

ЧИСЛЕННЫЕ МЕТОДЫ МАТЕМАТИКИ.

ЧАСТЬ II.

Решение систем линейных алгебраических уравнений

Цель работы: *изучение операций по созданию и работе с массивами numpy, индексации произвольной группы элементов, операций над матрицами. Освоение методов решения систем линейных алгебраических уравнений (СЛАУ).*

Задание

1. Написать программу, создающую матрицу, заданную в соответствии с вариантом в [Приложении №1](#). Программа не должна использовать вложенные циклы, конструкции ветвления и встроенные функции python по работе с матрицами (вроде rot90).
2. Выполнить расчёт токов в ветвях электрической схемы, заданной в [Приложении №2](#).
 - 2.1. По заданной системе уравнений Кирхгофа получить матрицу коэффициентов и матрицу свободных членов системы уравнений.
 - 2.2. Написать программу, выполняющую решение полученной системы уравнений методом, заданным в варианте и соответствующую [требованиям](#).
 - 2.3. Протестировать полученный алгоритм путем сравнения результата с решением, полученным при помощи функции numpy.linalg.solve.
3. Составить отчёт по проделанной работе. Отчет должен содержать:
 - 3.1. Титульный лист, цель работы, задание. Все страницы, кроме первой, должны быть пронумерованы в верхнем правом углу. В верхнем колонтитуле в одной строке с номером страницы слева должна быть размещена подпись ФИО студента и надпись “Лабораторная работа №1”.
 - 3.2. Постановку каждой задачи.
 - 3.3. Блок-схемы всех программ.
 - 3.4. Листинги программ в виде текста, оформленного в дополнении Code Blocks, язык python, тема github.
 - 3.5. Результаты решения задач

Библиотека numpy. Создание массивов.

Массив - совокупность переменных одного типа, имеющих один и тот же идентификатор (имя массива) и различающихся между собой одним или несколькими индексами. Индекс - порядковый номер элемента массива. Размерность массива - количество индексов, необходимое для однозначной идентификации элемента массива. Различают одно, двух и многомерные массивы.

Одномерный массив часто называют “вектором”. В математике вектор - это объект, характеризующийся величиной и направлением. Если задана система координат, то вектор однозначно задаётся в виде набора своих координат (набора чисел). Поэтому “вектором” в математике (и в программировании) часто называют любой упорядоченный набор чисел.

Двумерный массив чисел часто называют “матрицей”, т.к. эти понятия имеют один и тот же математический смысл (матрица - объект, описываемый прямоугольной таблицей чисел). Вектор является частным случаем матрицы, одна из размерностей которой равна 1. В зависимости от того, какая размерность матрицы равна 1, различают вектор-строку (равна 1 вертикальная размерность) и вектор-столбец (равна 1 горизонтальная размерность).

В языке Python основные операции над массивами и матрицами находятся в библиотеке numpy. Одной из особенностей языка Python при работе с массивами является очень медленное выполнение алгоритмов обработки массивов при помощи циклов for и while. Это связано с тем, что язык Python является языком-интерпретатором, и интерпретация кода программы выполняется на каждой итерации цикла.

Большинство функций библиотеки numpy написано на языках C и fortran и скорость их выполнения существенно превосходит скорость выполнения аналогичных алгоритмов на языке Python. Поэтому использование этих функций для работы с массивами и матрицами предпочтительнее реализации алгоритмов на языке Python.

Модификация алгоритма или кода программы таким образом, что скалярные операции, обрабатывающие по одному элементу массива за раз заменяются на операции с векторами (или матрицами), обрабатывающими за раз большую группу чисел, называется “векторизация”. В Python для выполнения векторизации в основном используются функции библиотеки numpy и срезы массивов.

В качестве примера можно рассмотреть расчет суммы элементов числовой последовательности большого размера. Для измерения времени выполнения фрагмента кода можно использовать функцию time() библиотеки time. Ниже приведена программа, рассчитывающая сумму элементов числовой

последовательности при помощи цикла python, засекающая и выводящая время выполнения программы.

```
import numpy as np
import time
t0 = time.time()
S = 0
for k in range(100000):
    S = S + k
te = time.time()-t0
print('Сумма ', S)
print('Время выполнения', te)
```

Результатом работы программы будут два числа: сумма элементов последовательности и время выполнения программы.

```
Сумма 4999950000
Время выполнения 0.02834486961364746
```

Ниже приведен векторизованный код, выполняющий ту же самую операцию с использованием функций библиотеки numpy.

```
t0 = time.time()
X = np.arange(0,100000)
S = np.sum(X)
te = time.time()-t0
print('Сумма ', S)
print('Время выполнения', te)
```

Результат работы программы:

```
Сумма 4999950000
Время выполнения 0.00266265869140625
```

Видно, что время выполнения векторизованной программы меньше примерно на порядок.

Создание массивов numpy в Python.

Можно выделить три основных способа создания массивов и матриц типа numpy.array в python.

1. Прямое преобразование списков и кортежей python в массив numpy.array.

Выполняется при помощи функции numpy.array. Работает как с типом данных list (прямоугольные скобки), так и с типом данных tuple (круглые скобки). Ниже приведен пример:

```
import numpy as np
```

```
A = np.array([1,2,3])
B = np.array([4,5,6])
print(A)
print(B)
```

Результат:

```
[1 2 3]
[4 5 6]
```

Аналогичным образом можно создавать (преобразовывать) и многомерные массивы:

```
import numpy as np
A = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
print(A)
```

Результат:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Обратите внимание, если строка кода заканчивается запятой, то python автоматически присоединяет к этой строке содержимое следующей строки. Эту особенность синтаксиса можно использовать при создании матриц с большим количеством элементов, чтобы не записывать всё в одну длинную, не помещающуюся на экране строку.

Важным моментом при создании массивов при помощи функции `numpy.array` является задание типа элемента. Тип задается в зависимости от типа элементов, перечисленных в списке или кортеже python. Если там только целые числа, будет выбран тип `integer`. Если есть хотя бы одно дробное число - все числа массива будут иметь тип `float`. Но если при создании массива его элементы получили целочисленный тип данных, то попытка присвоения элементу массива дробного значения приведет к его преобразованию в целое число (потере дробной части числа). Ниже приведен пример.

```
import numpy as np
A = np.array([1,2,3])
A[1] = 1.5
print(A)
```

Результат:

```
[1 1 3]
```

В результате выполнения этого кода, вместо 1.5 второму элементу массива будет присвоено значение 1 (дробная часть потеряна). Эта особенность автоматического задания типа массива `numpy.array` может приводить к трудно обнаруживаемым ошибкам в работе программы. Поэтому целесообразно при создании массива, прямо указывать необходимый тип его элементов:

```
import numpy as np
A = np.array([1,2,3],dtype=float)
A[1] = 1.5
print(A)
```

Результат:

```
[1.  1.5 3. ]
```

2. Использование функций `numpy.arange` и `numpy.linspace`.

Обе эти функции создают последовательность чисел от начального до конечного с некоторым шагом. Отличие заключается в следующем: функция `numpy.arange` создаёт числовую последовательность с заданным шагом, при этом переданное в функцию конечное значение последовательности в неё не включается:

```
import numpy as np
A = np.arange(1,10)
print(A)
```

Результат:

```
[1 2 3 4 5 6 7 8 9]
```

Видно, что число 10 не вошло в созданную числовую последовательность.

Можно опускать начальное значение последовательности, тогда отсчет будет вестись от нуля, и шаг последовательности (по умолчанию будет равен 1):

```
import numpy as np
A = np.arange(10)
print(A)
```

Результат:

```
[0 1 2 3 4 5 6 7 8 9]
```

Шаг последовательности может быть отрицательным, или не равным 1.

```
import numpy as np
```

```
A = np.arange(10,0,-1)
print(A)
B = np.arange(1,10,3)
print(B)
```

Результат:

```
[10  9  8  7  6  5  4  3  2  1]
[ 1  4  7]
```

Если необходимо создать числовую последовательность от A до B с шагом H, чтобы число B обязательно входило в созданную числовую последовательность, то при использовании функции `numpy.arange` необходимо указать следующие данные: `numpy.arange(A,B+H,H)`.

Функция `numpy.arange` выбирает тип элемента создаваемого массива в зависимости от типов переданных в неё аргументов. Если там целые числа, то будет выбран тип `integer`, что может приводить к описанной выше ошибке (для функции `numpy.array`). Для избежания этой ошибки, рекомендуется указывать необходимый тип данных в качестве аргумента функции `arange`:

```
import numpy as np
A = np.arange(10,dtype=float)
print(A)
```

Результат:

```
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

Функция `numpy.linspace` также создаёт числовую последовательность от A до B, но в отличие от `numpy.arange`, число B будет входить в создаваемую последовательность, а третьим аргументом функции является число элементов последовательности:

```
import numpy as np
A = np.linspace(1,10,10)
print(A)
```

Результат:

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

Обратите внимание, функция `linspace` по умолчанию задает тип элементов массива - `float`. Начальное, конечное значение и шаг числовой последовательности связаны с числом элементов выражением:

$$N = \text{floor}\left(\frac{\text{End}-\text{Start}}{\text{Step}}\right) + 1,$$

где: Start - первое число в числовой последовательности; End - последнее число; Step - шаг; N - число элементов последовательности; floor - операция округления к ближайшему меньшему целому. Соответственно, в функции numpy.linspace задаются числа Start, End и N и даже если три этих числа целые, шаг Step из приведенного выше выражения весьма вероятно может принять дробное значение (а значит - дробными будут и все элементы числовой последовательности, кроме первого и последнего):

```
import numpy as np
A = np.linspace(1,10,8)
print(A)
```

Результат:

```
[ 1.  2.28571429  3.57142857  4.85714286  6.14285714
 7.42857143  8.71428571 10. ]
```

Видимо, в связи с этим, функция numpy.linspace сразу устанавливает тип элементов создаваемого массива, как float.

3. Специальные функции генерации массивов

Некоторые специальные функции для генерации массивов и матриц приведены в таблице 1. Все приведенные в табл.1 функции по умолчанию устанавливают тип float для элементов создаваемого массива. Все функции, кроме numpy.eye могут быть использованы для создания матрицы произвольного размера, при этом число строк и столбцов матрицы передаются в функцию в виде кортежа tuple из двух значений: число строк, число столбцов (в круглых скобках).

Таблица 1 – Функции создания массивов с исходными данными

numpy.zeros(n) numpy.zeros((n,m))	Функция создает вектор-строку размером n или матрицу размером $n \times m$ и присваивает всем элементам ноль.
numpy.eye(n)	Функция создает единичную матрицу размером $n \times n$
numpy.empty(n) numpy.empty((n,m))	Функция создает вектор-строку размером n или матрицу размером $n \times m$ и присваивает всем элементам случайное значение, зависящее от состояния памяти.
numpy.ones(n)	Функция создает вектор-строку размером n или

<code>numpy.ones((n,m))</code>	матрицу размером $n \times m$ и присваивает всем элементам единицу.
--------------------------------	---

Срезы и копирование массивов

Доступ к элементам массива (индексация) в python осуществляется при помощи указания индекса элемента в квадратных скобках после имени массива. Нумерация элементов в python начинается с нуля. При доступе к элементу многомерного массива, индексы указываются через запятую, первым следует номер строки, вторым - номер столбца.

```
import numpy as np
A = np.array([1,2,3])
print('A[2] = ',A[2])
B = np.array([[1,2,3],[4,5,6]])
print('B[1,2] = ',B[1,2])
```

Результат (нумерация элементов - с нуля):

```
A[2] = 3
B[1,2] = 6
```

В качестве индексов можно указывать отрицательные числа, в этом случае отсчет элементов будет вестись с конца массива, последний элемент массива будет иметь индекс -1, предпоследний -2 и т.д.

```
import numpy as np
A = np.array([1,2,3])
print('A[-1] = ',A[-1])
```

Результат:

```
A[-1] = 3
```

Многомерные массивы в python фактически являются вложенными линейными массивами, т.е. массивом, элементами которого являются массивы. Поэтому, например, в двумерном массиве (матрице) можно получить доступ сразу ко всей строке матрицы, просто опустив второй индекс в обращении к элементу:

```
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('Вторая строка: ', A[1])
```

Результат:

Вторая строка: [4 5 6]

Приведенным выше способом можно получить доступ только к строке матрицы, если необходимо получить доступ ко всем элементам столбца, то необходимо использовать операцию среза (slice) массива.

В python можно получить доступ не только к одному элементу, а сразу к нескольким элементам по списку индексов. Эта операция называется срезом массива. Срез выполняется при помощи использования символа ':' (двоеточие). Общий синтаксис записи среза:

```
array_name[start:stop:step],
```

где: array_name - имя массива; start - стартовый номер списка индексов; stop - последний номер списка индексов (**не включается в список!**); step - шаг числовой последовательности списка индексов.

Любая из составляющих списка индексов среза может быть опущена. Тогда она принимает значение по умолчанию. Так start по умолчанию равен 0 (от начала массива); stop по умолчанию принимает значение последнего индекса (до последнего элемента); step по умолчанию равен 1. Если step не задается, то второй символ ':' (двоеточие) в объявлении среза можно опустить.

Ниже приведены различные варианты использования среза в одномерном массиве:

```
import numpy as np
A = np.arange(10)
print('Весь массив ', A)
print('Срез с 2-го по 7-й элемент ', A[2:7])
print('Срез с 3-го эл-та до конца ', A[3:])
print('Срез с начала до 5-го эл-та ', A[:5])
print('Срез каждого 2-го элемента ', A[::2])
print('Срез элементов в обратном порядке ', A[::-1])
```

Результат:

```
Весь массив [0 1 2 3 4 5 6 7 8 9]
Срез с 2-го по 7-й элемент [2 3 4 5 6]
Срез с 3-го эл-та до конца [3 4 5 6 7 8 9]
Срез с начала до 5-го эл-та [0 1 2 3 4]
Срез каждого 2-го элемента [0 2 4 6 8]
Срез элементов в обратном порядке [9 8 7 6 5 4 3 2 1 0]
```

При обращении к многомерному массиву срез можно выполнять по столбцу, по строке или одновременно (для выделения подматрицы):

```
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('Выделение 2-го столбца ', A[:,1])
print('Выделение 2-й строки ',A[1,:])
print('Выделение подматрицы ')
print(A[0:2,0:2])
```

Результат

```
Выделение 2-го столбца  [2 5 8]
Выделение 2-й строки  [4 5 6]
Выделение подматрицы
[[1 2]
 [4 5]]
```

Срезы массивов можно использовать в левой части оператора присваивания (т.е. присваивать значения конкретным элементам массива).

```
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('Исходная матрица \n',A)
# Присвоение нулей 2-й строке матрицы
A[1,:] = 0
# Замена 1-й и 3-й строк матрицы местами
C = A[0,:].copy()
A[0,:] = A[2,:].copy()
A[2,:] = C.copy()
print('Итоговая матрица \n',A)
```

Результат:

```
Исходная матрица
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Итоговая матрица
[[7 8 9]
 [0 0 0]
 [1 2 3]]
```

В приведенном выше примере в операторе присваивания используется метод `copy()` массива `numpy.array`. Это важное обстоятельство, связанное с *поверхностным* и *глубоким* копированием массивов при присваивании.

Оператор присваивания python по умолчанию выполняет поверхностное копирование массива. При поверхностном копировании в новую переменную переносятся лишь адреса элементов исходного массива. Это делается с целью экономии оперативной памяти. Изменение элементов во вновь созданном массиве приведёт к изменению элементов в исходном массиве.

```
import numpy as np
A = np.arange(1,10)
print('Исходный массив ', A)
B = A[2:7]
print('Массив B ', B)
# Изменяем элемент массива B
B[2] = 0
print('Массивы A и B после изменения массива B \n',A, '\n',B)
```

Результат:

```
Исходный массив  [1 2 3 4 5 6 7 8 9]
Массив B  [3 4 5 6 7]
Массивы A и B после изменения массива B
[1 2 3 4 0 6 7 8 9]
[3 4 0 6 7]
```

Видно, что при изменении элемента в массиве B (срез массива A), происходит и изменение элемента в массиве A. Фактически элементы массива B - это те же элементы массива A, просто выбранные по определённому списку. Поверхностное копирование массивов в python может приводить к трудно-обнаруживаемым ошибкам при выполнении программ.

С целью избежания изменения элементов исходного массива необходимо выполнять глубокое копирование массива. В этом случае специальная функция `copy()` создает новый массив, в который переносит элементы исходного массива. Изменение элементов вновь созданного массива уже не приведёт к изменению элементов исходного массива. Для глубокого копирования можно использовать одну из перечисленных ниже функций:

- `numpy.copy()` - функция библиотеки `numpy`, выполняющая копирование массива;
- метод `copy()` класса `numpy.array`, может быть вызван для любого массива, имеющего этот тип данных.

```

import numpy as np
A = np.arange(1,10)
print('Исходный массив ', A)
# Копирование через numpy.copy
B = np.copy(A)
# Копирование через метод copy
C = A.copy()
B[2] = 0
C[2] = 0
print('Массивы после изменения\n',A, '\n',B, '\n',C)

```

Результат:

```

сходный массив  [1 2 3 4 5 6 7 8 9]
Массивы после изменения
[1 2 3 4 5 6 7 8 9]
[1 2 0 4 5 6 7 8 9]
[1 2 0 4 5 6 7 8 9]

```

Матричные операции

Библиотека `numpy` содержит большое количество функций, выполняющих различные операции над массивами и матрицами. В этом разделе рассматриваются некоторые из этих функций, которые могут понадобиться при программировании в `python` алгоритмов решения СЛАУ, приведенных в следующем разделе.

1. Функция `len(A)` определения размера массива. Эта функция входит в ядро `python`, для её вызова не требуется подключение библиотек. Возвращает число элементов одномерного массива. Для матриц (многомерных массивов) возвращает число строк (подмассивов первого уровня).

```

A = np.arange(1,10)
B = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('Длина одномерного массива A ',len(A))
print('Число строк матрицы B ',len(B))

```

2. Функция `numpy.sum(A)`. Выполняет суммирование элементов массива.

```

import numpy as np
A = np.arange(1,11)
print('Массив A ',A)

```

```
print('Сумма элементов A ',np.sum(A))
```

```
Массив A [ 1  2  3  4  5  6  7  8  9 10]  
Сумма элементов A 55
```

3. Функция `numpy.column_stack((A,B))` выполняет добавление массива B к массиву A по столбцам. В качестве аргумента принимает кортеж (tuple), содержащий объединяемые массивы. Число элементов в столбцах A и B должно быть одинаково. A или B по отдельности или оба массива сразу могут быть одномерными, при этом произойдет добавление столбца к матрице или объединение двух столбцов в матрицу. Функция осуществляет глубокое копирование (изменение элементов результирующего массива не повлечет изменение исходных массивов). Функция может быть использована для создания расширенной матрицы в методе Гаусса.

```
import numpy as np  
# Матрица коэффициентов  
A = np.array([[1,2,0],[3,0,5],[4,6,0]])  
# Матрица свободных членов  
B = np.array([1,-1,2])  
# Создание расширенной матрицы  
# B добавляется к A справа  
C = np.column_stack((A,B))  
print('Расширенная матрица\n',C)
```

```
Расширенная матрица  
[[ 1  2  0  1]  
 [ 3  0  5 -1]  
 [ 4  6  0  2]]
```

4. Функция `numpy.delete(arr,obj,axis=None)` используется для удаления элементов из массива: `arr` - изменяемый массив; `obj` - индекс, массив индексов или срез удаляемых элементов; `axis` - ось, в направлении которой осуществляется удаление (0 - удаление строки; 1 - удаление столбца). `Numpy.delete` выполняет глубокое копирование элементов исходного массива. Функция может быть использована для удаления строк и столбцов массива в методе Крамера.

```
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
# Копируем матрицу A в C
C = A.copy()
# Удаляем 1-ю строку матрицы C
C = np.delete(C,0,axis=0)
# Удаляем 2-й столбец матрицы C
C = np.delete(C,1,axis=1)
print('Исходная матрица\n',A,'\n')
print('Матрица C после удалений\n',C)
```

Исходная матрица

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Матрица C после удалений

```
[[4 6]
 [7 9]]
```

5. Функция `numpy.transpose(A)`, выполняющая транспонирование элементов матрицы `A`. При применении к одномерному массиву возвращает исходный массив без изменений.

```
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
B = np.transpose(A)
print('Исходная матрица\n', A)
print('Транспонированная матрица\n', B)
```

Исходная матрица

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Транспонированная матрица

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Важно: функция `numpy.transpose` выполняет поверхностное копирование элементов исходной матрицы. В приведенном выше примере изменение элементов матрицы `B` повлечет изменение элементов матрицы `A`.

6. Функции умножения матриц. Для умножения матриц (не поэлементного перемножения, а по правилам умножения матриц) можно использовать один из трех приведенных ниже способов:

- Функция `numpy.matmul(A,B)`;
- Метод `dot()` класса `numpy.array`;
- Специальный символ '@'.

При умножении матрицы `A` на матрицу `B` число столбцов матрицы `A` должно быть равным числу строк матрицы `B`. Ниже приведен пример перемножения матриц с использованием всех трех способов.

```
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
B = np.array([1,2,3])
C = np.matmul(A,B)
D = A.dot(B)
E = A@B
print('Матрица A\n', A)
print('Матрица B\n', B)
print('Умножение при помощи numpy.matmul\n', C)
print('Умножение при помощи метода dot\n', D)
print('Умножение при помощи символа @\n', E)
```

Матрица A

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Матрица B

```
[1 2 3]
```

Умножение при помощи `numpy.matmul`

```
[14 32 50]
```

Умножение при помощи метода `dot`

```
[14 32 50]
```

Умножение при помощи символа @

```
[14 32 50]
```

7. Функция `numpy.linalg.det(A)`. Выполняет расчет определителя матрицы A. Матрица A должна быть квадратной (число строк равно числу столбцов).

```
import numpy as np
A = np.array([[1,-1,0],[1,1,0],[-1,0,-1]])
print('Определитель A = ',np.linalg.det(A))
```

Определитель A = -2.0

8. Функция `numpy.linalg.eigvals(A)`. Выполняет расчет собственных чисел матрицы A

```
import numpy as np
A = np.array([[1,2,0],[3,2,0],[5,0,-5]])
L = np.linalg.eigvals(A)
print('Собственные числа A = ',L)
```

Собственные числа A = [-5. 4. -1.]

Методы решения СЛАУ

Система линейных алгебраических уравнений - это система уравнений, в которой каждое уравнение является алгебраическим уравнением первой степени. Общий вид системы линейных алгебраических уравнений (1):

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\ &\dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= b_n \end{aligned} \quad (1)$$

Решением системы (1) является такая последовательность значений $c_1 \dots c_n$, при подстановке которой вместо $x_1 \dots x_n$ все уравнения системы (1) обращаются в тождества. Матрица (2):

$$\mathbf{A} = \begin{vmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{vmatrix} \quad (2)$$

называется матрицей коэффициентов, вектор-столбец, образованный значениями $b_1 \dots b_n$ - столбцом свободных членов. С учетом этого система уравнений (1) может быть записана в матричной форме:

$$AX = B. \quad (3)$$

Задача решения СЛАУ имеет следующие особенности:

- Матрица коэффициентов (2) не должна быть вырожденной, т.е. её определитель должен быть отличен от нуля $\det(A) \neq 0$;
- Система (1) должна быть неоднородной, т.е. хотя бы одно из значений $b_1 \dots b_n$ должно быть отличным от нуля;
- Число уравнений должно равняться числу неизвестных (матрица коэффициентов должна быть квадратной);

Практически все методы решения СЛАУ являются численными, т.к. решение находится в виде таблицы чисел, а не в виде математических выражений (правда, если в прямых методах решения, вместо чисел в исходных матрицах использовать буквенные обозначения, то можно получить аналитическое решение). Численные методы можно разделить на два класса:

- **Прямые методы.** Эти методы позволяют найти решение за заранее известное число строго определенных операций. К прямым методам относятся методы Крамера, Гаусса, нахождения решения через обратную матрицу, LU-разложение и т.д.
- **Итерационные методы.** В этих методах решение находится в результате последовательных приближений (итерационного процесса). К итерационным относят методы простых итераций, Зейделя и т.д.

Метод Гаусса

Метод основан на приведении матрицы коэффициентов СЛАУ к ступенчатому виду путём элементарных преобразований над строками. Элементарными преобразованиями матрицы являются преобразования, в результате которых сохраняется эквивалентность матриц. При элементарных преобразованиях матриц, описывающих СЛАУ, решение системы не будет меняться. К элементарным преобразованиям над строками относят:

- перестановку местами двух любых строк матрицы;

- умножение любой строки матрицы на произвольную константу $\lambda \neq 0$;
- прибавление к произвольной строке матрицы любой другой строки, умноженной на константу.

Матрицу, полученную в результате элементарных преобразований, и эквивалентную исходной обычно обозначают: $A \sim B$.

Ступенчатым видом матрицы по строкам называется такой вид матрицы, при котором:

- все ненулевые (имеющие хотя бы один ненулевой элемент) строки располагаются строго над нулевыми;
- ведущий элемент (первый ненулевой элемент при отсчете слева направо) каждой ненулевой строки располагается строго правее ведущего элемента в ненулевой строке, расположенной выше данной.

Пример матрицы ступенчатого вида по строкам (3):

$$\mathbf{A} = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{vmatrix} \quad (3)$$

Алгоритм метода Гаусса состоит из двух этапов, называемых “прямым ходом метода Гаусса” и “обратным ходом метода Гаусса”. Перед первым этапом матрица коэффициентов A и вектор-столбец свободных членов B объединяются в расширенную матрицу $(A|B)$:

$$\mathbf{A} = \left[\begin{array}{ccc|c} a_{11} & \dots & a_{1n} & b_1 \\ \dots & \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} & b_n \end{array} \right] \quad (4)$$

Все дальнейшие преобразования выполняются над расширенной матрицей. На этапе прямого хода метода Гаусса все элементы k -го столбца ниже главной диагонали (ниже элемента a_{kk}) зануляются путем умножения k -й строки

на соответствующий коэффициент $-\frac{a_{ik}}{a_{kk}}$, где a_{ik} - коэффициент в i -й строке k -го столбца ($i > k$), и прибавления результирующей строки к строкам ниже k -й. В результате этих преобразований левая часть расширенной матрицы (4), соответствующая матрице коэффициентов СЛАУ, будет преобразована в ступенчатую матрицу по строкам. В последней строке будет только один ненулевой коэффициент, и можно сразу вычислить решение n -го уравнения системы: $x_n = \hat{b}_n / \hat{a}_{nn}$, где \hat{b} - последний столбец преобразованной расширенной матрицы, \hat{a} - преобразованная (приведенная к ступенчатому виду) матрица коэффициентов, т.е. расширенная матрица без крайнего правого столбца. Из предпоследней строки можно получить решение $n-1$ -го уравнения x_{n-1} и т.д. В этом заключается смысл обратного хода метода Гаусса - вычисление решений СЛАУ в обратном порядке, начиная с n -го (5):

$$x_k = \frac{\hat{b}_k - \sum_{i=k+1}^{n-1} x_i a_{ki}}{\hat{a}_{kk}}, k = (n, n-1, n-2, \dots, 1). \quad (5)$$

Метод Гаусса требует $O(n^3)$ арифметических операций (n - число уравнений в СЛАУ) для поиска решения и является одним из наименее вычислительно-сложных методов решения СЛАУ.

У метода есть недостаток, в некоторых случаях, затрудняющий его применение: все коэффициенты на главной диагонали матрицы A должны быть ненулевыми ($a_{kk} \neq 0$). Это условие **всегда** выполнимо, т.к. существует теорема о приведении матрицы к ступенчатому виду: любую матрицу можно привести к ступенчатому виду путём элементарных преобразований только над строками. Однако для больших и разреженных (с большим количеством нулевых элементов) матриц приведение к ступенчатому виду может стать нетривиальной задачей. Обязательное отличие от нуля элементов главной диагонали матрицы коэффициентов также усложняет алгоритм метода Гаусса при его реализации на ПК.

Для приведения расширенной матрицы к виду, где на главной диагонали нет нулевых элементов, можно предложить следующий алгоритм (рис.1). В

цикле k меняется от 0 до $N-1$ с шагом 1, где N - число строк расширенной матрицы (число уравнений СЛАУ):

- Для k -го столбца матрицы находится максимальный по модулю элемент в столбце, начиная с k , запоминается его индекс m ;
- Меняются местами строки с номерами k и m и алгоритм переходит к следующему столбцу.

Блок-схема, реализующая описанный выше алгоритм приведена на рис.3. Алгоритм является достаточно устойчивым, однако не гарантирует 100% устранение нулей на главной диагонали, т.к. используется только одна элементарная операция над матрицей (перестановка строк). Для некоторых матриц, чтобы убрать нули на главной диагонали, потребуется сложение строк (умноженных на подходящие коэффициенты), данный алгоритм это не обеспечивает.



Рисунок 1 - Демонстрация работы алгоритма по устранению нулей на главной диагонали расширенной матрицы

После устранения нулей на главной диагонали расширенной матрицы, применяется алгоритм прямого хода метода Гаусса, для приведения матрицы к ступенчатому виду и затем - алгоритм обратного хода для нахождения корней. Блок-схема алгоритма прямого хода метода Гаусса приведена на рис.2

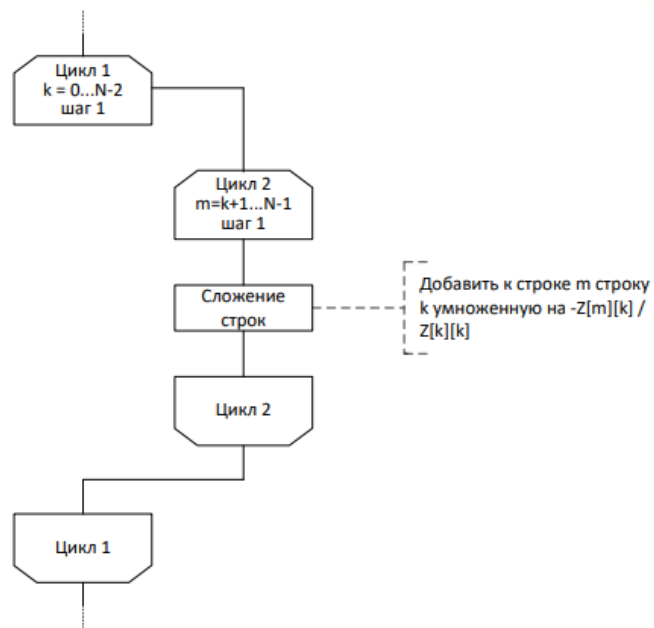


Рисунок 2 - Алгоритм прямого хода метода Гаусса

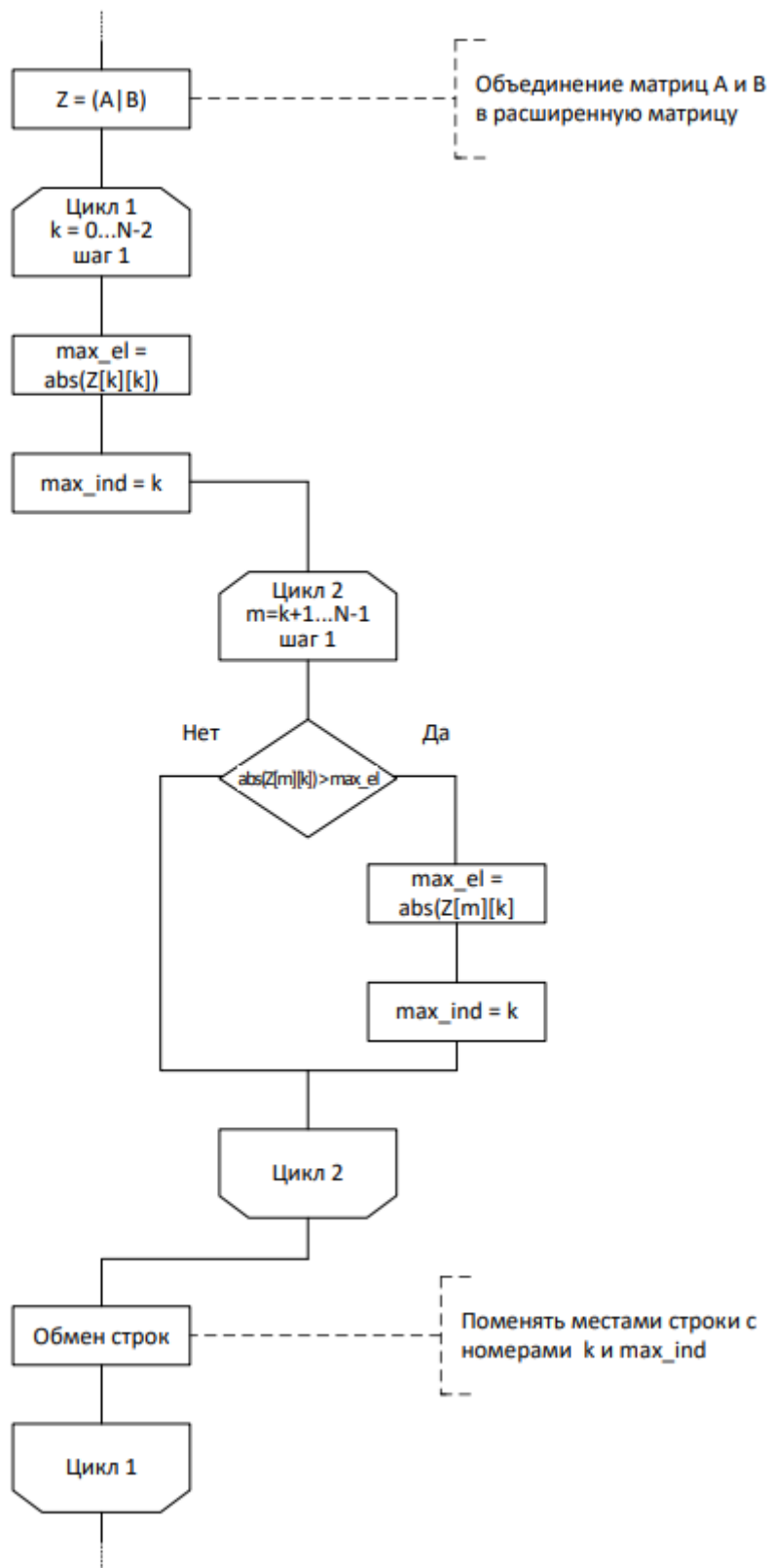


Рисунок 3 - Блок-схема устранения нулей на главной диагонали расширенной матрицы

На рис.4 приведена блок-схема алгоритма обратного хода метода Гаусса.

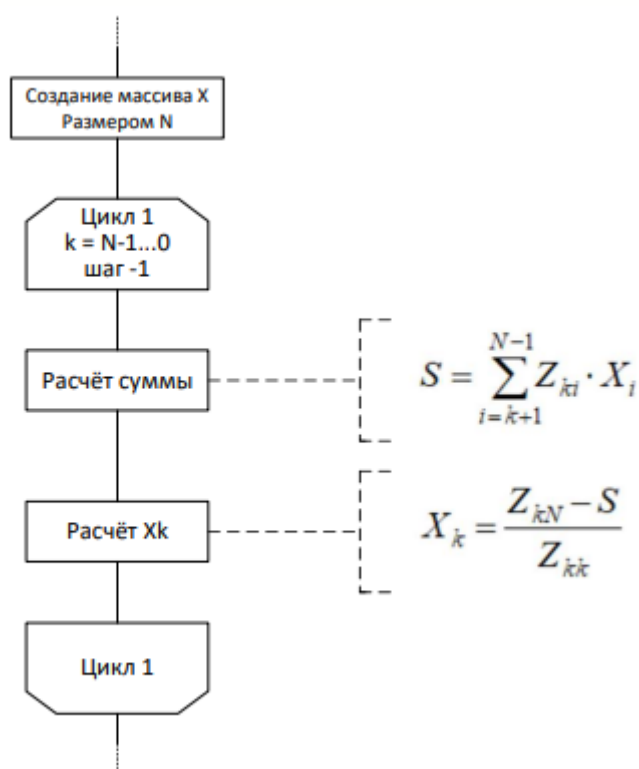


Рисунок 4 - Блок-схема обратного хода метода Гаусса

В результате применения алгоритма на рис.4 будет получен вектор X решений СЛАУ, описываемых матрицами A и B .

Метод LU-разложения.

Метод LU-разложения является частным случаем метода Гаусса. Его суть заключается в нахождении таких матриц L и U , чтобы их матричное произведение было равно исходной матрице коэффициентов A :

$$A = L \cdot U. \quad (6)$$

При этом сами матрицы L и U должны являться ступенчатыми (7):

$$L = \begin{vmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ \dots & & & \dots & 0 \\ l_{n1} & l_{n2} & l_{n3} & l_{nn} \end{vmatrix} \quad U = \begin{vmatrix} 1 & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & 1 & u_{23} & & u_{2n} \\ 0 & 0 & 1 & & u_{3n} \\ \dots & & & \dots & \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix} \quad (7)$$

Если матрицы L и U , удовлетворяющие выражениям (6) и (7) найдены, то дальнейшее решение системы уравнений $AX = B$ сводится к последовательному решению двух следующих систем линейных алгебраических уравнений:

$$\begin{aligned} L \cdot Y &= B, \\ U \cdot X &= Y. \end{aligned} \tag{8}$$

Так как в системах (8) матрицы коэффициентов L и U являются ступенчатыми, то для решения этих систем достаточно применить алгоритм обратного хода метода Гаусса (поэтому метод LU-разложения и называют модификацией метода Гаусса). Вычислительная сложность метода LU-разложения в точности такая же, как и у метода Гаусса: $O(n^3)$. Как и в случае применения метода Гаусса, возможность применения LU-разложения критически зависит от отсутствия нулей на главной диагонали матрицы A коэффициентов исходной системы уравнений. Поэтому **первым шагом алгоритма LU-разложения является применение алгоритма устранения нулей на главной диагонали матрицы A** . Для этого применяем алгоритм, приведенный на рис.3, после чего вновь разделяем результирующую матрицу Z на новую матрицу A и новый вектор B (отрезаем от Z крайний правый столбец).

После этого рассчитываются коэффициенты матриц L и U из выражений (9):

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot u_{kj}, \text{ при } i \geq j, \tag{9}$$

$$u_{ij} = \frac{1}{l_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik} \cdot u_{kj} \right), \text{ при } i < j.$$

При расчёте выражений (9) важное значение имеет порядок вычисления коэффициентов. Сначала все коэффициенты l_{ij} и u_{ij} приравниваются к нулю, затем два коэффициента приравниваются: $u_{11} = 1, l_{11} = a_{11}$. После этого для каждого сочетания i и j сначала рассчитывается l_{ij} и только затем - u_{ij} . Блок-схема алгоритма расчета коэффициентов l_{ij} и u_{ij} приведена на рис.5.

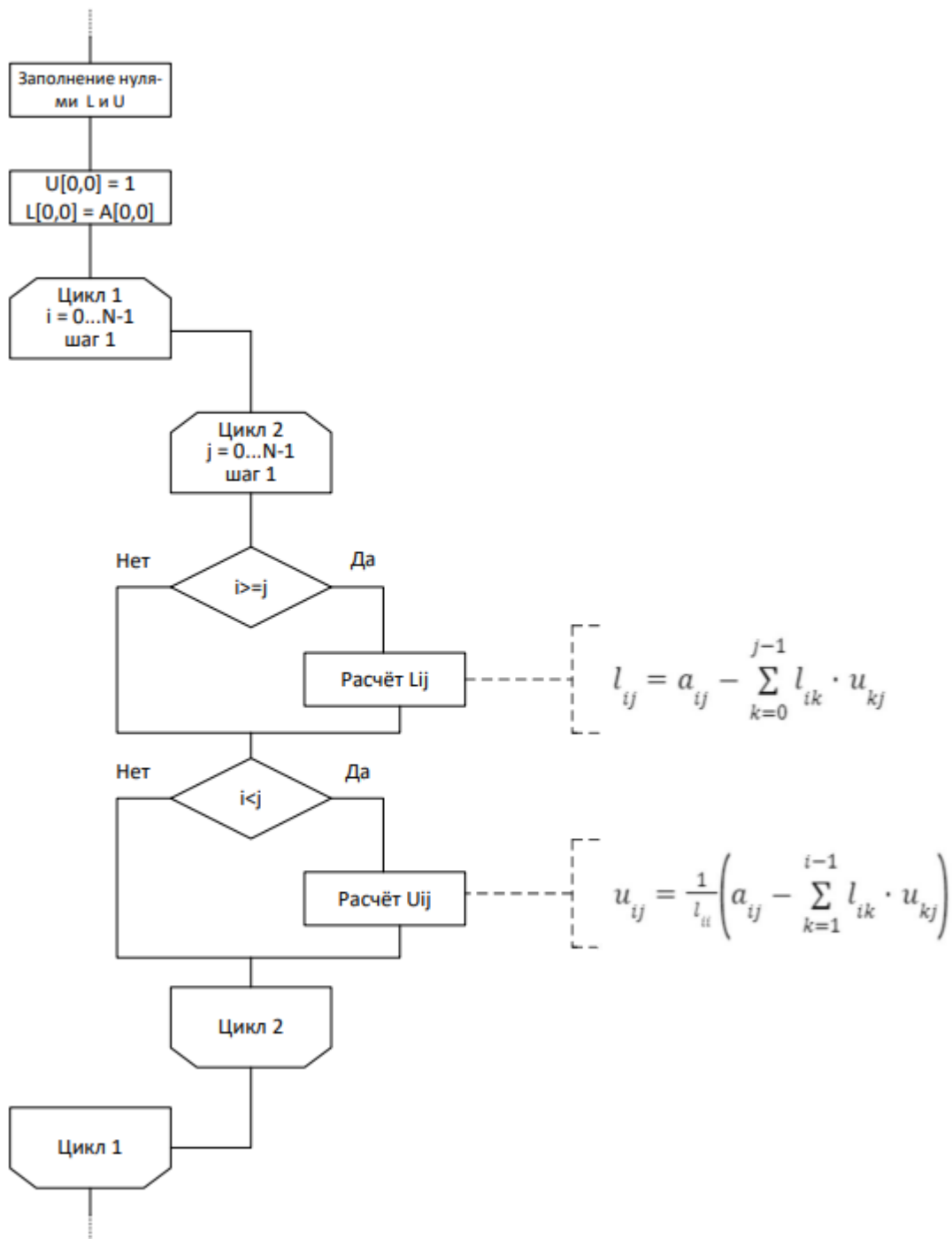


Рисунок 5 - Блок-схема разложения матрицы A на матрицы L и U.

Общий алгоритм нахождения решения системы $AX = B$ при помощи LU-разложения следующий:

- Объединение матриц A и B в расширенную матрицу и применение алгоритма на рис.3 (устранение нулей на главной диагонали);

- Выделение из результирующей расширенной матрицы (без нулей на главной диагонали) новых матриц A и B;
- Расчет элементов матриц L и U (алгоритм на рис. 5)
- Матрицы L и B объединяются в расширенную матрицу Z;
- К матрице Z применяется алгоритм прямого хода Гаусса, приведенный на рис. 2. В итоге матрица Z примет каноническую форму (ненулевые элементы останутся только на главной диагонали и в крайнем правом столбце);
- Находятся промежуточные решения путём деления правого столбца матрицы Z на главную диагональ: $Y_k = Z_{k,N+1} / Z_{kk}$;
- Матрицы U и Y объединяются в расширенную матрицу F;
- К матрице F применяется алгоритм обратного хода Гаусса, приведенный на рис. 4. В результате получаем вектор X решений исходной СЛАУ.

Метод Крамера.

Метод Крамера основан на вычислении определителей и является одним из наименее алгоритмически сложных методов. В общем случае решение записывается следующим образом (10):

$$x_k = \frac{\Delta_k}{\Delta}, \quad (10)$$

где: Δ - определитель исходной матрицы коэффициентов; Δ_k - определитель матрицы коэффициентов, у которой k -й столбец заменен столбцом свободных членов СЛАУ. Решение СЛАУ таким образом сводится к расчёту $n + 1$ определителя (n -число уравнений системы). Общая вычислительная сложность метода выше, чем у метода Гаусса и составляет: $O(n^4)$. Применимость метода не зависит от наличия/отсутствия нулей главной диагонали матрицы коэффициентов и вообще от каких-либо особенностей ее заполнения (обязательное условие лишь $\det(A) \neq 0$). Сложность алгоритма метода Крамера заключается в выбранном методе расчета определителя. Алгоритм метода Крамера приведен на рис.6



Рисунок 6 - Алгоритм метода Крамера

Для расчета определителя целесообразно использовать рекурсивный алгоритм разложения по строке (11). Ещё один способ вычисления определителей - привести матрицу к ступенчатому виду при помощи прямого хода алгоритма Гаусса (у ступенчатой матрицы определитель равен произведению элементов на главной диагонали), однако это потребует исключения нулей на главной диагонали и, в целом, усложнит алгоритм.

$$\Delta = \sum_{k=1}^N (-1)^{k+1} \cdot a_{1k} \cdot M_{1k}, \quad (11)$$

где: M_{1k} - дополнительный минор к элементу a_{1k} . Дополнительным минором M_{ij} матрицы A размерности N называется определитель меньшего порядка для матрицы, получаемой из A удалением i -й строки и j -го столбца. Очевидно, что т.к. для расчета определителя произвольной матрицы требуется расчет N определителей матриц меньшего размера, алгоритм (11) является рекурсивным.

Терминальной ветвью рекурсии может являться расчет определителя матрицы 2×2 , который можно выполнить по простому выражению (12):

$$\Delta = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}. \quad (12)$$

Алгоритм рекурсивной функции `determ`, выполняющей расчет определителя методом разложения по строке приведен на рис. 7.

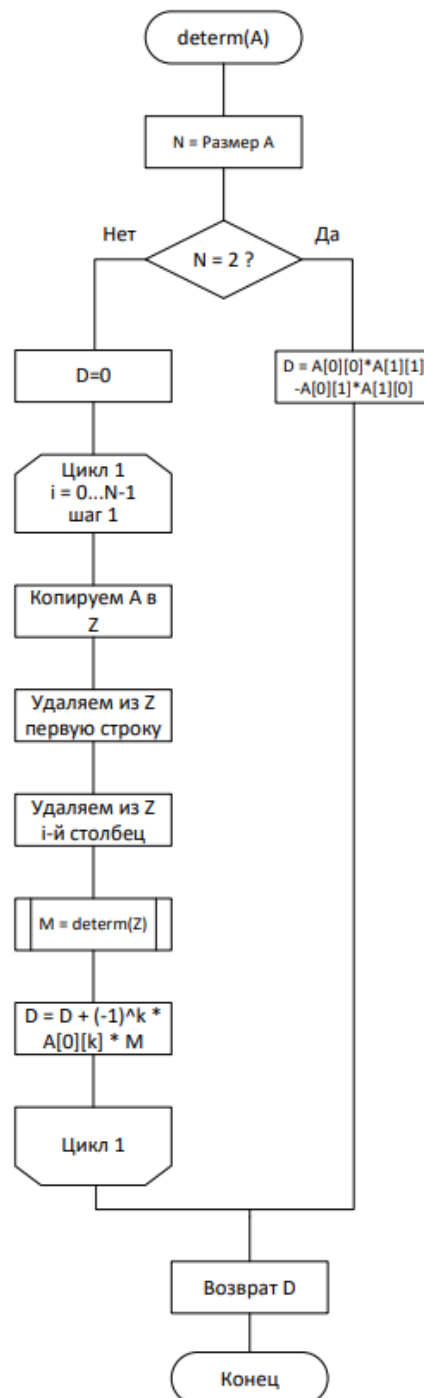


Рисунок 7 - Алгоритм рекурсивной функции `determ`, вычисляющей определитель произвольной матрицы

Расчёт через обратную матрицу.

Решение системы уравнений (3) может быть получено в виде (13):

$$X = A^{-1}B, \quad (13)$$

где: A^{-1} - матрица, обратная матрице коэффициентов. Обратная матрица - это такая матрица, произведение которой на исходную дает единичную матрицу: $A^{-1}A = E$. Обратная матрица определяется из выражения (14):

$$A^{-1} = \frac{1}{\det(A)} \cdot C^T, \quad (14)$$

где: C - союзная матрица для матрицы A . Союзная матрица - это матрица, состоящая из алгебраических дополнений (15):

$$C_{ij} = (-1)^{i+j} \cdot M_{ij}, \quad (15)$$

где: M_{ij} - дополнительный минор матрицы A , т.е. определитель матрицы, получаемой при вычеркивании из A i -й строки и j -го столбца. Расчет при помощи обратной матрицы является одним из наиболее алгоритмически простых, особенно с учетом того, что функции определения обратной матрицы обычно уже имеются в библиотеках математических функций используемого языка программирования. Основной недостаток метода - он имеет максимальную вычислительную сложность, т.к. требует расчета n^2 определителей матрицы, размером $n \times n$, т.е. вычислительная сложность - $O(n^5)$.

Блок-схема алгоритма расчета обратной матрицы приведена на рис. 8. Для расчета определителя матрицы можно использовать функцию `determ`, алгоритм которой приведен на рис.7.



Рисунок 8 - Блок-схема алгоритма расчета обратной матрицы

Метод простых итераций.

Рассмотренные выше методы Крамера, Гаусса, LU-разложения и вычисления обратной матрицы относятся к прямым численным методам решения СЛАУ. Но существует также большая группа итерационных методов, в

которых решение СЛАУ находится, как последовательное приближение некоторого начального вектора. К таким методам относится метод простых итераций. Суть метода заключается в следующем: от системы уравнений, записанной в виде $AX = B$ переходим к эквивалентной системе уравнений (16):

$$X = \alpha X + \beta. \quad (16)$$

Эквивалентность означает, что вектор решений X для исходной СЛАУ и для системы уравнений (16) одинаков.

Какой-либо вектор значений X_0 принимается в качестве начального приближения. Далее путём многократного повторения расчета выражения (17) выполняется уточнение решения.

$$X_{k+1} = \alpha X_k + \beta. \quad (17)$$

Теорема о сходимости: для сходимости метода простых итераций при любых X_0 и β необходимо и достаточно, чтобы все собственные числа матрицы α были по модулю меньше единицы (18)

$$|\lambda_i(\alpha)| < 1, i = 0, 1, 2 \dots n. \quad (18)$$

Если найдена матрица α , удовлетворяющая условиям (16) и (18), то алгоритм нахождения решения тривиален (рис. 9).

Основная сложность метода простых итераций заключается в нахождении матрицы α . Основной подход заключается в приведении при помощи элементарных преобразований исходной матрицы A к матрице с преобладанием диагональных элементов, т.е. у которой $|a_{kk}|$ больше, чем сумма модулей всех остальных элементов в k -й строке. К сожалению, алгоритм, реализующий этот подход, будет достаточно сложным.

Можно предложить следующий алгоритм нахождения матрицы α :

- Матрицы A и B объединяются в расширенную матрицу Z и к ней применяется алгоритм устранения нулей на главной диагонали (рис.3) и алгоритм прямого хода метода Гаусса (рис.2);
- В векторе D запоминаются элементы главной диагонали полученной ступенчатой матрицы Z ;
- Элементам главной диагонали матрицы Z присваиваются нули;
- Все остальные столбцы матрицы Z поэлементно делятся на вектор $-D$ (вектор D взятый с обратным знаком);

- Матрица α получается из N первых столбцов итоговой матрицы Z , а матрица β - это крайний правый ($N+1$ -й) столбец матрицы Z , взятый с обратным знаком.

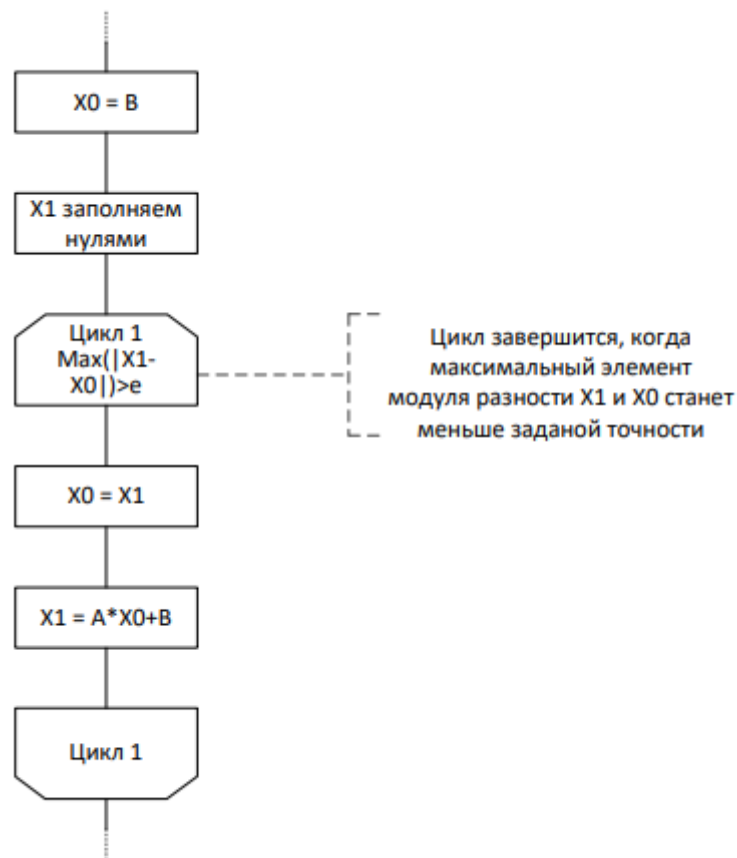


Рисунок 9 - Алгоритм метода простых итераций

Методические указания

Первое задание лабораторной работы посвящено созданию матрицы, заполненной числами по определённой схеме. Рассмотрим два примера, приведенных на рис.10.

<pre> [[1. 2. 3. 4. 5.] [0. 2. 3. 4. 5.] [0. 0. 3. 4. 5.] [0. 0. 0. 4. 5.] [0. 0. 0. 0. 5.]] </pre>	<pre> [[5. 5. 5. 5. 5.] [4. 4. 4. 4. 0.] [3. 3. 3. 0. 0.] [2. 2. 0. 0. 0.] [1. 0. 0. 0. 0.]] </pre>
а)	б)

Рисунок 10 - Примеры заполнения матриц

В задании также указано, что для заполнения матриц нельзя использовать вложенные циклы и функцию поворота матриц `numpy.rot90`. Данную задачу

можно решить следующим образом: первоначально матрица заполняется нулями (функция `numpy.zeros`), затем приводится к требуемому виду путём заполнения строк или столбцов матрицы в цикле при помощи срезов. Для использования срезов сначала необходимо определить закономерность заполнения матрицы числами. Рассмотрим пример на рис.10, а. Вывод закономерности заполнения приведен на рис.11.

```
[[1. 2. 3. 4. 5.] - в 0-й строке заполняются элементы с 0 до 4 последовательностью 1...5
 [0. 2. 3. 4. 5.] - в 1-й строке заполняются элементы с 1 до 4 последовательностью 2...5
 [0. 0. 3. 4. 5.] - в 2-й строке заполняются элементы с 2 до 4 последовательностью 3...5
 [0. 0. 0. 4. 5.]
 [0. 0. 0. 0. 5.]] - в k-й строке заполняются элементы с k до конца строки,
                    последовательностью k+1:N, где N - размер матрицы
```

Рисунок 11 - Вывод закономерности заполнения матрицы на рис.10, а

Общая закономерность на рис.11: в k-й строке заполняются элементы с номерами от k и до конца строки числовой последовательностью от k+1 до N, где N - размер матрицы (число столбцов или строк). Числовую последовательность можно создать при помощи функции `numpy.arange`, при этом следует помнить, что чтобы последним элементом последовательности было число N, последовательность нужно создавать до N+1 (`numpy.arange` не включает последнее число в создаваемую последовательность). Число k меняется в диапазоне от 0 до N-1. Код, создающий матрицу на рис.10,а приведен ниже.

```
import numpy as np
N = 5
A = np.zeros((N,N))
for k in range(N):
    A[k,k:] = np.arange(k+1,N+1)
print(A)
```

На рис.12 приведен вывод закономерности заполнения матрицы на рис.10, б.

```
[[5. 5. 5. 5. 5.] - в 0-й строке заполняются элементы с 0 до 4 числом 5
 [4. 4. 4. 4. 0.] - в 1-й строке заполняются элементы с 0 до 3 числом 4
 [3. 3. 3. 0. 0.] - в 2-й строке заполняются элементы с 0 до 2 числом 3
 [2. 2. 0. 0. 0.]
 [1. 0. 0. 0. 0.]] - в k-й строке заполняются элементы с 0 до N-1-k числом N-k
```

Рисунок 12 - Вывод закономерности заполнения матрицы на рис.10, б.

Код, создающий матрицу на рис.10, б приведен ниже:

```
import numpy as np
N = 5
A = np.zeros((N,N))
for k in range(N):
    A[k,:N-k] = N-k
print(A)
```

Обратите внимание, срез массива A создаётся от начала строки (в срезе не указан параметр start перед двоеточием) до элемента $N-k$, но т.к. в python последний элемент никогда не входит в создаваемую числовую последовательность, то на самом деле срез будет создан от начала строки до $N-k-1$.

Второе задание лабораторной работы посвящено расчету электрической схемы путем решения системы уравнений, полученных на основании правил Кирхгофа. Рассмотрим составление системы уравнений Кирхгофа и её решение в python для схемы, приведенной на рис.13.

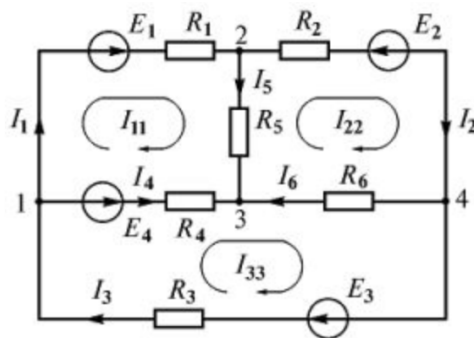


Рисунок 13 - Электрическая схема, для которой составляется система уравнений Кирхгофа.

В системе уравнений Кирхгофа неизвестными являются токи в ветвях схемы I_1, I_2, \dots, I_6 . Уравнения составляются по следующим двум правилам:

- Сумма токов в узле схемы всегда равна нулю. Если ток втекает в узел, он берётся со знаком '+', если вытекает - со знаком '-';
- Сумма падений напряжения в контуре равна сумме источников напряжения в этом контуре. Если направление обхода контура совпадает с направлением тока, то падение напряжения на резисторе берётся со знаком '+', иначе - со знаком '-'. Аналогично - с источниками напряжения (направление ЭДС совпадает с направлением обхода контура - берётся со знаком '+', иначе - со знаком '-')

Составленные по правилам Кирхгофа уравнения, в которых неизвестными являются токи в ветвях схемы, называются уравнениями Кирхгофа (системой уравнений Кирхгофа). Уравнения Кирхгофа для схемы на рис.13 приведены ниже (19):

$$\left\{ \begin{array}{l} I_1 + I_4 - I_3 = 0 \\ I_2 + I_5 - I_1 = 0 \\ I_4 + I_5 + I_6 = 0 \\ I_1 R_1 + I_5 R_5 - I_4 R_4 = E_1 - E_4 \\ I_2 R_2 + I_6 R_6 - I_5 R_5 = -E_2 \\ I_4 R_4 + I_3 R_3 - I_6 R_6 = E_3 + E_4 \end{array} \right. \quad (19)$$

Систему уравнений (19) необходимо представить в виде $AX = B$, где A - матрица коэффициентов, B - вектор-столбец свободных членов. Матрицы для системы уравнений (19) будут следующими (20):

$$A = \begin{bmatrix} 1 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ R_1 & 0 & 0 & -R_4 & R_5 & 0 \\ 0 & R_2 & 0 & 0 & -R_5 & R_6 \\ 0 & 0 & R_3 & R_4 & 0 & -R_6 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ E_1 - E_4 \\ -E_2 \\ E_3 + E_4 \end{bmatrix} \quad (20)$$

Рассмотрим решение системы уравнений (19) в python при помощи функции `numpy.linalg.solve`.

```
import numpy as np
# Расчёт схемы по правилам Кирхгофа
R1 = 10
R2 = 5
R3 = 15
R4 = 15
R5 = 5
R6 = 15
E1 = 100
```

```

E2 = 50
E3 = 25
E4 = 50
# Матрица коэффициентов
A = np.array([[1,0,-1,1,0,0],
              [-1,1,0,0,1,0],
              [0,0,0,1,1,1],
              [R1,0,0,-R4,R5,0],
              [0,R2,0,0,-R5,R6],
              [0,0,R3,R4,0,-R6]])
# Матрица свободных членов
B = np.array([0,0,0,E1-E4,-E2,E3+E4])
# Решение системы
I = np.linalg.solve(A,B)
# Вывод результатов решения
for k in range(6):
    print('I{:d} = {:.f}'.format(k+1,I[k]))

```

Результат работы программы:

```

I1 = 3.125000
I2 = 0.312500
I3 = 2.812500
I4 = -0.312500
I5 = 2.812500
I6 = -2.500000

```

В задании 2 система уравнений Кирхгофа уже составлена, выводить её не требуется. Решение системы при помощи `numpy.linalg.solve` необходимо дополнить реализацией алгоритма заданного в варианте метода решения СЛАУ и сравнить результаты (должны быть одинаковыми).

Контрольные задания

1. Объясните алгоритм метода Гаусса.
2. Объясните алгоритм метода Крамера.
3. Объясните алгоритм метода LU-разложения.
4. Объясните алгоритм метода простых итераций решения СЛАУ.
5. Создать массив из 10 элементов и заполнить его четными числами начиная с 2. Не использовать ввод элементов вручную.
6. Создать массив из 10 элементов, у которого элементы с нечетными индексами равны 1 а с четными - равны 0. Не использовать ввод элементов вручную.
7. Создать массив целых чисел от 1 до 10 и присвоить 0 первым четырем его элементам. Не использовать ввод элементов вручную.
8. Создать матрицу, не используя ввод элементов вручную:

```
[[5. 0. 0. 0. 0.]  
 [5. 4. 0. 0. 0.]  
 [5. 4. 3. 0. 0.]  
 [5. 4. 3. 2. 0.]  
 [5. 4. 3. 2. 1.]]
```

9. В матрице из задания 4 обнулить (присвоить 0 всем элементам) 3-ю строку.
10. В матрице из задания 4 обнулить (присвоить 0 всем элементам) 3-й столбец.
11. В матрице из задания 4 поменять местами 2-ю и 4-ю строки.
12. В матрице из задания 4 поменять местами 2-й и 4-й столбец.
13. Для элемента $A_{0,0}$ матрицы коэффициентов из задания 4 найти минор $M_{0,0}$.
14. Выполнить транспонирование матрицы из задания 4.
15. Найти собственные числа матрицы из задания 4.

Приложение № 1

Таблица П.1 - Варианты для задания №1

№ вар	Матрица	№ вар	Матрица
1	[[1 0 0 0 0 0 0 0 0 0] [2 2 0 0 0 0 0 0 0 0] [3 3 3 0 0 0 0 0 0 0] [4 4 4 4 0 0 0 0 0 0] [5 5 5 5 5 0 0 0 0 0] [6 6 6 6 6 6 0 0 0 0] [7 7 7 7 7 7 7 0 0 0] [8 8 8 8 8 8 8 8 0 0] [9 9 9 9 9 9 9 9 9 0] [10 10 10 10 10 10 10 10 10 10]]	16	[[0 0 0 0 0 0 0 0 0 10] [0 0 0 0 0 0 0 0 9 10] [0 0 0 0 0 0 0 8 9 10] [0 0 0 0 0 0 7 8 9 10] [0 0 0 0 0 6 7 8 9 10] [0 0 0 0 5 6 7 8 9 10] [0 0 0 4 5 6 7 8 9 10] [0 0 3 4 5 6 7 8 9 10] [0 2 3 4 5 6 7 8 9 10] [1 2 3 4 5 6 7 8 9 10]]
2	[[10 10 10 10 10 10 10 10 10 10] [0 9 9 9 9 9 9 9 9 9] [0 0 8 8 8 8 8 8 8 8] [0 0 0 7 7 7 7 7 7 7] [0 0 0 0 6 6 6 6 6 6] [0 0 0 0 0 5 5 5 5 5] [0 0 0 0 0 0 4 4 4 4] [0 0 0 0 0 0 0 3 3 3] [0 0 0 0 0 0 0 0 2 2] [0 0 0 0 0 0 0 0 0 1]]	17	[[10 9 8 7 6 5 4 3 2 1] [10 9 8 7 6 5 4 3 2 0] [10 9 8 7 6 5 4 3 0 0] [10 9 8 7 6 5 4 0 0 0] [10 9 8 7 6 5 0 0 0 0] [10 9 8 7 6 0 0 0 0 0] [10 9 8 7 0 0 0 0 0 0] [10 9 8 0 0 0 0 0 0 0] [10 9 0 0 0 0 0 0 0 0] [10 0 0 0 0 0 0 0 0 0]]
3	[[1 0 0 0 0 0 0 0 0 0] [1 2 0 0 0 0 0 0 0 0] [1 2 3 0 0 0 0 0 0 0] [1 2 3 4 0 0 0 0 0 0] [1 2 3 4 5 0 0 0 0 0] [1 2 3 4 5 6 0 0 0 0] [1 2 3 4 5 6 7 0 0 0] [1 2 3 4 5 6 7 8 0 0] [1 2 3 4 5 6 7 8 9 0] [1 2 3 4 5 6 7 8 9 10]]	18	[[0 0 0 0 0 0 0 0 0 10] [0 0 0 0 0 0 0 0 9 9] [0 0 0 0 0 0 0 8 8 8] [0 0 0 0 0 0 7 7 7 7] [0 0 0 0 0 6 6 6 6 6] [0 0 0 0 5 5 5 5 5 5] [0 0 0 4 4 4 4 4 4 4] [0 0 3 3 3 3 3 3 3 3] [0 2 2 2 2 2 2 2 2 2] [1 1 1 1 1 1 1 1 1 1]]
4	[[10 9 8 7 6 5 4 3 2 1] [0 9 8 7 6 5 4 3 2 1] [0 0 8 7 6 5 4 3 2 1] [0 0 0 7 6 5 4 3 2 1] [0 0 0 0 6 5 4 3 2 1] [0 0 0 0 0 5 4 3 2 1] [0 0 0 0 0 0 4 3 2 1] [0 0 0 0 0 0 0 3 2 1] [0 0 0 0 0 0 0 0 2 1] [0 0 0 0 0 0 0 0 0 1]]	19	[[1 1 1 1 1 1 1 1 1 1] [2 2 2 2 2 2 2 2 2 0] [3 3 3 3 3 3 3 3 0 0] [4 4 4 4 4 4 4 0 0 0] [5 5 5 5 5 5 0 0 0 0] [6 6 6 6 6 0 0 0 0 0] [7 7 7 7 0 0 0 0 0 0] [8 8 8 0 0 0 0 0 0 0] [9 9 0 0 0 0 0 0 0 0] [10 0 0 0 0 0 0 0 0 0]]

5	[[0 0 0 0 5 0 0 0 0 0] [0 0 0 4 5 0 0 0 0 0] [0 0 3 4 5 0 0 0 0 0] [0 2 3 4 5 0 0 0 0 0] [1 2 3 4 5 0 0 0 0 0] [6 6 6 6 6 6 0 0 0 0] [7 7 7 7 7 7 0 0 0 0] [8 8 8 8 8 8 0 0 0 0] [9 9 9 9 9 9 0 0 0 0] [10 10 10 10 10 10 10 10 10 10]]	20	[[5 5 5 5 5 0 0 0 0 0] [0 4 4 4 4 0 0 0 0 0] [0 0 3 3 3 0 0 0 0 0] [0 0 0 2 2 0 0 0 0 0] [0 0 0 0 1 0 0 0 0 0] [6 6 6 6 6 6 0 0 0 0] [7 7 7 7 7 7 0 0 0 0] [8 8 8 8 8 8 0 0 0 0] [9 9 9 9 9 9 0 0 0 0] [10 10 10 10 10 10 10 10 10 10]]
6	[[1 0 0 0 0 0 0 0 0 0] [2 2 0 0 0 0 0 0 0 0] [3 3 3 0 0 0 0 0 0 0] [4 4 4 4 0 0 0 0 0 0] [5 5 5 5 5 0 0 0 0 0] [6 6 6 6 6 0 0 0 0 10] [7 7 7 7 7 0 0 0 9 10] [8 8 8 8 8 0 0 8 9 10] [9 9 9 9 9 0 7 8 9 10] [10 10 10 10 10 6 7 8 9 10]]	21	[[1 0 0 0 0 0 0 0 0 0] [2 2 0 0 0 0 0 0 0 0] [3 3 3 0 0 0 0 0 0 0] [4 4 4 4 0 0 0 0 0 0] [5 5 5 5 5 0 0 0 0 0] [6 6 6 6 6 10 10 10 10 10] [7 7 7 7 7 0 9 9 9 9] [8 8 8 8 8 0 0 8 8 8] [9 9 9 9 9 0 0 0 7 7] [10 10 10 10 10 0 0 0 0 6]]
7	[[10 9 8 7 6 10 10 10 10 10] [10 9 8 7 0 9 9 9 9 9] [10 9 8 0 0 8 8 8 8 8] [10 9 0 0 0 7 7 7 7 7] [10 0 0 0 0 6 6 6 6 6] [0 0 0 0 0 5 5 5 5 5] [0 0 0 0 0 0 4 4 4 4] [0 0 0 0 0 0 0 3 3 3] [0 0 0 0 0 0 0 0 2 2] [0 0 0 0 0 0 0 0 0 1]]	22	[[6 0 0 0 0 10 10 10 10 10] [7 7 0 0 0 9 9 9 9 9] [8 8 8 0 0 8 8 8 8 8] [9 9 9 9 0 7 7 7 7] [10 10 10 10 10 6 6 6 6 6] [0 0 0 0 0 5 5 5 5 5] [0 0 0 0 0 0 4 4 4 4] [0 0 0 0 0 0 0 3 3 3] [0 0 0 0 0 0 0 0 2 2] [0 0 0 0 0 0 0 0 0 1]]
8	[[10 10 10 10 10 10 10 10 10 10] [0 9 9 9 9 9 9 9 9 9] [0 0 8 8 8 8 8 8 8 8] [0 0 0 7 7 7 7 7 7] [0 0 0 0 6 6 6 6 6] [0 0 0 0 0 5 4 3 2 1] [0 0 0 0 0 5 4 3 2 0] [0 0 0 0 0 5 4 3 0 0] [0 0 0 0 0 5 4 0 0 0] [0 0 0 0 0 5 0 0 0 0]]	23	[[10 10 10 10 10 10 10 10 10 10] [0 9 9 9 9 9 9 9 9 9] [0 0 8 8 8 8 8 8 8 8] [0 0 0 7 7 7 7 7 7] [0 0 0 0 6 6 6 6 6] [0 0 0 0 0 1 0 0 0 0] [0 0 0 0 0 2 2 0 0 0] [0 0 0 0 0 3 3 3 0 0] [0 0 0 0 0 4 4 4 4 0] [0 0 0 0 0 5 5 5 5 5]]
9	[[10 9 8 7 6 1 0 0 0 0] [10 9 8 7 6 2 2 0 0 0] [10 9 8 7 6 3 3 0 0 0] [10 9 8 7 6 4 4 4 0 0] [10 9 8 7 6 5 5 5 5 5] [10 9 8 7 6 0 0 0 0 0] [10 9 8 7 0 0 0 0 0 0] [10 9 8 0 0 0 0 0 0 0] [10 9 0 0 0 0 0 0 0 0] [10 0 0 0 0 0 0 0 0 0]]	24	[[10 9 8 7 6 0 0 0 0 5] [10 9 8 7 6 0 0 0 4 5] [10 9 8 7 6 0 0 3 4 5] [10 9 8 7 6 0 2 3 4 5] [10 9 8 7 6 1 2 3 4 5] [10 9 8 7 6 0 0 0 0 0] [10 9 8 7 0 0 0 0 0 0] [10 9 8 0 0 0 0 0 0 0] [10 9 0 0 0 0 0 0 0 0] [10 0 0 0 0 0 0 0 0 0]]

10	[[10 9 8 7 6 5 4 3 2 1] [10 9 8 7 6 5 4 3 2 0] [10 9 8 7 6 5 4 3 0 0] [10 9 8 7 6 5 4 0 0 0] [10 9 8 7 6 5 0 0 0 0] [6 0 0 0 0 0 0 0 0 0] [7 7 0 0 0 0 0 0 0 0] [8 8 8 0 0 0 0 0 0 0] [9 9 9 9 0 0 0 0 0 0] [10 10 10 10 10 0 0 0 0 0]]	25	[[10 9 8 7 6 5 4 3 2 1] [10 9 8 7 6 5 4 3 2 0] [10 9 8 7 6 5 4 3 0 0] [10 9 8 7 6 5 4 0 0 0] [10 9 8 7 6 5 0 0 0 0] [0 0 0 0 10 0 0 0 0 0] [0 0 0 9 10 0 0 0 0 0] [0 0 8 9 10 0 0 0 0 0] [0 7 8 9 10 0 0 0 0 0] [6 7 8 9 10 0 0 0 0 0]]
11	[[0 0 0 0 0 10 10 10 10 10] [0 0 0 0 0 0 9 9 9 9] [0 0 0 0 0 0 0 8 8 8] [0 0 0 0 0 0 0 0 7 7] [0 0 0 0 0 0 0 0 0 6] [0 0 0 0 5 6 7 8 9 10] [0 0 0 4 5 6 7 8 9 10] [0 0 3 4 5 6 7 8 9 10] [0 2 3 4 5 6 7 8 9 10] [1 2 3 4 5 6 7 8 9 10]]	26	[[0 0 0 0 0 10 9 8 7 6] [0 0 0 0 0 10 9 8 7 0] [0 0 0 0 0 10 9 8 0 0] [0 0 0 0 0 10 9 0 0 0] [0 0 0 0 0 10 0 0 0 0] [0 0 0 0 5 6 7 8 9 10] [0 0 0 4 5 6 7 8 9 10] [0 0 3 4 5 6 7 8 9 10] [0 2 3 4 5 6 7 8 9 10] [1 2 3 4 5 6 7 8 9 10]]
12	[[0 0 0 0 0 0 0 0 0 10] [0 0 0 0 0 0 0 0 9 10] [0 0 0 0 0 0 0 8 9 10] [0 0 0 0 0 0 7 8 9 10] [0 0 0 0 0 6 7 8 9 10] [5 5 5 5 5 6 7 8 9 10] [0 4 4 4 4 6 7 8 9 10] [0 0 3 3 3 6 7 8 9 10] [0 0 0 2 2 6 7 8 9 10] [0 0 0 0 1 6 7 8 9 10]]	27	[[0 0 0 0 0 0 0 0 0 10] [0 0 0 0 0 0 0 0 9 10] [0 0 0 0 0 0 0 8 9 10] [0 0 0 0 0 0 7 8 9 10] [0 0 0 0 0 6 7 8 9 10] [5 4 3 2 1 6 7 8 9 10] [5 4 3 2 0 6 7 8 9 10] [5 4 3 0 0 6 7 8 9 10] [5 4 0 0 0 6 7 8 9 10] [5 0 0 0 0 6 7 8 9 10]]
13	[[10 9 8 7 6 10 10 10 10 10] [10 9 8 7 0 9 9 9 9 9] [10 9 8 0 0 8 8 8 8 8] [10 9 0 0 0 7 7 7 7 7] [10 0 0 0 0 6 6 6 6 6] [0 0 0 0 0 5 5 5 5 5] [0 0 0 0 0 0 4 4 4 4] [0 0 0 0 0 0 0 3 3 3] [0 0 0 0 0 0 0 0 2 2] [0 0 0 0 0 0 0 0 0 1]]	28	[[6 0 0 0 0 10 10 10 10 10] [7 7 0 0 0 9 9 9 9 9] [8 8 8 0 0 8 8 8 8 8] [9 9 9 9 0 7 7 7 7 7] [10 10 10 10 10 6 6 6 6 6] [0 0 0 0 0 5 5 5 5 5] [0 0 0 0 0 0 4 4 4 4] [0 0 0 0 0 0 0 3 3 3] [0 0 0 0 0 0 0 0 2 2] [0 0 0 0 0 0 0 0 0 1]]
14	[[10 10 10 10 10 10 10 10 10 10] [0 9 9 9 9 9 9 9 9 9] [0 0 8 8 8 8 8 8 8 8] [0 0 0 7 7 7 7 7 7 7] [0 0 0 0 6 6 6 6 6 6] [0 0 0 0 0 5 4 3 2 1] [0 0 0 0 0 5 4 3 2 0] [0 0 0 0 0 5 4 3 0 0] [0 0 0 0 0 5 4 0 0 0] [0 0 0 0 0 5 0 0 0 0]]	29	[[10 10 10 10 10 10 10 10 10 10] [0 9 9 9 9 9 9 9 9 9] [0 0 8 8 8 8 8 8 8 8] [0 0 0 7 7 7 7 7 7 7] [0 0 0 0 6 6 6 6 6 6] [0 0 0 0 0 1 0 0 0 0] [0 0 0 0 0 2 2 0 0 0] [0 0 0 0 0 3 3 3 0 0] [0 0 0 0 0 4 4 4 4 0] [0 0 0 0 0 5 5 5 5 5]]

15	[[0 0 0 0 0 10 10 10 10 10 10] [0 0 0 9 10 9 9 9 9 9 9] [0 0 8 9 10 8 8 8 8 8 8] [0 7 8 9 10 7 7 7 7 7 7] [6 7 8 9 10 6 6 6 6 6 6] [0 0 0 0 0 5 5 5 5 5 5] [0 0 0 0 0 0 4 4 4 4 4] [0 0 0 0 0 0 0 3 3 3 3] [0 0 0 0 0 0 0 0 2 2 2] [0 0 0 0 0 0 0 0 0 0 1]]	30	[[10 10 10 10 10 10 10 10 10 10 10] [0 9 9 9 9 9 9 9 9 9 9] [0 0 8 8 8 8 8 8 8 8 8] [0 0 0 7 7 7 7 7 7 7 7] [0 0 0 0 6 6 6 6 6 6 6] [0 0 0 0 0 0 0 0 0 0 5] [0 0 0 0 0 0 0 0 0 4 5] [0 0 0 0 0 0 0 0 3 4 5] [0 0 0 0 0 0 2 3 4 5] [0 0 0 0 0 1 2 3 4 5]]
----	--	----	--

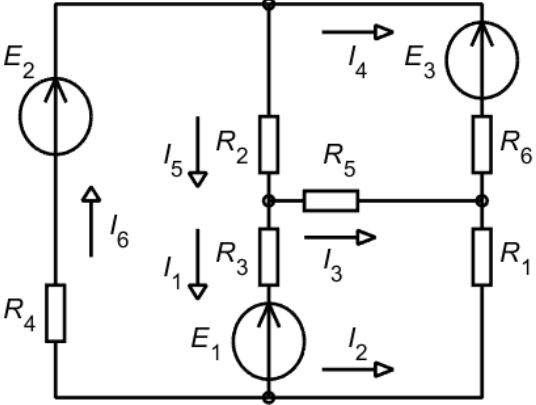
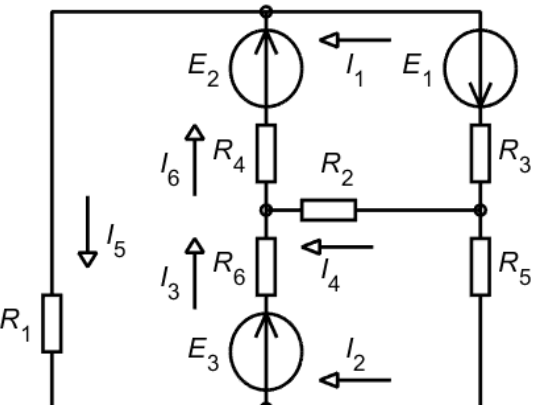
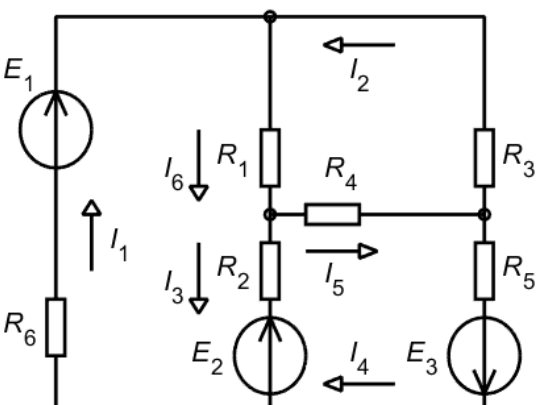
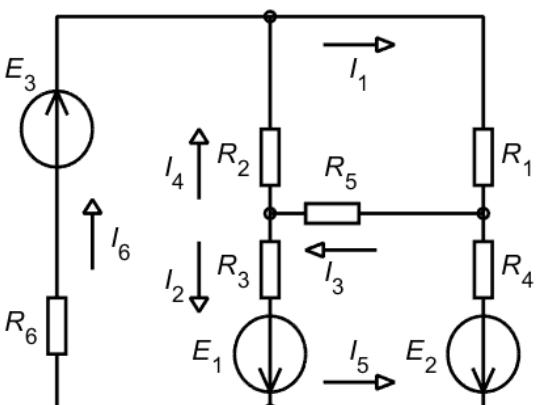
Приложение № 2

Таблица П.2 - Схемы, системы уравнений и метод решения для задания №2

№ вар.	Электрическая схема	Численный метод и система уравнений для решения
1		<p>Гаусса</p> $\begin{cases} I_1 + I_5 - I_3 = 0 \\ -I_5 - I_6 + I_4 = 0 \\ I_6 - I_2 - I_1 = 0 \\ -I_1 R_5 + I_5 R_6 - I_6 R_4 = -E_1 \\ -I_5 R_6 - I_3 R_1 - I_4 R_3 = E_3 \\ -I_4 R_3 - I_2 R_2 - I_6 R_4 = E_3 + E_2 - E_1 \end{cases}$
2		<p>Крамера</p> $\begin{cases} -I_5 - I_3 + I_6 = 0 \\ I_3 + I_4 + I_1 = 0 \\ -I_4 - I_2 + I_5 = 0 \\ -I_5 R_5 + I_3 R_3 - I_4 R_6 = -E_3 - E_1 \\ I_3 R_3 + I_6 R_4 - I_1 R_2 = -E_2 \\ -I_1 R_2 - I_2 R_1 + I_4 R_6 = E_1 \end{cases}$
3		<p>LU-разложение</p> $\begin{cases} I_4 - I_5 + I_3 = 0 \\ I_5 + I_2 + I_1 = 0 \\ -I_2 - I_6 - I_4 = 0 \\ I_4 R_2 + I_5 R_1 - I_2 R_6 = E_3 + E_1 - E_2 \\ -I_5 R_1 - I_3 R_3 + I_1 R_4 = -E_1 \\ I_1 R_4 + I_6 R_5 - I_2 R_6 = -E_2 \end{cases}$

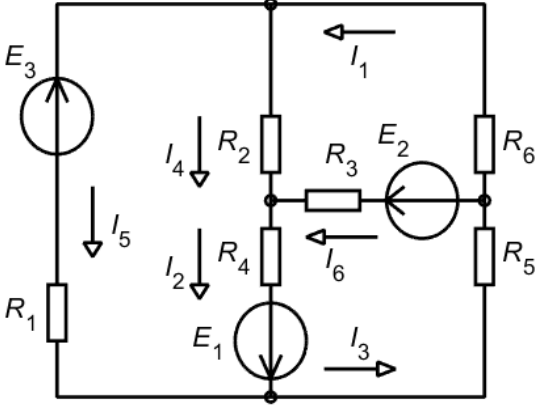
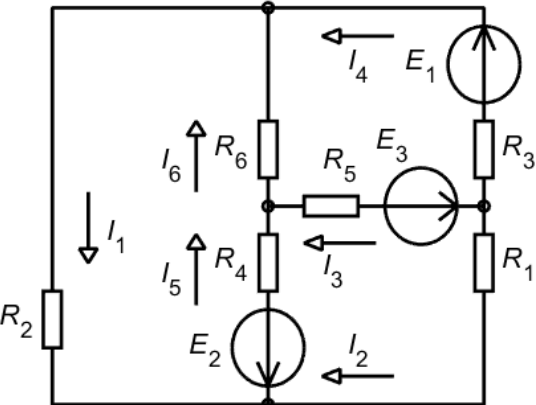
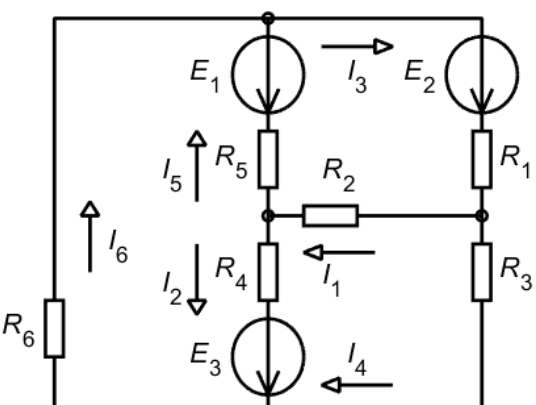
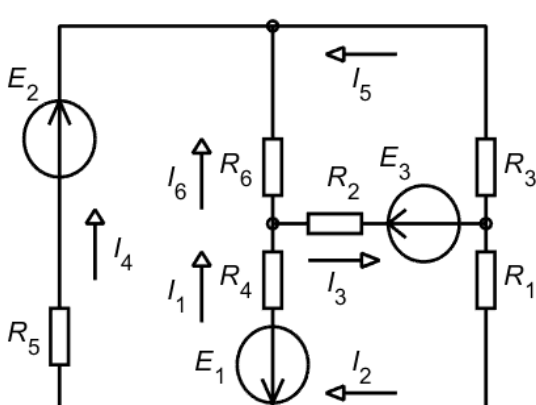
4		<p>Через обратную матрицу</p> $\begin{cases} I_2 - I_5 - I_6 = 0 \\ I_5 + I_3 - I_1 = 0 \\ -I_3 - I_4 - I_2 = 0 \\ I_2 R_5 + I_5 R_1 - I_3 R_4 = -E_3 + E_2 \\ -I_5 R_1 + I_6 R_2 - I_1 R_6 = E_3 - E_1 \\ I_1 R_6 - I_4 R_3 + I_3 R_4 = -E_2 \end{cases}$
5		<p>Метод простых итераций</p> $\begin{cases} -I_3 + I_2 - I_6 = 0 \\ -I_2 + I_5 - I_4 = 0 \\ -I_5 - I_1 + I_3 = 0 \\ -I_3 R_1 - I_2 R_2 - I_5 R_4 = -E_2 \\ I_2 R_2 + I_6 R_3 - I_4 R_6 = E_3 \\ -I_4 R_6 + I_1 R_5 - I_5 R_4 = E_1 \end{cases}$
6		<p>Гаусса</p> $\begin{cases} -I_4 - I_5 + I_6 = 0 \\ I_5 + I_1 - I_2 = 0 \\ -I_1 - I_3 + I_4 = 0 \\ I_4 R_6 - I_5 R_4 + I_1 R_3 = E_2 + E_3 \\ I_5 R_4 + I_6 R_2 + I_2 R_5 = 0 \\ -I_2 R_5 + I_3 R_1 - I_1 R_3 = E_1 - E_3 \end{cases}$

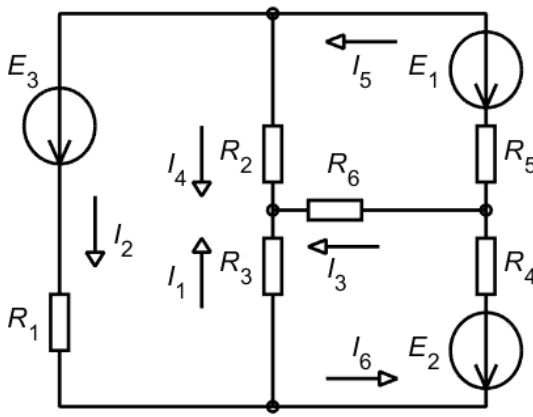
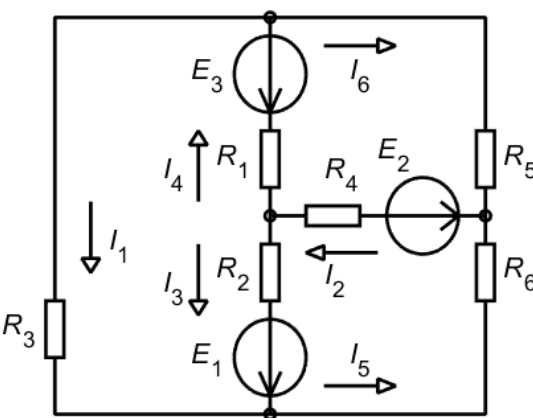
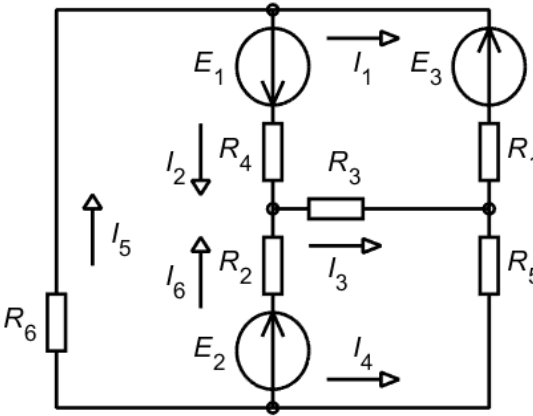
7		<p>Крамера</p> $\begin{cases} I_3 + I_5 - I_4 = 0 \\ -I_5 - I_1 - I_2 = 0 \\ I_1 - I_6 - I_3 = 0 \\ -I_3 R_4 + I_5 R_2 - I_1 R_5 = E_3 \\ I_5 R_2 + I_4 R_3 - I_2 R_1 = E_1 + E_2 \\ -I_2 R_1 + I_6 R_6 + I_1 R_5 = E_2 \end{cases}$
8		<p>LU-разложение</p> $\begin{cases} -I_2 + I_4 - I_1 = 0 \\ -I_4 - I_6 - I_5 = 0 \\ I_6 - I_3 + I_2 = 0 \\ I_2 R_4 + I_4 R_5 - I_6 R_2 = -E_1 - E_3 + E_2 \\ -I_4 R_5 - I_1 R_3 + I_5 R_6 = E_3 \\ -I_5 R_6 - I_3 R_1 + I_6 R_2 = -E_2 \end{cases}$
9		<p>Через обратную матрицу</p> $\begin{cases} -I_2 + I_1 - I_4 = 0 \\ -I_1 - I_5 + I_6 = 0 \\ I_5 - I_3 + I_2 = 0 \\ -I_2 R_4 - I_1 R_6 + I_5 R_2 = -E_3 \\ -I_1 R_6 - I_4 R_1 - I_6 R_3 = -E_3 + E_1 \\ I_6 R_3 - I_3 R_5 + I_5 R_2 = -E_1 + E_2 \end{cases}$
10		<p>Метод простых итераций</p> $\begin{cases} I_5 - I_1 - I_3 = 0 \\ I_1 - I_2 + I_6 = 0 \\ I_2 - I_4 - I_5 = 0 \\ I_5 R_5 + I_1 R_3 + I_2 R_2 = E_1 - E_3 \\ I_1 R_3 - I_3 R_4 - I_6 R_6 = E_1 - E_2 \\ -I_6 R_6 - I_4 R_1 - I_2 R_2 = -E_2 + E_3 \end{cases}$

11		<p>Гаусса</p> $\begin{cases} I_6 - I_5 - I_4 = 0 \\ I_5 - I_1 - I_3 = 0 \\ I_1 - I_2 - I_6 = 0 \\ I_6 R_4 + I_5 R_2 + I_1 R_3 = E_2 - E_1 \\ I_5 R_2 - I_4 R_6 + I_3 R_5 = E_3 \\ I_3 R_5 - I_2 R_1 - I_1 R_3 = E_1 \end{cases}$
12		<p>Крамера</p> $\begin{cases} -I_5 + I_6 + I_1 = 0 \\ -I_6 + I_3 + I_4 = 0 \\ -I_3 - I_2 + I_5 = 0 \\ I_5 R_1 + I_6 R_4 + I_3 R_6 = E_2 + E_3 \\ I_6 R_4 - I_1 R_3 + I_4 R_2 = E_2 + E_1 \\ I_4 R_2 - I_2 R_5 - I_3 R_6 = -E_3 \end{cases}$
13		<p>LU-разложение</p> $\begin{cases} I_1 - I_6 + I_2 = 0 \\ I_6 - I_3 - I_5 = 0 \\ I_3 - I_4 - I_1 = 0 \\ I_1 R_6 + I_6 R_1 + I_3 R_2 = E_1 - E_2 \\ I_6 R_1 + I_2 R_3 + I_5 R_4 = 0 \\ I_5 R_4 + I_4 R_5 - I_3 R_2 = E_3 + E_2 \end{cases}$
14		<p>Через обратную матрицу</p> $\begin{cases} I_6 + I_4 - I_1 = 0 \\ -I_4 - I_2 + I_3 = 0 \\ I_2 - I_5 - I_6 = 0 \\ I_6 R_6 - I_4 R_2 + I_2 R_3 = E_3 + E_1 \\ -I_4 R_2 - I_1 R_1 - I_3 R_5 = 0 \\ -I_3 R_5 - I_5 R_4 - I_2 R_3 = E_2 - E_1 \end{cases}$

15		<p>Метод простых итераций</p> $\begin{cases} I_3 - I_5 - I_2 = 0 \\ I_5 - I_1 + I_4 = 0 \\ I_1 - I_6 - I_3 = 0 \\ -I_3 R_4 - I_5 R_1 - I_1 R_3 = E_3 \\ I_5 R_1 - I_2 R_5 - I_4 R_2 = -E_2 + E_1 \\ -I_4 R_2 + I_6 R_6 - I_1 R_3 = E_1 + E_3 \end{cases}$
16		<p>Гаусса</p> $\begin{cases} -I_5 + I_3 - I_6 = 0 \\ -I_3 - I_4 + I_1 = 0 \\ I_4 - I_2 + I_5 = 0 \\ -I_5 R_4 - I_3 R_6 + I_4 R_3 = 0 \\ I_3 R_6 + I_6 R_1 + I_1 R_2 = -E_2 + E_3 \\ -I_1 R_2 + I_2 R_5 - I_4 R_3 = -E_3 - E_1 \end{cases}$
17		<p>Крамера</p> $\begin{cases} -I_5 + I_1 + I_6 = 0 \\ -I_1 - I_4 - I_3 = 0 \\ I_4 - I_2 + I_5 = 0 \\ -I_5 R_4 - I_1 R_5 + I_4 R_6 = E_1 + E_2 \\ -I_1 R_5 + I_6 R_3 + I_3 R_2 = 0 \\ -I_3 R_2 + I_2 R_1 + I_4 R_6 = E_3 + E_2 \end{cases}$
18		<p>LU-разложение</p> $\begin{cases} I_1 - I_5 - I_3 = 0 \\ I_5 + I_2 + I_6 = 0 \\ -I_2 - I_4 - I_1 = 0 \\ I_1 R_5 + I_5 R_3 - I_2 R_6 = E_3 - E_2 \\ -I_5 R_3 + I_3 R_2 + I_6 R_4 = 0 \\ -I_6 R_4 + I_4 R_1 + I_2 R_6 = E_1 + E_2 \end{cases}$

19		<p>Через обратную матрицу</p> $\begin{cases} I_3 - I_2 - I_6 = 0 \\ I_2 + I_5 - I_4 = 0 \\ -I_5 - I_1 - I_3 = 0 \\ -I_3 R_3 - I_2 R_4 + I_5 R_5 = 0 \\ -I_2 R_4 + I_6 R_6 - I_4 R_1 = -E_2 - E_1 \\ I_4 R_1 + I_1 R_2 + I_5 R_5 = E_1 + E_3 \end{cases}$
20		<p>Метод простых итераций</p> $\begin{cases} I_3 + I_2 + I_6 = 0 \\ -I_2 + I_1 + I_5 = 0 \\ -I_1 - I_4 - I_3 = 0 \\ I_3 R_5 - I_2 R_6 - I_1 R_2 = E_3 + E_2 \\ I_2 R_6 - I_6 R_3 + I_5 R_4 = -E_1 \\ -I_5 R_4 - I_4 R_1 + I_1 R_2 = -E_2 \end{cases}$
21		<p>Гаусса</p> $\begin{cases} -I_2 - I_1 + I_6 = 0 \\ I_1 - I_3 - I_5 = 0 \\ I_3 - I_4 + I_2 = 0 \\ -I_2 R_5 + I_1 R_3 + I_3 R_6 = -E_1 \\ -I_1 R_3 - I_6 R_1 - I_5 R_2 = -E_3 \\ I_5 R_2 - I_4 R_4 - I_3 R_6 = E_3 - E_2 + E_1 \end{cases}$

22		<p>Крамера</p> $\begin{cases} -I_5 - I_4 + I_1 = 0 \\ I_4 - I_2 + I_6 = 0 \\ I_2 - I_3 + I_5 = 0 \\ I_5 R_1 - I_4 R_2 - I_2 R_4 = -E_3 - E_1 \\ I_4 R_2 + I_1 R_6 - I_6 R_3 = -E_2 \\ -I_6 R_3 - I_3 R_5 - I_2 R_4 = -E_2 - E_1 \end{cases}$
23		<p>LU-разложение</p> $\begin{cases} -I_1 + I_6 + I_4 = 0 \\ -I_6 + I_5 + I_3 = 0 \\ -I_5 - I_2 + I_1 = 0 \\ -I_1 R_2 - I_6 R_6 - I_5 R_4 = E_2 \\ I_6 R_6 - I_4 R_3 + I_3 R_5 = -E_1 - E_3 \\ -I_3 R_5 + I_2 R_1 + I_5 R_4 = E_3 - E_2 \end{cases}$
24		<p>Через обратную матрицу</p> $\begin{cases} I_6 + I_5 - I_3 = 0 \\ -I_5 - I_2 + I_1 = 0 \\ I_2 - I_4 - I_6 = 0 \\ I_6 R_6 - I_5 R_5 + I_2 R_4 = E_1 + E_3 \\ -I_5 R_5 - I_3 R_1 - I_1 R_2 = E_1 - E_2 \\ I_1 R_2 - I_4 R_3 + I_2 R_4 = E_3 \end{cases}$
25		<p>Метод простых итераций</p> $\begin{cases} I_4 + I_6 + I_5 = 0 \\ -I_6 + I_1 - I_3 = 0 \\ -I_1 - I_2 - I_4 = 0 \\ I_4 R_5 - I_6 R_6 - I_1 R_4 = E_2 + E_1 \\ -I_6 R_6 + I_5 R_3 + I_3 R_2 = -E_3 \\ I_3 R_2 + I_2 R_1 + I_1 R_4 = -E_3 - E_1 \end{cases}$

26		<p>Гаусса</p> $\begin{cases} -I_2 - I_4 + I_5 = 0 \\ I_4 + I_1 + I_3 = 0 \\ -I_1 - I_6 + I_2 = 0 \\ -I_2 R_1 + I_4 R_2 - I_1 R_3 = -E_3 \\ I_4 R_2 + I_5 R_5 - I_3 R_6 = -E_1 \\ -I_3 R_6 - I_6 R_4 + I_1 R_3 = E_2 \end{cases}$
27		<p>Крамера</p> $\begin{cases} -I_1 + I_4 - I_6 = 0 \\ -I_4 - I_3 + I_2 = 0 \\ I_3 - I_5 + I_1 = 0 \\ I_1 R_3 + I_4 R_1 - I_3 R_2 = -E_3 - E_1 \\ I_4 R_1 + I_6 R_5 + I_2 R_4 = -E_3 - E_2 \\ I_2 R_4 + I_5 R_6 + I_3 R_2 = -E_2 + E_1 \end{cases}$
28		<p>LU-разложение</p> $\begin{cases} I_5 - I_2 - I_1 = 0 \\ I_2 + I_6 - I_3 = 0 \\ -I_6 - I_4 - I_5 = 0 \\ -I_5 R_6 - I_2 R_4 + I_6 R_2 = -E_1 + E_2 \\ -I_2 R_4 + I_1 R_1 - I_3 R_3 = -E_1 - E_3 \\ I_3 R_3 - I_4 R_5 + I_6 R_2 = E_2 \end{cases}$

29		<p>Через обратную матрицу</p> $\begin{cases} -I_1 + I_6 + I_5 = 0 \\ -I_6 - I_2 - I_4 = 0 \\ I_2 - I_3 + I_1 = 0 \\ I_1 R_5 + I_6 R_4 - I_2 R_3 = -E_3 + E_2 \\ -I_6 R_4 + I_5 R_2 + I_4 R_1 = E_1 \\ I_4 R_1 - I_3 R_6 - I_2 R_3 = E_1 + E_2 \end{cases}$
30		<p>Метод простых итераций</p> $\begin{cases} I_5 - I_6 + I_4 = 0 \\ I_6 + I_1 - I_3 = 0 \\ -I_1 - I_2 - I_5 = 0 \\ I_5 R_6 + I_6 R_2 - I_1 R_4 = E_1 - E_2 \\ -I_6 R_2 - I_4 R_5 - I_3 R_1 = E_3 \\ I_3 R_1 + I_2 R_3 + I_1 R_4 = -E_3 + E_2 \end{cases}$

Таблица П.3 - Значения ЭДС и сопротивлений к вариантам в таблице П.2

№ вар	R_1 , Ом	R_2 , Ом	R_3 , Ом	R_4 , Ом	R_5 , Ом	R_6 , Ом	E_1 , В	E_2 , В	E_3 , В
1	15	15	15	10	15	25	80	105	130
2	10	30	15	20	25	30	80	75	15
3	10	15	35	25	5	25	70	95	115
4	25	10	20	30	10	30	80	110	95
5	10	10	20	10	10	20	70	85	40
6	25	10	30	20	15	15	60	55	40
7	5	10	20	5	10	15	30	75	95
8	10	35	20	30	25	10	110	90	25
9	20	10	30	5	30	20	20	105	95

10	15	30	10	5	20	10	85	25	15
11	15	15	15	10	10	15	70	45	40
12	15	25	25	25	30	35	125	25	15
13	25	25	20	5	25	25	80	35	120
14	15	15	25	30	15	15	10	70	125
15	15	20	30	25	25	20	80	15	105
16	10	30	15	5	30	20	30	60	90
17	30	25	30	20	20	20	85	65	35
18	15	20	10	15	10	10	40	50	20
19	10	30	20	20	10	20	100	5	80
20	10	10	10	20	10	10	40	100	30
21	15	10	15	5	10	15	60	95	45
22	10	10	10	15	10	15	20	125	45
23	15	10	10	20	10	20	95	120	105
24	20	35	25	15	25	30	100	115	85
25	5	10	15	25	10	5	60	95	25
26	10	20	20	20	30	15	115	95	10
27	30	10	30	25	30	35	115	20	15
28	35	5	25	5	25	5	110	85	80
29	25	10	30	5	35	35	115	70	105
30	10	5	10	10	20	5	35	80	45

Приложение № 3

Требования к программе для решения второго задания ЛР (решение СЛАУ одним из численных методов)

1. Алгоритм должен быть применим к любым СЛАУ с произвольным количеством уравнений. Размерность задачи N должна определяться в начале алгоритма путем измерения размера (стороны) матрицы коэффициентов СЛАУ.
2. Не использовать более двух вложенных циклов. Если нужен третий вложенный цикл, заменить его какой-либо встроенной функцией python. Например, в алгоритме перемножения матриц суммирование произведений элементов i -й строки на j -й столбец выполнять при помощи поэлементных операций и функции `numpy.sum`.
3. В программах можно использовать следующие функции:
 - 3.1. `numpy.column_stack` - добавление столбца к матрице;
 - 3.2. `numpy.zeros` - создание матрицы, заполненной нулями;
 - 3.3. `numpy.ones` - заполнение матрицы единицами;
 - 3.4. `len` - определение размера массива/матрицы;
 - 3.5. `numpy.delete` - удаление из матрицы строки или столбца;
 - 3.6. `numpy.sum` - суммирование элементов массива;
 - 3.7. `numpy.transpose` - транспонирование матрицы;
 - 3.8. `numpy.matmul` - умножение матриц;
 - 3.9. `numpy.linalg.eigvals` - расчет собственных чисел матрицы.
4. Для всех остальных функций и алгоритмов, необходимых для решения задачи, написать программный код самостоятельно.