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 №9
«Расчет реактора изомеризации легких бензиновых фракций»

Томск – 2023 г.

Листинги

Листинг 1 – Модуль констант `constants.py`

```
import numpy as np


MR = np.array(
    [
        72.15, 72.15, 70.00, 2.02, 86.18, 86.18,
        84.16, 78.11, 100.21, 100.21, 98.19, 92.14,
        132.00, 132.00, 130.00, 124.00, 44.00,
    ]
)

#Ideal gas densities kg / m**3
DENSITIES = np.array(
    [
        3.219153774, 3.219153774, 3.123226115, 0.090127382,
        3.845137523, 3.845137523, 3.755010141, 3.485074169,
        4.471121272, 4.471121272, 4.380993889, 4.111057918,
        5.889512103, 5.889512103, 5.800277071, 5.532571976,
        1.963170701
    ]
)

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, 0.00000000000000637],
        [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, 0.00000000000000158],
        [-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, 0.00000000000000351],
        [-0.70438, 0.00396, -0.0000026, 0.000000000973, -0.00000000000015],
        [-0.40807, 0.003038, -0.0000016, 0.000000000438, -0.00000000000005],
    ]
)

PREDEXP = np.array(
    [
        6.50737128e+14,  4.50039235e+12,  8.14767423e+10,  6.33560136e+09,
        4.38460466e+09,  4.92806984e+10,  1.94359453e+07,  0.00000000e+00,
        9.21320594e+08,  2.72093636e+12,  2.85483012e+07,  1.96159861e+06,
        1.04946695e+10,  2.13964788e+08,  3.68370523e+11,  2.22047032e+08,
        8.26480631e+11,  4.33828170e+09,  4.06633908e+07,  7.13770863e+12,
        1.59172009e+12,  1.09581311e+14,  1.65999518e+13,  1.83828592e+12,
        9.69949213e+12,  5.70976501e+10,  1.34815298e+08,  1.04974344e+10,
        1.55205848e+09,  5.30897948e+08,  8.95362445e+09,  1.23329037e+15,
        6.22558126e-04,  1.67138894e-03,  4.03935911e+09,  7.34862663e+09,
        7.06764011e+09,  5.15216787e+09,  4.31721853e+09,  4.46839658e+09,
        2.21399563e+09,  9.83099110e+07,  5.81334566e+11,  1.92057423e+07,
    ]
)

EA = np.array(
    [
        120.5, 120.5, 125.52, 125.52, 125.52, 125.52, 105.86, 105.86,
        158.99, 167.36, 104.6, 104.6, 158.99, 158.99, 104.6, 104.6,
        146.44, 146.44, 112.97, 112.97, 112.97, 112.97, 112.97, 112.97,
        112.97, 112.97, 112.97, 112.97, 112.97, 112.97, 112.97, 112.97,
```

```
        112.97, 112.97, 112.97, 112.97, 112.97, 112.97, 112.97, 112.97,
        112.97, 112.97, 112.97, 112.97,
    ]
)

indxs = np.array([0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ])

bed_params = [
    {
        'diameter': 1.2,
        'height': .2,
    },

    {
        'diameter': 1.2,
        'height': .3,
    },

    {
        'diameter': 1.2,
        'height': .5,
    },

    {
        'diameter': 1.2,
        'height': 1.7,
    },

    {
        'diameter': 1.5,
        'height': .25,
    },

    {
        'diameter': 1.5,
        'height': .35,
    },

    {
        'diameter': 1.5,
        'height': .55,
    },

    {
        'diameter': 1.5,
        'height': 1.75,
    }
]


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 convert_mass_to_molar_fractions(
    mf: np.ndarray,
    mw: np.ndarray,
    bulk_d20: float
) -> np.ndarray:
    return mf / mw * bulk_d20


def convert_molar_to_mass_fractions(
    molar_frac: np.ndarray,
    mw: np.ndarray,
    density: float
) -> np.ndarray:
    return molar_frac * mw / density


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__': ...
```

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

```
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,
        pressure: float
    ) -> None:

        self.mass_flow_rate = mass_flow_rate
        self.mass_fractions = mass_fractions
        self.temperature = temperature
        self.pressure = pressure

        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)

        self.molar_fractions = conv.convert_mass_to_molar_fractions(
            self.mass_fractions, const.MR, self.ideal_gas_density
        )

        self.ideal_gas_volume_flow_rate = (
            self.mass_flow_rate / self.ideal_gas_density
        )

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

    @property
    def ideal_gas_density(self) -> float:
        return (
            self.pressure * self.average_mol_mass / (8.314 * self.temperature)
        )


if __name__ == '__main__':
    x = np.random.randint(1, 5, 17)
    x = conv.normalize(x)
    t = 500
    p = 101.325
    g = 1000
    f = Flow(
        mass_flow_rate=g,
        mass_fractions=x,
        temperature=t,
```

```
        pressure=p
    )
    print(f.volume_fractions)
    print(f.molar_fractions)
    print(f.density)
    print(f.average_mol_mass)
```

Листинг 4 – Описание класса `Mixer` (`mixer.py`)

```
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]
        )
        pressure = min(flow.pressure for flow in self.flows)
        self.mixture = Flow(
            mass_flow_rate=mass_flow_rate,
            mass_fractions=mass_fractions,
            temperature=t_mean,
            pressure=pressure
        )
        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 кинетической схемы (kinetic.py)

```python
from __future__ import annotations
import numpy as np
import pandas as pd
import constants as const


kinetic_matrix = pd.read_excel('kinetic_matrix.xlsx')
kinetic_matrix = kinetic_matrix.iloc[:, 1:].to_numpy()


def arrhenius_law(
    temperature: float | np.ndarray,
    ea: float | np.ndarray,
    predexp: float | np.ndarray
) -> float | np.ndarray:
    return predexp * np.exp(-ea * 1e3 / (8.314 * temperature))


def kinetic_scheme(
    time: float,
    c: np.ndarray,
    temperature: float,
    predexp: np.ndarray = const.PREDEXP,
    ea: np.ndarray = const.EA,
    stoich_matrix: np.ndarray = kinetic_matrix
) -> np.ndarray:
    k = arrhenius_law(temperature, ea, predexp)
    mask = stoich_matrix < 0
    p = (c ** -(stoich_matrix * mask)).prod(axis=1)
    reaction_rates = p * k

    right_parts = (
        (stoich_matrix.T * reaction_rates).sum(axis=1)
    )

    return right_parts


if __name__ == '__main__':
    print(kinetic_matrix)
```

Листинг 6 – Описание класса Reactor (reactors.py)

```
import numpy as np
from scipy.integrate import solve_ivp
from flows import Flow
import constants as const
import converters_and_functions as conv
from kinetic import kinetic_scheme
from mixer import Mixer


class Bed:
    """Класс, используемый для описания полки катализатора"""
    def __init__(
            self,
            diameter: float = None,
            height: float = None,
            volume: float = None
    ) -> None:

        self.residence_time: float = 0

        if volume and height:
            self.volume = volume
            self.height = height
            self.diameter = (self.volume / (np.pi * self.height)) ** 0.5 * 2

        elif volume and diameter:
            self.volume = volume
            self.diameter = diameter
            self.height = self.volume / (np.pi * (self.diameter / 2) ** 2)

        elif diameter and height:
            self.diameter = diameter
            self.height = height
            self.volume = np.pi * (self.diameter / 2) ** 2 * self.height

        else:
            print('not enough data to create a catalyst bed')
            return

        self.area = np.pi * (self.diameter / 2) ** 2

    def __repr__(self) -> str:
        return f'Volume: {self.volume:.2f};\narea: {self.area:.2f};\nd: {self.diameter:.2f
    };\nh: {self.height:.2f}'

    def calculate(
        self,
        kinetic_scheme: callable,
        feedstock: Flow,
        ea: np.ndarray,
        predexp: np.ndarray
    ) -> Flow:
        self.residence_time = (
            self.volume / feedstock.ideal_gas_volume_flow_rate * 3600
        )
        solution = solve_ivp(
            kinetic_scheme,
            t_span=(0, self.residence_time),
            y0=feedstock.molar_fractions,
            args=(
                feedstock.temperature,
                predexp,
                ea
            )
        )
        molar_fractions = solution.y[:, -1]
```

```
        mf = conv.convert_molar_to_mass_fractions(
            molar_fractions, const.MR, feedstock.ideal_gas_density
        )
        product = Flow(
            mass_flow_rate=feedstock.mass_flow_rate,
            mass_fractions=mf,
            temperature=feedstock.temperature,
            pressure=feedstock.pressure
        )
        return product


class Reactor:
    """Класс для описания реактора, представленного в виде массива полок"""
    def __init__(
        self,
        *bed_params: dict
    ) -> None:
        self.beds = [Bed(**params) for params in bed_params]

    def calculate(
        self,
        kinetic_scheme: callable,
        feedstock: Flow,
        ea: np.ndarray = const.EA,
        predexp: np.ndarray = const.PREDEXP
    ) -> Flow:
        self.feedstock = feedstock
        f = self.feedstock

        for bed in self.beds:
            f = bed.calculate(
                kinetic_scheme=kinetic_scheme,
                feedstock=f,
                ea=ea,
                predexp=predexp
            )
        self.product = f
        return self.product

    def performance(self) -> dict:
        perf = {
            'Выход изомеризата, % масс.': (
                self.product.mass_fractions[const.indxs].sum()
                / self.feedstock.mass_fractions[const.indxs].sum()
            ) * 100,
            'Выход изоалканов, % масс.': (
                self.product.mass_fractions[[1, 5, 9, 13]].sum()
                / (1 - self.product.mass_fractions[[3, 16]].sum())
            ) * 100,
        }
        return perf


if __name__ == '__main__':
    mf = np.array(
        [
            0.1784, 0.0557, 0.0223, 0., 0.0948, 0.1171, 0.0446,
            0.0056, 0.1587, 0.146,  0.1066, 0.0168, 0.0123,
            0.0293, 0.0118, 0., 0.,
        ]
    )
    f = Flow(
        50_000,
        mf,
        temperature=403.15,
        pressure=3001.325
```

```
    )
    h2_mf = np.zeros_like(mf)
    h2_mf[3] = 1.
    h2 = Flow(
        mass_flow_rate=80,
        mass_fractions=h2_mf,
        temperature=403.15,
        pressure=3001.325
    )
    mxr = Mixer()
    feedstock = mxr.mix(f, h2)
    r = Reactor(*const.bed_params)
    product = r.calculate(kinetic_scheme, feedstock, predexp=const.PREDEXP)
    perf = r.performance()
    for key in perf:
        print(f'{key}: {perf[key]:.2f}')
    print(product.mass_fractions)
```

Листинг 7 – Описание основного модуля (`main.py`)

```
import numpy as np
import pandas as pd
import constants as const
from kinetic import kinetic_scheme
from flows import Flow
from mixer import Mixer
from reactors import Reactor


def calculation(t: float) -> tuple[float, float]:
    mf = np.array(
        [
            0.1784, 0.0557, 0.0223, 0., 0.0948, 0.1171, 0.0446,
            0.0056, 0.1587, 0.146,  0.1066, 0.0168, 0.0123,
            0.0293, 0.0118, 0., 0.,
        ]
    )
    f = Flow(
        50_000,
        mf,
        temperature=t,
        pressure=3001.325
    )
    h2_mf = np.zeros_like(mf)
    h2_mf[3] = 1.
    h2 = Flow(
        mass_flow_rate=80,
        mass_fractions=h2_mf,
        temperature=t,
        pressure=3001.325
    )
    mxr = Mixer()
    feedstock = mxr.mix(f, h2)
    r = Reactor(*const.bed_params)
    r.calculate(kinetic_scheme, feedstock, predexp=const.PREDEXP)
    perf = r.performance()
    isomerate_yield = perf['Выход изомеризата, % масс.']
    isoalkanes_yield = perf['Выход изоалканов, % масс.']
    return isomerate_yield, isoalkanes_yield



def main() -> None:
    t = np.linspace(350, 500, 20)
    vcalculation = np.vectorize(calculation)
    isomerate, isoalkanes = vcalculation(t)
    print('Температура, K',
          'Выход изомеризата, % масс.', 'Выход изоалканов, % масс.')
    for ti, isom, isoalk in zip(t, isomerate, isoalkanes):
        print(f'{ti:>14.2f} {isom:>26.2f} {isoalk:>25.2f}')
    results = pd.DataFrame(
        {
            'Температура, K': t,
            'Выход изомеризата, % масс.': isomerate,
            'Выход изоалканов, % масс.': isoalkanes
        }
    )
    results.to_excel('results.xlsx', index=False)


if __name__ == '__main__':
    main()
```