高等工程热力学编程部分作业

何飏　3123101186

整个编程部分作业均通过 Python 语言完成，程序均在 Windows 10 操作系统下的 Anaconda3 环境中编写和运行，Python 版本为 3.13.2，所用主要第三方库包括 Numpy 和 Matplotlib，用于数值计算和绘图。

编程思路大致如下：

(1) 导入物质相关参数如临界温度、临界压力、偏心因子、摩尔质量等；

(2) 利用导入相关参数计算 PR 方程所需相关参数如 a、b、A、B 等；

(3) 使用牛顿法根据 PR 方程计算压缩因子 Z，计算时分别考虑液相和气相两种情况，液相初值取 0.001，气相初值取 1.1；

(4) 根据不同题目利用计算得到的压缩因子计算比体积、焓、熵等热力学性质，或绘制相图等。

然而，使用牛顿法进行压缩因子求解依赖初值，由于牛顿法就近收敛，初值不同会收敛到不同根，如液相迭代收敛到气相根，而且在接近临界压力或温度时，迭代慢，可以考虑采用解析解直接一次性拿到全部实根，再按相态选择。

第三章

3-10

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

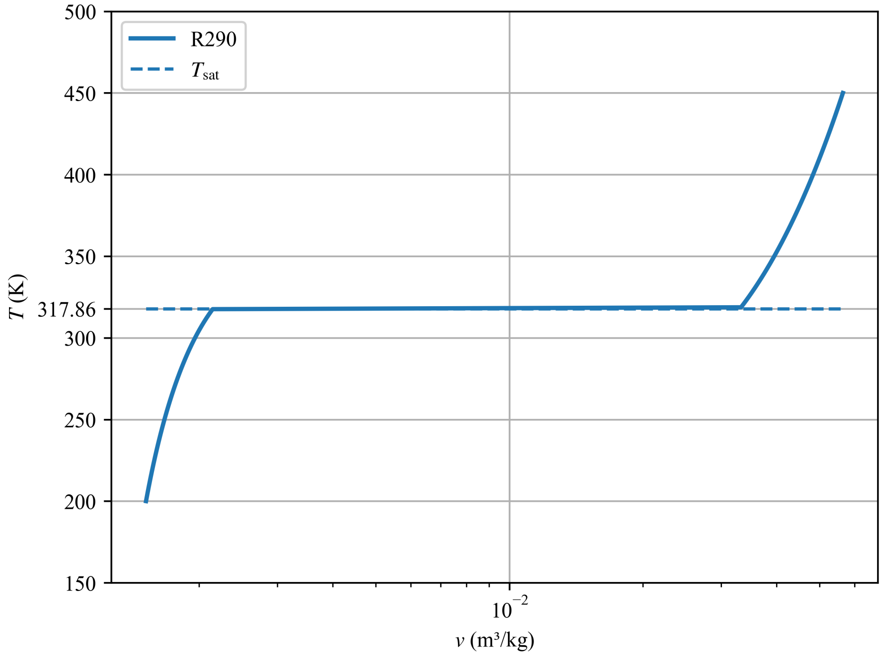

图 1: R290 在 1.4MPa 下的 T-v 图

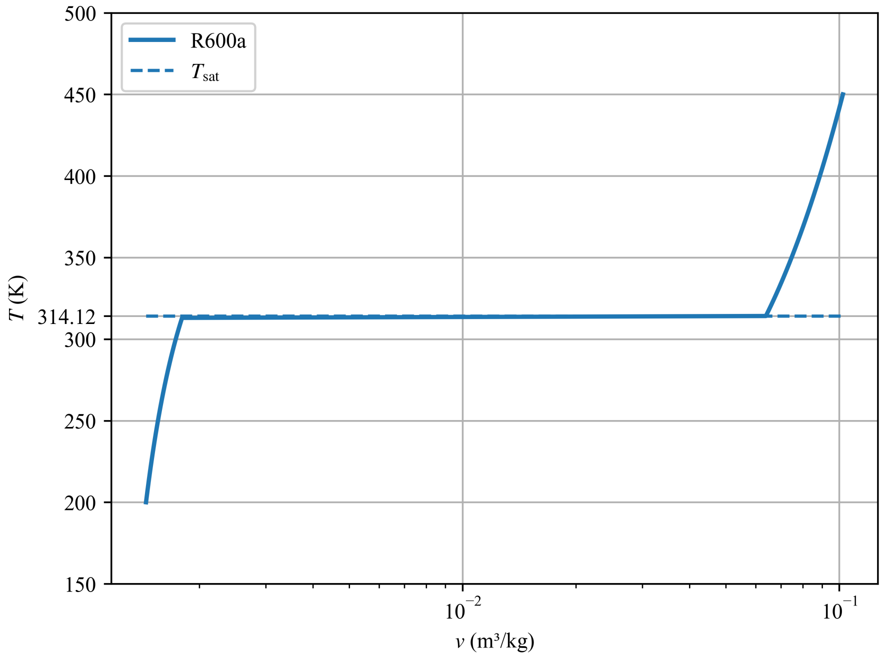

图 2: R600a 在 0.6MPa 下的 T-v 图

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"

class PR310:
    def __init__(self, Tc, Pc, omega, M):
        self.Tc = Tc # 输入K
        self.Pc = Pc * 1e6 # 输入MPa
        self.omega = omega # 无量纲
        self.M = M / 1000 # 输入g/mol

    R = 8.314462618 # J/(mol*K)

    # 计算a和b
    def params(self, T):
        kappa = 0.37464 + 1.54226 * self.omega - 0.26992 * self.omega**2
        Tr = T / self.Tc
        alpha = (1 + kappa * (1 - Tr**0.5)) ** 2
        a = 0.45724 * self.R**2 * self.Tc**2 / self.Pc * alpha
        b = 0.07780 * self.R * self.Tc / self.Pc
        return a, b

    # 计算A和B
    def AB(self, T, p):
        a, b = 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 = 1.1 # 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 * self.M)
        return vl

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

    # 画图
    def plot_Tv(
        self,
        fluid_name, # 流体名称
        p, # 压力 Pa
        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)
        elif T > Tsat:
            v_grid[i] = self.vg(T, p)
        else:
            v_grid[i] = 0.5 * (self.vl(T, p) + self.vg(T, p))
    fig, ax = plt.subplots() # 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)")
    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__))
```

```python
        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)

R290 = PR310(369.89, 4.2512, 0.1521, 44.096)
R290.plot_Tv("R290", 1.4, 317.86, 200, 450)

R600a = PR310(407.81, 3.629, 0.184, 58.122)
R600a.plot_Tv("R600a", 0.6, 314.12, 200, 450)
```

3-13

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

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

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

压力为 0.1MPa，温度为 35℃=308.15K 时，以上三种制冷剂均为气相，调用题 3-10 中程序计算三种制冷剂的 v_g。计算结果为：$v_{\mathrm{R134a}} = 0.24679\mathrm{m^3/kg}$，$v_{\mathrm{R1234yf}} = 0.22031\mathrm{m^3/kg}$，$v_{\mathrm{R1234ze(E)}} = 0.22007\mathrm{m^3/kg}$

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。

程序如下：

```python
import PR310 # 导入PR310模块

R134a = PR310.PR310(374.21, 4.0593, 0.326, 102.03)
R1234yf = PR310.PR310(367.85, 3.382, 0.276, 114.04)
R1234zeE = PR310.PR310(382.51, 3.635, 0.313, 114.04)

print(R134a.vg(308.15, 0.1))
print(R1234yf.vg(308.15, 0.1))
print(R1234zeE.vg(308.15, 0.1))
```

3-15

在压力 $p=0.1\text{MPa}$、0.2MPa、0.3MPa，温度 $T=300\text{K}$ 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

计算程序如下：

```
class PR315:
    def __init__(self, Tc1, Tc2, pc1, pc2, omega1, omega2, M1, M2, x1, kij):
        self.Tc1 = Tc1 # K
        self.Tc2 = Tc2 # K
        self.pc1 = pc1 * 1e6
        self.pc2 = pc2 * 1e6
        self.omega1 = omega1
        self.omega2 = omega2
        self.M1 = M1 / 1e3
        self.M2 = M2 / 1e3
        self.x1 = x1
        self.x2 = 1 - x1
        self.kij = kij

    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
```

```python
        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
        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
        return a, b

    # 计算A和B
    def AB(self, T, p):
        a, b = 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
```

```python
    def Zg(self, T, p):
        C2, C1, C0 = self.C(T, p)
        # 牛顿法求解Z
        Zg = 1.1 # 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 * (self.x1 * self.M1 + self.x2 * self.M2))
        ) # p从MPa转换为Pa
        return vl

    # 气相
    def vg(self, T, p):
        Zg = self.Zg(T, p)
        vg = (
            Zg * self.R * T / (p * 1e6 * (self.x1 * self.M1 + self.x2 * self.M2))
        ) # p从MPa转换为Pa
        return vg


R290_R600a_1 = PR315(
    369.89, 407.81, 4.2512, 3.629, 0.1521, 0.184, 44.096, 58.122, 0.5, 0.064
)
R290_R600a_2 = PR315(
    369.89, 407.81, 4.2512, 3.629, 0.1521, 0.184, 44.096, 58.122, 0.5, 0.1
)
R290_R600a_3 = PR315(
    369.89, 407.81, 4.2512, 3.629, 0.1521, 0.184, 44.096, 58.122, 0.5, 0
)
R290_R600a_4 = PR315(
    369.89, 407.81, 4.2512, 3.629, 0.1521, 0.184, 44.096, 58.122, 0.5, -0.1
)
print(R290_R600a_1.vg(300, 0.1))
print(R290_R600a_2.vg(300, 0.1))
print(R290_R600a_3.vg(300, 0.1))
print(R290_R600a_4.vg(300, 0.1))
```

```
print(R290_R600a_1.vg(300, 0.2))
print(R290_R600a_2.vg(300, 0.2))
print(R290_R600a_3.vg(300, 0.2))
print(R290_R600a_4.vg(300, 0.2))
print(R290_R600a_1.vg(300, 0.3))
print(R290_R600a_2.vg(300, 0.3))
print(R290_R600a_3.vg(300, 0.3))
print(R290_R600a_4.vg(300, 0.3))
```

第四章

4-13

利用程序分别计算在 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

程序如下：

```
import numpy as np

class PR413:
    def __init__(self, Tc, pc, omega, M, ps0):
        self.Tc = Tc # K
        self.pc = pc * 1e6
        self.omega = omega
        self.M = M / 1e3
        self.ps0 = ps0

    R = 8.314462618 # J/(mol*K)

    # 计算a和b
    def params(self, T):
        kappa = 0.37464 + 1.54226 * self.omega - 0.26992 * self.omega**2
        Tr = T / self.Tc
```

```python
        alpha = (1 + kappa * (1 - Tr**0.5)) ** 2
        a = 0.45724 * self.R**2 * self.Tc**2 / self.pc * alpha
        da = (
            -0.45724
            * self.R**2
            * self.Tc**2
            / self.pc
            * kappa
            * (1 + kappa * (1 - Tr**0.5))
            * (Tr**-0.5)
            / self.Tc
        )
        b = 0.07780 * self.R * self.Tc / self.pc
        return a, b, da

    # 计算A和B
    def AB(self, T, p):
        a, 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):
```

```python
        C2, C1, C0 = self.C(T, p)
        # 牛顿法求解Z
        Zg = 1.1 # 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

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

    # 计算焓的余函数
    # 液相
    def h_res_l(self, T, p):
        a, 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):
        a, 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
```

```python
    # 液相
    def s_res_l(self, T, p):
        a, 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):
        a, 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.M # 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.M
        ) # 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.M # 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.M
        ) # J/(kg*K)
        return sg


R290 = PR413(369.89, 4.2512, 0.1521, 44.096, 0.47446)
R600a = PR413(407.81, 3.629, 0.184, 58.122, 0.15696)

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。

程序如下：

```
import numpy as np
```

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	170.450	170.056	0.885	0.889	0.232	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	240.885	241.124	1.141	1.150	0.099	0.783
300	266.428	266.154	1.228	1.231	0.103	0.244

```

class PR415:
    def __init__(self, Tc1, pc1, omega1, M1, x1, Tc2, pc2, omega2, M2, kij, ps0):
        self.Tc1 = Tc1 # K
        self.pc1 = pc1 * 1e6
        self.omega1 = omega1
        self.M1 = M1 / 1e3
        self.x1 = x1

        self.Tc2 = Tc2 # K
        self.pc2 = pc2 * 1e6
        self.omega2 = omega2
        self.M2 = M2 / 1e3
        self.x2 = 1 - x1

        self.ps0 = ps0

        self.kij = kij

    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 a, b, da

    # 计算A和B
    def AB(self, T, p):
        a, 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
```

```python
        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 = 1.1 # 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

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

    # 计算焓的余函数
    # 液相
```

```
    def h_res_l(self, T, p):
        a, 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):
        a, 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):
        a, 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):
        a, 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, A1, A2, B1, B2, C1, C2, D1, D2):
        cp = (
            (A1 + A2) * 0.5 * (T - 273.15)
            + (B1 + B2) * 0.5 / 2 * (T**2 - 273.15**2)
```

```
            + (C1 + C2) * 0.5 / 3 * (T**3 - 273.15**3)
            + (D1 + D2) * 0.5 / 4 * (T**4 - 273.15**4)
        )
        return cp

    # 计算c_p/T积分
    def cpT(self, T, A1, A2, B1, B2, C1, C2, D1, D2):
        cp = (
            (A1 + A2) * 0.5 * np.log(T / 273.15)
            + (B1 + B2) * 0.5 * (T - 273.15)
            + (C1 + C2) * 0.5 / 2 * (T**2 - 273.15**2)
            + (D1 + D2) * 0.5 / 3 * (T**3 - 273.15**3)
        )
        return cp

    # 计算焓和熵
    # 液相
    def h_l(self, T, A1, A2, B1, B2, C1, C2, D1, D2, p):
        h_r_ps_0 = self.h_res_l(273.15, self.ps0)
        cp0 = self.cp(T, A1, A2, B1, B2, C1, C2, D1, D2)
        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, A1, A2, B1, B2, C1, C2, D1, D2, p):
        s_r_ps_0 = self.s_res_l(273.15, self.ps0)
        cpT = self.cpT(T, A1, A2, B1, B2, C1, C2, D1, D2)
        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, A1, A2, B1, B2, C1, C2, D1, D2, 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, A1, A2, B1, B2, C1, C2, D1, D2)
        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, A1, A2, B1, B2, C1, C2, D1, D2, 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, A1, A2, B1, B2, C1, C2, D1, D2)
        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


R290R600a = PR415(
    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下计算比焓和比熵
print(
    R290R600a.h_l(
        300,
        -95.80,
        -23.91,
        6.945,
        6.605,
        -3.597 * 1e-3,
        -3.176 * 1e-3,
        7.290 * 1e-7,
        4.981 * 1e-7,
        1.0,
    )
)
```

```python
print(
    R290R600a.s_l(
        300,
        -95.80,
        -23.91,
        6.945,
        6.605,
        -3.597 * 1e-3,
        -3.176 * 1e-3,
        7.290 * 1e-7,
        4.981 * 1e-7,
        1.0,
    )
)
```

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：

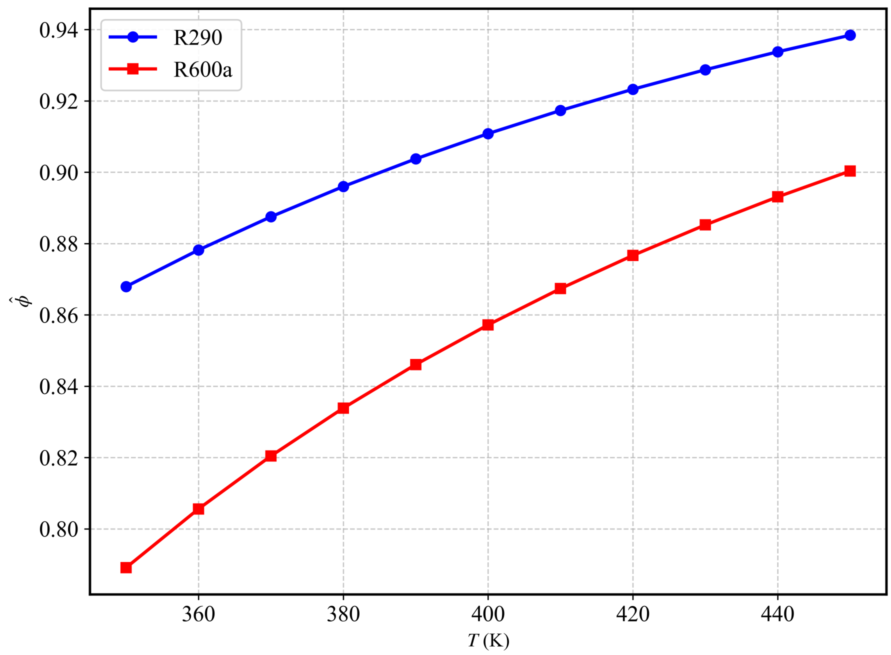

图 3: 1.4MPa 下 R290/R600a 混合制冷剂的 $\hat{\phi}$-T 图

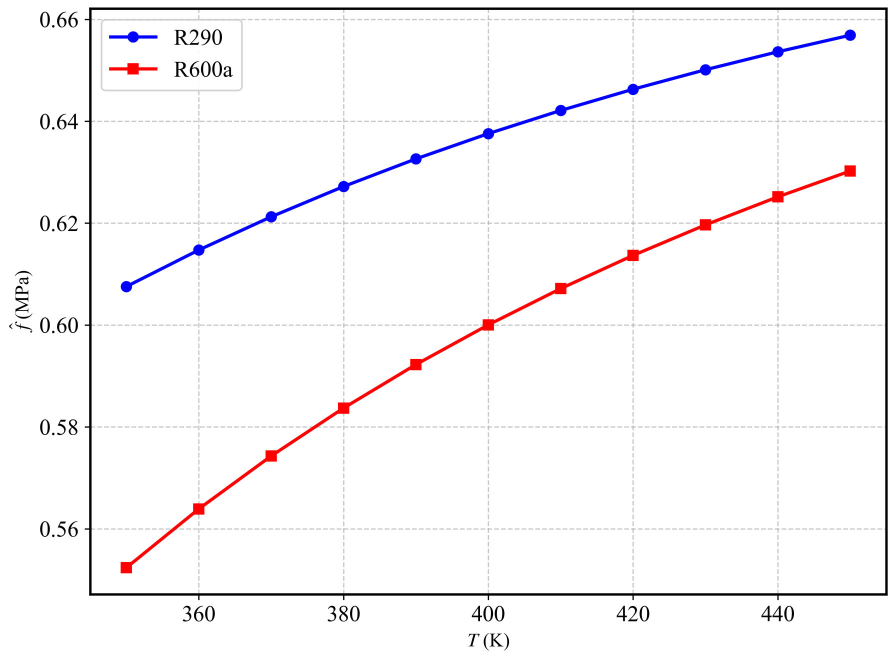

图 4: 1.4MPa 下 R290/R600a 混合制冷剂的 $\hat{f}$-T 图

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 PR611:
    def __init__(self, Tc1, pc1, omega1, Tc2, pc2, omega2, x1, kij):
        self.Tc1 = Tc1
        self.pc1 = pc1 * 1e6
        self.omega1 = omega1
        self.Tc2 = Tc2
        self.pc2 = pc2 * 1e6
        self.omega2 = omega2
        self.x1 = x1
        self.x2 = 1 - x1
        self.kij = kij

    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
    # 气相
    def Zg(self, T, p):
```

```python
        C2, C1, C0 = self.C(T, p)
        # 牛顿法求解Z
        Zg = 1.1 # 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
    # 气相
    def vg(self, T, p):
        Zg = self.Zg(T, p)
        vg = Zg * self.R * T / (p * 1e6)
        return vg # m³/mol

    # 计算逸度系数
    # 气相
    def phi_g(self, T, p):
        Zg = self.Zg(T, p)
        a1, a2, a, b1, b2, b, da = self.params(T)
        A, B = self.AB(T, p)
        phi_g1 = 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))
        )
        phi_g2 = 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))
        )
```

```python
        return phi_g1, phi_g2

    # 计算逸度
    # 气相
    def f_g(self, T, p): # MPa
        phi_g1, phi_g2 = self.phi_g(T, p)
        f_g1 = self.x1 * phi_g1 * p
        f_g2 = self.x2 * phi_g2 * p
        return f_g1, f_g2

    # 绘制溶液气相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):
            phi_g1, phi_g2 = self.phi_g(T, p)
            f_g1, f_g2 = self.f_g(T, p)
            phi_grid1[i] = phi_g1
            phi_grid2[i] = phi_g2
            f_grid1[i] = f_g1
            f_grid2[i] = f_g2

        # T-phi图
        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.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")
        fig_phi.savefig(savepath_phi, dpi=600, bbox_inches="tight", transparent=False)
        plt.close(fig_phi)

        # T-f图
        fig_f = plt.figure(figsize=(8, 6))
```

```python
        ax_f = fig_f.add_subplot(111)
        ax_f.plot(T_grid, f_grid1, "b-o", linewidth=2, label="R290", markersize=6)
        ax_f.plot(T_grid, f_grid2, "r-s", linewidth=2, label="R600a", markersize=6)
        ax_f.set_xlabel(r"$T$ (K)", fontsize=12)
        ax_f.set_ylabel(r"$\hat{f}$ (MPa)", fontsize=12)
        ax_f.grid(True, linestyle="--", alpha=0.7)
        ax_f.legend(loc="best", frameon=True, fancybox=True, framealpha=0.9)
        fig_f.tight_layout()
        savepath_f = os.path.join(fig_dir, f"{fluid_name}_f.png")
        fig_f.savefig(savepath_f, dpi=600, bbox_inches="tight", transparent=False)
        plt.close(fig_f)


R290R600a = PR611(
    Tc1=369.89,
    pc1=4.2512,
    omega1=0.1521,
    Tc2=407.81,
    pc2=3.629,
    omega2=0.184,
    x1=0.5,
    kij=0.064,
)

R290R600a.plot_fT("R290R600a", 1.4, 350, 450, 11)
```

第七章

7-3

式 (7.13) 为：

$$\ln p_\mathrm{r} = \ln T_\mathrm{r}[A_1 + A_2\tau^{1.89} + A_3\tau^{5.67}]$$

其中 $p_\mathrm{r} = p/p_\mathrm{c}$，$T_\mathrm{r} = T/T_\mathrm{c}$，$\tau = 1 - T_\mathrm{r}$。整理得：

$$p = p_\mathrm{c}(\frac{T}{T_\mathrm{c}})^{A_1 + A_2(1-\frac{T}{T_\mathrm{c}})^{1.89} + A_3(1-\frac{T}{T_\mathrm{c}})^{5.67}}$$

由书中表 7.1 可知甲烷、乙烷、丙烷和丁烷项-谭方程的相关常数，带入求解。绘制的蒸汽压曲线如下图所示：

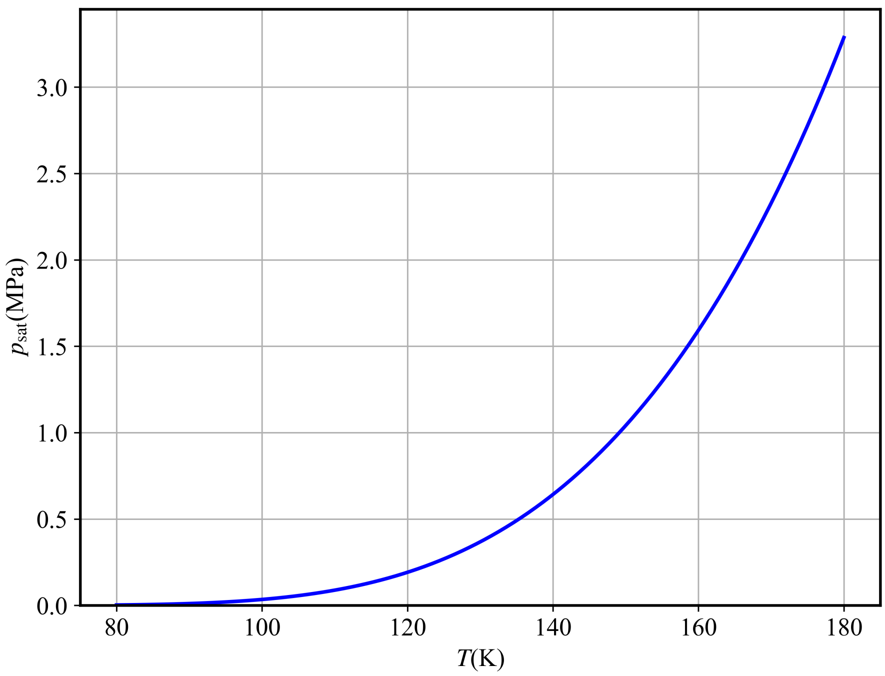

图 5: 甲烷的蒸汽压曲线

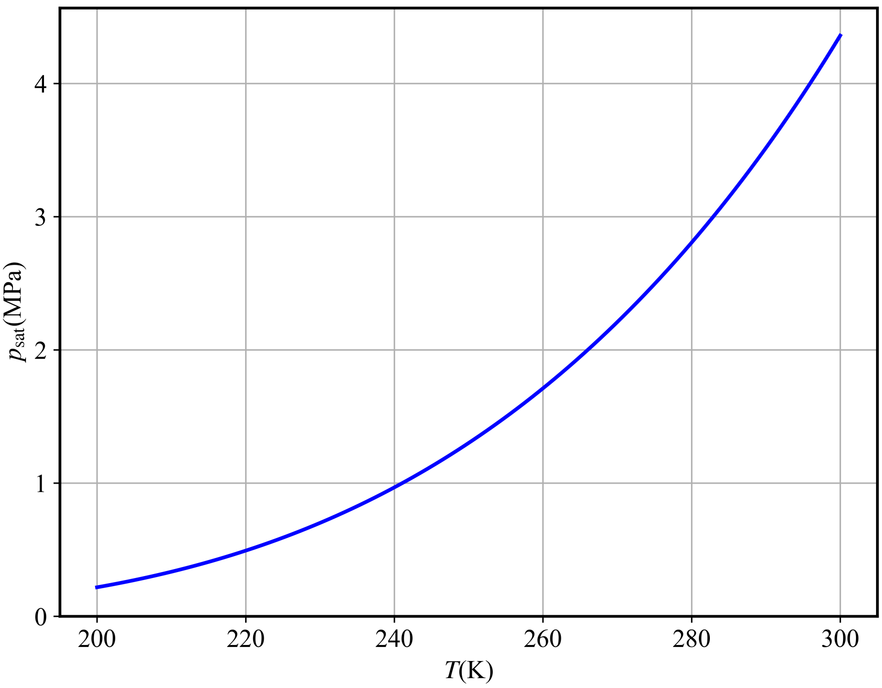

图 6: 乙烷的蒸汽压曲线

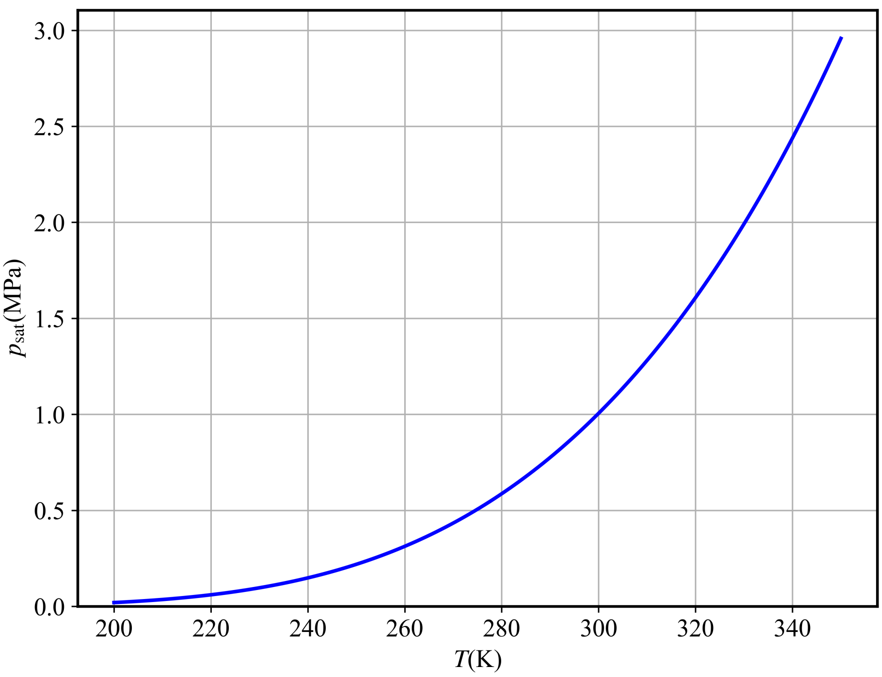

图 7: 丙烷的蒸汽压曲线

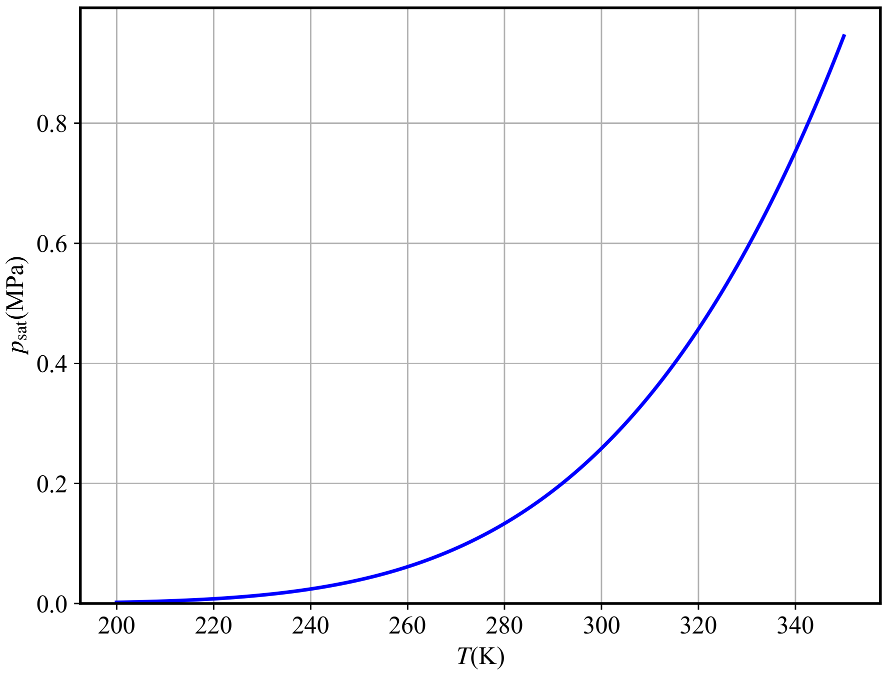

图 8: 丁烷的蒸汽压曲线

程序如下：

```
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 PR73:
    def __init__(self, Tc, pc, A1, A2, A3):
        self.Tc = Tc # K
        self.pc = pc * 1e6 # Pa, 输入MPa
        self.A1 = A1
        self.A2 = A2
        self.A3 = A3

    def psat(self, T):
        Tr = T / self.Tc
        p = self.pc * (Tr) ** (
            self.A1 + self.A2 * (1 - Tr) ** 1.89 + self.A3 * (1 - Tr) ** 5.67
        )
        return p / 1e6 # 输出MPa
```

```python

    def plot_psat(self, fluid_name, Tmin, Tmax, nt):
        T_grid = np.linspace(Tmin, Tmax, nt)
        psat_grid = np.zeros(nt)
        for i, T in enumerate(T_grid):
            psat_grid[i] = self.psat(T)

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

        fig = plt.figure(figsize=(8, 6))
        ax = fig.add_subplot(1, 1, 1)
        ax.plot(T_grid, psat_grid, "b-", linewidth=2)
        ax.set_ylim(bottom=0)
        ax.set_xlabel("T (K)", fontsize=14)
        ax.set_ylabel(r"$P_\mathrm{sat}$(MPa)", fontsize=14)
        ax.grid(True)

        # 保存图像
        savepath = os.path.join(fig_dir, f"{fluid_name}.png")
        fig.savefig(savepath, dpi=600, bbox_inches="tight", transparent=False)
        plt.close(fig)


CH4 = PR73(190.551, 4.5992, 5.87304544, 6.23280143, 13.0721578)
CH4.plot_psat("甲烷", 80, 180, 100)
C2H6 = PR73(305.33, 4.8717, 6.30717658, 7.47042131, 17.0958137)
C2H6.plot_psat("乙烷", 200, 300, 100)
C3H8 = PR73(369.80, 4.239, 6.50580501, 8.6776247, 18.0116214)
C3H8.plot_psat("丙烷", 200, 350, 150)
C4H10 = PR73(425.2, 3.8, 6.81692028, 8.77671813, 23.7680492)
C4H10.plot_psat("丁烷", 200, 350, 150)
```

7-4

制冷剂 R290、R600a、R1234yf 和 R1234ze(E) 的 p-T 相图如下所示：

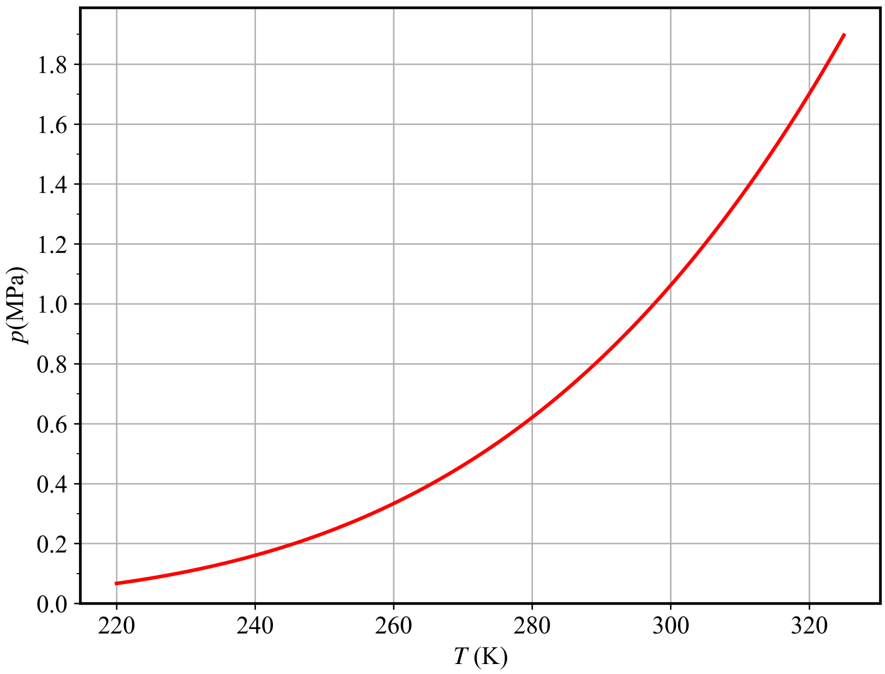

图 9: R290 的 p-T 相图

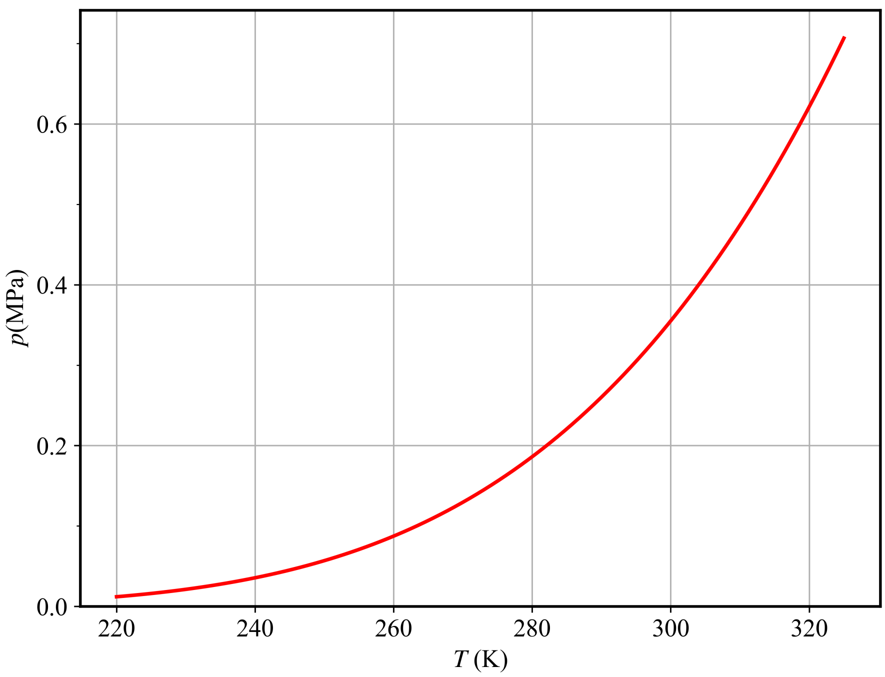

图 10: R600a 的 p-T 相图

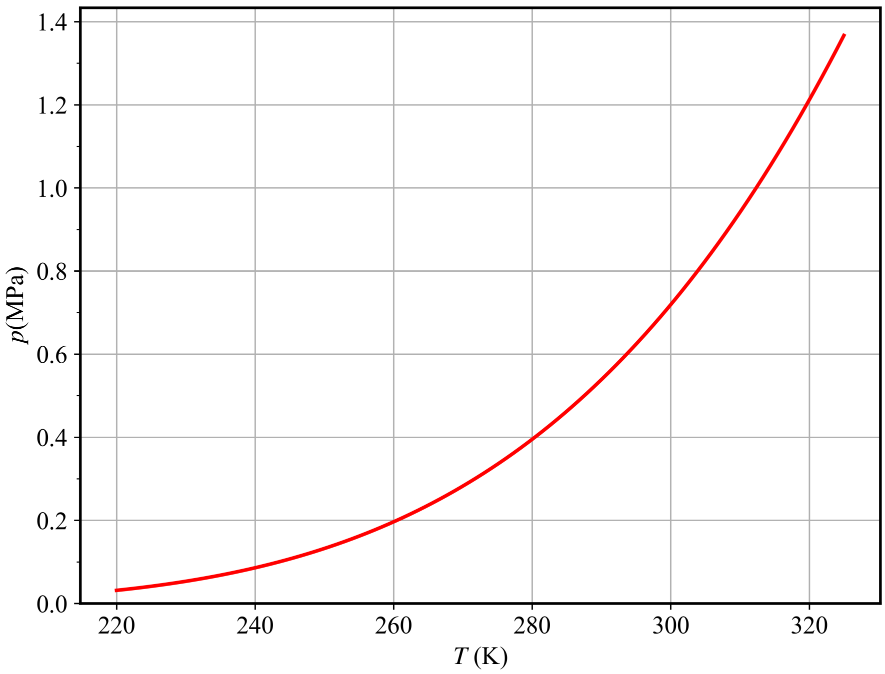

图 11: R1234yf 的 p-T 相图

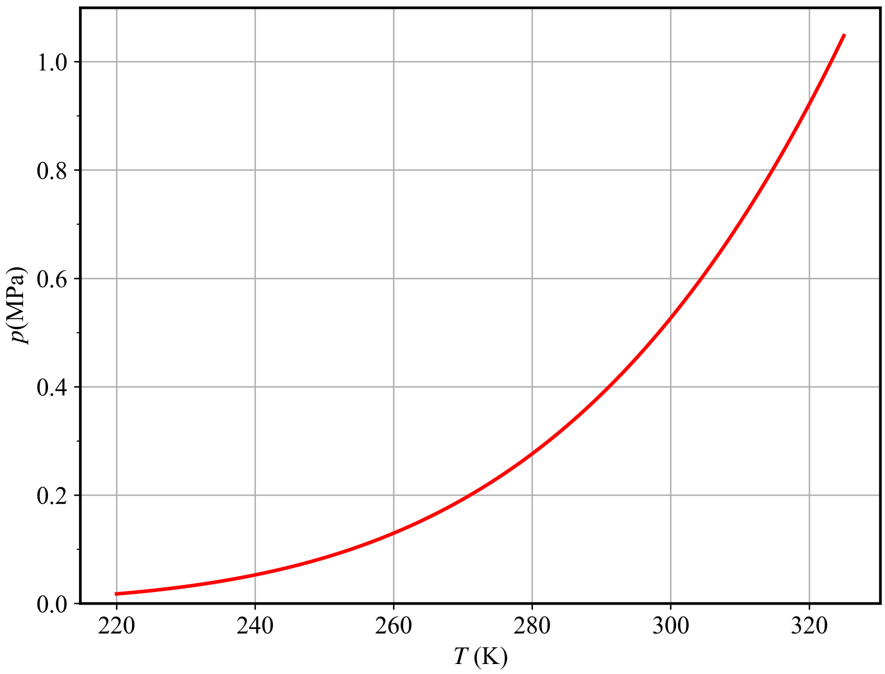

图 12: R1234ze(E) 的 p-T 相图

程序如下：

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
import os

# 使用 Times New Roman 作为 matplotlib 全局字体
```

```python
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 PR74:
    def __init__(self, Tc, pc, omega):
        self.Tc = Tc
        self.pc = pc * 1e6
        self.omega = omega

    R = 8.314462618 # J/(mol*K)

    def params(self, T):
        kappa = 0.37464 + 1.54226 * self.omega - 0.26992 * self.omega**2
        Tr = T / self.Tc
        alpha = (1 + kappa * (1 - Tr**0.5)) ** 2
        a = 0.45724 * self.R**2 * self.Tc**2 / self.pc * alpha
        b = 0.07780 * self.R * self.Tc / self.pc
        da = (
            -0.45724
            * self.R**2
            * self.Tc**2
            / self.pc
            * kappa
            * (1 + kappa * (1 - Tr**0.5))
            * (Tr**-0.5)
            / self.Tc
        )
        return a, b, da

    def AB(self, T, p):
        a, b, da = self.params(T)
        A = a * p * 1e6 / (self.R * T) ** 2
        B = b * p * 1e6 / (self.R * T)
        return A, B

    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

    def Zl(self, T, p):
```

```
        C2, C1, C0 = self.C(T, p)
        Zl = 0.001
        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

    def Zg(self, T, p):
        C2, C1, C0 = self.C(T, p)
        Zg = 1.1
        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

    def vl(self, T, p):
        Zl = self.Zl(T, p)
        vl = Zl * self.R * T / (p * 1e6)
        return vl

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

    def phi_l(self, T, p):
        Zl = self.Zl(T, p)
        a, b, da = self.params(T)
        A, B = self.AB(T, p)
        ln_phi_l = (
            Zl
            - 1
            - np.log(Zl - B)
            - A
            / (2 * (2**0.5) * B)
            * np.log((Zl + (1 + 2**0.5) * B) / (Zl + (1 - 2**0.5) * B))
        )
        phi_l = np.exp(ln_phi_l)
        return phi_l
```

```python

    def phi_g(self, T, p):
        Zg = self.Zg(T, p)
        a, b, da = self.params(T)
        A, B = self.AB(T, p)
        ln_phi_g = (
            Zg
            - 1
            - np.log(Zg - B)
            - A
            / (2 * (2**0.5) * B)
            * np.log((Zg + (1 + 2**0.5) * B) / (Zg + (1 - 2**0.5) * B))
        )
        phi_g = np.exp(ln_phi_g)
        return phi_g

    # 计算饱和压力
    def psat(self, T):
        from scipy.optimize import fsolve

        def objective(p):
            return self.phi_l(T, p) - self.phi_g(T, p)

        p_initial = 0.0001 # 初始猜测值，单位MPa
        psat_solution = fsolve(objective, p_initial)
        return psat_solution[0] # 返回平衡压力，单位MPa

    # 绘制p-T相图
    def plot_pT(self, T_min, T_max, savepath):
        T_grid = np.linspace(T_min, T_max, 100)
        p_grid = np.zeros_like(T_grid)
        for i, T in enumerate(T_grid):
            p_grid[i] = self.psat(T)

        fig = plt.figure(figsize=(8, 6))
        ax = fig.add_subplot(1, 1, 1)
        ax.plot(T_grid, p_grid, "r-", linewidth=2)
        ax.set_ylim(bottom=0)
        ax.yaxis.set_major_locator(MultipleLocator(0.2))
        ax.yaxis.set_minor_locator(MultipleLocator(0.1))
        ax.set_xlabel(r"$T$ (K)")
        ax.set_ylabel(r"$p$(MPa)")
        ax.grid(True)

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

```
        savepath = os.path.join(fig_dir, savepath)

        fig.savefig(savepath, dpi=600, bbox_inches="tight", transparent=False)
        plt.close(fig)


R290 = PR74(Tc=366.8, pc=4.248, omega=0.152)
R290.plot_pT(220, 325, "R290_pT.png")
R600a = PR74(Tc=407.8, pc=3.796, omega=0.227)
R600a.plot_pT(220, 325, "R600a_pT.png")
R1234yf = PR74(Tc=367.85, pc=3.3822, omega=0.276)
R1234yf.plot_pT(220, 325, "R1234yf_pT.png")
R1234ze = PR74(Tc=382.45, pc=3.6349, omega=0.313)
R1234ze.plot_pT(220, 325, "R1234ze(E)_pT.png")
```

7-5

按照书中图 7.22 的思路进行程序编写，绘制的 $p = 0.1\text{MPa}$ 和 $p = 1.0\text{MPa}$ 下溶液 R290/R600a 不同成分的泡点和露点温度的 T-x 相图如下所示：

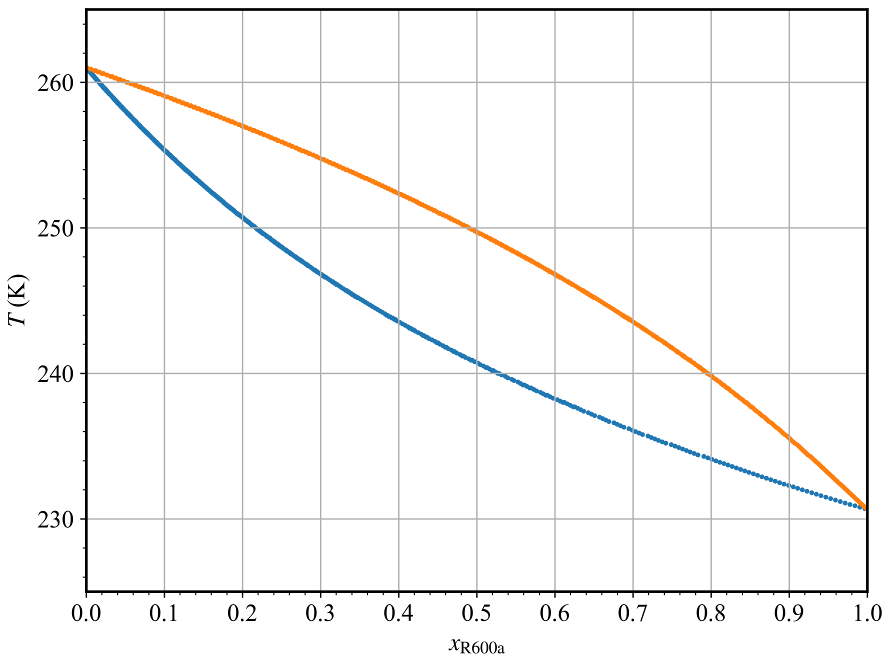

图 13: R290/R600a 在 0.1MPa 下的 T-x 相图

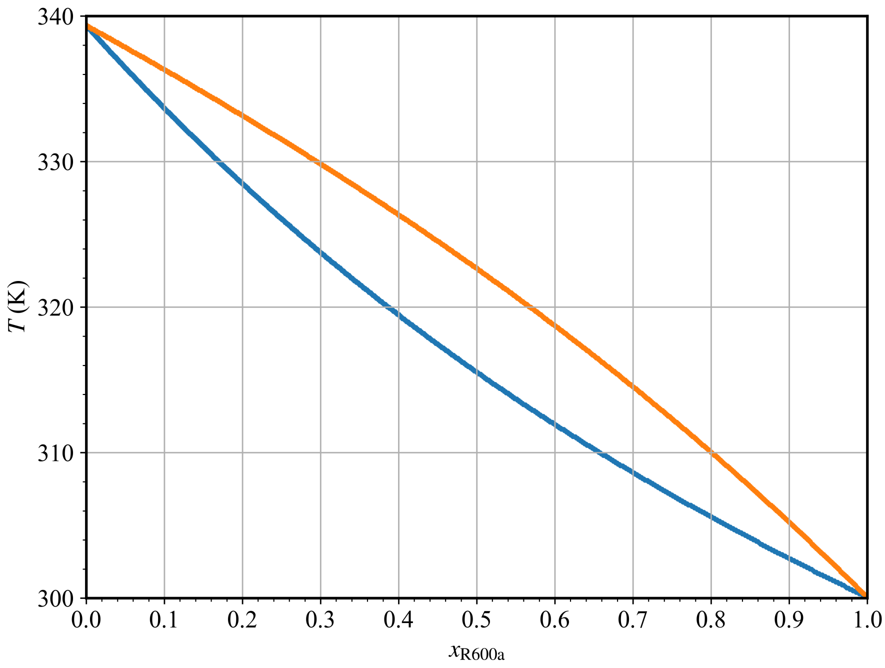

图 14: R290/R600a 在 1.0MPa 下的 T-x 相图

程序如下：

```
import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator

plt.rcParams["font.family"] = "serif"
plt.rcParams["font.serif"] = ["Times New Roman"]
plt.rcParams["mathtext.fontset"] = "stix"
plt.rcParams["font.size"] = 14
plt.rcParams["axes.linewidth"] = 1.5


class PR75:
    def __init__(self, Tc1, pc1, omega1, Tc2, pc2, omega2, kij):
        self.Tc1 = float(Tc1)
        self.pc1 = float(pc1) * 1e6
        self.omega1 = float(omega1)
        self.Tc2 = float(Tc2)
        self.pc2 = float(pc2) * 1e6
        self.omega2 = float(omega2)
        self.kij = float(kij)

    R = 8.314462618 # J/(mol·K)

    # 计算 a_i, b_i 以及混合 a, b; mu1 为该相中组分1摩尔分数
    def params(self, T, mu1):
```

```python
        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 - np.sqrt(Tr1))) ** 2
        alpha2 = (1 + kappa2 * (1 - np.sqrt(Tr2))) ** 2
        a1 = 0.45724 * (self.R * self.Tc1) ** 2 / self.pc1 * alpha1
        a2 = 0.45724 * (self.R * self.Tc2) ** 2 / self.pc2 * alpha2
        b1 = 0.07780 * self.R * self.Tc1 / self.pc1
        b2 = 0.07780 * self.R * self.Tc2 / self.pc2
        a = (
            (mu1**2) * a1
            + (1 - mu1) ** 2 * a2
            + 2 * mu1 * (1 - mu1) * np.sqrt(a1 * a2) * (1 - self.kij)
        )
        b = mu1 * b1 + (1 - mu1) * b2
        return a1, a2, a, b1, b2, b

    # 计算 A 与 B
    def AB(self, T, p, mu1):
        a1, a2, a, b1, b2, b = self.params(T, mu1)
        p_Pa = p * 1e6
        A = a * p_Pa / (self.R**2 * T**2)
        B = b * p_Pa / (self.R * T)
        return A, B

    # PR 三次的系数
    def C(self, T, p, mu1):
        A, B = self.AB(T, p, mu1)
        C2 = B - 1.0
        C1 = A - 3.0 * B**2 - 2.0 * B
        C0 = -(A * B - B**2 - B**3)
        return C2, C1, C0

    # 液相 Z
    def Zl(self, T, p, mu1):
        C2, C1, C0 = self.C(T, p, mu1)
        Z = 1.0e-3
        for _ in range(10000):
            f = Z**3 + C2 * Z**2 + C1 * Z + C0
            df = 3.0 * Z**2 + 2.0 * C2 * Z + C1
            Z_new = Z - f / df
            if abs(Z_new - Z) < 1e-6:
                Z = Z_new
                break
            Z = Z_new
        return Z
```

```python

    # 气相 Z
    def Zg(self, T, p, mu1):
        C2, C1, C0 = self.C(T, p, mu1)
        Z = 1.1
        for _ in range(10000):
            f = Z**3 + C2 * Z**2 + C1 * Z + C0
            df = 3.0 * Z**2 + 2.0 * C2 * Z + C1
            Z_new = Z - f / df
            if abs(Z_new - Z) < 1e-6:
                Z = Z_new
                break
            Z = Z_new
        return Z

    # 气相逸度系数
    def phi_g(self, T, p, mu1):
        Zg = self.Zg(T, p, mu1)
        a1, a2, a, b1, b2, b = self.params(T, mu1)
        A, B = self.AB(T, p, mu1)
        # 组分1
        lnphi1 = (
            b1 / b * (Zg - 1.0)
            - np.log(Zg - B)
            - A
            / (2.0 * np.sqrt(2.0) * B)
            * (
                2.0 * ((1 - mu1) * (1 - self.kij) * np.sqrt(a1 * a2) + mu1 * a1) / a
                - b1 / b
            )
            * np.log((Zg + 2.414 * B) / (Zg - 0.414 * B))
        )
        # 组分2
        lnphi2 = (
            b2 / b * (Zg - 1.0)
            - np.log(Zg - B)
            - A
            / (2.0 * np.sqrt(2.0) * B)
            * (
                2.0 * (mu1 * (1 - self.kij) * np.sqrt(a1 * a2) + (1 - mu1) * a2) / a
                - b2 / b
            )
            * np.log((Zg + 2.414 * B) / (Zg - 0.414 * B))
        )
        return np.exp(lnphi1), np.exp(lnphi2)

    # 液相逸度系数
```

```python
    def phi_l(self, T, p, mu1):
        Zl = self.Zl(T, p, mu1)
        a1, a2, a, b1, b2, b = self.params(T, mu1)
        A, B = self.AB(T, p, mu1)

        lnphi1 = (
            b1 / b * (Zl - 1.0)
            - np.log(Zl - B)
            - A
            / (2.0 * np.sqrt(2.0) * B)
            * (
                2.0 * ((1 - mu1) * (1 - self.kij) * np.sqrt(a1 * a2) + mu1 * a1) / a
                - b1 / b
            )
            * np.log((Zl + 2.414 * B) / (Zl - 0.414 * B))
        )

        lnphi2 = (
            b2 / b * (Zl - 1.0)
            - np.log(Zl - B)
            - A
            / (2.0 * np.sqrt(2.0) * B)
            * (
                2.0 * (mu1 * (1 - self.kij) * np.sqrt(a1 * a2) + (1 - mu1) * a2) / a
                - b2 / b
            )
            * np.log((Zl + 2.414 * B) / (Zl - 0.414 * B))
        )
        return np.exp(lnphi1), np.exp(lnphi2)

    def plot_Tx(self, p, T0):
        T_list = []
        x_bub_list = [] # 泡点: x1 vs T
        y_dew_list = [] # 露点: y1 vs T

        # y1 扫描
        y1_values = np.linspace(0.0, 1.0, 1001)

        for y1 in y1_values:
            y1 = float(y1)
            y2 = 1.0 - y1

            # 初值: 液相 x 猜 0.1/0.9; T 从 T0 逐步增加
            T = float(T0)
            x1 = 0.1
            s = 0.0 # s = Σ k_i y_i

```

```python
            # 外层调温: 使 s → 1（露点判据）
            # 注: 气相侧 ^v 用 y1; 液相侧 ^l 用 x1
            it_guard = 0
            while abs(s - 1.0) >= 1e-3:
                T += 0.1
                phi_g1, phi_g2 = self.phi_g(T, p, y1) # 气相
                phi_l1, phi_l2 = self.phi_l(T, p, x1) # 液相

                k1 = phi_g1 / phi_l1 # = 1/K1
                k2 = phi_g2 / phi_l2 # = 1/K2

                denom = k1 * y1 + k2 * y2
                if denom <= 1e-16:
                    break

                x1 = (k1 * y1) / denom
                s_prev = s
                s = denom

                # 细化循环: 仅重算液相侧
                inner_guard = 0
                while abs(s - s_prev) > 1e-6:
                    phi_l1, phi_l2 = self.phi_l(T, p, x1)
                    k1 = phi_g1 / phi_l1
                    k2 = phi_g2 / phi_l2
                    denom = k1 * y1 + k2 * y2
                    if denom <= 1e-16:
                        break
                    x1 = (k1 * y1) / denom
                    s_prev = s
                    s = denom

                    inner_guard += 1
                    if inner_guard > 2000: # 防止极端情况
                        break

                it_guard += 1
                if it_guard > 20000: # 防止极端情况
                    break

            # 收集边界点
            T_list.append(T)
            x_bub_list.append(x1) # 泡点边界（液相组成）
            y_dew_list.append(y1) # 露点边界（气相组成）

        fig, ax = plt.subplots(1, figsize=(8, 6))
        ax.scatter(x_bub_list, T_list, s=3, label=r"bubble: $T$-$x$")
```

```python
        ax.scatter(y_dew_list, T_list, s=3, label=r"dew: $T$ - $y$")
        ax.set_xlim(0.0, 1.0)
        ax.xaxis.set_major_locator(MultipleLocator(0.1))
        ax.xaxis.set_minor_locator(MultipleLocator(0.02))
        ax.yaxis.set_major_locator(MultipleLocator(10))
        ax.yaxis.set_minor_locator(MultipleLocator(2))
        ax.grid(True)

        ax.set_xlabel(r"$x_\mathrm{R600a}$")
        ax.set_ylabel(r"$T$ (K)")

        base_dir = os.path.dirname(os.path.abspath(__file__))
        fig_dir = os.path.join(base_dir, "figs")
        os.makedirs(fig_dir, exist_ok=True)
        filename = f"{p:.3f}MPa.png"
        savepath = os.path.join(fig_dir, filename)
        fig.savefig(savepath, dpi=600, bbox_inches="tight", transparent=False)
        plt.close(fig)


R290R600a = PR75(
    Tc1=369.89,
    pc1=4.2512,
    omega1=0.1521, # R290
    Tc2=407.81,
    pc2=3.629,
    omega2=0.184, # R600a
    kij=0.01,
)
R290R600a.plot_Tx(p=0.1, T0=215.0)
R290R600a.plot_Tx(p=1.0, T0=290.0)
```