ОТЧЕТ ПО ПЕДАГОГИЧЕСКОЙ ПРАКТИКЕ

Содержание

1. ПОСТАНОВКА ЗАДАЧИ ПОТОРАСПРЕДЕЛЕНИЯ И ПРЕДСТАВЛЕНИЕ ТРУБОПРОВОДНОЙ СЕТИ В ВИДЕ ГРАФА

Задачу потокорапределения в гидравлических сетях можно сформулировать следующим образом: определить распределение потоков жидкости (или газа) по сети трубопроводов, а также в расчете давления или притоков/оттоков в различных точках сети с учетом заданных краевых условий.

В настоящем курсе будет рассмотрен стационарный изотермический расчет произвольной газотранспортной сети методом глобального градиента (МГГ).

Для проведения расчетов гидравлическую систему разумно представить в виде ориентированного графа, в котором направление ребер определяется направлением течения жидкости в трубопроводе. Направленные ребра также именуются дугами.

Расчетный граф состоит из двух типов объектов:

1. Узлы, которые хранят в себе состояние гидравлической системы в конкретной точке и информацию о входящих и выходящих дугах;
2. Дуги, которые содержат в себе конкретную реализацию гидравлических и термодинамических законов для объектов системы, например, трубопровод, кран-регулятор или компрессор. Если обобщить, то дуга – это объект, который описывает связь между двумя узлами.

Так как расчеты выполняются на компьютере, то направленный граф удобно представить в виде матрицы инциденций A, где столбцы характеризуют дуги, строки – вершины. Значения матрицы описывают направление дуг графа:

1. Если узел является начальной вершиной дуги, то значение равно 1;
2. Если узел является конечной вершиной дуги – -1;
3. Если узел не относится к дуге – 0.

Ход урока 1

Найдем матрицу инцидентности для графа на рисунке 1.

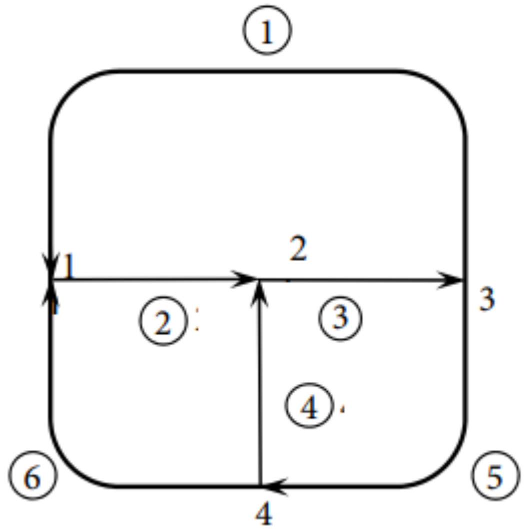

Рисунок 1 – Направленный граф

$$A = \left\| \begin{matrix} -1 & 1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 \end{matrix} \right\|$$

Дальше студентам демонстрируется шаблонный код на языке Python, который им предстоит заполнить.

Доступ к коду можно получить через GitHub по адресу: https://github.com/Paralamg/gga-solver/tree/student

Данной шаблон уже содержит классы объектов графа: узлы и дуги. Данный код можно запустить через main.py и увидеть информацию по созданному графу. Однако, решать задачу потокораспределения данный модуль пока не может. Этот функционал необходимо реализовать студентам.

На первом занятии необходимо заполнить пропуски (pass) в классе Graph:

```
class Graph:
    def __init__(self):
        self.nodes = []
        self.arcs = []
```

```python
        self.is_normal_result = False

    def add_node(self, node: Node):
        """
        Добавляет узел в граф
        :param node: узел
        """
        self.nodes.append(node)

    def add_arc(self, start: Node, end: Node, model):
        """
        Добавляет дугу в граф
        :param start: узел начала дуги
        :param end: узел конца дуги
        :param model: расчетная модель узла
        """
        arc = Arc(start, end, model)
        self.arcs.append(arc)

    def get_incidence_matrix(self) -> np.array:
        """
        Создает матрицу инцидентности

        :return: матрица инцидентности
        """
        pass

    def get_m(self) -> int:
        """
        Возвращает количество узлов в графе

        :return: количество узлов в графе
        """
        pass

    def get_n(self) -> int:
        """
        Возвращает количество дуг в графе

        :return: количество дуг в графе
        """
        pass

    def get_k(self) -> int:
        """
        Возвращает количество узлов с заданным притоком/оттоком
        :return:количество узлов с заданным притоком/оттоком
        """
        pass
```

```python
    def get_sorted_nodes(self) -> list[Node]:
        """
        Возвращает список узлов отсортированных по следующему правилу:
        узлы с известным притоком/оттоком находятся вначале списка,
        узлы с известным давлением

        :return: отсортированный список узлов
        """
        pass
```

Пример заполненного класса:

```python
class Graph:
    def __init__(self):
        self.nodes = []
        self.arcs = []
        self.is_normal_result = False

    def add_node(self, node: Node):
        """
        Добавляет узел в граф
        :param node: узел
        """
        self.nodes.append(node)

    def add_arc(self, start: Node, end: Node, model):
        """
        Добавляет дугу в граф
        :param start: узел начала дуги
        :param end: узел конца дуги
        :param model: расчетная модель узла
        """
        arc = Arc(start, end, model)
        self.arcs.append(arc)

    def get_incidence_matrix(self) -> np.array:
        """
        Создает матрицу инцидентности

        :return: матрица инцидентности
        """
        m = self.get_m()
        n = self.get_n()
        A = np.zeros((m, n))
        sorted_nodes = self.get_sorted_nodes()
        sorted_nodes[0].pressure = 1
        for num, arc in enumerate(self.arcs):
            A[sorted_nodes.index(arc.start_node), num] = 1
```

```
            A[sorted_nodes.index(arc.end_node), num] = -1
        return A

    def get_m(self) -> int:
        """
        Возвращает количество узлов в графе

        :return: количество узлов в графе
        """
        return len(self.nodes)

    def get_n(self) -> int:
        """
        Возвращает количество дуг в графе

        :return: количество дуг в графе
        """
        return len(self.arcs)

    def get_k(self) -> int:
        """
        Возвращает количество узлов с заданным притоком/оттоком
        :return:количество узлов с заданным притоком/оттоком
        """
        node_with_sign_flow = [node for node in self.nodes if node.sign ==
'flow']
        return len(node_with_sign_flow)

    def get_sorted_nodes(self) -> list[Node]:
        """
        Возвращает список узлов отсортированных по следующему правилу:
        узлы с известным притоком/оттоком находятся вначале списка,
        узлы с известным давлением

        :return: отсортированный список узлов
        """
        node_with_sign_flow = [node for node in self.nodes if node.sign ==
'flow']
        node_with_sign_pressure = [node for node in self.nodes if node.sign ==
'pressure']
        return node_with_sign_flow + node_with_sign_pressure
```

2. МЕТОД НЬЮТОНА

Метод Ньютона (или метод Ньютона-Рафсона) — это итерационный численный метод для решения нелинейных уравнений вида

$$f(x) = 0$$

Для того, чтобы найти корень уравнения методом Ньютона необходимо выполнить следующий итерационный алгоритм (рис. 1):

1. Задать начальное приближение $x^{(0)}$;
2. В точке O с координатами $\left(x^{(0)}, f\left(x^{(0)}\right)\right)$ проводится касательная к кривой $y = f(x)$;
3. Пересечении касательной с осью абсцисс дает первое приближение $x^{(1)}$;
4. Алгоритм повторяется с первого шага, где начальное приближение равно $x^{(1)}$.

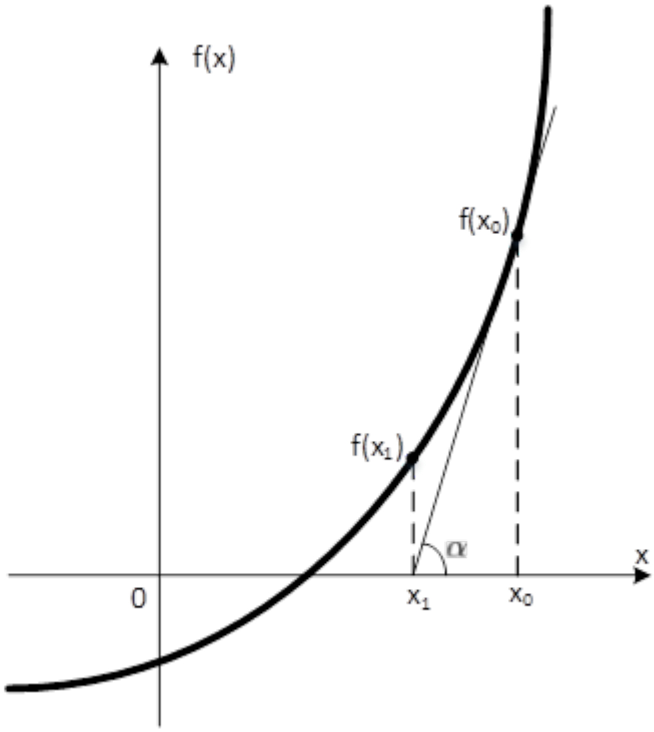

Рисунок 1 – Метод Ньютона

Введем следующее обозначение:

$$\Delta x^{(N)} = x^{(N+1)} - x^{(N)},$$

тогда с точностью до малых первого порядка относительно $\Delta x^{(N)}$получим:

$$f\left(x^{(N)} + \Delta x^{(N)}\right) \approx f\left(x^{(N)}\right) + f\left(x^{(N)}\right)' \Delta x^{(N)}.$$

Приравняв правую часть к нулю, получим уравнение касательной к графику функции $y = f(x)$:

$$f\left(x^{(N)}\right) + f'\left(x^{(N)}\right)\Delta x^{(N)} = 0$$

Откуда,

$$\Delta x^{(N)} = -\frac{f\left(x^{(N)}\right)}{f'(x^{(N)})}$$

Основное преимущество метода Ньютона быстрая сходимость (обычно квадратичная), если начальное приближение близко к корню. Однако, если начального приближения выбрано неудачно, метод может не сойтись или сходится медленно. Вместе с тем требуется вычисление производной $f'\left(x^{(N)}\right)$, что может быть затратно для сложных функций.

Метод Ньютона можно обобщить для решения систем нелинейных уравнений вида:

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0},$$

где $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ – вектор переменных;

$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_n(\boldsymbol{x}) \end{bmatrix}$ – вектор функций.

Принцип решения совпадает с приведенным ранее для нелинейного уравнения. На $(N+1)$-м шаге ищется вектор

$$\boldsymbol{x}^{(N+1)} = \boldsymbol{x}^{(N)} + \Delta\boldsymbol{x}^{(N)},$$

где $\Delta\boldsymbol{x}^{(N)}$ является решением линейной системы уравнений:

$$\boldsymbol{f}\left(\boldsymbol{x}^{(N)}\right) + \boldsymbol{f}'\left(\boldsymbol{x}^{(N)}\right)\Delta\boldsymbol{x}^{(N)} = \boldsymbol{0},$$

где $\boldsymbol{f}'\left(\boldsymbol{x}^{(N)}\right)$ – матрица Якоби, состоящая из первых частных производных функций

$$\boldsymbol{f}'\left(\boldsymbol{x}^{(N)}\right) = \begin{bmatrix} \dfrac{\partial f_1(x_1)}{\partial x_1} & \cdots & \dfrac{\partial f_1(x_n)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_n(x_1)}{\partial x_1} & \cdots & \dfrac{\partial f_n(x_n)}{\partial x_n} \end{bmatrix}$$

Ход урока 2

Пример решения нелинейного уравнения методом Ньютона проходился на курсе “Компьютерное моделирование технологических процессов трубопроводного транспорта углеводородов”, поэтому в данной лабораторной работе его подробно разбирать не будем.

Вместо этого на втором уроке студентам предлагается ознакомиться с классом Arc и Pipe. Внутри последнего необходимо реализовать два метода get_pressure_losses и get_pressure_derivatives, которые соответствуют функции $f(x)$ и производной этой функции $f'\left(x^{(N)}\right)$. Для этого формулу движения газа

$$q = 3{,}32 \cdot 10^{-6} \cdot d^{2,5}\sqrt{\frac{p_{\text{н}}^2 - p_{\text{к}}^2}{\lambda\Delta Z_{\text{ср}}T_{\text{ср}}L}}$$

нужно представить в следующем виде

$$p_{\text{н}}^2 - p_{\text{к}}^2 = \Lambda|q|q,$$

где

$$\Lambda = \frac{\lambda \Delta Z_{\text{ср}} T_{\text{ср}} L}{(3{,}32 \cdot 10^{-6})^2 \cdot d^5}$$

$$f(q) = \Lambda |q| q$$

Соответственно производная от этой функции по расходу равняется:

$$f'(q) = 2\Lambda |q|$$

В качестве облегчения задачи базовые методы для расчета газопровода были реализованы заранее.

Метод get_idem() постоянную часть для расчёта потенциала Λ.

Метод set_pressure() находит среднее значение давления на участке трубопровода и считает $p_{\text{кр}}$

Метод get_z_sto() возвращает значение коэффициента сжатия по формуле приведённой в СТО Газпром 202-3.5-051-2006

Теперь необходимо реализовать в класс Pipe методы get_pressure_losses и get_pressure_derivatives, которые соответствуют функциям выше:

```
class Pipe:
    def __init__(self, id: int, length: float, diameter: float, roughness: float):
        # Получить исходные данные для расчёта
        self.l = length
        self.d = diameter
        self.k = roughness
        self.id = id
        self.mu = 17.5
        self.t = 290
        self.p_sto = 101325
        self.t_crit = 200
        self.p_crit = 4.75 * 10 ** 6

        # Рассчитать постоянные значения
        self.gas_const = 8314 / self.mu
        self.t_pr = self.t / self.t_crit
        self.p_pr = self.p_sto / self.p_crit
        self.lamb = 0.067 * (2 * self.k / self.d) ** 0.2

    def get_pressure_losses(self, flow_rate: float, inlet_pressure: float, outlet_pressure: float):
        """
        Находит значение функции F(x) для конкретной модели
```

```
        :param flow_rate: расход
        :param inlet_pressure: давление на входе
        :param outlet_pressure: давление на выходе
        :return: значение функции F(x)
        """
        pass

    def get_pressure_derivatives(self, flow_rate: float, inlet_pressure: float,
outlet_pressure: float):
        """
        Находит значение производной функции F'(x) для конкретной модели

        :param flow_rate: расход
        :param inlet_pressure: давление на входе
        :param outlet_pressure: давление на выходе
        :return:
        """
        pass


    def get_idem(self):
        """
        Возвращает постоянную часть для расчёта потенциала
        """
        return 16 * self.lamb * self.get_z_sto() * self.gas_const * self.t *
self.l / (math.pi ** 2 * self.d ** 5)

    def set_pressure(self, inlet_pressure: float, outlet_pressure: float):
        """
        Находит среднее значение давления на участке трубопровода по формуле
приведённой в СТО Газпром 202-3.5-051-2006
        inlet_pressure - давление на входе
        outlet_pressure - давление на выходе
        """
        self.p_sto = 2 / 3 * (inlet_pressure + outlet_pressure ** 2 /
(inlet_pressure + outlet_pressure))
        self.p_pr = self.p_sto / self.p_crit

    def get_z_sto(self):
        """
        Возвращает значение коэффициента сжатия по формуле приведённой в СТО
Газпром 202-3.5-051-2006
        """
        self.z_A1 = -0.39 + 2.03 / self.t_pr - 3.16 / self.t_pr ** 2 + 1.09 /
self.t_pr ** 3
        self.z_A2 = 0.0423 - 0.1812 / self.t_pr + 0.2124 / self.t_pr ** 2
        self.z_sto = 1 + self.z_A1 * self.p_pr + self.z_A2 * self.p_pr ** 2
        return self.z_sto
```

Пример заполненных классов Arc и Pipe

```
class Arc:
    def __init__(self, start, end, model):
        self.start_node = start
        self.end_node = end
        self.model = model
        self.id = model.id
        self.flow_rate_calculated = 0

    def __str__(self):
        return f'{self.id:3}\t{self.start_node.id:>5} ->
{self.end_node.id:<3}\t{self.flow_rate_calculated:10.2f}
m3/s\t\t{self.start_node.pressure_calculated / 1e6:10.4f} MPa ->
{self.end_node.pressure_calculated / 1e6:10.4f} MPa'

    def get_pressure_losses(self, flow_rate: float):
        """
        Находит значение функции F(x) для дуги
        :param flow_rate: расход на дуге
        :return: значение функции F(x)
        """
        return self.model.get_pressure_losses(flow_rate,
self.start_node.pressure_calculated, self.end_node.pressure_calculated)

    def get_pressure_derivatives(self, flow_rate: float):
        """
        Находит значение производной функции F'(x) для дуги
        :param flow_rate: расход на дуге
        :return: значение производной функции F'(x)
        """
        return self.model.get_pressure_derivatives(flow_rate,
self.start_node.pressure_calculated, self.end_node.pressure_calculated)
class Pipe:
    def __init__(self, id: int, length: float, diameter: float, roughness:
float):
        # Получить исходные данные для расчёта
        self.l = length
        self.d = diameter
        self.k = roughness
        self.id = id
        self.mu = 17.5
        self.t = 290
        self.p_sto = 101325
        self.t_crit = 200
        self.p_crit = 4.75 * 10 ** 6

        # Рассчитать постоянные значения
        self.gas_const = 8314 / self.mu
        self.t_pr = self.t / self.t_crit
```

```
        self.p_pr = self.p_sto / self.p_crit
        self.lamb = 0.067 * (2 * self.k / self.d) ** 0.2

    def get_pressure_losses(self, flow_rate: float, inlet_pressure: float,
outlet_pressure: float):
        """
        Находит значение функции F(x) для конкретной модели

        :param flow_rate: расход
        :param inlet_pressure: давление на входе
        :param outlet_pressure: давление на выходе
        :return: значение функции F(x)
        """
        self.set_pressure(inlet_pressure, outlet_pressure)
        idem = self.get_idem()
        return idem * flow_rate * abs(flow_rate)

    def get_pressure_derivatives(self, flow_rate: float, inlet_pressure: float,
outlet_pressure: float):
        """
        Находит значение производной функции F'(x) для конкретной модели

        :param flow_rate: расход
        :param inlet_pressure: давление на входе
        :param outlet_pressure: давление на выходе
        :return:
        """
        self.set_pressure(inlet_pressure, outlet_pressure)
        idem = self.get_idem()
        return 2 * idem * abs(flow_rate)


    def get_idem(self):
        """
        Возвращает постоянную часть для расчёта потенциала
        """
        return 16 * self.lamb * self.get_z_sto() * self.gas_const * self.t *
self.l / (math.pi ** 2 * self.d ** 5)

    def set_pressure(self, inlet_pressure: float, outlet_pressure: float):
        """
        Находит среднее значение давления на участке трубопровода по формуле
приведённой в СТО Газпром 202-3.5-051-2006
        inlet_pressure - давление на входе
        outlet_pressure - давление на выходе
        """
        self.p_sto = 2 / 3 * (inlet_pressure + outlet_pressure ** 2 /
(inlet_pressure + outlet_pressure))
        self.p_pr = self.p_sto / self.p_crit
```

```
    def get_z_sto(self):
        """
        Возвращает значение коэффициента сжатия по формуле приведённой в СТО Газпром 202-3.5-051-2006
        """
        self.z_A1 = -0.39 + 2.03 / self.t_pr - 3.16 / self.t_pr ** 2 + 1.09 / self.t_pr ** 3
        self.z_A2 = 0.0423 - 0.1812 / self.t_pr + 0.2124 / self.t_pr ** 2
        self.z_sto = 1 + self.z_A1 * self.p_pr + self.z_A2 * self.p_pr ** 2
        return self.z_sto
```

3. МЕТОД ГЛОБАЛЬНОГО ГРАДИЕНТА

Метод глобального градиента основывается на законах Кирхгофа.

Первый закон Кирхгофа для гидравлической цепи утверждает: в любом узле гидравлической цепи сумма потоков, входящих в узел, равна сумме потоков, выходящих из узла.

Векторно-матричная форма записи первого закона Кирхгофа имеет вид:

$$\boldsymbol{Ax} = \boldsymbol{Q}$$

Второй закон Кирхгофа утверждает, что для любого замкнутого контура в гидравлической цепи сумма изменений давления на всех элементах контура равна нулю:

$$\boldsymbol{By} = \boldsymbol{0},$$

где $\boldsymbol{B}$ – цикломатическая матрица, или матрица контуров.

Однако последнее уравнение заменяется на другое, таким образом рассматривается система:

$$\boldsymbol{A}^T\boldsymbol{P} = \boldsymbol{F}(\boldsymbol{x})$$
$$\boldsymbol{Ax} = \boldsymbol{Q}$$

Матрицу $\boldsymbol{A}$ и векторы $\boldsymbol{P}, \boldsymbol{Q}$ необходимо записать в блочном виде, так чтобы узлы с известными из условия притоками и оттоками оказались в верхнем блоке, а в нижнем – с известными потенциалами. Таким образом, получим:

$$\boldsymbol{A} = \begin{Vmatrix} \boldsymbol{A_1} \\ \boldsymbol{A_2} \end{Vmatrix};\ \boldsymbol{P} = \begin{Vmatrix} \boldsymbol{P_1} \\ \boldsymbol{P_2} \end{Vmatrix};\ \boldsymbol{A} = \begin{Vmatrix} \boldsymbol{Q_1} \\ \boldsymbol{Q_2} \end{Vmatrix}$$

где $\boldsymbol{A_1}, \boldsymbol{P_1}, \boldsymbol{Q_1}$ – это матрица инцидентности и векторы давления и притоков соответственно, которые отвечают вершина с заданными внешними притоками-отборами.

$\boldsymbol{A_2}, \boldsymbol{P_2}, \boldsymbol{Q_2}$ - отвечают вершинам с известным значением потенциала.

Размер матрицы $\boldsymbol{A_1}$ равен $k \times n$, а матрицы $\boldsymbol{A_2}$ – $(m - k) \times n$, где k – это количество вершин с известным притоком-отбором. Соответственно для размер векторов $\boldsymbol{P_1}, \boldsymbol{Q_1}$ равен k, а $\boldsymbol{P_2}, \boldsymbol{Q_2}$ – $(m - k)$.

Приведем систему уравнений к следующему виду:

$$\boldsymbol{A_1^T P_1} = \boldsymbol{F}(\boldsymbol{x}) - \boldsymbol{A_2^T P_2}$$
$$\boldsymbol{A_1 x} = \boldsymbol{Q_1}$$

Данная система уравнений решается методом Ньютона непосредственно. Для этого произведем линеаризацию вектор-функции $\boldsymbol{F}(\boldsymbol{x})$

$$\boldsymbol{F}\left(\boldsymbol{x}^{(N)} + \boldsymbol{\Delta x}^{(N)}\right) \approx \boldsymbol{F}\left(\boldsymbol{x}^{(N)}\right) + \boldsymbol{F}'\left(\boldsymbol{x}^{(N)}\right)\boldsymbol{\Delta x}^{(N)},$$

где $\boldsymbol{F}'\left(\boldsymbol{x}^{(N)}\right)$ – диагональная матрица Якоби.

Учитываю формулу () на N шаге решается линейная система уравнений:

$$\boldsymbol{A_1^T P_1^{(N+1)}} = \boldsymbol{F}\left(\boldsymbol{x}^{(N)}\right) + \boldsymbol{F}'\left(\boldsymbol{x}^{(N)}\right)\boldsymbol{\Delta x}^{(N)} - \boldsymbol{A_2^T P_2}$$
$$\boldsymbol{A_1 x}^{(N)} + \boldsymbol{A_1 \Delta x}^{(N)} = \boldsymbol{Q_1}$$

Из первого уравнения можно без труда выразить $\boldsymbol{\Delta x}^{(N)}$:

$$\boldsymbol{\Delta x}^{(N)} = \boldsymbol{F}'\left(\boldsymbol{x}^{(N)}\right)^{-1}\left(\boldsymbol{A_1^T P_1^{(N+1)}} - \boldsymbol{F}\left(\boldsymbol{x}^{(N)}\right) + \boldsymbol{A_2^T P_2}\right)$$

Стоит отметить, что обратная матрицы $\boldsymbol{F}'\left(\boldsymbol{x}^{(N)}\right)^{-1}$ вычисляется просто, так как $\boldsymbol{F}'\left(\boldsymbol{x}^{(N)}\right)$ является диагональной.

Затем подставляем уравнение () в ()

$$\boldsymbol{A_1 x^{(n)}} + \boldsymbol{A_1 F'}\left(\boldsymbol{x}^{(N)}\right)^{-1}\left(\boldsymbol{A_1^T P_1^{(N+1)}} - \boldsymbol{F}\left(\boldsymbol{x}^{(N)}\right) + \boldsymbol{A_2^T P_2}\right) = \boldsymbol{Q_1}$$

После преобразования получим:

$$\boldsymbol{M}\left(\boldsymbol{x}^{(N)}\right)\boldsymbol{P_1^{(N+1)}} = \boldsymbol{Q_1} - \boldsymbol{A_1 x^{(n)}} + \boldsymbol{A_1 F'}\left(\boldsymbol{x}^{(N)}\right)^{-1}\left(\boldsymbol{F}\left(\boldsymbol{x}^{(N)}\right) - \boldsymbol{A_2^T P_2}\right),$$

где $\boldsymbol{M}\left(\boldsymbol{x}^{(N)}\right) = \boldsymbol{A_1 F'}\left(\boldsymbol{x}^{(N)}\right)^{-1}\boldsymbol{A_1^T}$ – матрица Максвелла.

Преобразовав получим:

$$\boldsymbol{P_1^{(N+1)}} = \boldsymbol{M}\left(\boldsymbol{x}^{(N)}\right)^{-1}\left(\boldsymbol{Q_1} - \boldsymbol{A_1 x^{(n)}} + \boldsymbol{A_1 F'}\left(\boldsymbol{x}^{(N)}\right)^{-1}\left(\boldsymbol{F}\left(\boldsymbol{x}^{(N)}\right) - \boldsymbol{A_2^T P_2}\right)\right),$$

Следующее приближение по расходу можно определить по формулы

$$\boldsymbol{x}^{(N+1)} = \boldsymbol{x}^{(N)} + \boldsymbol{\Delta x}^{(N)} = \boldsymbol{x}^{(N)} + \boldsymbol{F'}\left(\boldsymbol{x}^{(N)}\right)^{-1}\left(\boldsymbol{A_1^T P_1^{(N+1)}} - \boldsymbol{F}\left(\boldsymbol{x}^{(N)}\right) + \boldsymbol{A_2^T P_2}\right)$$

или

$$\boldsymbol{x}^{(N+1)} = \boldsymbol{x}^{(N)} + \boldsymbol{F'}\left(\boldsymbol{x}^{(N)}\right)^{-1}\left(\boldsymbol{A^T P^{(N+!)}} - \boldsymbol{F}\left(\boldsymbol{x}^{(N)}\right)\right)$$

Ход занятия 3

Студентам необходимо реализовать в классе GgaSolver основной расчет ГТС. Рекомендуется заполнять существующий шаблон:

```
class GgaSolver:

    def __init__(self):
        self.graph = None
        self.ACCURACY = 1000
```

```python
    def solve(self, graph: Graph):
        """
        Производит расчет ГТС методом глобального градиента

        :param graph: расчетный граф
        """
        self.graph = graph
        self.__initialize()
        self.__main_loop()

    def __initialize(self):
        """
        Инициализирует метод глобального градиента:
        1. Задает постоянные величины
        2. Задает начальное приближение по расходу и давлению
        """
        pass

    def __main_loop(self):
        """
        Выполняет итерационный алгоритм:
        1. Определяет значение вектор-функции F(x) и обратную матрицу F'(x)^-1
        2. Выполняет шаг по давлению
        3. Выполняет шаг по расходу
        4. Обновляет значение давления и притока/оттока в узле
        5. После цикла обновляет расход на дугах
        """
        step = 0
        mistake = math.inf
        while mistake > self.ACCURACY and step < 100:
            break
        else:
            pass


    def __get_f_vector(self, x_vector: np.array) -> np.array:
        pass

    def __get_f_diff_inv(self, x_vector: np.array) -> np.array:
        pass

    def __update_node(self, p_vector: np.array, q_vector: np.array):
        pass

    def __update_arc(self, x_vector: np.array):
        pass
```

Пример заполненного шаблона:

```python
class GgaSolver:

    def __init__(self):
        self.graph = None
        self.ACCURACY = 1000

    def solve(self, graph: Graph):
        """
        Производит расчет ГТС методом глобального градиента

        :param graph: расчетный граф
        """
        self.graph = graph
        self.__initialize()
        self.__main_loop()

    def __initialize(self):
        """
        Инициализирует метод глобального градиента:
        1. Задает постоянные величины
        2. Задает начальное приближение по расходу и давлению
        """
        self.m = self.graph.get_m()
        self.n = self.graph.get_n()
        self.k = self.graph.get_k()
        self.A = self.graph.get_incidence_matrix()
        self.A1 = self.A[:self.k]
        self.A2 = self.A[self.k:]
        self.X_vector = np.random.rand(self.n)
        self.sorted_nodes = self.graph.get_sorted_nodes()
        self.P_vector = np.array([node.pressure* node.pressure for node in
self.sorted_nodes])
        self.Q_vector = np.array([node.flow_rate for node in self.sorted_nodes])

    def __main_loop(self):
        """
        Выполняет итерационный алгоритм:
        1. Определяет значение вектор-функции F(x) и обратную матрицу F'(x)^-1
        2. Выполняет шаг по давлению
        3. Выполняет шаг по расходу
        4. Обновляет значение давления и притока/оттока в узле
        5. После цикла обновляет расход на дугах
        """
        step = 0
        mistake = math.inf
        while mistake > self.ACCURACY and step < 100:
            # Найти F(x), F`(x) и матрицу Максвела в соответсвии с формулами
            f_vector = self.__get_f_vector(self.X_vector)
            f_diff_inv = self.__get_f_diff_inv(self.X_vector)
            maxwell = self.A1 @ f_diff_inv @ self.A1.T  # (m-k)x(m-k)
```

```
            # Сделать шаг итераци в соответсвии с выведенными формулами. Получить 
вектор давлений и расходов по ребрам
            self.P_vector[:self.k] = (np.linalg.inv(maxwell) @
                                      (self.Q_vector[:self.k] - self.A1 @ 
self.X_vector - self.A1 @ f_diff_inv @
                                       (self.A2.T @ self.P_vector[self.k:] - 
f_vector)))
            self.X_vector = self.X_vector + f_diff_inv @ (self.A.T @ 
self.P_vector - f_vector)
            self.Q_vector[self.k:] = self.A2 @ self.X_vector

            self.__update_node(self.P_vector, self.Q_vector)

            # Опеределить ошибку
            mistake = np.abs(self.A.T @ self.P_vector - f_vector).max()
            step += 1
        else:
            self.__update_arc(self.X_vector)
            if mistake < self.ACCURACY:
                self.graph.is_normal_result = True
            else:
                print("Error")


    def __get_f_vector(self, x_vector: np.array) -> np.array:
        return np.array([arc.get_pressure_losses(flow_rate) for arc, flow_rate in 
zip(self.graph.arcs, x_vector)])

    def __get_f_diff_inv(self, x_vector: np.array) -> np.array:
        f_diff_inv = np.zeros((self.n, self.n))
        for i in range(self.n):
            f_diff_inv[i, i] = 1 / 
self.graph.arcs[i].get_pressure_derivatives(x_vector[i])
        return f_diff_inv

    def __update_node(self, p_vector: np.array, q_vector: np.array):
        p_vector = np.sqrt(p_vector[:self.k])
        q_vector = q_vector[self.k:]
        for node, pressure in zip(self.sorted_nodes[:self.k], p_vector):
            node.pressure_calculated = pressure

        for node, flow_rate in zip(self.sorted_nodes[self.k:], q_vector):
            node.flow_rate_calculated = flow_rate

    def __update_arc(self, x_vector: np.array):
        for arc, flow_rate in zip(self.graph.arcs, x_vector):
            arc.flow_rate_calculated = flow_rate
```

4. ВАЛИДАЦИЯ РАСЧЕТА

Ход занятия 4

Для проверки расчета студентам необходимо рассчитать 2 схемы ГТС. Первая схема (рисунок 2) выполняется всеми студентами по одному варианту задания. Данный граф уже реализован. Чтобы его создать необходимо вызвать метод factories.graph_factory.create_graph_with_three_pipes()

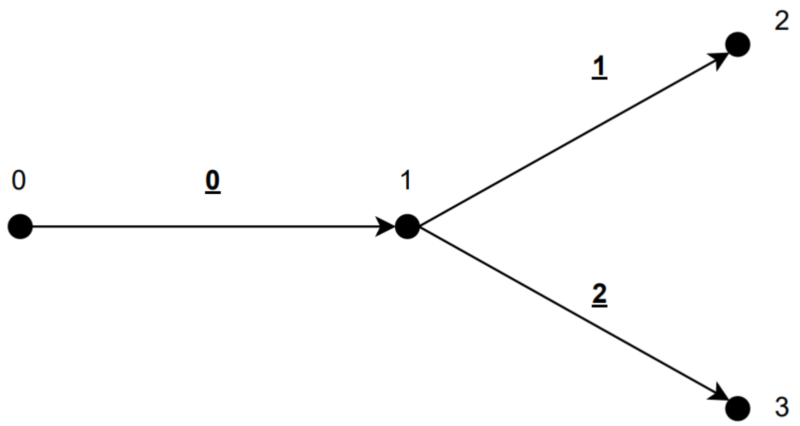

Рисунок 2 – Схема ГТС

Исходные данные приведены в таблице 1.

Таблица 1 – Исходные данные для узлов

Id	Sign	Pressure, МПа	Flow rate, $\frac{м^3}{с}$
0	“pressure”	5	-
1	“flow”	-	0
2	“pressure”	2	-
3	“pressure”	2.2	-

Все трубы имеют длину 40 км, внутренний диаметр 1196 мм и эквивалентная шероховатость труб 0,003 м.

Определить давление в точке 1 и расход в узлах 0, 2, 3. На рисунке 3 показан результат работы программы:

```
Nodes:
 Id     Sign                  Flow_rate       Pressure
  0     pressure             487.61 m3/s     5.0000 MPa
  1     flow                   0.00 m3/s     2.9448 MPa
  2     pressure            -255.74 m3/s     2.0000 MPa
  3     pressure            -231.88 m3/s     2.2000 MPa

 Id     Start -> End          Flow_rate          Inlet_pressure -> Outlet_pressure
  0         0 -> 1           487.61 m3/s            5.0000 MPa ->      2.9448 MPa
  1         2 -> 1          -255.74 m3/s            2.0000 MPa ->      2.9448 MPa
  2         1 -> 3           231.88 m3/s            2.9448 MPa ->      2.2000 MPa
```

Вторая схема ГТС (рисунок 4) решается студентами по вариантам.

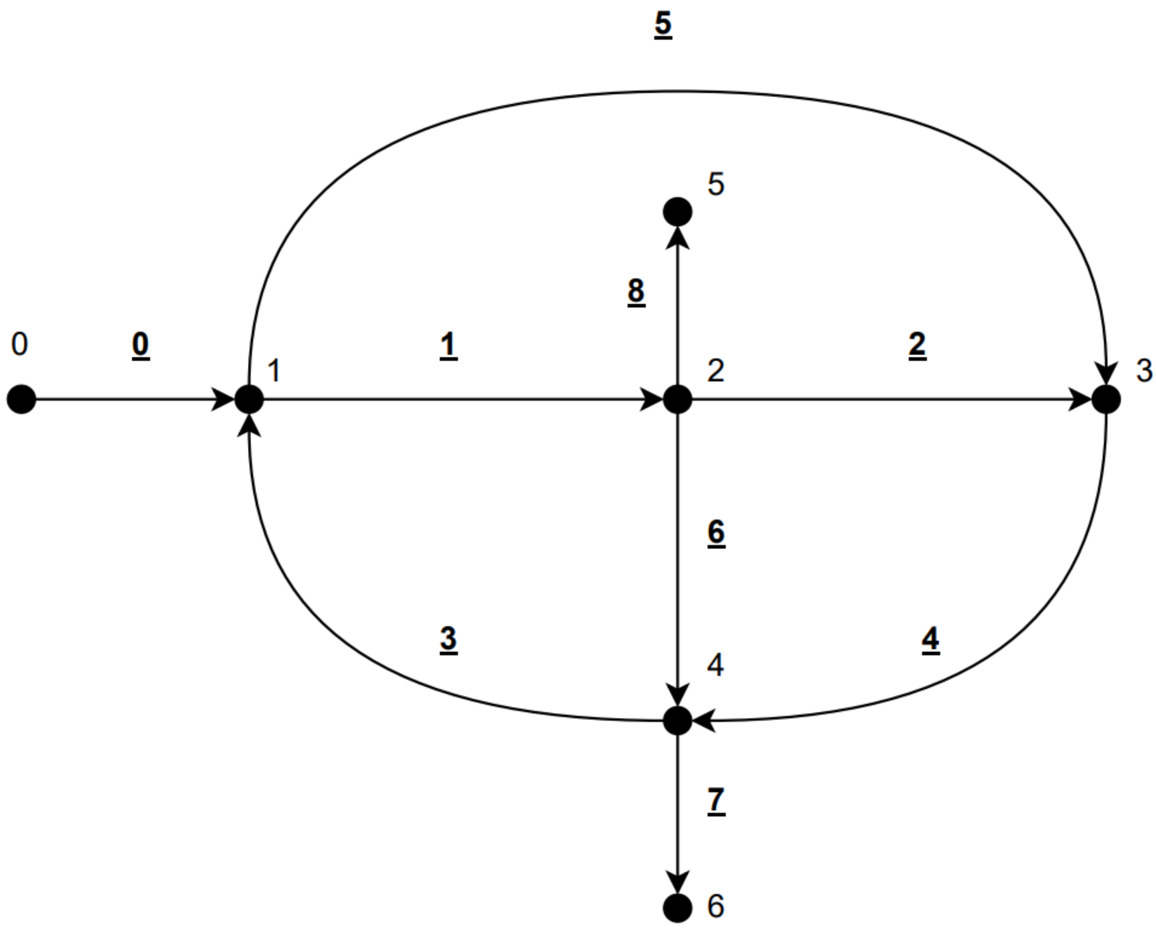

Рисунок 4 – Схема ГТС

Так как данная схема еще не построена, студентам необходимо создать метод, который будет формировать расчетный граф. Например:

```python
def create_graph_contest() -> Graph:
    graph = Graph()

    nodes = [Node(0, 'pressure', pressure=5e6)]
    for i in range(1, 5):
        nodes.append(Node(i))
    nodes.append(Node(5, 'pressure', pressure=2e6))
    nodes.append(Node(6, 'pressure', pressure=2.2e6))

    for node in nodes:
        graph.add_node(node)

    pipeline = [
        Pipe(0, 40e3, 1.22, 0.003),
        Pipe(1, 40e3, 1.22, 0.003),
        Pipe(2, 40e3, 1.22, 0.003),
        Pipe(3, 40e3, 1.22, 0.003),
        Pipe(4, 40e3, 1.22, 0.003),
        Pipe(5, 40e3, 1.22, 0.003),
        Pipe(6, 40e3, 1.22, 0.003),
        Pipe(7, 40e3, 1.22, 0.003),
        Pipe(8, 40e3, 1.22, 0.003)]

    graph.add_arc(nodes[0], nodes[1], pipeline[0])
    graph.add_arc(nodes[1], nodes[2], pipeline[1])
    graph.add_arc(nodes[2], nodes[3], pipeline[2])
    graph.add_arc(nodes[4], nodes[1], pipeline[3])
    graph.add_arc(nodes[3], nodes[4], pipeline[4])
    graph.add_arc(nodes[1], nodes[3], pipeline[5])
    graph.add_arc(nodes[2], nodes[4], pipeline[6])
    graph.add_arc(nodes[4], nodes[6], pipeline[7])
    graph.add_arc(nodes[2], nodes[5], pipeline[8])

    return graph
```

На рисунке 5 показан результат работы программы

```
Nodes:
 Id     Sign                 Flow_rate       Pressure
  0     pressure            465.86 m3/s     5.0000 MPa
  1     flow                  0.00 m3/s     3.1830 MPa
  2     flow                  0.00 m3/s     2.8812 MPa
  3     flow                  0.00 m3/s     2.9449 MPa
  4     flow                  0.00 m3/s     2.8829 MPa
  5     pressure           -245.27 m3/s     2.0000 MPa
  6     pressure           -220.59 m3/s     2.2000 MPa

Arcs:
 Id     Start -> End         Flow_rate         Inlet_pressure -> Outlet_pressure
  0         0 -> 1          465.86 m3/s            5.0000 MPa ->      3.1830 MPa
  1         1 -> 2          161.16 m3/s            3.1830 MPa ->      2.8812 MPa
  2         2 -> 3          -72.45 m3/s            2.8812 MPa ->      2.9449 MPa
  3         4 -> 1         -160.74 m3/s            2.8829 MPa ->      3.1830 MPa
  4         3 -> 4           71.51 m3/s            2.9449 MPa ->      2.8829 MPa
  5         1 -> 3          143.96 m3/s            3.1830 MPa ->      2.9449 MPa
  6         2 -> 4          -11.66 m3/s            2.8812 MPa ->      2.8829 MPa
  7         4 -> 6          220.59 m3/s            2.8829 MPa ->      2.2000 MPa
  8         2 -> 5          245.27 m3/s            2.8812 MPa ->      2.0000 MPa
```

Рисунок 5 – Результат работы программы

ЗАКЛЮЧЕНИЕ

Таким образом студенты освоят основы работы с графами, метод Ньютона и МГГ. Данные методы и их модификации регулярно применяются на практике при моделировании и оптимизации.

Прохождение данной дисциплины позволит студентам разработать применить свои знания в области гидравлики на реальной задаче, которая соответствует профилю магистерской дисциплины «21.04.01.64 Управление системами транспорта углеводородов в условиях цифровой трансформации»