpT = self.cpT(T, A, B, C, D)
        sr_l = self.s_res_l(T, p)
        sl = (
            1e3
            + cpT
            + (s_r_ps_0 - self.R * np.log(p / self.ps0) - sr_l)
            / (self.x1 * self.M1 + self.x2 * self.M2)
        ) # J/(kg*K)
        return sl

    # 气相
    def h_g(self, T, A, B, C, D, p):
        h_r_ps_0 = self.h_res_l(273.15, self.ps0) # 使用液相作为基准
        cp0 = self.cp(T, A, B, C, D)
        h_res_g = self.h_res_g(T, p)
        hg = (
            200 * 1e3
            + cp0
            + (h_r_ps_0 - h_res_g) / (self.x1 * self.M1 + self.x2 * self.M2)
        ) # J/kg
        return hg

    def s_g(self, T, A, B, C, D, p):
        s_r_ps_0 = self.s_res_l(273.15, self.ps0) # 使用液相作为基准
        cpT = self.cpT(T, A, B, C, D)
        sr_g = self.s_res_g(T, p)
        sg = (
            1e3
            + cpT
```

```
            + (s_r_ps_0 - self.R * np.log(p / self.ps0) - sr_g)
            / (self.x1 * self.M1 + self.x2 * self.M2)
        ) # J/(kg*K)
        return sg

    # 计算逸度系数
    def phi(self, T, p):
        Zg = self.Zg(T, p)
        a1, a2, a, b1, b2, b, da = self.params(T)
        A, B = self.AB(T, p)
        phi1 = np.exp(
            (b1 / b) * (Zg - 1)
            - np.log(Zg - B)
            - A
            / (B * np.sqrt(8))
            * (
                2 * (self.x2 * (1 - self.kij) * (a1 * a2) ** 0.5 + self.x1 * a1)
                    / a
                - b1 / b
            )
            * np.log((Zg + 2.414 * B) / (Zg - 0.414 * B))
        )
        phi2 = np.exp(
            (b2 / b) * (Zg - 1)
            - np.log(Zg - B)
            - A
            / (B * np.sqrt(8))
            * (
                2 * (self.x1 * (1 - self.kij) * (a1 * a2) ** 0.5 + self.x2 * a2)
                    / a
                - b2 / b
            )
            * np.log((Zg + 2.414 * B) / (Zg - 0.414 * B))
        )
        return phi1, phi2

    # 计算逸度
    def f(self, T, p): # MPa
        phi1, phi2 = self.phi(T, p)
        f1 = self.x1 * phi1 * p
```

```
        f2 = self.x2 * phi2 * p
        return f1, f2

    # 绘制f-T、phi-T图
    def plot_fT(self, fluid_name, p, T_min, T_max, nT=11):
        T_grid = np.linspace(T_min, T_max, nT) # 温度网格
        phi_grid1 = np.empty_like(T_grid) # 组分1逸度系数网格
        phi_grid2 = np.empty_like(T_grid) # 组分2逸度系数网格
        f_grid1 = np.empty_like(T_grid) # 组分1逸度网格
        f_grid2 = np.empty_like(T_grid) # 组分2逸度网格

        base_dir = os.path.dirname(os.path.abspath(__file__))
        fig_dir = os.path.join(base_dir, "figs")
        os.makedirs(fig_dir, exist_ok=True)

        # 计算逸度系数和逸度
        for i, T in enumerate(T_grid):
            phi1, phi2 = self.phi(T, p)
            f1, f2 = self.f(T, p)
            phi_grid1[i] = phi1
            phi_grid2[i] = phi2
            f_grid1[i] = f1
            f_grid2[i] = f2

        # T-phi
        fig_phi = plt.figure(figsize=(8, 6))
        ax_phi = fig_phi.add_subplot(1, 1, 1)
        ax_phi.plot(T_grid, phi_grid1, "b-o", linewidth=2, label="R290",
            markersize=6)
        ax_phi.plot(T_grid, phi_grid2, "r-s", linewidth=2, label="R600a",
            markersize=6)
        ax_phi.set_xlabel(r"$T$ (K)", fontsize=12)
        ax_phi.set_ylabel(r"$\hat{\phi}$", fontsize=12)
        # ax_phi.set_title(
        # rf"{fluid_name} $\hat{{\phi}}$ at $p$ = {p:.1f} MPa", fontsize=12
        # )
        ax_phi.grid(True, linestyle="--", alpha=0.7)
        ax_phi.legend(loc="best", frameon=True, fancybox=True, framealpha=0.9)
        fig_phi.tight_layout()
        savepath_phi = os.path.join(fig_dir, f"{fluid_name}_phi.png")
```

```
399         fig_phi.savefig(savepath_phi, dpi=600, bbox_inches="tight", transparent=
                False)
400         plt.close(fig_phi)
401 
402         # T-f
403         fig_f = plt.figure(figsize=(8, 6))
404         ax_f = fig_f.add_subplot(111)
405         ax_f.plot(T_grid, f_grid1, "b-o", linewidth=2, label="R290", markersize
                =6)
406         ax_f.plot(T_grid, f_grid2, "r-s", linewidth=2, label="R600a", markersize
                =6)
407         ax_f.set_xlabel(r"$T$ (K)", fontsize=12)
408         ax_f.set_ylabel(r"$\hat{f}$ (MPa)", fontsize=12)
409         # ax_f.set_title(rf"{fluid_name} $\hat{{f}}$ at $p$ = {p:.1f} MPa",
                fontsize=12)
410         ax_f.grid(True, linestyle="--", alpha=0.7)
411         ax_f.legend(loc="best", frameon=True, fancybox=True, framealpha=0.9)
412         fig_f.tight_layout()
413         savepath_f = os.path.join(fig_dir, f"{fluid_name}_f.png")
414         fig_f.savefig(savepath_f, dpi=600, bbox_inches="tight", transparent=
                False)
415         plt.close(fig_f)
416 
417     if __name__ == "__main__": # 主程序
418         pass
```

3-10

R290 在 1.4MPa 下的 T-v 图和 R600a 在 0.6MPa 下的 T-v 图如下：

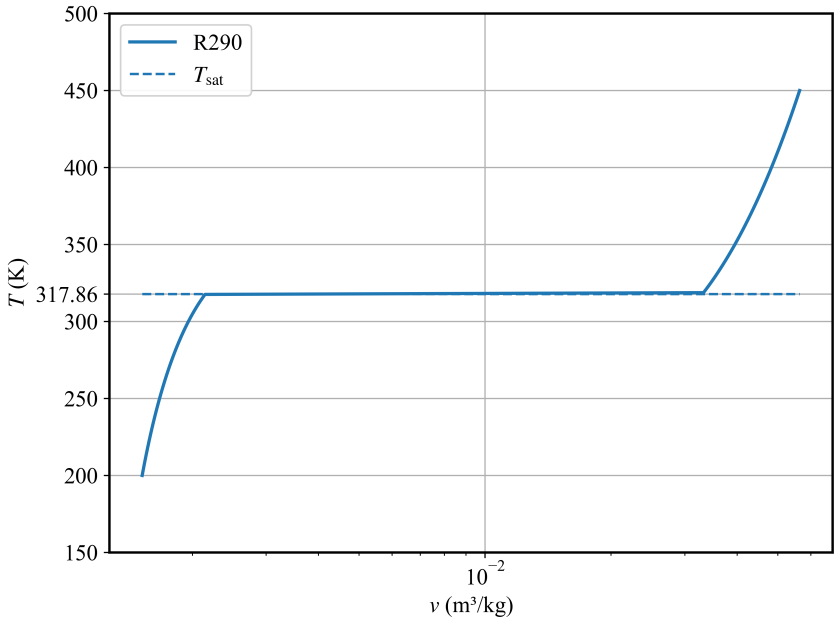

图 1: R290 在 1.4MPa 下的 T-v 图

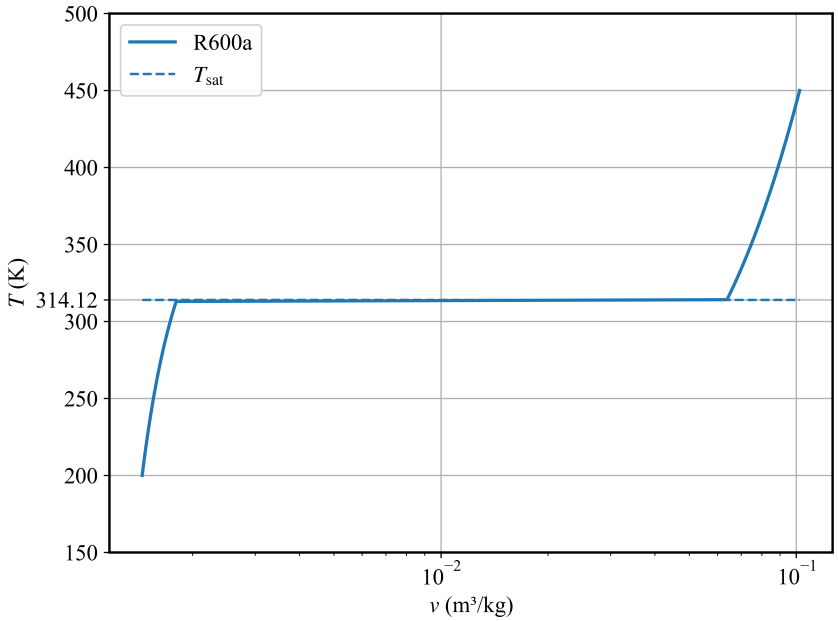

图 2: R600a 在 0.6MPa 下的 T-v 图

主程序如下：

```
R290 = PR(
    Tc1=369.89, # K
    pc1=4.2512, # MPa
```

```
        omega1=0.1521, # 无量纲
        M1=44.096, # g/mol
        x1=1.0,
        Tc2=369.89,
        pc2=4.2512,
        omega2=0.1521,
        M2=44.096,
        kij=0.064,
        ps0=0.47446,
    )

    R600a = PR(
        Tc1=407.81,
        pc1=3.629,
        omega1=0.184,
        M1=58.122, # R600a
        x1=1.0,
        Tc2=407.81,
        pc2=3.629,
        omega2=0.184,
        M2=58.122, # R600a
        kij=0.0,
        ps0=0.15696,
    )

    R290.plot_Tv("R290", 1.4, 317.86, 200, 450)
    R600a.plot_Tv("R600a", 0.6, 314.12, 200, 450)
```

3-13

查物性库得，对于 R134a，各参数为：$T_c = 374.21\text{K}$，$p_c = 4.0593\text{MPa}$，$\omega = 0.326$，$M = 102.03\text{g/mol}$。

对于 R1234yf，各参数为：$T_c = 367.85\text{K}$，$p_c = 3.3822\text{MPa}$，$\omega = 0.276$，$M = 114.04\text{g/mol}$；

对于 R1234ze(E)，各参数为：$T_c = 382.75\text{K}$，$p_c = 3.6349\text{MPa}$，$\omega = 0.313$，$M = 114.04\text{g/mol}$；

压力为 0.1MPa，温度为 35℃=308.15K 时，以上三种制冷剂均为气相，利用程序进行计算 v_g。计算结果为：$v_{\text{R134a}} = 0.24679\text{m}^3/\text{kg}$，$v_{\text{R1234yf}} = 0.22031\text{m}^3/\text{kg}$，$v_{\text{R1234ze(E)}} = 0.22006\text{m}^3/\text{kg}$

可以看出，三种制冷剂的比体积相差不大，R134a 的比体积略大于另外两种，故采用 R1234yf 和 R1234ze(E) 作为 R134a 的替代品是合理的。

主程序如下：

```
R134a = PR(
    Tc1=374.21,
    pc1=4.0593,
    omega1=0.326,
    M1=102.03, # R134a
    x1=1.0,
    Tc2=374.21,
    pc2=4.0593,
    omega2=0.326,
    M2=102.03, # R134a
    kij=0.0,
    ps0=0.57245,
)
R1234yf = PR(
    Tc1=367.85,
    pc1=3.3822,
    omega1=0.276,
    M1=114.04, # R1234yf
    x1=1.0,
    Tc2=367.85,
    pc2=3.3822,
    omega2=0.276,
    M2=114.04, # R1234yf
    kij=0.0,
    ps0=0.42483,
)
R1234ze = PR(
    Tc1=382.75,
    pc1=3.6349,
    omega1=0.313,
    M1=114.04, # R1234ze
    x1=1.0,
    Tc2=382.75,
    pc2=3.6349,
    omega2=0.313,
    M2=114.04, # R1234ze
```

```
        kij=0.0,
        ps0=0.49314,
    )

    print(R134a.vg(308.15, 0.1))
    print(R1234yf.vg(308.15, 0.1))
    print(R1234ze.vg(308.15, 0.1))
```

3-15

在压力 $p = 0.1$MPa、0.2MPa、0.3MPa，温度 $T = 300$K 时，不同的 k_{ij} 条件下，混合制冷剂 R290/R600a 的比体积计算结果与计算偏差如表 1 所示，表中计算偏差是相对于 $k_{ij} = 0.064$ 时的比体积计算结果而言的。可以看出，k_{ij} 取 0.1、0 和-0.1 时，计算结果与 $k_{ij} = 0.064$ 时的比体积计算结果偏差逐渐增大，且偏差均小于 1%。

表 1: 不同 k_{ij} 条件下混合制冷剂 R290/R600a 的比体积计算结果与计算偏差

p (MPa)	k_{ij}	v (m^3/mol)	误差 (%)
0.1	0.064	0.47838	
	0.1	0.47859	0.04390
	0	0.47802	0.07525
	-0.1	0.47744	0.19650
0.2	0.064	0.23422	
	0.1	0.23443	0.08966
	0	0.23384	0.16224
	-0.1	0.23324	0.41841
0.3	0.064	0.15273	
	0.1	0.15295	0.14405
	0	0.15233	0.26190
	-0.1	0.15171	0.66785

代码如下：

```
    # kij = 0.064, p=0.1MPa情况下代码
    R290R600a = PR(
        Tc1=369.89,
        pc1=4.2512,
        omega1=0.1521,
        M1=44.096, # R290
        x1=0.5,
```

```
        Tc2=407.81,
        pc2=3.629,
        omega2=0.184,
        M2=58.122, # R600a
        kij=0.064,
        ps0=0.32979,
    )

    print(
        R290R600a.vg(300, 0.1)
        / (R290R600a.x1 * R290R600a.M1 + R290R600a.x2 * R290R600a.M2)
    )
```

第四章

4-13

利用主程序分别计算在 1.4MPa 下不同温度 T 下 R290 的液相焓和熵，以及在 0.6MPa 下不同温度 T 下 R600a 的液相焓和熵，计算结果与标准值对比如表 2 和表 3 所示，可以看出，计算结果与标准值误差均小于 1%。

表 2: 1.4MPa 下不同温度 T 下 R290 的液相焓和熵计算结果与标准值对比

T (K)	h (kJ/kg)	$h_{标准}$(kJ/kg)	s (kJ/(kg • K))	$s_{标准}$(kJ/(kg • K))	h 误差%	s 误差%
260	168.686	170.083	0.876	0.881	0.821	0.567
270	192.542	194.346	0.966	0.973	0.928	0.719
280	217.443	219.262	1.057	1.063	0.830	0.723
290	243.544	244.914	1.149	1.153	0.559	0.347
300	271.043	271.376	1.242	1.243	0.123	0.080

表 3: 0.6MPa 下不同温度 T 下 R600a 的液相焓和熵计算结果与标准值对比

T (K)	h (kJ/kg)	$h_{标准}$(kJ/kg)	s (kJ/(kg • K))	$s_{标准}$(kJ/(kg • K))	h 误差%	s 误差%
260	171.745	171.556	0.891	0.891	0.110	0
270	193.349	193.946	0.973	0.975	0.308	0.205
280	215.677	216.839	1.054	1.058	0.536	0.378
290	238.773	240.279	1.135	1.140	0.627	0.438
300	262.694	264.277	1.216	1.222	0.158	0.491

主程序如下：

```
R290 = PR(
    Tc1=369.89, # K
    pc1=4.2512, # MPa
    omega1=0.1521, # 无量纲
    M1=44.096, # g/mol
    x1=1.0,
    Tc2=369.89,
    pc2=4.2512,
    omega2=0.1521,
    M2=44.096,
    kij=0.064,
    ps0=0.47446,
)
```

```
R600a = PR(
    Tc1=407.81,
    pc1=3.629,
    omega1=0.184,
    M1=58.122, # R600a
    x1=1.0,
    Tc2=407.81,
    pc2=3.629,
    omega2=0.184,
    M2=58.122, # R600a
    kij=0.0,
    ps0=0.15696,
)
# 300K下结果
print(R290.h_l(300, -95.80, 6.945, -3.597 * 1e-3, 7.290 * 1e-7, 1.4))
print(R290.s_l(300, -95.80, 6.945, -3.597 * 1e-3, 7.290 * 1e-7, 1.4))
print(R600a.h_l(300, -23.91, 6.605, -3.176 * 1e-3, 4.981 * 1e-7, 0.6))
print(R600a.s_l(300, -23.91, 6.605, -3.176 * 1e-3, 4.981 * 1e-7, 0.6))
```

4-15

取二元作用系数 $k_{ij} = 0.064$,在 p=1.0MPa 下不同温度下计算 R290/R600a(50%/50%) 混合制冷剂的焓和熵，计算结果如表 4 所示，结果表明，计算结果与标准值误差很小。

表 4: 1.0MPa 下不同温度 T 下 R290/R600a(50%/50%) 混合制冷剂的液相焓和熵计算结果与标准值对比

T (K)	h (kJ/kg)	$h_{标准}$(kJ/kg)	s (kJ/(kg • K))	$s_{标准}$(kJ/(kg • K))	h 误差%	s 误差%
260	170.449	170.056	0.885	0.889	0.231	0.450
270	193.011	193.144	0.970	0.979	0.069	0.919
280	216.459	216.813	1.055	1.066	0.163	1.032
290	240.887	241.124	1.141	1.150	0.098	0.783
300	266.430	266.154	1.228	1.231	0.103	0.244

主程序如下：

```
R290R600a = PR(
    Tc1=369.89,
    pc1=4.2512,
```

```
        omega1=0.1521,
        M1=44.096, # R290
        x1=0.5,
        Tc2=407.81,
        pc2=3.629,
        omega2=0.184,
        M2=58.122, # R600a
        kij=0.064,
        ps0=0.32979,
    )
    # 300K下结果
    print(R290R600a.h_l(260, -59.81, 6.775, -3.386 * 1e-3, 6.135 * 1e-7, 1))
    print(R290R600a.s_l(260, -59.81, 6.775, -3.386 * 1e-3, 6.135 * 1e-7, 1))
```

第六章

6-11

推导过程见作业手写部分，1.4MPa 下 R290/R600a 混合制冷剂的 $\hat{\phi}$-T 图和 $\hat{f}$-T 图如下：

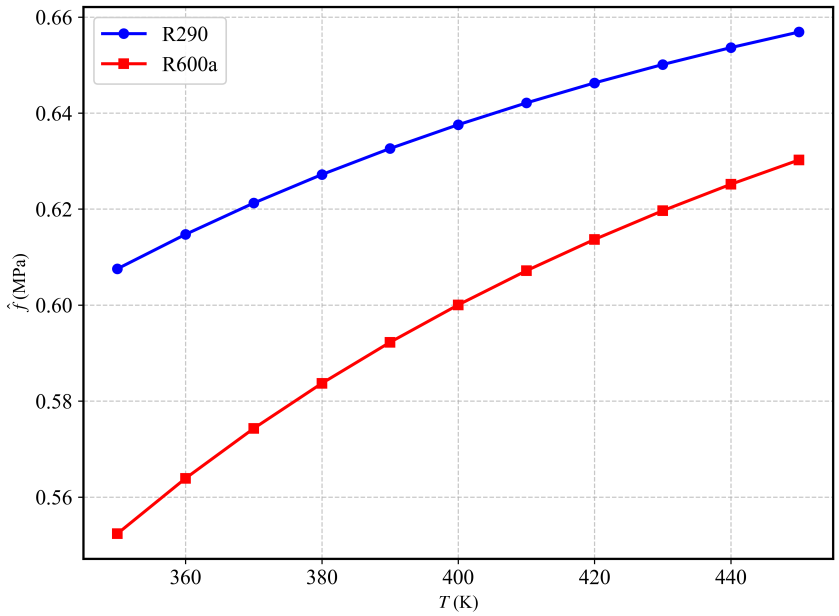

图 3: 1.4MPa 下 R290/R600a 混合制冷剂的 $\hat{\phi}$-T 图

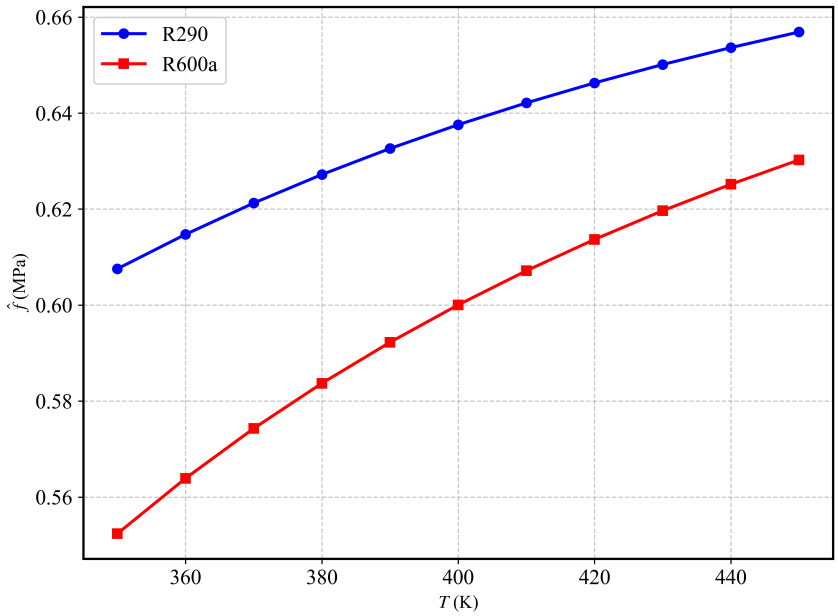

图 4: 1.4MPa 下 R290/R600a 混合制冷剂的 $\hat{f}$-T 图

主程序如下：

```
R290R600a = PR(
    Tc1=369.89,
    pc1=4.2512,
    omega1=0.1521,
    M1=44.096, # R290
    x1=0.5,
    Tc2=407.81,
    pc2=3.629,
    omega2=0.184,
    M2=58.122, # R600a
    kij=0.064,
    ps0=0.32979,
)
R290R600a.plot_fT("R290R600a", 1.4, 350, 450, 11)
```