МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

ПРОГРАММНЫЙ КОД

Системный анализ процессов химической технологии
Лабораторная работа №7
«Расчет химико-технологической системы переменной структуры»

Томск – 2023 г.

Листинги

Листинг 1 – Модуль констант constants.py

```
import numpy as np


MR = np.array(
    [
        128.1332, 2, 131.82935, 34, 214.3767, 253.8023, 103.615405,
        54.2102, 165.34111, 165.0947, 227.066815, 188.50365,
        82.6582345, 17, 97.455155, 205.99876, 144.39626, 159.18086,
        273.5151, 219.3049, 431.2175, 295.692, 129.36525, 283.3715
    ]
)

DENSITIES = np.array(
    [
        0.821537454674234, 8.56839230991066E-05, 0.634118153548788, 0.0138331933625558,
        0.825421792521393, 0.825421792521393, 1.02060976934116, 0.000184506047740076,
        1.11674713105835, 0.818624201288865, 1.21579774616092, 1.1604459318389,
        0.939038674550809, 0.0406332204207763, 0.953507833031478, 1.26240980032683,
        1.18472304338365, 1.06139531673633, 0.845814566218981, 1.21385557723734,
        2.3306027082957, 0.776867569431899, 0.679111733288024, 0.747735035578203
    ]
)

HEATCAPACITYCOEFFS = np.array(
    [
        [0.071254, 0.002979, -0.0000007, 0, 0],
        [13.83761, 0.0003, 0.000000346, -0.000000000097, 0.00000000000000773],
        [-0.09689, 0.003473, -0.0000013, 0.000000000256, -0.000000000000014],
        [0.9985, -0.00018, 0.000000557, -0.00000000032, 0.0000000000000637],
        [0.14303, 0.002308, -0.00000061, 0.0000000000547, 7.57E-22],
        [0.15129, 0.002146, -0.0000008, 0.000000000135, 6.1E-22],
        [-0.36405, 0.002663, -0.0000015, 0.000000000373, 0],
        [0.395, 0.002114, 0.000000396, -0.00000000067, 0.000000000000168],
        [-0.35765, 0.003037, -0.000002, 0.000000000774, -0.00000000000013],
        [-0.4231, 0.003185, -0.0000014, 0.000000000327, -0.000000000000021],
        [-0.73442, 0.003418, -0.0000019, 0.000000000421, 0.0000000000000158],
        [-0.53669, 0.003315, -0.0000017, 0.000000000386, 0],
        [-0.21475, 0.002651, -0.000001, 0.000000000176, 0],
        [1.9937, -0.00053, 0.00000206, -0.0000000013, 0.000000000000305],
        [-0.28424, 0.002585, -0.00000096, 0.0000000000691, 0.0000000000000351],
        [-0.70438, 0.00396, -0.0000026, 0.000000000973, -0.00000000000015],
        [-0.40807, 0.003038, -0.0000016, 0.000000000438, -0.00000000000005],
        [-0.18474, 0.002402, -0.0000008, 0.00000000000559, 0.0000000000000432],
        [-0.33844, 0.003705, -0.0000014, 0.000000000218, 5.85E-22],
        [-0.39433, 0.003057, -0.0000016, 0.000000000427, -0.000000000000047],
        [0, 0, 0, 0, 0],
        [-0.0581, 0.003377, -0.0000013, 0.000000000205, 4.22E-23],
        [0.38489, 0.001757, 0.000000762, -0.000000001, 0.000000000000272],
        [0.093835, 0.002912, -0.00000074, 0.00000000000262, -3.7E-24]
    ]
)

if __name__ == '__main__':
    ...
```

Листинг 2 – Вспомогательные функции (converters_and_functions.py)

```python
import numpy as np


def convert_mass_to_volume_fractions(
    mass_fractions: np.ndarray,
    density: np.ndarray
) -> np.ndarray:
    x = mass_fractions / density
    s = x.sum()
    return x / s


def convert_mass_to_mole_fractions(
    mass_fractions: np.ndarray,
    mr: np.ndarray
) -> np.ndarray:
    x = mass_fractions / mr
    s = x.sum()
    return x / s


def get_flow_density(
    mass_fractions: np.ndarray,
    density: np.ndarray
) -> float:
    return (mass_fractions / density).sum() ** -1


def get_average_mol_mass(
    mass_fractions: np.ndarray,
    mr: np.ndarray
) -> float:
    return (mass_fractions / mr).sum() ** -1


def get_flow_cp(
    mass_fractions: np.ndarray,
    coeffs: np.ndarray,
    temperature: float
) -> float:
    p = np.arange(coeffs.shape[1])
    component_cp = ((p + 1) * coeffs * temperature ** p).sum(axis=1)
    return (component_cp * mass_fractions).sum()


def normalize(x: np.ndarray) -> np.ndarray:
    return x / x.sum()


if __name__ == '__main__':
    import constants as const
    x = np.random.randint(1, 5, 24)
    x = normalize(x)
    t = 273.15
    cp = get_flow_cp(x, const.HEATCAPACITYCOEFFS, t)
    print(cp)
```

Листинг 3 – Описание класса `Flow` (`flows.py`)

```python
import numpy as np
import constants as const
import converters_and_functions as conv


class Flow:
    def __init__(
        self,
        mass_flow_rate: float,
        mass_fractions: np.ndarray,
        temperature: float
    ) -> None:

        self.mass_flow_rate = mass_flow_rate
        self.mass_fractions = mass_fractions
        self.temperature = temperature
        self.mole_fractions = conv.convert_mass_to_mole_fractions(
            self.mass_fractions, const.MR
        )
        self.volume_fractions = conv.convert_mass_to_volume_fractions(
            self.mass_fractions, const.DENSITIES
        )
        self.density = conv.get_flow_density(
            self.mass_fractions, const.DENSITIES
        )
        self.average_mol_mass = conv.get_average_mol_mass(
            self.mass_fractions, const.MR
        )
        self.mole_flow_rate = self.mass_flow_rate / self.average_mol_mass
        self.volume_flow_rate = self.mass_flow_rate / (self.density * 1e3)

    @property
    def flow_cp(self) -> float:
        flow_cp = conv.get_flow_cp(
            self.mass_fractions, const.HEATCAPACITYCOEFFS, self.temperature
        )
        return flow_cp


if __name__ == '__main__':
    x = np.random.randint(1, 5, 24)
    x = conv.normalize(x)
    t = 500
    g = 1000
    f = Flow(mass_flow_rate=g, mass_fractions=x, temperature=t)
    print(f.volume_fractions)
    print(f.mole_fractions)
    print(f.density)
    print(f.average_mol_mass)
    print(f.flow_cp)
```

Листинг 4 – Описание класса Mixer (mixer.py)

```python
import numpy as np
from scipy.optimize import fsolve
from flows import Flow


class Mixer:
    def mix(self, *flows: Flow) -> Flow:
        self.flows = flows
        mass_flow_rate = np.sum(
            [flow.mass_flow_rate for flow in self.flows]
        )
        mass_fractions = np.sum(
            [flow.mass_fractions * flow.mass_flow_rate for flow in self.flows],
            axis=0,
        ) / mass_flow_rate
        t_mean = np.mean(
            [flow.temperature for flow in self.flows]
        )
        self.mixture = Flow(
            mass_flow_rate=mass_flow_rate,
            mass_fractions=mass_fractions,
            temperature=t_mean
        )
        self.mixture.temperature = self.__calculate_temperature()
        return self.mixture

    def __calculate_temperature(self) -> float:
        def func(t):
            self.mixture.temperature = t
            t_ = np.sum(
                [flow.mass_flow_rate * flow.flow_cp * flow.temperature for flow in self.flows]
            ) / (self.mixture.mass_flow_rate * self.mixture.flow_cp)
            return t - t_

        temperature, = fsolve(func, self.mixture.temperature)
        return temperature


if __name__ == '__main__':
    import converters_and_functions as conv

    f1 = Flow(
        mass_flow_rate=100,
        mass_fractions=conv.normalize(np.random.randint(1, 5, 24)),
        temperature=200
    )
    f2 = Flow(
        mass_flow_rate=100,
        mass_fractions=conv.normalize(np.random.randint(1, 5, 24)),
        temperature=300
    )
    m = Mixer()
    fmixture = m.mix(f1, f2)
    print(fmixture.temperature)
```

Листинг 5 – Описание класса HeatExchanger (heat_exchanger.py)

```
import numpy as np
from flows import Flow


class HeatExchanger:
    def __init__(
        self,
        d_in: float = .1,
        d_out: float = .25,
        length: float = 3.0,
        k: float = 4900
    ) -> None:
        self.d_in = d_in
        self.d_out = d_out
        self.length = length
        self.k = k

    def calculate(
        self,
        hot: Flow,
        cold: Flow,
        h: float = .01
    ) -> tuple[Flow]:
        cold_space_velocity = (
            cold.volume_flow_rate
            / (np.pi * self.d_out ** 2 / 4 * self.length
            - np.pi * self.d_in ** 2 / 4 * self.length)
        )
        hot_space_velocity = (
            hot.volume_flow_rate
             / (np.pi * self.d_in ** 2 / 4 * self.length)
        )
        cold_ = Flow(
            mass_flow_rate=cold.mass_flow_rate,
            mass_fractions=cold.mass_fractions,
            temperature=cold.temperature
        )
        hot_ = Flow(
            mass_flow_rate=hot.mass_flow_rate,
            mass_fractions=hot.mass_fractions,
            temperature=hot.temperature
        )

        l = 0
        while l <= self.length:
            hot_.temperature -= (
                self.k * np.pi * self.d_in
                / (hot_space_velocity * hot_.density * 1e3 * hot_.flow_cp)
                * (hot_.temperature - cold_.temperature) * h
            )
            cold_.temperature += (
                self.k * np.pi * self.d_in
                / (cold_space_velocity * cold_.density * 1e3 * cold_.flow_cp)
                * (hot_.temperature - cold_.temperature) * h
            )
            l += h

        return hot_, cold_


if __name__ == '__main__':
    import converters_and_functions as conv
    mf1 = np.zeros(24)
    mf1[0] = .5
    mf1[2] = .3
    mf1[4] = .2
```

```
    mf2 = mf1
    cold = Flow(
        mass_flow_rate=1000,
        mass_fractions=mf1,
        temperature=273
    )
    hot = Flow(
        mass_flow_rate=1000,
        mass_fractions=mf2,
        temperature=300
    )
    he = HeatExchanger()
    h, c = he.calculate(hot, cold)
    print(h.temperature, c.temperature)
```

Листинг 6 – Описание класса `Splitter` (`splitter.py`)

```python
from flows import Flow


class Splitter:
    def calculate(self, flow: Flow, *ratio: float) -> list[Flow]:
        results = [
            Flow(
                mass_flow_rate=r * flow.mass_flow_rate,
                mass_fractions=flow.mass_fractions,
                temperature=flow.temperature
            )
            for r in ratio
        ]
        return results


if __name__ == '__main__':
    import numpy as np
    import converters_and_functions as conv

    f = Flow(
        mass_flow_rate=300,
        mass_fractions=conv.normalize(np.random.randint(1, 5, 24)),
        temperature=273.15
    )
    spl = Splitter()
    f1, f2, f3 = spl.calculate(f, .33333, .33333, .33)
    print(f1.mass_flow_rate, f2.mass_flow_rate, f3.mass_flow_rate)
```